1 The Inverse Element

1.1 Inverse Elements

In a group, any element a has an inverse a', such that $a \cdot a' = e$, where e is the neutral element of that group. We have already met some groups, permutation groups and finite groups of modular arithmetic. We will now look at infinite groups, namely the additive group over $\mathbb{Z}$, the integers, and the multiplicative group over $\mathbb{Q}$, the rationals.

As you may have guessed already, the inverse element of a natural number n in the additive group over integers is the *negation* of n, $-n$. It is easy to see that n plus its inverse, $-n$, is just 0, the neutral element of addition: $n + (-n) = n - n = 0$. Since it is, of course, also true that $-n + n = n - n = 0$, the inverse element of a negative number $-n$ is its positive counterpart n. With the law of double negation, $-(-n) = n$, that is the negation of the negation of n is n, the formula $n + (-n) = 0$ is universally true for both positive and negative numbers. When we are talking about n, the positive, we just have : $n + (-n) = n - n = 0$. When we are talking about $-n$, the negative, we have: $-n + (-(-n)) = -n + n = n - n = 0$.

Note that with this identity, we can solve any equation of the form $a + x = b$ in the additive group of $\mathbb{Z}$. We just add the inverse of a to both sides of the equation: $a + (-a) + x = b + (-a)$, which, of course, is $x = b - 1$. Without negative numbers, there were equations we could not solve in this way, for instance $4 + x = 3$. We had gaps, so to speak, in our additive equations. But now, in the group over integers, there are no such gaps anymore: we just add the inverse of 4 on both sides and get $x = 3 - 4 = -1$.

What is the inverse element of integers in the multiplicative group? Well, in this group, it must still hold that $a \cdot a' = e$, where $\cdot$ is multiplication and e, the neutral element, is unity. We, hence, have $a \times a' = 1$. We easily find a solution with division. We divide a on both sides and get $a' = \frac{1}{a}$ and that is the answer: the inverse element of a natural number n in the multiplicative group is $\frac{1}{n}$.

We see that we now can solve any multiplicative equation of the form $ax = b$, just by multiplying the inverse of a on both sides: $ax\frac{1}{a} = b\frac{1}{a}$, which is $x\frac{a}{a} = \frac{b}{a}$. The left-hand side reduces to $1x = x$ and we have $x = \frac{b}{a}$. As we have already seen in finite groups, prime numbers simply "disappear" with fractions, since we can now reach them with multiplication, for instance: $3x = 5$ is equivalent to $x = \frac{5}{3}$.

Let us go on and build something bigger: a field consisting of addition and multiplication

where the distributive law holds. We already know that the distributive law holds in the world of natural numbers: $a(b+c) = ab + ac$. But what, when we have creatures like $-n$ and $\frac{1}{n}$? We obviously need rules to add fractions and to negate products.

Let us start with fractions. We may add fractions with the same denominator by simply adding the numerators, *i.e.* $\frac{a}{n} + \frac{b}{n} = \frac{a+b}{n}$. With different denominators, we first have to manipulate the fractions in a way that they have the same denominator, but without changing their value. The simplest way to do that is to multiply one denominator by the other and to multiply the numerators correspondingly: $\frac{a}{n} + \frac{b}{m} = \frac{am}{nm} + \frac{bn}{mn} = \frac{am+bn}{mn}$.

We can reduce the computational complexity of this operation in most cases by using the *lcm* instead of the product nm. As you may remember, the *lcm* of two numbers, a and b is $\frac{ab}{\gcd(a,b)}$. We would still multiply the numerator by the value by which the denominator changes, *i.e.* the *lcm* divided by the denominator and would get

$$\frac{a}{n} + \frac{b}{m} = \frac{a \times \frac{lcm(n,m)}{n} + b \times \frac{lcm(n,m)}{m}}{lcm(n,m)}. \tag{1.1}$$

That looks complicated, but is simple when we take two concrete numbers: $\frac{1}{6} + \frac{2}{9}$. We first compute the $lcm(6,9)$ as

$$\frac{6 \times 9}{\gcd(6,9)}.$$

The gcd of 6 and 9 is 3, the product $6 \times 9 = 54$ and $54/3 = 18$. So, we have:

$$\frac{1 \times \frac{18}{6} + 2 \times \frac{18}{9}}{lcm(6,9) = 18} = \frac{3+4}{18} = \frac{7}{18}.$$

It looks simpler now, but still it seems that we do need more steps than by just multiplying the respective other denominator to numerator and denominator of both fractions. However, when we do this, we have to operate with greater numbers and, at the end, reduce the fractions to their *canonical form*, which is

$$\frac{a}{b} = \frac{a/\gcd(a,b)}{b/\gcd(a,b)}. \tag{1.2}$$

For the example, this would mean

$$\frac{1 \times 9}{6 \times 9} + \frac{2 \times 6}{9 \times 6} = \frac{9}{54} + \frac{12}{54} = \frac{21}{54}.$$

Now, we reduce to canonical form:

$$\frac{21/\gcd(21,54)}{54/\gcd(21,54)},$$

which is

$$\frac{\frac{21}{3}=7}{\frac{54}{3}=18}.$$

How to multiply negative numbers of the form $-a(b+c)$? Let us look at an example: the additive inverse of 6 is -6. We would therefore expect $-6+6=0$. Furthermore, we have $2\times 3=6$. Now, if we add the inverse of 3 to 3 once, we get 0: $-3+3=0$. What should we get, if we add the inverse of 3 twice to twice 3, *i.e.* $2\times 3+2\times -3$? We expect it to be 0, correct? Therefore and since 2×3 is 6, 2×-3 must be the inverse of 6, hence, -6.

The same is true, the other way round, thus $2\times 3+(-2)\times 3=0$. That means that we can move the minus sign in a product, such that $a\times -b=-a\times b$. To any such product we can simply add the factor 1 without changing the result: $a\times b=1\times a\times b$. We, therefore, have $1\times a\times -b=1\times -a\times b=-1\times a\times b$. This facilitates life a bit: we can handle one minus sign as the additional factor -1. In other words, multiplying by -1 has the same effect has negating: $-1n=-n$.

Going back to the question of how to handle products of the form $-a(b+c)$, we now can say that $-a(b+c)=-1a(b+c)$. Multiplying a out in terms of the distributive law, we get: $-1(ab+ac)$. Now, we multiply -1, just as we did above: $(-1)ab+(-1)ac$ and, since we know that $-1n=-n$, we derive $-ab-ac$.

What if we have more than one minus sign in a product like, for instance: -2×-3? We saw above that $2\times -3+2\times 3=0$ and we can reformulate this as $-1\times 2\times 3+2\times 3=0$. Now, if we have two negative factors, we add one more minus sign, *i.e.* one more factor -1: $-1\times -1\times 2\times 3+2\times 3=$? We just substitute one -1 after the other by negation. We first get $-1\times -(2\times 3)\ldots$ and then $-(-(2\times 3))$. We now see clearly that this should be the negation, *i.e.* the inverse of $-(2\times 3)$. The inverse of $-(2\times 3)$, however, is just 2×3 and that is 6. Therefore: $-2\times -3+2\times 3=12$.

So, what happens if we multiply negative numbers and fractions? When we just follow multiplication rules we get $-1\times\frac{1}{n}=\frac{-1}{n}$. We, hence, would say, according to the rules derived above, that $\frac{-1}{n}$ is the additive inverse of $\frac{1}{n}$. What about $\frac{1}{-n}$? This should be the multiplicative inverse of $-n$, such that $-n\times\frac{1}{-n}=1$. We again can move the minus sign around to get $-1\times n\times\frac{1}{-n}$ leading to $-1\times\frac{n}{-n}=1$. Dividing n on both sides gives $-1\times\frac{1}{-n}=\frac{1}{n}$ and multiplying -1 gives $\frac{1}{-n}=\frac{-1}{n}$. In other words, a fraction with a minus sign in it, independently of where it appears, in the numerator or the denominator, is the additive inverse of the same fraction without the minus sign: $\frac{a}{b}+\frac{-a}{b}=\frac{a}{b}+\frac{a}{-b}=\frac{a}{b}-\frac{a}{b}=0$.

In summary, we can define a set of rules on multiplication that are independent of whatever a and b are:

$$\begin{array}{rcrcr} a & \times & b & = & ab \\ -a & \times & -b & = & ab \\ -a & \times & b & = & -ab \\ a & \times & -b & = & -ab \end{array}$$

With this, we have established an important theoretical result, namely that $\mathbb{Q}$, including negative numbers, is an infinite field with addition and multiplication. But let us go on. There are still operations we have seen for natural numbers combining multiplication and addition that may and should have an interpretation with integers and fractions, namely exponentiation.

From the table above, we see immediately that products with an even number of negative numbers are positive – a fact that we already used when discussing prime numbers. In general, any number raised to an even exponent is positive independent of that number being positive or negative.

This leads to a difficulty with the root operation, since even roots may have two different results: a positive number or its additive inverse. For instance, $\sqrt{4}$ could be 2 and -2. Even further, the operation cannot be applied to a negative number: $\sqrt{-1}$ has no meaning – at least not with the creatures we have met so far. There may be an object i that fulfil equations of the form $i = \sqrt{-1}$, but such objects are beyond our imagination at this point in time.

Now, what is the effect of having negative numbers or fractions in the exponent?

We will first look at fractions as exponents and investigate powers of the form $a^{\frac{1}{n}}$. To clarify the meaning of such expression, we will use the rules we know so far to observe what happens, when we multiply two powers with the same base and with fractional exponents:

$$x = a^{\frac{1}{n}} \times a^{\frac{1}{m}} = a^{\frac{1}{n}+\frac{1}{m}}. \tag{1.3}$$

Let us look at the special case of the exponent $\frac{1}{2}$:

$$x = a^{\frac{1}{2}} \times a^{\frac{1}{2}} = a^{\frac{1}{2}+\frac{1}{2}}. \tag{1.4}$$

Obviously, $\frac{1}{2} + \frac{1}{2} = 2 \times \frac{1}{2} = 1$. In other words, x, in this case, is just a. Furthermore, we see that there is a number $n = a^{\frac{1}{2}}$, such that $n \times n = n^2 = x$. For which number does

this hold? Well, it is just the definition of the square root: $\sqrt{x} \times \sqrt{x} = (\sqrt{x})^2 = x$. We would conclude that $a^{\frac{1}{n}}$ is equivalent to $\sqrt[n]{a}$:

$$a^{\frac{1}{n}} = \sqrt[n]{a}. \tag{1.5}$$

This, in fact, makes a lot of sense. We would expect, for instance, that $(a^{\frac{1}{n}})^n$ is just a, since $(\sqrt[n]{a})^n = a$. When we multiply it out, we get indeed $a^{\frac{1}{n} \times n} = a^1 = a$. We would also expect that $(a^{\frac{1}{n}})^{\frac{1}{m}}$ is $\sqrt[m]{\sqrt[n]{a}} = \sqrt[mn]{a}$, *e.g.* $(a^{\frac{1}{2}})^{\frac{1}{2}} = \sqrt{\sqrt{a}} = \sqrt[4]{a}$. When we multiply it out again, we indeed obtain $a^{\frac{1}{2} \times \frac{1}{2}} = a^{\frac{1}{4}}$.

Now, what about negative exponents? We adopt the same technique, *i.e.* we multiply two powers with the same base:

$$a \times a^{-1}.$$

We can write this as $a^1 \times a^{-1}$ and this is

$$a^1 \times a^{-1} = a^{1-1} = a^0 = 1.$$

We see a^{-1} is the multiplicative inverse of a. But we already know that the inverse of a is $\frac{1}{a}$. We conclude that

$$a^{-n} = \frac{1}{a^n}. \tag{1.6}$$

This conjecture would imply that $a^{-n} \times a^n = 1$, since $\frac{1}{a^n} \times a^n = 1$ and, indeed: $a^{-n} \times a^n = a^{-n+n} = a^0 = 1$. It would also imply that $(a^{-n})^{-1} = a^n$, since $a^{-n} = \frac{1}{a^n}$, whose inverse is a^n. Indeed, we have $(a^{-n})^{-1} = a^{-n \times -1} = a^n$.

A side effect of this rule is that we now have a very nice notation for the multiplicative inverse. Until now, we have used the symbol a' to denote the inverse of a. Since $'$ is also used in other contexts, the notation a^{-1} is much clearer and we will stick to it from now on.

1.2 $\mathbb{Z}$

We will now implement a number type that takes signedness into account. We will do so in a way that allows us to negate objects of different kind, basically any type of number. We therefore start by defining a parametrised data type:

$$\textbf{data}\ \mathit{Signed}\ a = \mathit{Pos}\ a \mid \mathit{Neg}\ a$$
$$\quad \textbf{deriving}\ (\mathit{Eq}, \mathit{Show})$$

The data type has two constructors, *Pos* and *Neg* for a positive and a negative a respectivley. The expression **let** $x = \mathit{Neg}\ 1$ would assign the value -1 to x. We would instantiate a concrete data type by giving a concrete type for the type parameter, *e.g.*

$$\textbf{type}\ \mathit{Zahl} = \mathit{Signed}\ \mathit{Natural}$$

This type is called *Zahl*, the German word for *number*, which was used for the designation of the set $\mathbb{Z}$ of the integers. When Abstract Algebra started to be a major field of mathematics, the University of Göttingen was the gravitational centre of the math world and, since it was not yet common to use English in scientific contributions, many German words slipped into math terminology.

For convenience, we add a direct conversion from *Zahl* to *Natural*:

$$\mathit{z2n} :: \mathit{Zahl} \to \mathit{Natural}$$
$$\mathit{z2n}\ (\mathit{Pos}\ n) = n$$
$$\mathit{z2n}\ (\mathit{Neg}\ \_) = \bot$$

Another convenience function should be defined for *Neg*, namely to guarantee that 0 is always positive. Otherwise, we could run into situations where we compare *Pos* 0 and *Neg* 0 and obtain a difference that does not exist. We therefore define

$$\mathit{neg0} :: (\mathit{Eq}\ a, \mathit{Num}\ a) \Rightarrow a \to \mathit{Signed}\ a$$
$$\mathit{neg0}\ 0 = \mathit{Pos}\ 0$$
$$\mathit{neg0}\ x = \mathit{Neg}\ x$$

We now make *Signed* an instance of *Ord*:

$$\textbf{instance}\ (\mathit{Ord}\ a) \Rightarrow \mathit{Ord}\ (\mathit{Signed}\ a)\ \textbf{where}$$
$$\quad \mathit{compare}\ (\mathit{Pos}\ a)\ (\mathit{Pos}\ b) = \mathit{compare}\ a\ b$$
$$\quad \mathit{compare}\ (\mathit{Neg}\ a)\ (\mathit{Neg}\ b) = \mathit{compare}\ b\ a$$
$$\quad \mathit{compare}\ (\mathit{Pos}\ \_)\ (\mathit{Neg}\ \_) = \mathit{GT}$$
$$\quad \mathit{compare}\ (\mathit{Neg}\ \_)\ (\mathit{Pos}\ \_) = \mathit{LT}$$

The difficult cases are implemented in the first two lines. When a and b have the same sign, we need to compare a and b themselves to decide which number is greater than the other. If the sign differs, we can immediately decide that the one with the negative sign is smaller.

Signed is also an instance of *Enum*:

$$\textbf{instance}\ (\mathit{Enum}\ a) \Rightarrow \mathit{Enum}\ (\mathit{Signed}\ a)\ \textbf{where}$$
$$\quad \mathit{toEnum}\ i \mid i \geqslant 0 \quad = \mathit{Pos}\ \$\ \mathit{toEnum}\ i$$
$$\qquad\qquad\quad \mid i < 0 \quad = \mathit{Neg}\ \$\ \mathit{toEnum}\ i$$
$$\quad \mathit{fromEnum}\ (\mathit{Pos}\ a) = \mathit{fromEnum}\ a$$
$$\quad \mathit{fromEnum}\ (\mathit{Neg}\ a) = \mathit{negate}\ (\mathit{fromEnum}\ a)$$

With this definition some more semantics comes in. We explicitly define that, converting an integer greater or equal to zero, we use the *Pos* constructor; converting an integer less than zero, we use the *Neg* constructor. Furthermore, when we convert in the opposite direction, *Pos a* is just an a, whereas *Neg a* is the negation of a.

Now we come ot arithmetic, first addition:

$$
\begin{aligned}
&\textbf{instance}\ (Ord\ a, Num\ a) \Rightarrow Num\ (Signed\ a)\ \textbf{where}\\
&\quad (Pos\ a) + (Pos\ b) = Pos\ (a + b)\\
&\quad (Neg\ a) + (Neg\ b) = neg0\ (a + b)\\
&\quad (Pos\ a) + (Neg\ b) \mid a > b = Pos\ (a - b)\\
&\qquad\qquad\qquad\quad \mid a < b = Neg\ (b - a)\\
&\qquad\qquad\qquad\quad \mid a \equiv b = Pos\ 0\\
&\quad (Neg\ a) + (Pos\ b) \mid a > b = Neg\ (a - b)\\
&\qquad\qquad\qquad\quad \mid a < b = Pos\ (b - a)\\
&\qquad\qquad\qquad\quad \mid a \equiv b = Pos\ 0
\end{aligned}
$$

The addition of two positive numbers is a positive sum. The addition of two negative numbers ($-a + (-b) = -a - b$) is a negative sum. The addition of a negative and a positive number results in a difference, which may be positive or negative depending on which number is greater: the negative or the positive one.

We define subtraction in terms of addition: subtracting a positive number b from any number a is the same as adding the negation of b to a. Vice versa, subtracting a negative number b is the same as adding a postive number.

$$
\begin{aligned}
&a - (Pos\ b) = a + (neg0\ b)\\
&a - (Neg\ b) = a + (Pos\ b)
\end{aligned}
$$

Multiplication:

$$
\begin{aligned}
&(Pos\ a) * (Pos\ b) = Pos\ (a * b)\\
&(Neg\ a) * (Neg\ b) = Pos\ (a * b)\\
&(Pos\ 0) * (Neg\ \_) = Pos\ 0\\
&(Pos\ a) * (Neg\ b) = Neg\ (a * b)\\
&(Neg\ \_) * (Pos\ 0) = Pos\ 0\\
&(Neg\ a) * (Pos\ b) = Neg\ (a * b)
\end{aligned}
$$

This is a straight forward implementation of the rules we have already seen above: the product of two positive numbers is positive; the product of two negative numbers is positive; the product of a positive and a negative number is negative.

The next method is *negate*. There is one minor issue we have to handle: what do we do if the number is 0? In this case, we assume the number is positive. But that is a mere convention. Without this convention, we would have to introduce a constructor for 0 that is neither postive nor negative.

$$
\begin{aligned}
&\mathit{negate}\ (\mathit{Pos}\ a) \mid \mathit{signum}\ a \equiv 0 = \mathit{Pos}\ a \\
&\qquad\qquad\quad \mid \mathit{otherwise}\quad\ = \mathit{Neg}\ a \\
&\mathit{negate}\ (\mathit{Neg}\ a) = \mathit{Pos}\ a
\end{aligned}
$$

Finally, we have *abs*, *signum* and *fromInteger*. There is nothing new in the implementation of these methods:

$$
\begin{aligned}
&\mathit{abs}\ (\mathit{Pos}\ a)\quad\ = \mathit{Pos}\ a \\
&\mathit{abs}\ (\mathit{Neg}\ a)\quad\ = \mathit{Pos}\ a \\
&\mathit{signum}\ (\mathit{Pos}\ a) = \mathit{Pos}\ (\mathit{signum}\ a) \\
&\mathit{signum}\ (\mathit{Neg}\ a) = \mathit{Neg}\ (\mathit{signum}\ \$\ \mathit{negate}\ a) \\
&\mathit{fromInteger}\ i \quad \mid i \geqslant 0 = \mathit{Pos}\ (\mathit{fromInteger}\ i) \\
&\qquad\qquad\quad\ \mid i < 0 = \mathit{Neg}\ (\mathit{fromInteger}\ \$\ \mathit{abs}\ i)
\end{aligned}
$$

We make *Signed* an instance of *Real*:

$$
\begin{aligned}
&\textbf{instance}\ (\mathit{Real}\ a) \Rightarrow \mathit{Real}\ (\mathit{Signed}\ a)\ \textbf{where} \\
&\quad \mathit{toRational}\ (\mathit{Pos}\ i) = \mathit{toRational}\ i \\
&\quad \mathit{toRational}\ (\mathit{Neg}\ i) = \mathit{negate}\ (\mathit{toRational}\ i)
\end{aligned}
$$

We also make *Signed* an instance of *Integral*:

$$
\begin{aligned}
&\textbf{instance}\ (\mathit{Enum}\ a, \mathit{Integral}\ a) \Rightarrow \mathit{Integral}\ (\mathit{Signed}\ a)\ \textbf{where} \\
&\quad \mathit{quotRem}\ (\mathit{Pos}\ a)\ (\mathit{Pos}\ b) = \textbf{let}\ (q, r) = \mathit{quotRem}\ a\ b\ \textbf{in}\ (\mathit{Pos}\ q, \mathit{Pos}\ \ r) \\
&\quad \mathit{quotRem}\ (\mathit{Neg}\ a)\ (\mathit{Neg}\ b) = \textbf{let}\ (q, r) = \mathit{quotRem}\ a\ b\ \textbf{in}\ (\mathit{Pos}\ q, \mathit{neg0}\ r) \\
&\quad \mathit{quotRem}\ (\mathit{Pos}\ a)\ (\mathit{Neg}\ b) = \textbf{let}\ (q, r) = \mathit{quotRem}\ a\ b\ \textbf{in}\ (\mathit{neg0}\ q, \mathit{Pos}\ r) \\
&\quad \mathit{quotRem}\ (\mathit{Neg}\ a)\ (\mathit{Pos}\ b) = \textbf{let}\ (q, r) = \mathit{quotRem}\ a\ b\ \textbf{in}\ (\mathit{neg0}\ q, \mathit{neg0}\ r) \\
&\quad \mathit{toInteger}\ (\mathit{Pos}\ a) = \mathit{toInteger}\ a \\
&\quad \mathit{toInteger}\ (\mathit{Neg}\ a) = \mathit{negate}\ (\mathit{toInteger}\ a)
\end{aligned}
$$

The implementation of *toInteger* contains nothing new. But have a look at the definition of *quotRem*. The first case, where both numbers are positive, is easy: we return a positive quotient and a positive remainder. When both numbers are negative, the quotient is postive and the remainder is negative. Indeed, when we have the equation $a = qb + r$ and both, a and b, are negative, then a postive q will bring b close to a. With $a = -5$ and $b = -2$, for instance, the quotient would be 2: $2 \times -2 = -4$. Now, what do we have to add to -4 to reach -5? Obviously, -1. Therefore the remainder must be negative.

Now, when we have a postive a and a negative b, for instance: $a = 5$ and $b = -2$; then a negative quotient will bring us close to a: $-2 \times -2 = 4$. Missing now is the positive remainder 1. Finally, when a is negative and b is positive, we need a negative quotient, *e.g.*: $a = -5$ and $b = 2$; then $-2 \times 2 = -4$. Missing, in this case, is again a negative remainder, namely -1.

In the future, there may arise the need to make number types signed that do no fit into the classes from which we derived *Signed* so far. In particular, a signed fraction should inherit from Fractional. We therefore make *Signed* an instance of Fractional:

instance $(Eq\ a, Ord\ a, Fractional\ a) \Rightarrow Fractional\ (Signed\ a)$ **where**
$(Pos\ a)\ /\ (Pos\ b) = Pos\ (a\ /\ b)$
$(Neg\ a)\ /\ (Neg\ b) = Pos\ (a\ /\ b)$
$(Pos\ 0)\ /\ (Neg\ b) = Pos\ (0\ /\ b)$
$(Pos\ a)\ /\ (Neg\ b) = Neg\ (a\ /\ b)$
$(Neg\ a)\ /\ (Pos\ b) = Neg\ (a\ /\ b)$

1.3 Negative Binomial Coefficients

The aim of this section is to generalise binomial coefficients to integers using the new data type *Zahl*. There are many ways how to approach this goal, which perhaps can be grouped into two main approaches: first, we can look for an application of such a generalisation in the "real world" and search an appropriate mathematical tool for this problem. We started with binomial coefficients by discussing combinatorial issues, but, of course, combinatorial problems like the ways to choose k objects out of n provide no meaningful interpretation for negative numbers – what should a negative number of possibilities mean? But there are other applications, such as the multiplication of sums.

The second basic approach is not to look for applications, but to investigate the formalism asking "what happens, when we change it?" This may appear much less interesting at the first sight, since there is no obvious use case for such an altered formalism. However, such formal approaches do not only help deepening the insight into specific mathematical problems, but they also lead to new tools, which may find their applications in the future. This has happened more than once in the history of mathematics. When David Hilbert, the champion of the formalistic approach, redefined geometry extending it from the two-dimensional plane and the three-dimensional space to n-dimensional manifolds, there was no concrete application in sight. It took only a short while, however, before John von Neumann and others started using the concepts of Hilbert's geometry to model quantum physics.

Anyhow, we start with the second approach. Still, there are many ways to go. We can start with the formula $\binom{n}{k}$ and ask ourselves what happens if $n < 0$ or $k < 0$. To begin with, let us assume $n < 0$ and $k \geq 0$. If we just apply the formula $\frac{n(n-1)\ldots(n-k+1)}{k!}$, we get for $\binom{-n}{k}$ a kind of falling factorial in the numerator that looks like a rising factorial with minus signs in front of the numbers, *e.g.* $\binom{-6}{3}$ is $\frac{-6\times-7\times-8}{6}$, which is $-1\times-7\times-8 = -56$. Obviously, the signedness of the result depends on whether k is even or odd. Since there are k negative factors in the numerator, the product will be positive if k is even and negative otherwise.

Here is a Haskell implementation for this version of negative binomial coefficients:

$$
\begin{aligned}
&\mathit{chooseNeg} :: \mathit{Zahl} \to \mathit{Natural} \to \mathit{Zahl}\\
&\mathit{chooseNeg}\ n\ k \mid n \geqslant 0 \wedge k \geqslant 0 = \mathit{Pos}\ (\mathit{choose}\ (\mathit{z2n}\ n)\ k)\\
&\qquad\qquad\quad \mid n \equiv k \qquad\quad = 1\\
&\qquad\qquad\quad \mid k \equiv 0 \qquad\quad = 1\\
&\qquad\qquad\quad \mid k \equiv 1 \qquad\quad = n\\
&\qquad\qquad\quad \mid \mathit{otherwise} \quad = (n > \mid (\mathit{Pos}\ k))\ \text{`}\mathit{div}\text{`}\ (\mathit{fac}\ (\mathit{Pos}\ k))
\end{aligned}
$$

This function accepts two arguments, one of type *Zahl* and the other of type *Natural*. For the moment, we want to avoid negative ks and to rule negative values out right from the beginning, we choose *Natural* as data type for k.

When both, n and k, are positive, we just use the old *choose* converting n to *Natural*. Then we handle the trivial cases. Finally, we just implement one of the formulas for binomial coefficients. Here are some values:

$\mathit{map}\ (\mathit{chooseNeg}\ (-1))\ [0\,..\,9]$
$[1, -1, 1, -1, 1, -1, 1, -1, 1, -1]$

$\mathit{map}\ (\mathit{chooseNeg}\ (-2))\ [0\,..\,9]$
$[1, -2, 3, -4, 5, -6, 7, -8, 9, -10]$

$\mathit{map}\ (\mathit{chooseNeg}\ (-3))\ [0\,..\,9]$
$[1, -3, 6, -10, 15, -21, 28, -36, 45, -55]$

$\mathit{map}\ (\mathit{chooseNeg}\ (-4))\ [0\,..\,9]$
$[1, -4, 10, -20, 35, -56, 84, -120, 165, -220]$

$\mathit{map}\ (\mathit{chooseNeg}\ (-5))\ [0\,..\,9]$
$[1, -5, 15, -35, 70, -126, 210, -330, 495, -715]$

$\mathit{map}\ (\mathit{chooseNeg}\ (-6))\ [0\,..\,9]$
$[1, -6, 21, -56, 126, -252, 462, -792, 1287, -2002]$

We see that the coefficients for each n alternate between positive and negative values depending on k being even or odd. We also see that there is no limit anymore from which on the coefficients are all zero. In the original definition, we had $\binom{n}{k} = 0$ for $k > n$. But now we have to give up that rule, because for $n < 0$ and $k \geq 0$, $k > n$ trivially holds for all cases. In consequence, we lose the nice symmetry we had in the original triangle. Of course, we can restore many of the old characteristics by changing the definition of *chooseNeg* for negative ns to $(n \mid > (\mathit{Pos}\ k))\ \text{`}\mathit{div}\text{`}\ (\mathit{fac}\ (\mathit{Pos}\ k))$. In this variant, let us call it *chooseNeg2*, we use the rising factorial, such that the absolute values of the numbers in the numerator equal the numbers in the numerator for $n \geq 0$. For instance:

$\mathit{map}\ (\mathit{chooseNeg2}\ (-2))\ [0\,..\,9]$
$[1, -2, 1, 0, 0, 0, 0, 0, 0, 0]$

map (*chooseNeg2* (-3)) $[0\,..\,9]$
$[1,-3,3,-1,0,0,0,0,0,0]$

map (*chooseNeg2* (-4)) $[0\,..\,9]$
$[1,-4,6,-4,1,0,0,0,0,0]$

map (*chooseNeg2* (-5)) $[0\,..\,9]$
$[1,-5,10,-10,5,-1,0,0,0,0]$

map (*chooseNeg2* (-6)) $[0\,..\,9]$
$[1,-6,15,-20,15,-6,1,0,0,0]$

The first solution, *chooseNeg*, however, is more faithful to the original definition of the binomial coefficients, even if its results do not resemble the original results. One of the characteristics that is preserved is Pascal's rule:

$$\binom{n}{k} = \binom{n-1}{k-1} + \binom{n-1}{k}. \tag{1.7}$$

Indeed, we could use Pascal's rule to find negative coefficients in the first place. If we are interested in the coefficient $\binom{n-1}{k}$, we just subtract $\binom{n-1}{k-1}$ from both sides and get

$$\binom{n-1}{k} = \binom{n}{k} - \binom{n-1}{k-1}. \tag{1.8}$$

We can use this equation to search the unknown territory of negative coefficients starting from the well-known territory of positive coefficients using each result as a base camp for further expeditions. We start with $\binom{0}{0}$ and want to know the value for $\binom{-1}{0}$. Since coefficients for $k = 0$ are defined as 1, this case turns out to be trivial: $\binom{-1}{0} = 1$. The next is $\binom{-1}{1}$, which is $\binom{0}{1} - \binom{-1}{0}$, hence $0-1=-1$. The next is $\binom{-1}{2} = \binom{0}{2} - \binom{-1}{1}$, which is $0-(-1) = 0+1 = 1$. Next: $\binom{-1}{3} = \binom{0}{3} - \binom{-1}{2}$, which is $0-1=-1$ and so on. Indeed, the coefficients of -1, as we have seen before, are just alternating positive and negative 1s:

map (*chooseNeg* (-1)) $[0\,..\,9]$
$[1,-1,1,-1,1,-1,1,-1,1,-1]$

On the basis of this result, we can investigate the coefficients of -2. We know that $\binom{-2}{0}$ is 1 and continue with $\binom{-2}{1}$, which is $\binom{-1}{1} - \binom{-2}{0}$, which is $-1-1=-2$. The next is $\binom{-2}{2} = \binom{-1}{2} - \binom{-2}{1}$ or $1-(-2) = 1+2 = 3$. We continue with $\binom{-2}{3} = \binom{-1}{3} - \binom{-2}{2}$, which is $-1-3=-4$. It turns out that the coefficients for $n = -2$ are

map (*chooseNeg* (-2)) $[0\,..\,9]$
$[1,-2,3,-4,5,-6,7,-8,9,-10]$,

which is a beautiful result. If we go on this way, we will reproduce the values observed above using *chooseNeg*.

Now, what about negative ks? There is no direct way to implement something like *chooseNeg* for negative ks, because we do not know what a negative factorial would mean. Of course, we can apply a trick very similar to that we used for *chooseNeg2*, *i.e.* we compute the factorial not as $fac\ n = n * fac\ (n-1)$, but as $fac\ n = n * fac\ (n+1)$ going up towards zero. In this case, we also have to take care of the rising or falling factorial of the numerator. For instance, we can use the falling factorial with the inverse of k, such that the value of the numerator remains the same independent of k being positive or negative. The result would resemble that of *chooseNeg2*, *i.e.* we would have alternating positive and negative coefficients for $n > 0$ and positive coefficients for $n < 0$.

More interesting is using Pascal's rule to create coefficients for negative ks. But we have to be careful. In equation 1.8, we used a coefficient for $k-1$ to find the coefficient for k. This will not work. When we look for $\binom{n}{-k}$, we do not yet know the value for $\binom{n}{-(k+1)}$, since we are entering the territory of negative numbers from above. Therefore, we need a variant of equation 1.8. Instead of subtracting $\binom{n-1}{k-1}$ from Pascal's rule, we subtract the other term $\binom{n-1}{k}$ and get

$$\binom{n-1}{k-1} = \binom{n}{k} - \binom{n-1}{k}. \tag{1.9}$$

It is obvious that any coefficient resulting from a positive n and a negative k in this way equals 0, since any coefficient with $k = 0$ is 1. So, $\binom{n}{-1} = 1 - 1 = 0$. However, we also have $\binom{0}{k} = 0$ for any k and $\binom{n}{n} = 1$. For $\binom{-1}{-2}$, we therefore have $\binom{0}{-1} - \binom{-1}{-1} = 0 - 1 = -1$. For $\binom{-1}{-3}$, we get $0 - \binom{-1}{-2} = 0 - (-1) = 0 + 1 = 1$. The next coefficient $\binom{-1}{-4}$, froreseeably, is $0 - \binom{-1}{-3}$, which is $0 - 1 = -1$. Again, for $n = -1$, we get a sequence of alternating negative and positive ones.

For $\binom{-2}{k}$, we get $\binom{-2}{-2} = 1$, $\binom{-2}{-3}$ is $\binom{-1}{-2} - \binom{-2}{-2} = -1 - 1 = -2$. $\binom{-2}{-4}$ then is $\binom{-1}{-3} - \binom{-2}{-3} = 1 - (-2) = 3$. The next coefficient is $\binom{-2}{-5} = \binom{-1}{-4} - \binom{-2}{-3}$, which is $-1 - 3 = -4$ and so on. The binomial coefficients for -2 with negative ks, thus, is just the sequence $1, -2, 3, -4, 5, -6, 7, -8, 9, -10, \ldots$, which we have already seen for positive ks above. The symmetry of the triangle, hence, is reinstalled. For coefficients with $n = -2$ and $k = -9, -8, \ldots, -1, 0, 1, \ldots, 7$, we get the sequence

$$-8, 7, -6, 5, -4, 3, -2, 1, 0, 1, -2, 3, -4, 5, -6, 7, -8.$$

To confirm this result, we implement the backward rule as

$$
\begin{aligned}
&\mathit{pascalBack} :: \mathit{Zahl} \to \mathit{Zahl} \to \mathit{Zahl}\\
&\mathit{pascalBack}\ n\ 0 = 1\\
&\mathit{pascalBack}\ n\ k \mid k \equiv n \qquad = 1\\
&\qquad\qquad\quad \mid k \equiv 0 \qquad = 1\\
&\qquad\qquad\quad \mid n \equiv 0 \qquad = 0\\
&\qquad\qquad\quad \mid n > 0 \wedge k < 0 = 0\\
&\qquad\qquad\quad \mid n > 0 \wedge k \geqslant 0 = \mathit{Pos}\ (\mathit{choose}\ (\mathit{z2n}\ n)\ (\mathit{z2n}\ k)\\
&\qquad\qquad\quad \mid k < 0 \qquad = \mathit{pascalBack}\ (n+1)\ (k+1) - \mathit{pascalBack}\ n\ (k+1)\\
&\qquad\qquad\quad \mid \mathit{otherwise} = \mathit{pascalBack}\ (n+1)\ k \qquad - \mathit{pascalBack}\ n\ (k-1)
\end{aligned}
$$

We first handle the trivial cases $n = k$, $k = 0$ and $n = 0$. When $n > 0$ and $k < 0$, the coefficient is 0. For n and k both greater 0, we use *choose*. For $k < 0$, we use the rule in equation 1.9 and for $n < 0$, with $k > 0$, we use the rule in equation 1.8. Now we map *pascalBack* for specific ns to a range of ks:

$\mathit{map}\ (\mathit{pascalBack}\ (-1))\ [-9, -8\,..\,9]$
$[1, -1, 1, -1, 1, -1, 1, -1, 1, 1, -1, 1, -1, 1, -1, 1, -1, 1]$

$\mathit{map}\ (\mathit{pascalBack}\ (-2))\ [-9, -8\,..\,9]$
$[-8, 7, -6, 5, -4, 3, -2, 1, 0, 1, -2, 3, -4, 5, -6, 7, -8, 9, -10]$

$\mathit{map}\ (\mathit{pascalBack}\ (-3))\ [-9, -8\,..\,9]$
$[28, -21, 15, -10, 6, -3, 1, 0, 0, 1, -3, 6, -10, 15, -21, 28, -36, 45, -55]$

$\mathit{map}\ (\mathit{pascalBack}\ (-4))\ [-9, -8\,..\,9]$
$[-56, 35, -20, 10, -4, 1, 0, 0, 0, 1, -4, 10, -20, 35, -56, 84, -120, 165, -220]$

$\mathit{map}\ (\mathit{pascalBack}\ (-5))\ [-9, -8\,..\,9]$
$[70, -35, 15, -5, 1, 0, 0, 0, 0, 1, -5, 15, -35, 70, -126, 210, -330, 495, -715]$

The symmetry of positive and negative ks is not perfect. The coefficients for $n < 0$ and $n < k < 0$ are all 0. The sequence, in consequence, is mirrowed with a delay of $|n| - 1$ ks, such that the coefficient that corresponds to the coefficient $\binom{n}{k}$ is not $\binom{n}{-k}$, but rather $\binom{n}{n-k}$. For instance: $\binom{-2}{4} = 5 = \binom{-2}{-6}$, $\binom{-3}{5} = -21 = \binom{-3}{-8}$ and $\binom{-5}{2} = 15 = \binom{-5}{-7}$.

Looking at the numbers of negative ns, I have the strange feeling that I already saw those sequences somewhere. But where? These are definitely not the rows of Pascal's triangle, but perhaps something else? Let us look at the triangle once again:

0										1									
1									1		1								
2								1		2		1							
3							1		3		3		1						
4						1		4		6		4		1					
5					1		5		10		10		5		1				
6				1		6		15		20		15		6		1			
7			1		7		21		35		35		21		7		1		
8		1		8		28		56		70		56		28		8		1	
9	1		9		36		84		126		126		84		36		9		1

Now follow the diagonals from the upper vertex left and right downwards. On both sides you see first the 1s, then the counting numbers $1, 2, 3, \ldots,$ then the sequence we saw with $\binom{-3}{k}$, then the sequence we saw with $\binom{-4}{k}$ and so on. In other words, if we turn the triangle by 90° counterclockwise, we obtain the sequences for negative n; the same also works with a clockwise turn, but then we read the sequence from right to left.

This boils down to the equations

$$\left|\binom{-n}{1}\right| = \binom{n}{1} = \binom{n}{n-1} \tag{1.10}$$

and

$$\left|\binom{-n}{k+1}\right| = \binom{n+k-1}{k+1} = \binom{n+k-1}{n-1}. \tag{1.11}$$

For instance:

$$\begin{aligned}
\left|\binom{-3}{1}\right| &= \binom{3}{1} = \binom{3}{2} = 3\\
\left|\binom{-3}{2}\right| &= \binom{4}{2} = \binom{4}{2} = 6\\
\left|\binom{-3}{3}\right| &= \binom{5}{3} = \binom{5}{2} = 10\\
\left|\binom{-3}{4}\right| &= \binom{6}{4} = \binom{6}{2} = 15\\
\left|\binom{-3}{5}\right| &= \binom{7}{5} = \binom{7}{2} = 21\\
\left|\binom{-3}{6}\right| &= \binom{8}{6} = \binom{8}{2} = 28
\end{aligned}$$

This result may look a bit surprising at the first sight. But when we look at the formula that actually generates the value, it is obvious:

$$\binom{n}{k} = \frac{n^{\underline{k}}}{k!}. \tag{1.12}$$

When we have a negative n, the falling factorial in the numerator is in fact a rising factorial with negative numbers:

$$-n^{\underline{k}} = -n \times -(n+1) \times \cdots \times -(n+k-1).$$

Each number in the product is one less than its predecessor, but, since the numbers are negative, the absolute value is greater than its predecessor. If we eliminate the minus signs, we obtain the rising factorial for n:

$$n^{\overline{k}} = n \times (n+1) \times \cdots \times (n+k-1),$$

which is just the falling factorial for $n+k-1$:

$$(n+k-1)^{\underline{k}} = (n+k-1) \times (n+k-2) \times \cdots \times n.$$

We can therefore conclude that

$$\left|\binom{-n}{k}\right| = \frac{(n+k-1)^{\underline{k}}}{k!} = \binom{n+k-1}{k}. \tag{1.13}$$

That $\binom{n+k-1}{k}$ also equals $\binom{n+k-1}{n-1}$ results from the fact that $n-1$ and k maintain the same distance from one of the sides of the triangle, *i.e.* from either $\binom{n+k-1}{0}$ or $\binom{n+k-1}{n+k-1}$. k is trivially k coefficients away from any $\binom{n}{0}$, whereas $n-1$ is $(n+k-1)-(n-1)$ away from $\binom{n+k-1}{n+k-1}$, which is $n+k-1-n+1=k$. This is just an implication of the triangle's symmetry.

Somewhat more difficult is to see the relation to Pascal's rule. In fact, we have never proven that Pascal's rule follows from the fraction $\frac{n^{\underline{k}}}{k!}$. If we can establish this relation, the backward rule will follow immediately. So, we definitely should try to prove Pascal's rule.

We want to establish that

$$\binom{n+1}{k} = \binom{n}{k} + \binom{n}{k-1} \tag{1.14}$$

and do this directly using the definitions

$$\binom{n}{k} = \frac{n^{\underline{k}}}{k!} \tag{1.15}$$

$$\binom{n}{k-1} = \frac{n^{\underline{k-1}}}{(k-1)!} \tag{1.16}$$

Our claim is that

$$\binom{n+1}{k} = \frac{n^{\underline{k}}}{k!} + \frac{n^{\underline{k-1}}}{(k-1)!} \tag{1.17}$$

In other words, when we can deduce the left-hand side of this equation from the right-hand side, we are done. So let us just add the two fractions on the right-hand side. We first convert them to a common denominator. We can do this simply by multiplying the second fraction by k:

$$\frac{kn^{\underline{k-1}}}{k(k-1)! = k!}$$

We can join the two fractions with the common denominator and obtain a sum in the numerator:

$$\frac{n^{\underline{k}} + kn^{\underline{k-1}}}{k!}$$

If we extend the falling factorials in the numerator we get

$$n(n-1)\dots(n-k+1) + n(n-1)\dots(n-(k-1)+1)k.$$

The terms have the same number of factors, where the last factor in the first term is $n-k+1$ and the one in the second term is k. The factor before the last in the second term is $n-(k-1)+1$, which is $n-k+1+1 = n-k+2$, and this term is also the last but second factor in the first term. In other words, the factors of the terms are equal with the exception of the last factor. We can factor the equal factors out. If we do this stepwise, this looks like (using brackets to indicate what remains within the sum):

$$n[(n-1)(n-2)\dots(n-k+1)+(n-1)(n-2)\dots(n-k+2)k]$$
$$n(n-1)[(n-2)\dots(n-k+1)+(n-2)\dots(n-k+2)k]$$
$$\dots$$
$$n(n-1)\dots(n-k+2)[n-k+1+k].$$

The remaining sum can now be simplified: $n-k+1+k=n+1$ and with this we recognise in the whole expression the falling factorial of $n+1$. The whole fraction is now

$$\frac{(n+1)^{\underline{k}}}{k!},$$

which is the definition of the binomial coefficient $\binom{n+1}{k}$ and that concludes the proof. □

The proof establishes the relation between the definition of binomial coefficients and Pascal's rule. This spares us from going through the laborious task of establishing the relation expressed in equation 1.11 using only Pascal's rule.

We now switch to the first approach mentioned at the beginning of this section, *i.e.* trying to find a mathematical formalism for a practical problem. The practical problem is multiplication. We want to know what happens with negative as or bs in products of the form

$$(a+b)^n.$$

Negative ns are not too interesting here, since $(a+b)^{-n}$ is just the inverse of $(a+b)^n$, which is $\frac{1}{(a+b)^n}$, without any effect on the coefficients themselves.

We could, of course go on by trying out this formula with concrete numbers a and b. It appears much more promising, however, to choose a symbolic approach manipulating strings of the form `"a"` and `"b"`. The idea is to use string operations to simulate multiplication and addition. We do so in two steps: first we combine strings, then we simplify them in a way mimicking the rules of addition and multiplication. Since we want to see the differences between positive and negative coefficients, we need a means to negate strings simulating negative numbers. To this end, we define the simple data type

```
data Sym = P String | N String
  deriving (Eq, Show)
```

where, as you may have guessed, the P-constructor represents positive strings and the N-constructor represents negative strings. We now combine two symbols to simulate multiplication:

$$
\begin{aligned}
&comb :: Sym \to Sym \to Sym \\
&comb\ (P\ a)\ (P\ b) = P\ (a +\!\!+ b) \\
&comb\ (P\ a)\ (N\ b) = N\ (a +\!\!+ b) \\
&comb\ (N\ a)\ (P\ b) = N\ (a +\!\!+ b) \\
&comb\ (N\ a)\ (N\ b) = P\ (a +\!\!+ b)
\end{aligned}
$$

You probably realise the pattern we already used to define the number type *Zahl*: two numbers of equal signedness result in a positive number and two numbers of different signedness result in a negative number. Multiplication itself is just the concatenation of the two strings. $comb\ (P\ \texttt{"a"})\ (P\ \texttt{"b"})$, hence, is $P\ \texttt{"ab"}$; $comb\ (N\ \texttt{"a"})\ (P\ \texttt{"b"})$ is $N\ \texttt{"ab"}$.

We represent addition as lists of strings. We then can formulate the distributive law as

$$
\begin{aligned}
&combN :: Sym \to [Sym] \to [Sym] \\
&combN\ x = map\ (comb\ x)
\end{aligned}
$$

where we multiply a number, x, with a sum by multiplying that number with each term of the sum, mapping the basic combinator operation *comb* on the list representing the sum. Based on $combN$, we can now define the multiplication of a sums by itself:

$$
\begin{aligned}
&combine :: [Sym] \to [Sym] \\
&combine\ xs = concat\ [combN\ x\ xs \mid x \leftarrow xs]
\end{aligned}
$$

Let us look at an example to get a grip on *combine*:

let $a = P\ \texttt{"a"}$
let $b = P\ \texttt{"b"}$
$combine\ [a, b] = concat\ [combN\ x\ [a, b] \mid x \leftarrow [a, b]]$.

The list comprehension will first apply $combN\ a\ [a, b]$, resulting in $[aa, ab]$, and then it will apply $combN\ b\ [a, b]$ resulting in $[ba, bb]$. These two lists are then merged using *concat* resulting in $[aa, ab, ba, bb]$. We will later have to simplify this list, since, as we know, ab and ba are equal and can be written $2ab$. We come back to this immediately, but first we want to implement one more function, namely a combinator that applies *combine* n times:

$$
\begin{aligned}
&combinator :: [Sym] \to Natural \to [Sym] \\
&combinator\ \_\ 0 = [\,] \\
&combinator\ xs\ n = go\ xs\ (n-1) \\
&\quad \textbf{where}\ go\ ys\ 0 = ys \\
&\qquad\qquad go\ ys\ n = go\ (concat\ [combN\ x\ ys \mid x \leftarrow xs])\ (n-1)
\end{aligned}
$$

Note that this function does not reuse *combine*, but implements it anew. The reason is that we do not want to combine the original input with itself, but with the result of the previous recursion. For instance, if $n = 3$, then we start with $[combN\ x\ [a, b] \mid x \leftarrow [a, b]]$ resulting in $[aa, ab, ba, bb]$. In the next round, we multiply this result by $[a, b]$: $[combN\ x\ [aa, ab, ba, bb] \mid x \leftarrow [a, b]]$ resulting in $[aaa, aab, aba, abb, baa, bab, bba, bbb]$

and corresponding to the expression $(a + b)^3$.

Another interesting aspect of *combinator* is the base case for $n = 0$, which is just defined as being the empty list. In this context, the empty list would, hence, represent the number 1, since 1, as we know, is the coefficient $\binom{0}{0}$.

Now we want to simplify results, such that $[aa, ab, ba, ba] = [aa, 2 * ab, ba]$. First, we need to express that `"ab"` and `"ba"` are the same thing. Therefore, we sort the string:

```
sortStr :: Sym → Sym
sortStr (P a) = P (sort a)
sortStr (N a) = N (sort a)
```

leading to a canonical form for strings. The call *sortStr* (*P* `"ba"`) would result in *P* `"ab"`.

To simplify a complete list, we compare all symbols in the lists and consolidate symbols with equal strings resulting in lists of the type (*Natural*, *Sym*), where the natural number counts the ocurrences of that symbol in the list:

```
simplify :: [Sym] → [(Natural, Sym)]
simplify xs = go (sort $ map sortStr xs) 1
  where go [] _ = []
        go [x] n = [(n, x)]
        go ((P x) : (P y) : zs)  n | x ≡ y       = go ((P y) : zs) (n + 1)
                                   | otherwise = (n, P x) : go ((P y) : zs) 1
        go ((N x) : (N y) : zs) n | x ≡ y       = go ((N y) : zs) (n + 1)
                                   | otherwise = (n, N x) : go ((N y) : zs) 1
        go ((N x) : (P y) : zs)  n | x ≡ y       = go zs 1
                                   | otherwise = (n, N x) : go ((P y) : zs) 1
        go ((P x) : (N y) : zs)  n | x ≡ y       = go zs 1
                                   | otherwise = (n, P x) : go ((N y) : zs) 1
```

We start by first transforming all symbols into the canonical form sorting their strings. Then we sort the list of symbols itself. The idea is that all strings of the same kind are listed in a row, such that we can easily count them. But, of course, we do not want to have the negative and positive symbols separated, we want all symbols with the same string in a row, independently of these symbols being positive or negative. We have to implement this notion of comparison ignoring the sign. To this end, we make *Sym* an instance of *Ord*:

```
instance Ord Sym where
  compare (P a) (P b) = compare a b
  compare (P a) (N b) = compare a b
  compare (N a) (P b) = compare a b
  compare (N a) (N b) = compare a b
```

We now *go* through the list of symbols sorted in this sense of comparison and check on

the first and the second of the remaining list on each step. If the signedness of first and second are equal and their strings are equal, we increment n and store (n, x), x the head of the list, whenever they differ starting the rest of the list with $n = 1$. Otherwise, when the signs are not equal, but the strings are, then we drop both symbols and continue with the rest of the list after the second.

Applied step for step on the list

$[aaa, aab, aba, abb, baa, bab, bba, bbb]$,

this would advance as follows. We first sort all strings resulting

$[aaa, aab, aab, abb, aab, abb, abb, bbb]$.

We next sort the symbols:

$[aaa, aab, aab, aab, abb, abb, abb, bbb]$.

We then weight according to the number of their appearance:

$[(1, aaa), (3, aab), (3, abb), (1, bbb)]$.

In this example, we ignore signedness, since all terms are positive anyway. The interesting thing thus is still to commence: the application to sums with negative terms. We do this with the simple function

$$\begin{aligned} &\mathit{powsum} :: [\mathit{Sym}] \to \mathit{Natural} \to [(\mathit{Natural}, \mathit{Sym})] \\ &\mathit{powsum}\ xs = \mathit{simplify} \circ \mathit{combinator}\ xs \end{aligned}$$

We now define two variables, a positive string a and a negative one b: **let** $a = P$ `"a"` and **let** $b = N$ `"b"` and just apply *powsum* with increasing ns:

powsum $[a, b]$ 1
$[(1, P$ `"a"`$), (1, N$ `"b"`$)]$

powsum $[a, b]$ 2
$[(1, P$ `"aa"`$), (2, N$ `"ab"`$), (1, P$ `"bb"`$)]$

powsum $[a, b]$ 3
$[(1, P$ `"aaa"`$), (3, N$ `"aab"`$), (3, P$ `"abb"`$), (1, N$ `"bbb"`$)]$

It appears that the absolute values of the coefficients do not change. We have, for instance, $\binom{3}{0} = 1$, $\binom{3}{1} = -3$, $\binom{3}{2} = 3$, $\binom{3}{3} = -1$. What, if we swap the negative sign: **let** $a = N$ `"a"` and **let** $b = P$ `"b"`?

powsum $[a, b]$ 1
$[(1, N$ `"a"`$), (1, P$ `"b"`$)]$

powsum $[a, b]$ 2
$[(1, P$ `"aa"`$), (2, N$ `"ab"`$), (1, P$ `"bb"`$)]$

powsum $[a, b]$ 3
$[(1, N$ `"aaa"`$), (3, P$ `"aab"`$), (3, N$ `"abb"`$), (1, P$ `"bbb"`$)]$

The result is just the same for even exponents. For odd exponents, the signs are just exchanged: $\binom{3}{0} = -1$, $\binom{3}{1} = 3$, $\binom{3}{2} = -3$, $\binom{3}{3} = 1$.

What if both terms are negative: **let** $a = N$ `"a"` and **let** $b = N$ `"b"`?

powsum $[a, b]$ 1
$[(1, N$ `"a"`$), (1, N$ `"b"`$)]$

powsum $[a, b]$ 2
$[(1, P$ `"aa"`$), (2, P$ `"ab"`$), (1, P$ `"bb"`$)]$

powsum $[a, b]$ 3
$[(1, N$ `"aaa"`$), (3, N$ `"aab"`$), (3, N$ `"abb"`$), (1, N$ `"bbb"`$)]$

Now, wonder of wonders, even exponents lead to positive coefficients, while odd exponents lead to negative coefficients: $\binom{3}{0} = -1$, $\binom{3}{1} = -3$, $\binom{3}{2} = -3$, $\binom{3}{3} = -1$.

This result appears quite logical, since, with even exponents, we multiply an even number of negative factors, while, with odd exponents, we multiply an odd number of negative factors.

To confirm these results with more data, we define one more function to extract the coefficients and, this way, making the output more readable:

$$\begin{aligned}&\mathit{coeffs} :: [\mathit{Sym}] \to \mathit{Natural} \to [\mathit{Natural}]\\&\mathit{coeffs}\ \mathit{xs} = \mathit{map}\ \mathit{getCoeff} \circ \mathit{powsum}\ \mathit{xs}\end{aligned}$$

where *getCoeff* is

$$\begin{aligned}&\mathit{getCoeff} :: (\mathit{Natural}, \mathit{Sym}) \to \mathit{Natural}\\&\mathit{getCoeff}\ (n, P\ \_) = n\\&\mathit{getCoeff}\ (n, N\ \_) = -n\end{aligned}$$

We now map *coeffs* on the numbers $[0\,..\,9]$ and, with a and b still both negative, see Pascal's triangle with signs alternating per row:

$[\,]$
$[-1, -1]$
$[1, 2, 1]$
$[-1, -3, -3, -1]$
$[1, 4, 6, 4, 1]$
$[-1, -5, -10, -10, -5, -1]$
$[1, 6, 15, 20, 15, 6, 1]$
$[-1, -7, -21, -35, -35, -21, -7, -1]$
$[1, 8, 28, 56, 70, 56, 28, 8, 1]$
$[-1, -9, -36, -84, -126, -126, -84, -36, -9, -1]$

For one of the value a and b positive and the other negative, we see the coefficients in one row alternating in signedness. For a positive and b negative, we see:

$[]$
$[1, -1]$
$[1, -2, 1]$
$[1, -3, 3, -1]$
$[1, -4, 6, -4, 1]$
$[1, -5, 10, -10, 5, -1]$
$[1, -6, 15, -20, 15, -6, 1]$
$[1, -7, 21, -35, 35, -21, 7, -1]$
$[1, -8, 28, -56, 70, -56, 28, -8, 1]$
$[1, -9, 36, -84, 126, -126, 84, -36, 9, -1]$

The other way round, a negative and b positive, we see just the same triangle where, for odd exponents, the minus signs are swapped. The triangle, hence, is the same as the one before with each row reversed:

$[]$
$[-1, 1]$
$[1, -2, 1]$
$[-1, 3, -3, 1]$
$[1, -4, 6, -4, 1]$
$[-1, 5, -10, 10, -5, 1]$
$[1, -6, 15, -20, 15, -6, 1]$
$[-1, 7, -21, 35, -35, 21, -7, 1]$
$[1, -8, 28, -56, 70, -56, 28, -8, 1]$
$[-1, 9, -36, 84, -126, 126, -84, 36, -9, 1]$

1.4 $\mathbb{Q}$

We now turn our attention to fractions and start by implementing a rational data type:

```
data Ratio = Q Natural Natural
  deriving (Show, Eq)
```

A *Ratio* has a constructor Q that takes two natural numbers. The name of the constructor is derived from the symbol for the set of rational numbers $\mathbb{Q}$ that was introduced by Giuseppe Peano and stems from the Italian word *Quoziente*.

It would be nice of course to have a function that creates a rational in its canonical form, *i.e.* reduced to two natural numbers that are coprime to each other. This is done by *ratio*:

$$
\begin{aligned}
&\mathit{ratio} :: \mathit{Natural} \to \mathit{Natural} \to \mathit{Ratio}\\
&\mathit{ratio}\ \_\ 0 = \mathit{error}\ \texttt{"division by zero"}\\
&\mathit{ratio}\ a\ b = \mathit{reduce}\ (Q\ a\ b)
\end{aligned}
$$

for which we define the infix %:

$$
\begin{aligned}
&\textbf{infix}\ \%\\
&(\%) :: \mathit{Natural} \to \mathit{Natural} \to \mathit{Ratio}\\
&(\%) = \mathit{ratio}
\end{aligned}
$$

so that we can create ratios with expressions like $5\ \%\ 2$, $8\ \%\ 4$ and so on. The function *reduce* called in *ratio* is defined as follows:

$$
\begin{aligned}
&\mathit{reduce} :: \mathit{Ratio} \to \mathit{Ratio}\\
&\mathit{reduce}\ (Q\ \_\ 0) = \mathit{error}\ \texttt{"division by zero"}\\
&\mathit{reduce}\ (Q\ 0\ \_) = Q\ 0\ 1\\
&\mathit{reduce}\ (Q\ n\ d) = Q\ (n\ `\mathit{div}`\ \mathit{gcd}\ n\ d)\\
&\qquad\qquad\qquad\qquad\ (d\ `\mathit{div}`\ \mathit{gcd}\ n\ d)
\end{aligned}
$$

which reduces numerator and denominator to the quotient of the greatest common divisor of numerator and denominator. If numerator and denominator are coprime, the *gcd* is just 1 and the numbers are not changed at all.

Useful would be to have access functions for numerator and denominator. We define them straight forward as

$$
\begin{aligned}
&\mathit{numerator} :: \mathit{Ratio} \to \mathit{Natural}\\
&\mathit{numerator}\ (Q\ n\ \_) = n\\
&\mathit{denominator} :: \mathit{Ratio} \to \mathit{Natural}\\
&\mathit{denominator}\ (Q\ \_\ d) = d
\end{aligned}
$$

We now make *Ratio* an instance of *Ord*:

$$
\begin{aligned}
&\textbf{instance}\ \mathit{Ord}\ \mathit{Ratio}\ \textbf{where}\\
&\quad \mathit{compare}\ x@(Q\ nx\ dx)\ y@(Q\ ny\ dy) \mid dx \equiv dy \quad = \mathit{compare}\ nx\ ny\\
&\qquad\qquad\qquad\qquad\qquad\qquad\qquad\quad \mid \mathit{otherwise} = \textbf{let}\ (x', y') = \mathit{unify}\ x\ y\\
&\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\quad\ \ \textbf{in}\ \mathit{compare}\ x'\ y'
\end{aligned}
$$

If the denominators are equal, we just compare the numerators, *i.e.* $\frac{1}{5} < \frac{2}{5} < \frac{3}{5}$ and so on. Otherwise, if the denominators differ, we must convert the fractions to a common denominator before we actually can compare them. This is done using *unify*:

$$
\begin{aligned}
&\mathit{unify} :: \mathit{Ratio} \to \mathit{Ratio} \to (\mathit{Ratio}, \mathit{Ratio})\\
&\mathit{unify}\ (Q\ nx\ dx)\ (Q\ ny\ dy) = (Q\ (nx * (\mathit{lcm}\ dx\ dy)\ `\mathit{div}`\ dx)\ (\mathit{lcm}\ dx\ dy),\\
&\qquad\qquad\qquad\qquad\qquad\qquad\quad\ Q\ (ny * (\mathit{lcm}\ dx\ dy)\ `\mathit{div}`\ dy)\ (\mathit{lcm}\ dy\ dx))
\end{aligned}
$$

This is the implementation of the logic already described before: we convert a fraction to the common denominator defined by the *lcm* of both denominators and multiply the numerators by the number we would have to multiply the denominator by to get the *lcm*, which trivially is the *lcm* divided by the denominator: $\mathit{lcm}(dx, dy) = dx \times \frac{\mathit{lcm}(dx,dy)}{dx}$.

This may appear a bit complicated, but it is much faster, whenever the denominators are not coprime to each other.

The next step is to make *Ratio* an instance of *Enum*:

```
instance Enum Ratio where
  toEnum i = Q (toEnum i) (toEnum 1)
  fromEnum (Q n d) = fromEnum (n `div` d)
```

which implies a conversion from and to an integer type. A plain integral number i is converted to a *Ratio* using the denominator 1. For the backward conversion, *div* is used leading to the loss of precision if the denominator does not divide the numerator.

Now we come to the heart of the data type making it instance of *Num*:

```
instance Num Ratio where
  x@(Q nx dx) + y@(Q ny dy) | dx ≡ dy   = (nx + ny) % dx
                            | otherwise = let (x', y') = unify x y
                                          in (x' + y')
  x@(Q nx dx) - y@(Q ny dy) | x ≡ y = Q 0 1
                            | x > y ∧ dx ≡ dy = (nx - ny) % dx
                            | x > y = let (x', y') = unify x y
                                      in x' - y'
                            | otherwise = error "Subtraction beyond zero!"
  (Q nx dx) * (Q ny dy) = (nx * ny) % (dx * dy)
  negate a = a
  abs a = a
  signum (Q 0 _) = 0
  signum (Q _ _) = 1
  fromInteger i = Q (fromIntegral i) 1
```

We add two fraction with the same denominator by reducing the result of $\frac{nx+ny}{d}$. If we add two fractions in canonical form, such as $\frac{1}{9}$ and $\frac{5}{9}$, we may arrive at a result that is not in canonical form like, in this example, $\frac{6}{9}$, which should be reduced to $\frac{2}{3}$. Otherwise, if the denominators differ, we first convert the fractions to a common denominator before we add them.

Since we have defined *Ratio* as a fraction of two natural numbers (and not of two integers), we have to be careful with subtraction. If the two fractions are equal, the result is zero, which is represented as $\frac{0}{1}$. If $x > y$, we use the same strategy as with addition. Otherwise, if $y > x$, subtraction is undefined.

Multiplication is easy: we just reduce the result of multiplying the two numerators and denominators by each other. The other functions do not add anything new. We just define *negate*, *abs* and *signum* as we have done before for plain natural numbers and we define the conversion from integer as we have done for *Enum* already.

The next step, however, is unique: we define *Ratio* as an instance of *Fractional*. The core of this is to define a division function and do so defining division as the inverse of multiplication:

$$rdiv :: Ratio \to Ratio \to Ratio$$
$$rdiv\ (Q\ nx\ dx)\ (Q\ ny\ dy) = (Q\ nx\ dx) * (Q\ dy\ ny)$$

The division of a fraction $\frac{nx}{dx}$ by another $\frac{ny}{dy}$ is just the multiplication of that fraction with the inverse of the second one, which is $\frac{dy}{ny}$. The complete implementation of the *Fractional* type then is

```
instance Fractional Ratio where
  (/)          = rdiv
  fromRational r = Q (fromIntegral $ R.numerator r)
                     (fromIntegral $ R.denominator r)
```

This is not a complete definition of $\mathbb{Q}$, however. $\mathbb{Q}$ is usually defined on top of the integers rather than on top of natural numbers. So, our data type should be signed. That, however, is quite easy to achieve:

```
type Quoz = Signed Ratio
```

1.5 Zeno's Paradox

The ancient Greek philosopher Zeno of Elea, who lived in the 5^{th} century BC, devised a number of paradoxes that came upon us indirectly through the work of Aristotle and its commentators. Zeno designed the paradoxes to defened the philosophy of the *One* developed by Zeno's teacher Parmenides. According to this philosophy everything is One, undivisible, motionless, eternal, everywhere and nowhere. That we actually see motion, distinguish and divide things around us, that everything "in this world" is volatile and that everything has its place, is either here or there, is, according to Parmenides, just an illusion.

Plato discusses the philosophy of Parmenides in one of his most intriguing dialogs, "Parmenides", where young Socrates, Zeno and Parmenides himself analyse contradictions that arise both in Plato's *theory of forms* as well as in Parmenides' theory of the One. The Parmenides dialog had a deep influence on European philosophy and religion. It was the main inspiration for the late-ancient *neoplatonism* and, for many centuries, it shaped the interpretation of ancient philosophy by medieval thinkers. Scholars today, however, are not so sure anymore what the dialog is about in the first place. Some see in it a critical discussion of the theory of forms, others hold it is a collection of exercises for students of Plato's academy and again others consider the dialog as highly ironic, actually criticising Parmenides and other philosophers for using terms that they know from everyday life in a context where the ideas associated with these terms do not hold anymore – very similar to the therapeutic approach of Ludwig Wittgenstein.

It is tempting to relate the philosophy of the One with the philosophical worldview of mathematical platonism. Constructivists would see the world as dynamic, as a chaotic process without meaning in itself or, pessimistically, as a thermodynamic process that tends to entropy. It is an effort of human beings to create order in this dynamic and perhaps chaotic world. Therefore, prime numbers – or any other mathematical object – do not exist, we define them and we have to invest energy to construct them. By contrast, mathematical platonists would hold that there is a static eternal structure that does not change at all. The mathematical objects are out there, perhaps like in a gigantic lattice around the universe or, like a skeleton, within the universe. It would then be absurd to say that we construct prime numbers. We find them travelling along the eternal metaphysical structure that is behind of what we can perceive directly with our senses.

Be that as it may, we are here much more interested in the math in Zeno's paradoxes. The most famous one is the race of Achilles and the tortoise. Achilles gives the tortoise a lead of, say, hundred meters. The question now is when Achilles will actually catch up with the tortoise. Zeno says: never, for it is impossible. To catch up, he must reach a point where the tortoise has been shortly before. But when he gets there, the tortoise is already ahead. Perhaps just a few metres, but definitely ahead. So, again, to reach that point, Achilles will need some time. When he reaches the point where the tortoise has been a second before, the same is already a bit further. To reach that point, Achilles again needs some time and in this time the tortoise again makes some progress and so it goes on and on.

A more concrete version of this paradox is given in the so called *Dichotomy* paradox. It states that, in general, it is impossible to move from A to B. Since, to do so, one has first to make half of the way arriving at a point C. To move from C to B, one now has to first make half of the way arriving at a point D. To move from D to B, one now has to first make half of the way arriving at yet another point and so on and so on.

This paradox appears to be at odds with what we observe in the physical world where it indeed appears to be possible to move from A to B quite easily. The paradox, however, draws our attention to the fact that, between any two rational numbers, there are infinitely many other rational numbers. Between 0 and 1, for instance, there is $\frac{1}{2}$. Between 0 and $\frac{1}{2}$, there is $\frac{1}{4}$. Between 0 and $\frac{1}{4}$, there is $\frac{1}{8}$. Between 0 and $\frac{1}{8}$, there is $\frac{1}{16}$ and, in general, between 0 and any number of the form $\frac{1}{2^k}$, there is a number $\frac{1}{2^{k+1}}$.

So, following these points as in Zeno's paradox, how close to B would we get after k steps? The problem can be represented as a sum of the form

$$\sum_{i=1}^{k} \frac{1}{2^i}$$

that would describe the distance we have travelled. After $k = 1$ step, we would have

travelled $\frac{1}{2}$ of the way. After $k = 2$ steps, we would have travelled

$$\frac{1}{2} + \frac{1}{2^2}.$$

We convert the fractions to a common denominator multiplying the first fraction by 2 and arrive at

$$\frac{2+1}{4} = \frac{3}{4}.$$

For $k = 3$ steps, we have

$$\frac{3}{4} + \frac{1}{2^3} = \frac{6+1}{8} = \frac{7}{8}.$$

These experiments suggest the general formula

$$\sum_{i=1}^{k} \frac{1}{2^i} = \frac{2^k - 1}{2^k}. \tag{1.18}$$

This equation cries out for an induction proof. Any of the examples above serves as base case. We then have to prove that

$$\frac{2^k - 1}{2^k} + \frac{1}{2^{k+1}} = \frac{2^{k+1} - 1}{2^{k+1}}. \tag{1.19}$$

We convert the fractions on the left-hand side of the equation to a common denominator multiplying the first fraction by 2:

$$\frac{2(2^k - 1) + 1}{2^{k+1}}.$$

We simplify the numerator: $2 \times 2^k = 2^{k+1}$ and $2 \times (-1) = -2$; we, hence, have in the numerator $2^{k+1} - 2 + 1$, which can be simplified to $2^{k+1} - 1$. This leads to the desired result

$$\frac{2^{k+1} - 1}{2^{k+1}}. \quad \square$$

That was easy! Can we generalise the result for any denominator n, such that

$$\frac{1}{n^k} + \frac{1}{n^{k+1}} = \frac{n^{k+1} - 1}{n^{k+1}}? \tag{1.20}$$

If we went a third of the way on each step instead of half of it, we had $\frac{1}{3^k} + \frac{1}{3^{k+1}}$, for instance: $\frac{1}{3} + \frac{1}{9}$. We convert the fraction to a common denominator multiplying the first by 3: $\frac{3+1}{9} = \frac{4}{9}$. So, equation 1.20 seems to be wrong. The nice and clean result with the denominator 2 appears to be one of those deceptions that are so common for small numbers, which often behave very differently from greater numbers.

But let us stop moaning. What actually is the rule for $n = 3$? After the next step, we would have

$$\frac{4}{9} + \frac{1}{27}.$$

We multiply the first fraction by 3 and have

$$\frac{12 + 1}{27} = \frac{13}{27}.$$

For $k = 4$, we would have

$$\frac{13}{27} + \frac{1}{81} = \frac{39 + 1}{81} = \frac{40}{81}.$$

The experiments this time suggest the rule

$$\sum_{i=1}^{k} \frac{1}{3^i} = \frac{(3^k - 1)/2}{3^k}. \tag{1.21}$$

We prove again by induction with any of the examples serving as base case. We have to prove that

$$\frac{(3^k - 1)/2}{3^k} + \frac{1}{3^{k+1}} = \frac{(3^{k+1} - 1)/2}{3^{k+1}}. \tag{1.22}$$

We multiply the first fraction by 3 in numerator and denominator and get in the numerator $\frac{3(3^k-1)}{2} = \frac{3^{k+1}-3}{2}$. We can now add the two fractions:

$$\frac{(3^{k+1} - 3)/2 + 1}{3^{k+1}}.$$

To add 1 to the fraction in the numerator we have to convert 1 to a fraction with the denominator 2, which, of course, is $\frac{2}{2}$. We, hence, have in the numerator $\frac{3^{k+1}-3+2}{2}$ and this leads to the desired result:

$$\frac{(3^{k+1}-1)/2}{3^{k+1}}. \quad \square \tag{1.23}$$

Before we dare to make a new conjecture based on equations 1.18 and 1.21, let us collect some more data. Since $n = 4$ is closely related to $n = 2$, we will immediately go to $n = 5$. For $k = 2$ we have

$$\frac{1}{5} + \frac{1}{25} = \frac{5+1}{25} = \frac{6}{25}.$$

For $k = 3$ we have

$$\frac{6}{25} + \frac{1}{125} = \frac{30+1}{125} = \frac{31}{125}.$$

For $k = 4$ we have

$$\frac{31}{125} + \frac{1}{625} = \frac{155+1}{625} = \frac{156}{625}.$$

In these examples, we see the relation

$$\sum_{i=1}^{k} \frac{1}{5^i} = \frac{(5^k-1)/4}{5^k}. \tag{1.24}$$

We prove easily by induction using any of the examples as base case. We have to show that

$$\frac{(5^k-1)/4}{5^k} + \frac{1}{5^{k+1}} = \frac{(5^{k+1}-1)/4}{5^{k+1}}. \tag{1.25}$$

We multiply the first fraction by 5, yielding the numerator $\frac{5^{k+1}-5}{4}$ and, when adding 1, we get $\frac{5^{k+1}-5}{4} + \frac{4}{4}$, which, of course, leads to the desired result. $\square$

To summarise: with $n = 2$, we see $\frac{n^k-1}{n^k}$; with $n = 3$, we see $\frac{(n^k-1)/2}{n^k}$; with $n = 5$, we see $\frac{(n^k-1)/4}{n^k}$. This suggests the general form

$$\sum_{i=1}^{k} \frac{1}{n^i} = \frac{(n^k - 1)/(n-1)}{n^k}, \tag{1.26}$$

which would nicely explain why we overlooked the division in the numerator for the case $n = 2$, since, here, $n - 1 = 1$ and anything divided by 1 is just that something.

It, again, does not appear to be too difficult to prove the result. We have a lot of base cases already and now want to prove that

$$\frac{(n^k - 1)/(n-1)}{n^k} + \frac{1}{n^{k+1}} = \frac{(n^{k+1} - 1)/(n-1)}{n^{k+1}}. \tag{1.27}$$

We multiply the first fraction by n in numerator and denominator and get in the numerator

$$\frac{n(n^k - 1)}{n-1} = \frac{n^{k+1} - n}{n-1}.$$

We now add 1 represented as the fraction $\frac{n-1}{n-1}$:

$$\frac{n^{k+1} - n}{n-1} + \frac{n-1}{n-1},$$

leading to

$$\frac{n^{k+1} - n + n - 1}{n-1} = \frac{n^{k+1} - 1}{n-1},$$

which is the desired result

$$\frac{(n^{k+1} - 1)/(n-1)}{n^{k+1}}. \quad \square$$

Could we not have come to this result in an easier way? Well, we should have realised that Zeno's problem is just an instance of a geometric series. A geometric series is defined by the equation

$$S_n = \frac{a(1 - r^k)}{1 - r}. \tag{1.28}$$

In our case, a and r are fractions. For the first case, we have $a = \frac{1}{2}$ and $r = \frac{1}{2}$. We therefore get

$$S_n = \frac{\frac{1}{2}(1 - \frac{1}{2^k})}{1 - \frac{1}{2}}. \tag{1.29}$$

When we multiply the numerator out, we get (just looking at the numerator):

$$\frac{1}{2}\left(1 - \frac{1}{2^k}\right) = \frac{1}{2} - \frac{1}{2^{k+1}}.$$

We multiply $\frac{1}{2}$ by 2^k in numerator and denominator and add the resulting terms:

$$\frac{2^k - 1}{2^{k+1}}.$$

Now we look at the denominator, which is $1 - \frac{1}{2}$. This is just $\frac{1}{2}$ and, since dividing by $\frac{1}{2}$ is the same as multiplying by 2, we can reduce the whole fraction to

$$\frac{2 \times (2^k - 1)}{2^{k+1}}.$$

The 2 in the numerator cancels against the 2^{k+1}, so we finally get

$$S_n = \frac{2^k - 1}{2^k}, \tag{1.30}$$

the same result we got above with some guessing around.

Now, to generalise the final result we set $a = \frac{1}{n}$ and $r = \frac{1}{n}$ and get the scary-looking equation

$$S_n = \frac{\frac{1}{n}\left(1 - \frac{1}{n^k}\right)}{1 - \frac{1}{n}}. \tag{1.31}$$

We start by looking at the numerator first again:

$$\frac{1}{n}\left(1 - \frac{1}{n^k}\right) = \frac{1}{n} - \frac{1}{n^{k+1}} = \frac{n^k}{n^{k+1}} - \frac{1}{n^{k+1}} = \frac{n^k - 1}{n^{k+1}}.$$

The denominator is $1 - \frac{1}{n}$, which is the same as $\frac{n-1}{n}$. Again, instead of dividing by this fraction, we can multiply by the inverse $\frac{n}{n-1}$:

$$\frac{n^k-1}{n^{k+1}} \times \frac{n}{n-1} = \frac{\frac{n^{k+1}}{n-1} - \frac{n}{n-1}}{n^{k+1}} = \frac{\frac{n^{k+1}-n}{n-1}}{n^{k+1}}$$

We can factor n out in the numerator to get

$$\frac{\frac{n\left(n^k-1\right)}{n-1}}{n^{k+1}} = \frac{n\frac{n^k-1}{n-1}}{n^{k+1}}$$

and, again, cancel n against the denominator resulting at

$$S_n = \frac{(n^k-1)/(n-1)}{n^k}. \tag{1.32}$$

1.6 Systems of Linear Equations

Systems of linear equations provide an excellent topic to get familar with structures that we will need a lot in algebra, namely *matrices*. Before we get there, we look at linear equations as such. Linear equations and the systems made of them belong to the oldest topics studied in algebra. There is a rich body of knowledge in Chinese and Indian books dating back to antiquity and early middle ages (in terms of European history). The famous "Nine Chapters of Mathematical Art", for instance, dates back to 179 AD. It contains advanced algorithms to solve systems of linear equations that were formulated in Europe only in the 19^{th} century.

This knowledge was brought to Europe through Arab and Persian scholars, most famously perhaps al-Hwarizmi, called Algoritmi in medieval Europe, and his book "Compendium on Calculation and Balancing", whose original title contains the word "al-gabr", which was latinised as *algebra*.

In this tradition, systems of linear equations were often worded in terms of *bird problems*. Bird problems are centered around the question of how many of n different kinds of birds can be bought for a specific amount of money. A typical problem is to buy 100 birds for 100 drachme. There are ducks, chickens and sparrows. For 1 drachme, you can either buy one chicken or 20 sparrows; for 5 drachme, you get a duck. This translates into the simple system of equations

$$\begin{array}{rcl} x+y+z & = & 100 \\ 5x+\frac{1}{20}y+z & = & 100 \end{array} \tag{1.33}$$

The first equation states that the sum of the number of birds shall be 100; the second equation states that the sum of the price of the birds shall be 100 drachme.

One way to solve such equations is to *eliminate* one of the variables. In the given system, we can solve for z in both equations:

$$z = 100 - x - y \tag{1.34}$$

and

$$z = 100 - 5x - \frac{1}{20}y. \tag{1.35}$$

We set the right-hand sides of the equations equal, subtract 100 and bring x to the left side of the equation and y to the right side. We get:

$$4x = \frac{19}{20}y \tag{1.36}$$

and divide by 4:

$$x = \frac{19}{80}y. \tag{1.37}$$

From here, we easily find a solution by assuming that $y = 80$, $x = 19$ and, in consequence, $z = 1$. The original equations with the variables substituted, then, read

$$\begin{array}{lcl} 19 + 80 + 1 & = & 100 \\ 5 \times 19 + \frac{1}{20} \times 80 + 1 & = & 100, \end{array} \tag{1.38}$$

which, as you can easily convince yourself, is correct in both cases.

The final step in the derivation was a mere guess based on the fact that we expected integer numbers as results in one of the equations. Without that restriction, *i.e.* when we define the system over the field of rational numbers, would there be a way to solve any such system? It turns out, there is. Furthermore, that algorithm is guaranteed to find a single solution to any well-defined system.

You might remember a similar claim that we proved for a special kind of systems in the previous chapter, namely the Chinese Remainder Theorem. Indeed, Chinese remainders are just a special case of linear equations in a finite field of modular arithmetic. For the general case, which includes infinite fields, such as the rational numbers, we have to restrict the claim adding the constraint that the system must be *well-defined*.

By this, we mean that the system is *consistent* and contains the same number of *independent* equations and unknowns; for the bird problem above this was not the case,

since there were only two equations for three unknowns (x, y and z). However, the bird problem was restricted to integers, and we were able to guess the result after some steps.

Have a look at the following system:

$$\begin{array}{rcrcl} x & + & y & = & 1 \\ 2x & + & 2y & = & 2 \end{array} \tag{1.39}$$

There are two unknowns, x and y, and two equations. Unfortunately, the two equations are not independent, since the second equation is equivalent to the first, *i.e.* it is just the first equation scaled up. Indeed, whenever one equation can be derived from the others by algebraic means, it is not independent and, hence, does not add new information to the system. A somewhat more subtle example of a system with a dependent equation is

$$\begin{array}{rcrcrcr} x & - & 2y & + & z & = & -1 \\ 3x & + & 5y & + & z & = & 8 \\ 4x & + & 3y & + & 2z & = & 7. \end{array} \tag{1.40}$$

Here, the third equation is the sum of equations 1 and 2, so it does not add new information.

Systems of equations that have more unknowns than independent equations are called *underdetermined*. They usually have no or infinitely many solutions. If a system has more equations than unknowns, it is *overdetermined* and, usually, has no solution. The system is then *inconsistent*, *i.e.* it contains a contradiction. An inconsistent system is, for instance

$$\begin{array}{rcrcrcr} x & - & 2y & + & z & = & -1 \\ 3x & + & 5y & + & z & = & 8 \\ 4x & + & 3y & + & 2z & = & 5. \end{array} \tag{1.41}$$

The sum of the left-hand side of equations 1 and 2 results in the left-hand side of equation 3. The right-hand side of equation 3, however, is not the sum of the right-hand side of equations 1 and 2. Any try of to solve this system will lead to a contradiction of the form $1 = 0$.

A consistent system, however, that has the same number of independent equations and unknowns has, within a field, always a unique solution and there is an algorithm that finds this solution. But before we present and implement the algorithm as such, we will look at the ideas, on which it is based.

The first approach is *elimination*. The idea is to solve one equation for one of the variables and then to substitute that variable in the other equations by the result. A concrete example:

$$\begin{array}{rcrcrcr} x & + & 3y & - & 2z & = & 5 \\ 3x & + & 5y & + & 6z & = & 7 \\ 2x & + & 4y & + & 3z & = & 8. \end{array} \tag{1.42}$$

We solve the first equation for x. We just subtract $3y$ from and add $2z$ to both side to obtain

$$x = 5 - 3y + 2z. \tag{1.43}$$

We substitute this result for x in the other equations and obtain:

$$\begin{array}{rcrcrcr} 3(5 - 3y + 2z) & + & 5y & + & 6z & = & 7 \\ 2(5 - 3y + 2z) & + & 4y & + & 3z & = & 8, \end{array} \tag{1.44}$$

which, after simplication and bringing the constant numbers to the right-hand side, translates to

$$\begin{array}{rcrcr} -4y & + & 12z & = & -8 \\ -2y & + & 7z & = & -2. \end{array} \tag{1.45}$$

Now we repeat the process, solving the first of these equations for y, which yields $-4y = -12z - 8$ and, after dividing both sides by -4, $y = 3z + 2$. We then substitute y into the second equation yielding $-2(3z + 2) + 7z = -2$. Simplifying again leads to $z - 4 = -2$ and, after adding 4 to both sides, $z = 2$.

Now, we just go backwards, first substituting z in the equation solved for y leading to

$$y = 3 \times 2 + 2 = 8 \tag{1.46}$$

and, second, substituting $z = 2$ and $y = 8$ in the first equation solved for x:

$$x = 5 - 3 \times 8 + 2 \times 2 = -19 + 4 = -15. \tag{1.47}$$

The complete result, hence, is

$$x = -15, y = 8, z = 2.$$

Notice that the approach aims to subsequently *eliminate* variables from the equations. This way, we simplify a system with n equations and unknowns to a system with $n - 1$

equations and unknowns and, then, we just repeat until we are left with one equation with one unknown.

We can reach this goal in a more direct manner by adding (or subtracting) one equation to (or from) the other such that one of the unknowns disappears, *i.e.* reduces to zero. Usually, we have to scale one of the equations to achieve this.

When we look at the previous system once again

$$\begin{array}{rcrcrcr} x & + & 3y & - & 2z & = & 5 \\ 3x & + & 5y & + & 6z & = & 7 \\ 2x & + & 4y & + & 3z & = & 8, \end{array} \tag{1.48}$$

we see that, if we scale the first equation by factor 3 and add it to the second equation, z would fall away:

$$\begin{array}{rcrcrcr} & 3x & + & 9y & - & 6z & = & 15 \\ + & 3x & + & 5y & + & 6z & = & 7 \\ = & 6x & + & 14y & & & = & 22. \end{array} \tag{1.49}$$

Likewise, we can scale the third equation by factor 2 and subtract it from the second equation:

$$\begin{array}{rcrcrcr} & 3x & + & 5y & + & 6z & = & 7 \\ - & 4x & + & 8y & + & 6z & = & 16 \\ = & -x & - & 3y & & & = & -9. \end{array} \tag{1.50}$$

This way, we obtain two equations with two unknowns. We can eliminate one more unknown by scaling the second of these new equations by factor 6 and add it to the first one:

$$\begin{array}{rcrcrcr} & 6x & + & 14y & = & 22 \\ + & -6x & - & 18y & = & -54 \\ = & & & -4y & = & -32. \end{array} \tag{1.51}$$

When we divide both sides of the result by -4, we get $y = 8$, which is the same result we saw before with the elimination method.

We now can go on and eliminate other unknowns by scaling and adding. We should not be frightened to use fractions, when solving equations in a field. We can, for instance, isolate x by scaling the resulting equation 1.50 by the factor $\frac{14}{3}$ and add it to equation 1.49:

$$\begin{array}{rrrrrr} & 6x & + & 14y & = & 22 \\ + & -\frac{14}{3}x & - & 14y & = & -42 \\ = & \frac{4}{3}x & & & = & -20. \end{array} \tag{1.52}$$

After multiplying by 3 and dividing by 4 on both sides, we get $x = -15$, as before.

The generic algorithm is based on these principles of scaling and adding as well as elimination, but does so in a systematic way. In our manual process, we took decisions on which equation to solve and on which equations to add to or subtract from which other. Those decisions were driven by human motives, for instance, to avoid fractions whenever possible. For a systematic algorithm executed on a machine, such considerations are irrelevant. The machine has no peference for integers over fractions.

The algorithm is called *Gaussian elimination*, although it is known to Chinese and Indian mathematicians since late antiquity. We will here discuss the basic form of this algorithm. There is a more advanced form, called *Gauss-Jordan algorithm*, at which we look later. Interesting, however, is the second eponym of the algorithm, Wilhelm Jordan (1842 – 1899), a German geodesist. This underlines the fact that this method – as well as many other methods from linear algebra – has its roots in applied science rather than in pure mathematics.

Both algorithms are based on a data structure of fundamental importance in linear algebra, the *matrix*. We, here, introduce matrices as a mere tool that helps us doing calculations. In algebra, however, matrices are studied as a topic in itself.

Anyway, what is a matrix in the first place? Well, "matrix" is basically a fancy name for what we all know as "table". A matrix consists of rows and columns that are identified by a pair of indices (i, j), where i usually refers to the row and j to the column.

Here we use matrices to represent systems of equations. Each row contains one equation. Each column contains one coefficient, *i.e.* the numbers before the unknowns and, in the last column, we have the constant value on the right-hand side of the equations (this is often called an *augmented matrix*). Our equation above can be represented in matrix form as:

$$\begin{pmatrix} 1 & 3 & -2 & 5 \\ 3 & 5 & 6 & 7 \\ 2 & 4 & 3 & 8 \end{pmatrix}$$

In Haskell, we can define a matrix as a list of lists, where the inner lists represent rows, for instance:

```
data Matrix a = M [[a]]
  deriving (Show, Eq)
```

We can create a matrix for our system by

```
mysystem :: Matrix [Natural]
mysystem = let e1 = [1, 3, −2, 5]
               e2 = [3, 5, 6, 7]
               e3 = [2, 4, 3, 8]
           in  M [e1, e2, e3]
```

The following functions yield the rows and, respectively, the columns of the matrix:

```
rows :: Matrix a → [[a]]
rows (M rs) = rs

cols :: Matrix a → [[a]]
cols (M rs) = go rs
  where go [] = []
        go rs | null (head rs) = []
              | otherwise =
                heads rs : go (tails rs)

heads :: [[a]] → [a]
heads zs = [head z | z ← zs, ¬ (null z)]

tails :: [[a]] → [[a]]
tails = map tail z
```

Obtaining the rows is trivial: the function just returns the list of lists. Columns are bit more difficult. We recursively return the list of the heads of the inner lists, reducing these lists per step to their *tail*s until the lists are empty. This condition is checked on the first inner list. Since, in a matrix, all rows need to have the same size, the first list can act as a model for all lists.

Here are two helper functions to compute the length of one row in the matrix and to compute the length of one column in the matrix:

```
colen :: [[a]] → Int
colen = length

rowlen :: [[a]] → Int
rowlen []     = 0
rowlen [x: _] = length x

columnLength :: Matrix a → Int
columnLength (M ms) = colen ms

rowLength :: Matrix a → Int
rowLength (M ms) = rowlen ms
```

The column length is equivalent to the number of rows in the matrix; the row length is the length of the first row. Again, in a matrix, all rows shall have the same length; the first row, hence, serves as a pattern for the other rows.

Gaussian elimination consists of two steps (one of the improvements of Gauss-Jordan is

that it consists of only one step, but applies this step with more consequence). The first step brings the matrix into a special form, often called *echelon* form. In this form, the matrix contains a triangle of zeros in the lower-left corner like this:

$$\begin{pmatrix} a_{0,0} & a_{0,1} & a_{0,2} & a_{0,3} & a_{0,4} \\ 0 & a_{1,1} & a_{1,2} & a_{1,3} & a_{1,4} \\ 0 & 0 & a_{2,2} & a_{2,3} & a_{2,4} \\ 0 & 0 & 0 & a_{3,3} & a_{3,4} \end{pmatrix}.$$

The echelon form of our matrix is as follows:

$$\begin{pmatrix} 1 & 3 & -2 & 5 \\ 0 & 4 & -12 & 8 \\ 0 & 0 & 4 & 8 \end{pmatrix}$$

The echelon form corresponds to a system of equations where the last equation has been reduced to one unkown; the last but one to two unknowns and so one until the first that remains in its original form.

The second step consists in eliminating and backsubstituting coefficients remaining in the matrix. But let us first look at how to create the echelon form. In Haskell this may be implemented as follows:[1]

```
echelon :: (Eq a, Num a) ⇒ Matrix a → Matrix a
echelon (M ms) = M (go ms)
  where go :: (Eq a, Num a) ⇒ [[a]] → [[a]]
        go rs | null rs ∨
                null (head rs) = rs
              | null rs2       = map (0:) (go (map tail rs))
              | otherwise      = piv : map (0:) (go rs′)
          where rs′            = map (adjustWith piv) (rs1 ++ rs3)
                (rs1, rs2)     = span (λ(n: _) → n ≡ 0) rs
                (piv : rs3)    = rs2
adjustWith :: (Num a) ⇒ [a] → [a] → [a]
adjustWith (m : ms) (n : ns) = zipWith (−) (map (n∗) ms)
                                           (map (m∗) ns)
```

We first look at *adjustWith*. This function takes two lists (of equal length), drops the first elements, scales each of the lists multiplying by the first element of the respective other list and zips the result together by subtracting the corresponding elements. Note that, if we not dropped the first elements, they would be multiplied by the first element of the respective other list; in consequence, both lists would begin with nm. Subracting

[1] This code is based on Matrix.hs, part of the Hugs system

one list from the other would result in a list with a leading zero. The function, instead, just drops the leading element.

Now let us look at how *adjustWith* is used in *echelon*. The main work in *echelon* is done in the local function *go*. This function has two base cases:

1. If the input matrix *rs* is null (it contains no rows) or if its first element is null (it contains only empty rows), the input is already in echelon form and we give it back as is.

2. We look at the local variable *rs2*. This variable is generated as the second element of a tuple resulting from *span* $(\lambda(n\colon \_) \to n \equiv 0)$, i.e. *rs1* will contain the rows with leading zeros and *rs2* will contain those without leading zeros. If *rs2* is the empty list, all rows in *rs* have at least one leading zero; we, therefore, ignore this step and continue with the tail of all rows, adding the zero that we ignored here to the final result again.

Now, in the *otherwise* branch, we use the *adjustWith* function. It is used to generate the local variable rs'. Look at how this variable is generated: (*adjustWith piv*) is mapped on the concatenation *rs1* $+\!\!+$ *rs3*. We already know the variable *rs1*: it contains the rows of *rs* with leading zeros. The second list, *rs3*, is created from *rs2* as $(piv : rs3)$. The pivot (*piv*), hence, is the first row without leading zero and *rs3* consists of all other rows. In other words: we use one row (the pivot) to eliminate one variable from all rows. From here, it is simple: we just apply *go* once again on the result rs' until one of the base cases applies. On each step, we insert zero as head to all rows in the result matrix and, finally, add one more row: the pivot that now contains on more column with a value $\neq 0$ than the rows in the result matrix. The code looks a bit scary on the first sight, but, after going through it step by step, it turns out to be quite simple. But let us go through an example: in the first instance of *go*, we compute

$(rs1, rs2) = ([\,], m)$, where m contains all lines of the matrix;
$(piv, rs3) = ([1, 3, -2, 5], rs3)$, where *rs3* contains the last two lines.

For *adjustWith piv*, we compute, for the first line of *rs3*:

$$\begin{array}{rrrrr} & 3 & 9 & -6 & 15 \\ - & 3 & 5 & 6 & 7 \\ = & 0 & 4 & -12 & 8 \end{array} \tag{1.53}$$

and for the second:

$$\begin{array}{rrrrr} & 2 & 6 & -4 & 10 \\ - & 2 & 4 & 3 & 8 \\ = & 0 & 2 & -7 & 2 \end{array} \tag{1.54}$$

.

With these results, we repeat the process computing

$(rs1, rs2) = ([\,], m)$, where m now contains the two results computed above;
$(piv, rs3) = ([4, -12, 8], [[2, -7, 2]])$.

For *adjustWith piv*, we compute:

$$\begin{array}{rrrr} & 8 & -24 & 16 \\ - & 8 & -28 & 8 \\ = & 0 & 4 & 8 \end{array} \tag{1.55}$$

Now, going back, we add heading zeros to the rows and, per recursion, the pivot resulting in the matrix:

$$\begin{pmatrix} 1 & 3 & -2 & 5 \\ 0 & 4 & -12 & 8 \\ 0 & 0 & 4 & 8 \end{pmatrix}$$

It should be clear, by the way, that *echelon* just applies the second method we discussed above: it systematically scales equations (in *adjustWith*) and subtracts them from each other. Now you may guess that the second step of the algorithm applies the first method, *i.e.* eliminating variables by solving and back-substituting – and you are right:[2]

```
backsub :: Matrix Zahl → [Quoz]
backsub (M ms) = go ms []
  where go [] rs = rs
        go xs rs = go xs' (p : rs)
          where a        = (last xs) !! ((rowlen xs) − 2)
                c        = (last xs) !! ((rowlen xs) − 1)
                p        = c % a
                (M xs')  = eliminate p $ M (init xs)
eliminate :: Quoz → Matrix Zahl → Matrix Zahl
eliminate r (M ms) = M (map (simplify n d) ms)
  where n = numerator   r
        d = denominator r
        simplify n d row  = init (init row') ++ [d * lr − al * n]
          where lr        = last row
                al        = last (init row)
                row'      = map (*d) row
```

Note that, for sake of the topic of this section, we have adapted the code to a specific data type. The function *backsub* expects a system with integer coefficients and presents a result of rational numbers.

[2] This code is based on *Haskell Road*

In *backsub*, we call the local function *go*, which receives the input matrix and an empty result set. When the input matrix is exhausted, we yield the result set. Otherwise, we create the local variables xs' and p. The latter is a rational number generated by dividing the last element of the last row by the penultimate element of that same row. This number, the quotient of the last and the last but one element of the last row, is the first element of the result set.

What does that mean? Well, look at the last line of the matrix. It reads $0, 0, 4, 8$. That is: it contains only two elements. The penultimate element, 4, is the coefficient of the last unknown, z, while the last element, 8, is the constant value on the right-hand side of this equation with one unknown. The last row can thus be rephrased as:

$$4z = 8.$$

That we divide the last element by the last but one corresponds to the simple manipulation that divides both sides of the equation by 4, *i.e.*

$$z = \frac{8}{4} = 2.$$

The other variable xs' is generated by eliminating p from the other lines. Eliminating works as follows: We first multiply all elements in every row by the denominator of p. This corresponds to the following operation; the last but one row of the matrix, for instance, is

$$4y - 12z = 8.$$

When we substitute z by $\frac{8}{4}$ (which, of course, is 2, but let us look at the fraction), we get:

$$4y - \frac{12 \times 8}{4} = 8.$$

We now get rid of the denominator, by multiplying both sides by 4:

$$16y - 12 \times 8 = 32.$$

In the code above, we start by performing this second step, *i.e.* multiplying by the denominator. Note, however, that we later continue to compute with the last and the last but one element of *row*, not of *row′*. In other words, we multiply the denominator only by the elements that precede the penultimate and leave the last two elements as they are.

We then take the last two elements, multiply the last one by the denominator and the penultimate one by the numerator and subtract the latter from the former. That is, we get rid of the denominator, apply multiplication of the numerator to the value that represents z and subtract it from both sides. In abstract algebraic notation, that would look like:

$$ay + bz = c.$$

We know that $z = \frac{n}{d}$, so we can substitute the second term for $\frac{bn}{d}$. We multiply by d and get:

$$ady + bn = cd.$$

Now, we subtract bn from both sides and get

$$ady = cd - bn.$$

Voilà, we have reduced an equation with two unknowns to an equation with only one unknown, namely y. This elimination step is applied to all rows (but the last). Then, the process is repeated using as input the reduced rows.

Let us go through the whole example. The echelon form of our system is

$$\begin{pmatrix} 1 & 3 & -2 & 5 \\ 0 & 4 & -12 & 8 \\ 0 & 0 & 4 & 8 \end{pmatrix}$$

We look at the last line $0, 0, 4, 8$. We set

$$p = \frac{8}{4} = \frac{2}{1}.$$

We then call *eliminate* p on the first two lines of the matrix. Processing the first line, we compute *map* $(*1)$ *row*, which we can ignore. We then set

$$lr = 1 \times 5, al = 2 \times -2$$

and compute $lr - al$, which is 9. The complete result for the first row, hence, is $1, 3, 9$.

For the second row $0, 4, -12, 8$, *eliminate* computes

$$lr = 1 \times 8, al = 2 \times -12$$

and further computes $lr - al$, *i.e.* $8 + 24 = 32$. The complete result for the second row, hence, is $0, 4, 32$. After application of *eliminate*, xs' is thus:

$$\begin{pmatrix} 1 & 3 & 9 \\ 0 & 4 & 32 \end{pmatrix}$$

Now, in *backsub*, we repeat the process with this result. We, again, look at the last line, which now is $0, 4, 32$. We set

$$p = \frac{32}{4} = 8.$$

This goes into the result set and, as you may remember, is the result for y.

We apply *eliminate* on the remaining row, which is $1, 3, 9$. We set

$$lr = 1 \times 9 = 9, al = 8 \times 3 = 24$$

and compute $lr - al$, *i.e.* $9 - 24 = -15$. The complete result for this instance of *eliminate*, hence, is $1, -15$.

We, again, repeat the *backsub* process with this result. There is only one row left and from this line we compute p as

$$p = \frac{-15}{1} = -15,$$

which, as you may remember, is the result for x. Since *init xs* is now $[\,]$, *eliminate* will return $[\,]$ and this terminates the process with the correct result $[-15, 8, 2]$.

1.7 Binomial Coefficients are Integers

To prove that binomial coefficients are integers is quite easy. We will make our argument for coefficients of the form $1 \leq k \leq n$ in $\binom{n}{k}$. For cases outside of this range, the coefficients are defined as 1 for $k = 0$, and as 0 for $k > n$. So, there is nothing to prove in these cases.

We have the equation

$$\binom{n}{k} = \frac{n^{\underline{k}}}{k!}. \tag{1.56}$$

We can prove that $\binom{n}{k}$ is an integer for any n and any $k \leq \frac{n}{2}$ by induction. The induction argument holds, until we have $k > \frac{n}{2}$. At this moment, the factors in the numerator and denominator begin to intersect and the factors now common in numerator and denominator are cancelled out leading to a corresponding case in the lower half of ks, namely the case $k = n - k$. You can easily convince yourself by trying some examples that this is the reason for the symmetry in Pascal's triangle.

Since, in the proof, we have already handled the cases in the lower half of ks by induction up to $\frac{n}{2}$, there is nothing that still needs to be proven. The proof, therefore, consists in the induction argument that if $\binom{n}{k}$ is an integer, then $\binom{n}{k+1}$ is an integer too for $k \leq \frac{n}{2}$.

We first handle the trivial case $k = 1$. Here, we have $\frac{n}{1!=1}$, which, trivially, is an integer. Note that, for this case, the falling factorial, which is defined as the product of the consecutive numbers n to $n - k + 1$, is just n, since $n - 1 + 1 = n$.

For $k = 2$, we have

$$\frac{n(n-1)}{2}.$$

In the numerator we have a product of two consecutive numbers, one of which must be even. We, hence, divide the even one and the denominator by 2 and have an integer.

For $k = 3$, we have

$$\frac{n(n-1)(n-2)}{6}.$$

We now have three consecutive numbers as factors in the numerator and $3! = 6$ in the denominator. One of three consecutive numbers must be divided by 3, since there are only two numbers between any two multiples of 3. If, for instance, $n = 11$, then n and $n-1 = 10$ are not divided by 3, but $n-2 = 9$ is. So we reduce the problem to $k-1 = 2$. But note that there is a difference between the previous case $k = 2$ and the case $k - 1$ at which we are arriving now: In the previous case, we had two consecutive numbers in the numerator, but now we have three numbers that are not necessarily consecutive anymore. It may have been the middle number, $n - 1$, that we divided by 3; then, if n is odd, $n - 2$ is odd as well. However, if n and $n - 2$ are not even, then $n - 1$ must have been and then it must have been divisible by 3 and 2. In consequence, even though the numbers are not consecutive anymore, the divisibility argument still holds.

This is how the induction argument works: for any $k + 1$, we have $k + 1$ consecutive factors in the numerator and we have $(k + 1) \times k!$ in the denominator. Since there are $k + 1$ consecutive numbers in the numerator, all greater $k + 1$, one of them must be divided by $k + 1$. By dividing this number and the denominator by $k + 1$, we reduce the problem to k with k factors in the numerator and $k!$ in the denominator. Now, the factors are not consecutive anymore, but that does not affect the argument: either the

number that was reduced by dividing by $k+1$ is divided by k as well and then we have reduced the problem to $k-2$ already, or, if it is not divided by k, then one of the other numbers must be, because we started with $k+1$ numbers in the first place. $\square$

Let us quickly look at the next example, $k = 4$, just to illustrate the argument once again. With $k = 4$ we have the fraction

$$\frac{n(n-1)(n-2)(n-3)}{24}.$$

There are four consecutive numbers in the numerator, one of which must be divided by 4. We divide this number by 4 and the problem is reduced to the case $k = 3$. Again, the numbers are not consecutive anymore. But if the number that we reduce by dividing by 4 is divisible by 3, then we would have reduced the problem to $k = 2$ already. Otherwise, that number was just a number between two multiples of 3 and the argument does not suffer.

A concrete example makes the argument entirely clear. Consider $\binom{11}{4} = \frac{11\times10\times9\times8}{24}$. One of the numbers in the numerator must divide 4. 11 does not, 10 and 9 either, but 8 does. We divide numerator and denominator by 4 and, with that, reduce the problem to $\frac{11\times10\times9\times2}{6}$. There are 3 consecutive numbers in the numerator, one of which must be divided by 3. 11 and 10 are not divided by 3, but 9 is and, hence, we reduce the problem to $\frac{11\times10\times3\times2}{2}$, which was our base case. We now have the choice to either cancel 2 in the numerator and 2 in the denominator or to divide 10 by 2 in the numerator and cancel 2 in the denominator. If we choose the latter, we divide 10 by 2 and obtain $\frac{11\times5\times3\times2}{1} = 11 \times 5 \times 3 \times 2$, which is $55 \times 6 = 330$.

When calculating binomial coefficients by hand, we see that the main activity in this process is to cancel numbers in the numerator and the denominator. The question arises if we could not spare a lot of computing by using numbers that are already reduced before we start working with them. In particular, we see that the reduction of the numerator tends towards the prime factors of the binomial coefficient. We know that finding the prime factors is a very hard problem. But could there not be a shortcut to the binomial coefficient by using prime factors?

It turns out there is. The solution, however, sounds a bit surreal, since it combines facts that, on the first sight, are completely unrelated. The point is that there is a way to quickly decide for any prime number p whether it is part of the prime factorisation of a binomial coefficient and to even determine how often it occurs in its prime factorisation by counting the number of *borrows* we have to make subtracting k from n in the numeral system of base p.

To determine how often (or if at all) 2 appears in the prime factorisation of $\binom{6}{2}$, we would perform the subtraction $6-2$ in binary format. 6 in binary format is 110 and 2 is just 10. So, we subtract:

$$\begin{array}{cccc} & 1 & 1 & 0 \\ - & & 1 & 0 \\ = & 1 & 0 & 0 \end{array}$$

The final result 100 is 4 in decimal notation and, hence, correct. Since we have not borrowed once, 2 is not a factor of $\binom{6}{2}$. We check the next prime number 3. 6 in base 3 is 20 and 2 is just 2:

$$\begin{array}{cccc} & 2 & 0 \\ - & & 2 \end{array}$$

Now we have to borrow to compute 0 - 2, so we have:

$$\begin{array}{cccc} & 2 & 0 \\ - & \underline{1} & 2 \\ = & 1 & 1 \end{array}$$

11 base 3 is 4 in the decimal system, so the result is correct. Furthermore, we had to borrow once to compute this result. We, therefore, claim that 3 appears once in the prime factorisation of $\binom{6}{2}$.

We look at the next prime, 5. 6 in base 5 is 11 and 2 in base 5 is just 2 again. So we have:

$$\begin{array}{cccc} & 1 & 1 \\ - & & 2 \end{array}$$

Again, we have to borrow to compute 1 - 2:

$$\begin{array}{cccc} & 1 & 1 \\ - & \underline{1} & 2 \\ = & 0 & 4 \end{array}$$

Since 4 in base 5 is just decimal 4, the result, again, is correct. To reach it, we had to borrow once and we, therefore, claim that 5 appears once in the prime factorisation of $\binom{6}{2}$.

The next prime would be 7, but we do not need to go on, since 7 is greater than $n = 6$. Because we multiply n only by values less than n (namely: $n-1, n-2, \ldots, n-k+1$), 7 cannot be a factor of such a number. Our final result, thus, is: $\binom{6}{2} = 3^1 \times 5^1 = 3 \times 5 = 15$. Let us check this result against our usual method to compupte the binomial coefficient: $\frac{6\times5}{2} = 3 \times 5 = 15$. The result is correct.

But what, on earth, has the factorisation of binomial coefficients to do with the borrows in $n - k$? The link is the following theorem:

$$\binom{n}{k} \equiv \prod_{i=0}^{r} \binom{a_i}{b_i} \pmod{p}, \tag{1.57}$$

where the as and bs are the coefficients in the representation of n and k in base p:

$$n = a_r p^r + a_{r-1} p^{r-1} + \cdots + a_1 p + a_0 \tag{1.58}$$

and

$$k = b_r p^r + b_{r-1} p^{r-1} + \cdots + b_1 p + b_0. \tag{1.59}$$

The theorem, hence, claims that $\binom{n}{k}$ is congruent to the product of the coefficients in the representation base p modulo that p. We are back to congruences and modular arithmetic!

The theorem is a corollary of *Lucas' theorem*, which we will now introduce as a lemma to prove the theorem above. Lucas' theorem states that

$$\binom{n}{k} \equiv \binom{\lfloor n/p \rfloor}{\lfloor k/p \rfloor} \binom{n \bmod p}{k \bmod p} \pmod{p}, \tag{1.60}$$

which is exceptionally beautiful, since it decomposes n and k into the two parts of the Euclidian division, the quotient $\lfloor n/p \rfloor$ and the remainder $n \bmod p$. Let us rename the quotient and remainder of n and k, because we will refer to them quite often in this section: let $u = \lfloor n/p \rfloor$ and $v = n \bmod p$, such that $n = up + v$, and let $s = \lfloor k/p \rfloor$ and $t = k \bmod p$, such that $k = sp + t$. We can now rewrite the usual computation of the coefficient

$$\binom{n}{k} = \frac{n}{k} \times \frac{n-1}{k-1} \times \cdots \times \frac{n-k+1}{1} \tag{1.61}$$

as

$$\binom{n}{k} = \frac{up+v}{sp+t} \times \frac{up+v-1}{sp+t-1} \times \cdots \times \frac{up+v-k+1}{1}. \tag{1.62}$$

This formula leads to a cyclic repetition of denominators of the form

$$sp + t - 1, sp + t - 2, \ldots, sp + t - t.$$

We have to be careful with the denominators of the form $sp + t - t = sp$, since, modulo p, they are just zero and the corresponding fraction is thus undefined. But before we get into it, let us look at the t very first numbers, that is the fractions, before the formula reaches a multiple of p for the first time. These fractions are:

$$\frac{up+v}{sp+t} \times \frac{up+v-1}{sp+t-1} \times \cdots \times \frac{up+v-t+1}{sp+t-t+1},$$

which modulo p is

$$\frac{v}{t} \times \frac{v-1}{t-1} \times \cdots \times \frac{v-t+1}{1}.$$

This, in its turn, is just the usual way to define the binomial coefficient for v and t:

$$\binom{v}{t} = \frac{v}{t} \times \frac{v-1}{t-1} \times \cdots \times \frac{v-t+1}{1}. \tag{1.63}$$

But v and t are $n \bmod p$ and $k \bmod p$ respectively and substituting back these values for v and t in the equation leads to

$$\binom{n \bmod p}{k \bmod p} = \frac{n \bmod p}{k \bmod p} \times \frac{(n \bmod p)-1}{(k \bmod p)-1} \times \cdots \times \frac{(n \bmod p)-(k \bmod p)+1}{1} \tag{1.64}$$

and we conclude

$$\binom{n}{k} \equiv \binom{n \bmod p}{k \bmod p} X \pmod{p}, \tag{1.65}$$

where X is the rest of the product after the first t factors we are looking at right now.

Consider the example $\binom{90}{31}$ and the prime 7. For n, we get $u = 90/7 = 12$ and, since $12 \times 7 = 84$, we have $v = 90 \bmod 7 = 6$. For k, we get $s = 31/7 = 4$ and, since $4 \times 7 = 28$, we have $t = 31 \bmod 7 = 3$. The first t factors, we have looked at so far are

$$\frac{90 \times 89 \times 88}{31 \times 30 \times 29}.$$

We take all factors in numerator and denominator modulo 7:

$$\frac{6 \times 5 \times 4}{3 \times 2 \times 1},$$

which, as you can see, is just $\binom{6}{3} = \binom{90 \bmod 7}{31 \bmod 7}$.

Now we will look at the ominous X. Since X is the product with the first $k \bmod p$ factors cut off and the number of factors in the entire product is k, the number of the remaining

factors is a multiple of p. These factors fall into $\frac{k}{p}$ groups each of which contains in the numerator and the denominator one multiple of p and $p-1$ remainders of p. Let us look at the denominators of such a group:

$$\frac{\cdots}{sp} \times \frac{\cdots}{sp+1} \times \cdots \times \frac{\cdots}{sp+p-1}.$$

Since the whole is a product, these values are multiplied with each other and, as we know from Wilson's theorem, the factorial of $p-1$ is $(p-1)! \equiv p-1 \pmod{p}$. We certainly have the same remainders in the numerators modulo p, which, again according to Wilson, are $p-1$ modulo p and, therefore, cancel out. We are then left with the factors that are multiples of p.

Continuing the example, the first of such groups would be

$$\frac{87 \times 86 \times 85 \times 84 \times 83 \times 82 \times 81}{28 \times 27 \times 26 \times 25 \times 24 \times 23 \times 22}$$

Here, 84 in the numerator and 28 in the denominator are multiples of 7. All other numbers are remainders. In the denominator, we have a complete group of remainders $22 \ldots 27$, which are $1 \ldots 6$ modulo 7. These multiplied with each other are, according to Wilson's theorem, congruent to 6 modulo 7. In the numerator, we do not see the whole group at once. Instead, we see two different parts of two groups separated by 84, the multiple of 7, in the middle: $85, 86, 87$, which are congruent to 1, 2 and 3 modulo 7, and $81, 82, 83$, which are congruent to 4, 5 and 6 modulo 7. Multiplying these remainders is, again according to Wilson's theorem, congruent to 6 modulo 7. So, we cancel all these values in numerator and denominator and keep only

$$\frac{84}{28}.$$

As already observed, we have $\lfloor k/p \rfloor = \lfloor 31/7 \rfloor = 4$ of such groups. We are therefore left with 4 fractions with a multiple of 7 in the numerator and the denominator, namely the factors:

$$\frac{84}{28} \times \frac{77}{21} \times \frac{70}{14} \times \frac{63}{7}.$$

When we divide numerator and denominator by 7, we get

$$\frac{12}{4} \times \frac{11}{3} \times \frac{10}{2} \times \frac{9}{1}.$$

and see by this simple trick of black magic that the result is

$$\binom{12}{4} = \binom{\lfloor 90/7 \rfloor}{\lfloor 31/7 \rfloor} = \binom{\lfloor n/p \rfloor}{\lfloor k/p \rfloor}.$$

Unfortunately, there are very few scholars left who would accept magic as proof and so we must continue with the abstract reasoning. Note again that we have taken the first $t = k \bmod p$ terms out in the previous step. The denominators we are left with, when we arrive at fractions with multiples of p in the numerator and denominator, are therefore $k - (k \bmod p), k - (k \bmod p) - p, k - (k \bmod p) - 2p, \ldots, 1$. In the example above, 28 corresponds to $k - (k \bmod p)$: $31 - 3 = 28$, 21 corresponds to $k - (k \bmod p) - p$ and so on.

The numerators are not so clean, but very similar: $n - (k \bmod p) - x, n - (k \bmod p) - x - p, \ldots$ The x in this formula results from the fact that $n - (k \bmod p)$ does not necessarily result in a multiple of p. For instance, $90 - 3 = 87$ is not a multiple of p. x in this case is 3, since $90 - 3 - 3 = 84$, which is a multiple of p. In fact, we can determine the value of x more specifically as $(n \bmod p) - (k \bmod p)$, which is $6 - 3 = 3$, but we do not need to make use of this fact. It is sufficient to realise that each value must be divisible by p and, hence, $(k \bmod p) + x < p$. When we now divide by p, we get for each factor

$$\frac{\lfloor (n - (k \bmod p) - x_i - a_i p)/p \rfloor}{\lfloor (k - (k \bmod p) - b_i p)/p \rfloor},$$

where the as and bs run from 0 to the number of groups we have minus 1, *i.e.* $\lfloor k/p \rfloor - 1$.

Since the second term of the differences in numerator and denominator are remainders of p that, together with n and, respectively, k, are multiples of p, this is just the same as saying

$$\frac{\lfloor (n - a_i p)/p \rfloor}{\lfloor (k - b_i p)/p \rfloor},$$

which of course is

$$\frac{\lfloor n/p \rfloor - a_i}{\lfloor k/p \rfloor - b_i}.$$

Since the as and bs run from 0 to the number of the last group, we get this way the product

$$\frac{\lfloor n/p \rfloor}{\lfloor k/p \rfloor} \times \frac{\lfloor n/p \rfloor - 1}{\lfloor k/p \rfloor - 1} \times \frac{\lfloor n/p \rfloor - 2}{\lfloor k/p \rfloor - 2} \times \cdots \times \frac{\lfloor n/p \rfloor - \lfloor k/p \rfloor + 1}{\lfloor k/p \rfloor - \lfloor k/p \rfloor + 1},$$

which we immediately recognise as the computation for

$$\binom{\lfloor n/p \rfloor}{\lfloor k/p \rfloor}.$$

You, hopefully, remember that this is the X, we left over in equation 1.65. Substituting for X we derive the intended result:

$$\binom{n}{k} \equiv \binom{n \bmod p}{k \bmod p}\binom{\lfloor n/p \rfloor}{\lfloor k/p \rfloor} \pmod{p} \tag{1.66}$$

and this completes the proof. □

But we have to add an important remark. Binomial coefficients with $k > n$ are defined to be zero. The equation, thus, tells us that the prime p divides the coefficient if $k \bmod p > n \bmod p$. For instance $\binom{8}{3}$, which is $\frac{8\times7\times6}{6} = 8 \times 7 = 56$, is divided by 7, since 7 appears as a factor in the numerator and, indeed: $\binom{8 \bmod 7}{3 \bmod 7} = \binom{1}{3}$. This would also work with $\binom{9}{3}$, which is $\frac{9\times8\times7}{6} = 3 \times 4 \times 7 = 84$, where 7, again, appears as a factor in the numerator and $\binom{9 \bmod 7}{3 \bmod 7} = \binom{2}{3}$. It does not work with $\binom{9}{2}$, which is $\frac{9\times8}{2} = 9 \times 4 = 36$, since $\binom{9 \bmod 7}{2 \bmod 7} = \binom{2}{2} = 1$ and, indeed: $36 \bmod 7 = 1$. Let us memorise this result: a prime p divides a binomial coefficient $\binom{n}{k}$, if $k \bmod p > n \bmod p$.

We, finally, come to the corollary, which we wanted to prove in the first place. We need to prove that

$$\binom{n}{k} \equiv \prod_{i=0}^{r} \binom{a_i}{b_i} \pmod{p}, \tag{1.67}$$

where the as and bs are the coefficients in the representation of n and k base p. We now calculate u, v, s and t, as we have done before, as $u = \lfloor n/p \rfloor$, $v = n \bmod p$, which is just the last coefficient in the p-base representation of n a_0, $s = \lfloor k/p \rfloor$ and $t = k \bmod p$, which is just the last coefficient b_0.

The p-base representations of n and k are $n = a_r p^r + \cdots + a_1 p + a_0$ and $k = b_r p^r + \cdots + b_1 p + b_0$. If we divide those by p, we get $u = a_r p^{r-1} + \cdots + a_1$ and $s = b_r p^{r-1} + \cdots + b_1$ with a_0 and b_0 as remainders.

From Lucas' theorem we conclude that

$$\binom{n}{k} \equiv \binom{u}{v}\binom{a_0}{b_0} \pmod{p}. \tag{1.68}$$

Now we just repeat the process for u and v:

$$\binom{n}{k} \equiv \binom{\lfloor u/p \rfloor}{\lfloor v/p \rfloor}\binom{a_1}{b_1}\binom{a_0}{b_0} \pmod{p} \tag{1.69}$$

and continue until we have

$$\binom{n}{k} \equiv \binom{a_r}{b_r}\dots\binom{a_1}{b_1}\binom{a_0}{b_0} \pmod{p}, \tag{1.70}$$

which then concludes the proof. □

We see immediately that p divides $\binom{n}{k}$, when at least one digit of k in the p-base representation is greater than the corresponding digit of n, because, in this case, the corresponding binomial coefficient is zero and, in consequence, the whole product modulo p becomes zero.

That a digit of k is greater than the corresponding digit of n implies that, on subtracting k from n, we have to borrow from the next place. Therefore, if we have to borrow during subtraction, then p divides $\binom{n}{k}$ and, thus, is a prime factor of that number.

So, we divide $\binom{n}{k}$ by p leading to the product in equation 1.70 with one pair of digits removed and search again for a pair of digits where $k > n$. If we find one, the number $\binom{n}{k}$ is divided twice by p, so p appears twice in the factorisation of that number. We again divide by p and repeat the process until we do not find a pair $k > n$ anymore. Then we know how often p appears in the prime factorisation of $\binom{n}{k}$. If we do this for all primes $\leq n$, we learn the complete prime factorisation of $\binom{n}{k}$.

We now will implement this logic in Haskell. We start with the notion of borrows:

```
borrows :: Natural → Natural → Natural → Natural → Natural → Natural
borrows _ 0 _ _ _ = 0
borrows p u v s t | v < t = 1 + borrows p (u `div` p) (u `rem` p)
                                (s `div` p) ((s `rem` p) + 1)
                  | otherwise = borrows p (u `div` p) (u `rem` p)
                                (s `div` p) (s `rem` p)
```

The function *borrows* takes five arguments all of our old type *Natural*. The first argument is the prime; the next four arguments are u, v, s and t, that is $u = \lfloor n/p \rfloor$, $v = n \bmod p$, $s = \lfloor k/p \rfloor$ and $t = k \bmod p$.

If $u = 0$, we are through and no more borrows are to be found. Otherwise, if $v < t$, we have to borrow. The borrow is actually seen in the recursive call of borrows, where we increment $s \bmod p$ by 1. We also add 1 to the overall result. Otherwise, we call borrows with the quotients and remainders of u and s. The recursion implements the logic we see in equation 1.70: we reduce the product factor by factor by dividing by p; on each step, we check if a borrow occurs and continue with the next step.

This is the heart of our algorithm. However, we can improve on this. There are cases we can decide immediately without going through the whole process. Consider a prime $p \leq n-k$. Since the numerator in the computation of the binomial coefficient runs from $n \ldots (n-k+1)$, this prime will not appear directly in the numerator. It could of course still be a factor of one of the numbers $n, n-1, \ldots, n-k+1$ and, as such, be a factor of the resulting number. But then it must be less than or at most equal to the half one of those numbers. Otherwise, there would be no prime number by which we could multiply p to obtain one of those number. In consequence, when $p \leq n-k$ and $p > n/2$, p cannot divide $\binom{n}{k}$.

On the other hand, if $p > n-k$, then it appears in the numerator; if also $p > k$, then it will not appear in the denominator. In consequence, if $p > k$ and $p > n-k$, then it will not be cancelled out and is therefore prime factor of the binomial coefficient. Furthermore, it can appear only once in the numerator. There are only k consecutive numbers in the numerator and $p > k$. If we assume there is a factor ap with $a \geq 1$, then we will reach n in one direction and $n-k+1$ in the other direction, before we reach either $(a+1)p$ or $(a-1)p$. Therefore, $a = 1$ or, in other words, p appears only once in the prime factorisation.

Finally, if $n \bmod p < k \bmod p$, we know for sure that p divides the number at least once. If $p^2 > n$, then we know that p divides $\binom{n}{k}$ exactly once.

We will implement these one-step decisions as filters in a function that calls *borrows*:

```
powOfp :: Natural → Natural → Natural → Natural
powOfp n k p | p ⩽ n − k ∧ p > n `div` 2          = 0
             | p > k ∧ p > n − k                  = 1
             | p ∗ p > n ∧ n `rem` p < k `rem` p  = 1
             | otherwise = borrows p (n `div` p) (n `rem` p)
                                     (k `div` p) (k `rem` p)
```

Now we implement a new variant of *choose* making use of borrows:

```
choose3 :: Natural → Natural → Natural
choose3 n k  = product (map f ps)
  where ps   = takeWhile (⩽ n) allprimes
        f p  = p ↑ (powOfp n k p)
```

The implementation is quite simple and there is certainly room for optimisation. It just maps $p^{(\text{powOfp } n\ k\ p)}$ on all primes up to n and builds the product of the result. A possible improvement is to use *fold* instead of *map* and not to add primes to the result for which *powOfp* yields 0. That would reduce the size of the resulting list of primes drastically. With *map*, there are a lot of 1s, in fact, most of the elements are 1 in most cases.

But let us investigate the running time of *choose3* compared to that of $\frac{n^k}{k!}$ in more general terms. The running time of the fraction is a function of k, such as $2k-1$, since

we have $k-1$ multiplications in numerator and denominator and one division, as already discussed in a previous chapter. Since we are not too much interested in the details of the operations, *i.e.* the cost of single multiplications and divisions and the cost of *borrows* for one prime, we will use the *big-O* notation, for instance: $\mathcal{O}(2k-1)$. This notation tells us that there is a function of the form $2k-1$, which is a limit for our function, the running time of the fraction. In other words, for huge input values, the function within the $\mathcal{O}$ is equal to or greater than our function.

The running time of *choose3*, by contrast, is a function of $\Pi(n)$, that is the number of primes up to n, approxmimately $n/\ln n$. In big-O notation, we state that the running time of *choose3* is $\mathcal{O}(n/\ln n)$.

For large ns and small ks, *e.g.* $\binom{1000000}{2}$, the fraction appears to be much better. Of course, the multiplications in the numerator are heavy, since the complexity of multiplication grows with the number of digits of the factors, but there are still only few such multiplications. The multiplications in the denominator, in compensation, are trivial with small ks.

The complexity of *choose3* for this specific number is $1000000/\ln 1000000 \approx 72382$ and, as such, of course much worse. With larger ks, however, the picture changes. Even though we can reduce the complexity of the fraction for cases where $k > n/2$ to cases with $k < n/2$, as discussed before, there is sufficient room for ks around $n/2$ that makes *choose3* much more efficient than the fraction. In general, we can say that *choose3* is more efficient, whenever $\frac{n}{2\ln n} + 1 < k < n - \frac{n}{2\ln n} - 1$. For the specific example of $n = 1000000$, *choose3* is more efficient for $36192 < k < 963807$.

The fact underlying the algorithm, the relation between the number of occurrences of a prime p in the factorisation of a binomial coefficient and the number of borrows in the subtraction of n and k in base-p, was already known in the 19^{th} century, but was apparently forgotten during the 20^{th} century. It was definitely mentioned by German mathematician Ernst Kummer (1810 – 1893), but described as an algorithm apparently only in 1987 in a short, but nice computer science paper by French mathematician Pascal Goetgheluck who rediscovered the relation by computer analysis.

Lucas' theorem is named for Édouard Lucas (1842 – 1891), a French mathematician who worked mainly in number theory, but is also known for problems and solutions in recreational math. The *Tower of Hanoi* is of his invention. Lucas died of septicemia in consequence of a household accident where he was cut by a piece of broken crockery.

1.8 Euler's Totient Function

The constructor function *ratio* of our *Ratio* data type reduces fractions to their canonical form by dividing numerator and denominator by their *gcd*. If the *gcd* is just 1, numerator and denominator do not change. If the *gcd* is not 1, however, the numbers actually do

change. For instance, the fraction $\frac{1}{6}$ is already in canonical form. The fraction $\frac{3}{6}$, however, is not, since $gcd(6,3)$ is 3 and so we reduce: $\frac{3/3=1}{6/3=2} = \frac{1}{2}$.

This leads to the observation that not all fractions possible with one denominator manifest with the *Ratio* number type. For the denominator 6, for example, we have only $\frac{1}{6}$ and $\frac{5}{6}$. For other numerators, we have $\frac{2}{6} = \frac{1}{3}$, $\frac{3}{6} = \frac{1}{2}$ and $\frac{4}{6} = \frac{2}{3}$. We could write a function that shows all proper fractions for one denominator, *e.g.*

$$\begin{aligned}&\mathit{fracs1} :: \mathit{Natural} \to [\mathit{Ratio}]\\&\mathit{fracs1}\ n = [x \,\%\, n \mid x \leftarrow [1 \mathinner{..} n-1]]\end{aligned}$$

fracs1 6, for instance, gives:

$[Q\ 1\ 6, Q\ 1\ 3, Q\ 1\ 2, Q\ 2\ 3, Q\ 5\ 6]$,

which is easier to read in mathematical notation:

$$\frac{1}{6}, \frac{1}{3}, \frac{1}{2}, \frac{2}{3}, \frac{5}{6}.$$

How many proper fractions are there for a specific denominator? In the example above, there are only two proper fractions with the denominator 6. We devise a function to filter out the fractions that actually preserve the denominator 6:

$$\begin{aligned}&\mathit{fracs2} :: \mathit{Natural} \to [\mathit{Ratio}]\\&\mathit{fracs2}\ n = \mathit{filter}\ (\lambda(Q\ \_\ d) \to d \equiv n)\ (\mathit{fracs1}\ n)\end{aligned}$$

This function, again applied to 6 would give (in mathematical notation):

$$\frac{1}{6}, \frac{5}{6}.$$

Applied to 12, it would give:

$$\frac{1}{12}, \frac{5}{12}, \frac{7}{12}, \frac{11}{12}.$$

It is easy to see that the numerators of the fractions with a given denominator n, correspond to the group of numbers $g < n$ coprime to n. We, hence, could also create a function that just finds the coprimes from the range $0 \ldots n-1$:

$$\begin{aligned}&\mathit{coprimes} :: \mathit{Natural} \to [\mathit{Natural}]\\&\mathit{coprimes}\ n = [x \mid x \leftarrow [0 \mathinner{..} n-1], \mathit{gcd}\ n\ x \equiv 1]\end{aligned}$$

With little surprise, we see that *coprimes* 12 gives

$$1, 5, 7, 11.$$

Mathematicians like to quantify things and so it is no wonder that there is a well known function, Euler's *totient* function, often denoted $\varphi(n)$, that actually counts the coprimes. This is easily implemented:

$$tot :: Natural \to [Natural]$$
$$tot = fromIntegral \circ length \circ coprimes$$

Let us have a look at the results of the totient function for the first 20 or so numbers:

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	...
1	1	2	2	4	2	6	4	6	4	10	4	12	6	8	8	16	6	18	...

We first see that the totient of a prime p is $p-1$: $\varphi(2) = 1$, $\varphi(3) = 2$, $\varphi(5) = 4$, $\varphi(7) = 6$, $\varphi(11) = 10, \ldots$ Of course, that is the group of remainders of that prime – the previous chapter was dedicated almost entirely to that group!

The totients of composites are different. They slowly increase, but many numbers have the same totient. Since the difference between primes and composites lies in the fact that composites have divisors different from 1 and themselves, it is only natural to suspect that there is a relation between the totients of composites and their divisors. For instance, 2 has one divisor: 1. The totien of 1 is $\varphi(1) = 1$ and that of 2, $\varphi(2)$ is 1 as well. It may just be one of those peculiarities we see with small numbers, but what we see is the curious relation $\varphi(1) + \varphi(2) = 2$. We could formulate the hypothesis that the sum of the totients of the divisors of a number n is that number n. $n = \varphi(1)+\varphi(2)+\cdots+\varphi(n)$ or, more elegantly:

$$n = \sum_{d|n} \varphi(d). \tag{1.71}$$

Let us check this hypothesis. From 2, we go on to 3, but 3 is prime and has only the divisors 1 and 3 and, trivially, $\varphi(1) + \varphi(3) = 3$, since, for any prime p: $\varphi(p) = p - 1$. We go on to 4. The divisors of 4 are 1,2 and 4: $\varphi(1) + \varphi(2) + \varphi(4) = 1 + 1 + 2 = 4$. The next number, 5, again is prime and we go on to 6, which has the divisors 1, 2, 3, 6: $\varphi(1) + \varphi(2) + \varphi(3) + \varphi(6) = 1 + 1 + 2 + 2 = 6$. Until here, the equation is confirmed. Let us jump a bit forward: 12 has the divisors 1, 2, 3, 4, 6, 12: $\varphi(1) + \varphi(2) + \varphi(3) + \varphi(4) + \varphi(6) + \varphi(12) = 1 + 1 + 2 + 2 + 2 + 4 = 12$. The equation appears to be true. But can we prove it?

In fact, it follows from a number of fundamental, but quite simple theorems that one would probably tend to take for granted on first encounter. One of these theorems is related to cardinality of set union. The theorem states that

$$|S1 \cup S2| = |S1| + |S2| - |S1 \cap S2| \tag{1.72}$$

that is: the cardinality of the union of two sets equals the sum of the cardinalities of the two sets minus the cardinality of the intersection of the two sets.

Proof: The intersection of two sets $S1$ and $S2$ contains all elements that are both in $S1$ and $S2$. The union of two sets contains all elements of $S1$ and $S2$. But those elements that are in both sets will appear only once in the union, since this is the definition of the very notion of *set*. We can therefore first build a collection of all elements in the sets including the duplicates and then, in a second step, remove the duplicates. The elements that we remove, however, are exactly those that are also elements of the intersection. The number of elements in the union, hence, is exactly the sum of the numbers of elements of the individual sets minus the number of duplicates. □

There are two corollaries that immediately follow from this theorem. First, for two disjoint sets $S1$ and $S2$, *i.e.* sets for which $S1 \cap S2 = \varnothing$, the equation above simplifies to:

$$|S1 \cup S2| = |S1| + |S2|. \tag{1.73}$$

This is trivially true, since $|\varnothing| = 0$.

Second, for sets that are pairwise disjoint (but only for those!), we can derive the general case:

$$\left| \bigcup_{i=1}^{n} S_i \right| = \sum_{i=1}^{n} |S_i|, \tag{1.74}$$

where $\bigcup$ is the union operator for n sets, where n is not necessarily 2. It, hence, does for $\cup$ what $\sum$ does for $+$. Systems of such operators that are applied to an arbitrary number of operands are called σ-algebras. But, for the time being, that is just a fancy word.

The next fundamental theorem states that the sum of the divisors of a number n equals the sum of the fractions $\frac{n}{d_i}$, where $d_i = d_1, d_2, \ldots, d_r$ are the divisors of n. More formally, the theorem states:

$$\sum_{d|n} d = \sum_{d|n} \frac{n}{d}. \tag{1.75}$$

The point, here, is to see that if d is a divisor of n, then $\frac{n}{d}$ is a divisor too. That d is a divisor means exactly that: n divided by d results in another integer m, such that $d = \frac{n}{m}$ and $dm = n$. Since the set of divisors of n contains all divisors of n and the set of quotients $\frac{n}{d}$ contains quotients with all divisors, the two sets are equal. The only aspect that changes, when we see these sets as sequences of numbers, is the order. Since order has no influence on the result of the sum, the two sums are equal. □

For the example $n = 12$, the divisors are $1, 2, 3, 4, 6, 12$. The quotients generated by

dividing n by the divisors are $12, 6, 4, 3, 2, 1$. The sum of the first sequence is $1+2+3+4+6+12 = 28$. The sum of the second sequence is $12+6+4+3+2+1 = 28$.

It remains to note that, when we have a sum of functions, then still $\sum_{d|n} f(\frac{n}{d}) = \sum_{d|n} f(d)$, since the values to which the function is applied are still the same in both sets.

Equipped with these simple tools, we return to the sum of the totients of the divisors. We start by defining a set S_d that contains all numbers, whose *gcd* with n is d:

$$S_d = \{m \in \mathbb{N} : 1 \leq m \leq n, \gcd(n, m) = d\}. \tag{1.76}$$

In Haskell this would be:

$s :: Natural \to Natural \to [Natural]$
$s\ n\ d = [m \mid m \leftarrow [1\,..\,n], gcd\ n\ m \equiv d]$

When we map this function with $n = 12$ on the divisors of 12, $map\ s\ 12\ [1, 2, 3, 4, 6, 12]$, we get:

$[1, 5, 7, 11]$
$[2, 10]$
$[3, 9]$
$[4, 8]$
$[6]$
$[12]$.

We see six pairwise disjoint sets whose union equals the numbers $1 \ldots 12$. The first set contains the coprimes of 12, since we ask for m, such that $\gcd(12, m) = 1$. The next set contains the numbers, such that $\gcd(12, m) = 2$, the next, the numbers, such that $\gcd(12, m) = 3$ and so on. In other words, these lists together contain all numbers $1 \ldots 12$ partitioned according to their greatest common divisor with $n = 12$. Note that the lists together necessarily contain all the numbers in the range $1 \ldots n$, since, either a number does not have common divisors with n, then it is in the first set for $\gcd(n, m) = 1$, or it has a common divisor with n. Then it is in one of the other sets. This is just what the set S_d mapped on the divisors of n is about.

The sets are also necessarily disjoint from each other, since no number m would, on one occasion, have a *gcd* d_1 with n and, on another, a distinct *gcd* d_2 with the same n. It either shares d_1 as greatest common divisor with n or divisor d_2. It, hence, is either in set S_{d_1} or in set S_{d_2}.

But there is more. The set for divisor 2 contains 2 and 10. These numbers divided by 2 give 1 and 5. $\frac{12}{2}$ is 6 and 1 and 5 are the coprimes of 6. The set for divisor 3 contains 3 and 9; these numbers divided by 3 are 1 and 3. $\frac{12}{3}$ is 4 and 1 and 3 are the coprimes of 4 and so on. In other words, the sets that we see above contain numbers m that, divided by the corresponding divisor d, $\frac{m}{d}$, are the coprimes of $\frac{n}{d}$. This results from the fact that

these numbers and the divisor are related to each other by the gcd. When we have two numbers m and n and we compute their gcd: $d = \gcd(n, m)$, then $\frac{n}{d}$ is coprime to $\frac{m}{d}$, since we divide them by the biggest number that divides both. Therefore, all numbers in the set S_d are necessarily coprimes of $\frac{n}{d}$.

Can there be a coprime of $\frac{n}{d}$ (less than $\frac{n}{d}$) that is not in the set S_d? We created the list of coprimes by first computing m, such that $\gcd(n, m) = d$, and then $c = \frac{m}{d}$. Now, let us assume that there is a coprime c that escapes this filter. In other words, there is another number $k \neq d$, such that $\gcd(n, m) = k$ and $c = \frac{m}{k}$. To be a coprime of interest, we must have $\frac{m}{k} < \frac{n}{d}$. Since $\frac{dn}{d} = n$, we must have $\frac{dm}{k} < n$. This number must therefore appear in one of the S_{d_i}. We can ask: in which one? The answer is $\gcd(n, \frac{dm}{k})$. There are two candidates: d and $\frac{m}{k}$. But $\frac{m}{k}$ cannot be a divisor of n, since k is the greatest divisor m and n have in common. They do not share any other divisor, not even $\frac{m}{k}$. Therefore, d must be the greatest common divisor of n and $\frac{dm}{k}$. But then this number appears in S_d and $\frac{m}{k}$ does not escape our filter.

It follows that each of the sets S_d contains exactly those numbers that divided by d are the coprimes of $\frac{n}{d}$. The size of each of these sets is thus the totient number of $\frac{n}{d}$:

$$|S_d| = \varphi\left(\frac{n}{d}\right). \tag{1.77}$$

To complete the proof, we now have to extend the relation between one of those sets and the totient of one $\frac{n}{d}$ to that between the union of all the S_{d_i} and the sum of the totient numbers of the divisors. From cardinality of disjoint sets (equation 1.74) we know that the cardinality of the union of disjoint sets is the sum of the cardinality of each of the sets, so we have:

$$\left|\bigcup_{d|n} S_d\right| = \sum_{d|n} \varphi\left(\frac{n}{d}\right). \tag{1.78}$$

From sum of divisors (equation 1.75) we know even further that the sum of $\frac{n}{d}$ equals the sum of d, therefore:

$$\left|\bigcup_{d|n} S_d\right| = \sum_{d|n} \varphi(d). \tag{1.79}$$

We have seen that the union of the S_{d_i} for a given n contains all numbers in the range $1 \ldots n$:

$$\bigcup_{d|n} S_d = \{1 \ldots n\}. \tag{1.80}$$

Since the set $\{1 \ldots n\}$ contains n numbers, we can conclude that

$$\left| \bigcup_{d|n} S_d \right| = n, \tag{1.81}$$

from which, together with equation 1.79, we then can conclude that

$$\sum_{d|n} \varphi(d) = n. \quad \square \tag{1.82}$$

We could define a recursive function very similar to Pascal's rule that exploits this relation. We first define a function to get the divisors

$$\begin{aligned}
&\mathit{divs} :: \mathit{Natural} \to [\mathit{Natural}] \\
&\mathit{divs}\ n = [\, d \mid d \leftarrow [1\,..\,n], \mathit{rem}\ n\ d \equiv 0]
\end{aligned}$$

Then we add up the totients of these numbers (leaving n out, because that is the one we want to compute) and subtract the result from n and, this way, obtain the totient number of n:

$$\begin{aligned}
&\mathit{divsum} :: \mathit{Natural} \to [\mathit{Natural}] \\
&\mathit{divsum}\ 1 = 1 \\
&\mathit{divsum}\ 2 = 1 \\
&\mathit{divsum}\ n = n - \mathit{sum}\ [\mathit{divsum}\ d \mid d \leftarrow \mathit{divs}\ n, d < n]
\end{aligned}$$

Another property of the totient function is multiplicity of totients of coprimes, that is

$$\varphi(a) \times \varphi(b) = \varphi(ab), \text{if}\, \gcd(a, b) = 1. \tag{1.83}$$

For instance, the coprimes of 3 are 1 and 2; those of 5 are 1, 2, 3 and 4. $\varphi(3)$, hence, is 2 and $\varphi(5)$ is 4. $\varphi(3 \times 5 = 15)$ is 8, which also is 2×4. Indeed, the coprimes of 15 are 1, 2, 4, 7, 8, 11, 13 and 14. An example of two coprimes that are not both primes is 5 and 8. $\varphi(5) = 4$ and $\varphi(8) = 4$. $\varphi(5 \times 8 = 40) = 16$, which also is 4×4.

This property might look surprising at the first sight, but is becomes almost trivial in the light of the Chinese Remainder theorem. For two coprimes a and b and their sets of coprimes A and B, we can, for any $a_i \in A$ and $b_j \in B$ create congruence systems of the form

$$\begin{aligned}
x &\equiv a_i \pmod{a} \\
x &\equiv b_j \pmod{b}
\end{aligned}$$

The Chinese Remainder theorem guarantees that, for every case, their is a solution that is unique modulo ab, *i.e.* there are no two different systems with the same solution and there is no system without a solution. Since the solutions are unique modulo ab, there must be exactly one number in the group of coprimes of ab for any combination of a_i and b_j. Since there are $|A| \times |B|$ combinations of all elements of A and B, $\varphi(ab)$ must be $\varphi(a) \times \varphi(b)$.

To illustrate that, we can create all the combinations of as and bs and then apply the Chinese remainder on all of them. First we create all combinations of as and bs:

$$
\begin{aligned}
&\mathit{consys} :: \mathit{Natural} \to \mathit{Natural} \to [[\mathit{Natural}]] \\
&\mathit{consys}\ a\ b = \mathit{concatMap}\ \mathit{mm}\ (\mathit{coprimes}\ b) \\
&\quad \textbf{where}\ \mathit{mm}\ y = [[x, y] \mid x \leftarrow \mathit{coprimes}\ a]
\end{aligned}
$$

The result for *consys* 5 8, for instance, is:

$[1,1],[2,1],[3,1],[4,1],$
$[1,3],[2,3],[3,3],[4,3],$
$[1,5],[2,5],[3,5],[4,5],$
$[1,7],[2,7],[3,7],[4,7]$

Now we map *chinese* on this:

$$
\begin{aligned}
&\mathit{china} :: \mathit{Natural} \to \mathit{Natural} \to [\mathit{Natural}] \\
&\mathit{china}\ a\ b = \mathit{map}\ (\mathit{esenihc}\ [a, b])\ (\mathit{consys}\ a\ b) \\
&\quad \textbf{where}\ \mathit{esenihc} = \mathit{flip}\ \mathit{chinese}
\end{aligned}
$$

Note that, to map *chinese* on *consys*, we have to flip it. *chinese* expects first the congruences and then the moduli, but we need it the other way round.

This is the result for *china* 5 8:

$1, 11, 21, 31, 17, 27, 37, 7, 33, 3, 13, 23, 9, 19, 29, 39$

and this is the result for *coprimes* 40:

$1, 3, 7, 9, 11, 13, 17, 19, 21, 23, 27, 29, 31, 33, 37, 39.$

When you sort the result of *china* 5 8, you will see that the results are the same.

We are now approaching the climax of this section. There is a little fact we need, before we can go right to it, which may appear a tiny curiosity. This curiosity is yet another property of the totient function concerning the totient of powers of prime numbers:

$$\varphi(p^k) = p^k - p^{k-1}. \tag{1.84}$$

That is, if p is prime, then the totient of p^k equals the difference of this number p^k and the previous power of that prime p^{k-1}. An example is 27, which is 3^3. We compute $27 - 3^2 = 27 - 9 = 18$, which is indeed $\varphi(27)$.

The proof is quite simple. Since p is prime, its powers have only one prime factor, namely p. When we say *prime power*, we mean exactly this: a number whose factorisation consists of one prime raised to some $k \geq 1$: p^k. Therefore, the only numbers that share divisors with p^k are multiples of p less than or equal to p^k: p, $2p$, $3p$, $\ldots$, p^k. The 9 numbers that share divisors with 27 are:

$$3, 6, 9, 12, 15, 18, 21, 24, 27.$$

How many multiples of p less than or equal to p^k are there? There are p^k numbers in the range $1 \ldots p^k$, every p^{th} number of which is a multiple of p. There, hence, are $\frac{p^k}{p} = p^{k-1}$ numbers that divide p^k. The number of coprimes in this range is therefore p^k minus that number and, thus, $p^k - p^{k-1}$. □

We can play around a bit with this formula. The most obvious we can do is to factor p^{k-1} out to get $p^{k-1}(p-1)$. So, we could compute $\varphi(27) = 9 \times 2 = 18$. Even more important for the following, however, is the formula at which we arrive by factoring p^k out. We then get

$$\varphi\left(p^k\right) = p^k\left(1 - \frac{1}{p}\right) \tag{1.85}$$

This formula leads directly to a closed form for the totient function, namely *Euler's product formula*. Any number can be represented as a product of prime powers: $n = p_1^{k_1} p_2^{k_2} \ldots$ Since the ps in this formula are all prime, the resulting prime powers are for sure coprime to each other. That means that the multiplicity property of the totient function applies, *i.e.*

$$\varphi(n) = \varphi\left(p_1^{k_1}\right)\varphi\left(p_2^{k_2}\right)\ldots.$$

We now can substitute the totient computations of the prime powers on the right-hand side by the formula in equation 1.85 resulting in

$$\varphi(n) = p_1^{k_1}\left(1 - \frac{1}{p_1}\right)p_2^{k_2}\left(1 - \frac{1}{p_2}\right)\ldots$$

We regroup the formula a bit to get:

$$\varphi(n) = p_1^{k_1} p_2^{k_2} \ldots (1 - \frac{1}{p_1})\left(1 - \frac{1}{p_2}\right)\ldots$$

and see that we have all the prime factors of n and then the differences. The prime factors multiplied out result in n, so we can simplify and obtain Euler's product formula:

$$\varphi(n) = n \prod_{p|n} \left(1 - \frac{1}{p}\right). \tag{1.86}$$

Consider again the example $n = 12$. The formula claims that

$$\varphi(12) = 12 \times \left(1 - \frac{1}{2}\right)\left(1 - \frac{1}{3}\right),$$

since the prime factors of 12 are 2 and 3. Let us see: $1 - \frac{1}{2}$ is just $\frac{1}{2}$, $1 - \frac{1}{3}$ is $\frac{2}{3}$. We therefore get $12 \times \frac{1}{2} \times \frac{2}{3}$. $12 \times \frac{1}{2} = 6$ and $6 \times \frac{2}{3} = \frac{12}{3} = 4$, which, indeed, is $\varphi(12)$.

We can use this formula to implement a variant of *tot* in Haskell:

```
ptot :: Natural → Natural
ptot n = let r = ratio n 1
         in  case r * product [ratio (p − 1) p | p ← nub (trialfact n)] of
             Q t 1 → t
             _     → error "Totient not an integer"
```

Note that we use the *Ratio* number type. To convert it back to *Natural*, we check if the denominator is 1, *i.e.* if the resulting fraction is indeed an integer. If so, we return the numerator. Otherwise, we create an error, which, of course, should not occur if Euler was right.

But what is so special about this function? Well, it leads us quickly to an old friend. Consider a squarefree number n, that is a number where the exponents in the prime factorisation are all 1: $n = p_1 p_2 \ldots p_r$. We perform some simple transformations on 1.86. First, we transform the factors $1 - \frac{1}{p}$ to $\frac{p}{p} - \frac{1}{p} = \frac{p-1}{p}$. Then we have:

$$\varphi(n) = n \frac{p-1}{p} \frac{q-1}{q} \ldots \tag{1.87}$$

As you may have noticed, this is the form in which we implemented Euler's formula in *ptot*. Now we multiply this out:

$$\varphi(n) = \frac{n(p-1)(q-1)\ldots}{pq\ldots} \tag{1.88}$$

Since n is squarefree, the denominator, the product of the prime factors, equals n. n and the entire denominator, hence, cancel out. Now we have:

$$\varphi(n) = (p-1)(q-1)\ldots \tag{1.89}$$

That is, the totient number of a squarefree number is the product of the prime factors, each one reduced by one, a result that follows from multiplicity of totients of numbers that are coprime to each other.

The formula is quite interesting. It is very close to the formula we used for the RSA algorithm to find a number t for which

$$a^t \equiv 1 \pmod{n}.$$

It will probably not come as a surprise that there is indeed *Euler's theorem*, which states that, for a and n coprime to each other:

$$a^{\varphi(n)} \equiv 1 \pmod{n}, \tag{1.90}$$

which, as you will at once recognise, is a generalisation of Fermat's little theorem. Fermat's theorem expresses the same congruence for a prime number. Since, as we know, the totient number of a prime number p is $p - 1$, Fermat's theorem can be reduced to Euler's theorem. In fact, Euler's theorem is nothing new at all. We already know from the previous chapter that the powers of a up to a^k, such that $a^k \equiv 1 \pmod{n}$, establish a group or subgroup of the numbers coprime to n. Since $\varphi(n)$ is the size of the group of coprimes of n, any subgroup of n has either size $\varphi(n)$ or a size that divides $\varphi(n)$. But for sure, for any coprime a the group is at most $\varphi(n)$, since there are only $\varphi(n)$ elements in the group.

For the example 12, the groups are all trivial. The coprimes of 12 are 1, 5, 7 and 11. 1 establishes the trivial group $\{1\}$. 5 is in the likewise trivial group $\{1, 5\}$, since $5^2 = 25$ and, hence, $5^2 \equiv 1 \pmod{12}$. Since $7^2 = 49 \equiv 1 \pmod{12}$, the group of 7 is also trivial. Finally, since $11^2 = 121 \equiv 1 \pmod{12}$, 11 is in a trivial group as well.

A more interesting case is 14. Let us look at the groups of 14 using *generate*, which we defined in the previous chapter, like *map* (*generate* 14) (*coprimes* 14):

$[1]$
$[3, 9, 13, 11, 5, 1]$
$[5, 11, 13, 9, 3, 1]$
$[9, 11, 1]$
$[11, 9, 1]$
$[13, 1]$

We see four different groups. The trivial groups 1 and $n - 1$ and the non-trivial groups $\{1, 3, 5, 9, 11, 13\}$, identical to the coprimes of 14, and $\{1, 9, 11\}$, a subgroup with three elements.

From these examples it should be clear that not for all coprimes a $\varphi(n)$ is the first number for which the congruence in Euler's theorem is established. In fact, in many

cases, there are smaller numbers k that make $a^k \equiv 1 \pmod{n}$. For $\varphi(n)$, however, it is guarenteed for any a coprime to n that the congruence holds.

But, still, $\varphi(n)$ is not necessarily the smallest number that guarantees the congruence. In some cases, there is a smaller number that does the job and this number can be calculated by the *Carmichael function*, of which we have already used a part, when we discussed RSA.

The Carmichael function is usually denoted $\lambda(n)$ (but has nothing to do with the λ-calculus!). It is a bit difficult to give its definition in words. It is much easier, actually, to define it in Haskell:

```
lambda :: Natural → Natural
lambda 2 = tot 2
lambda 4 = tot 4
lambda n | twopower n     = (tot n) `div` 2
         | primepower n   = tot n
         | even n ∧ primepower (n `div` 2) = tot n
         | otherwise      = let ps = map lambda (simplify $ trialfact n)
                            in foldl' lcm 1 ps
  where simplify = map product ∘ group ∘ sort
```

There are two base cases stating that $\lambda(2)$ and $\lambda(4)$ equal $\varphi(2)$ and $\varphi(4)$ respectively. Otherwise, if n is a power of 2, then $\lambda(n)$ equals the half of $\varphi(n)$. An example is 8. We would expect that any group generated with coprimes of 8 has at most two members, since $\varphi(8) = 4$, and $\lambda(8) = 2$. We generate the groups with *map* (*generate* 8) (*coprimes* 8) and see:

[1]
[3, 1]
[5, 1]
[7, 1].

The prediction, hence, is correct. We saw a similar result for 12, but that has other reasons as we will see below.

The function *twopower*, by the way, is defined as

```
twopower :: Natural → Bool
twopower 1 = True
twopower 2 = True
twopower n | even n     = twopower (n `div` 2)
           | otherwise = False
```

The next line states that for any primepower n, $\lambda(n) = \varphi(n)$.

The function *primepower* is defined as

$primepower :: Natural \to Bool$
$primepower\ n = length\ (nub\ \$\ trialfact\ n) \equiv 1$

In other words, a primepower is a number whose prime factorisation consists of only one prime (which itself, however, may appear more than once). Since powers of 2 are handled in the previous guard, we are dealing here only with powers of odd primes. An example is 9, which is 3^2. The coprimes of 9 are 1, 2, 4, 5, 7 and 8. The totient number, hence, is 6. The groups are *map* (*generate* 9) (*coprimes* 9):

$[1]$
$[2, 4, 8, 7\ 5, 1]$
$[4, 7, 1]$
$[5, 7, 8, 4, 2, 1]$
$[7, 4, 1]$
$[8, 1]$.

We see four different groups. The two trivial groups and two groups with 3 and 6 members respectively. Again, the prediction is correct.

The next line states that, if n is even and half of n is a primepower, then again $\lambda(n) = \varphi(n)$. An example is 18, since 18 is even and the half of 18 is 9, which is a power of 3. $\varphi(18)$ is 6, so we would expect to see groups with at most 6 elements. Here is the result for *map* (*generate* 18) (*coprimes* 18):

$[1]$
$[5, 7, 17, 13, 11, 1]$
$[7, 13, 1]$
$[11, 13, 17, 7, 5, 1]$
$[13, 7, 1]$
$[17, 1]$.

We see, again, four groups, the two trivial groups 1 and $n-1$ and two non-trivial groups with 3 and 6 members respectively.

We come to the *otherwise* guard. If n is not 2 or 4, not a power of 2 nor a power of another prime and not twice a power of a prime, then we do the following: we compute the factorisation, order the factors, group by equal factors, compute the primepower that corresponds to each group of factors and map λ on the resulting numbers. Then we compute the *lcm* of the results. In short: $\lambda(n)$ is the *lcm* of the λ mappend to the prime powers in the prime factorisation of n.

An example for a number that is not a primepower nor twice a primepower is 20. The factorisation of 20 is $\{2, 2, 5\}$. We compute the primepowers resulting in $\{4, 5\}$. When we map λ on them, we should get $\lambda(4) = \varphi(4) = 2$ and $\lambda(5) = \varphi(5) = 4$. The *lcm* of 2 and 4 is 4 and, hence, $\lambda(20) = 4$. We, thus, should not see a group with more than 4 elements. We call *map* (*generate* 20) (*coprimes* 20) and see:

$[1]$
$[3, 9, 7, 1]$
$[7, 9, 3, 1]$
$[9, 1]$
$[11, 1]$
$[13, 9, 17, 1]$
$[17, 9, 13, 1]$
$[19, 1]$.

We see 6 different groups, the two trivial groups 1 and $n-1$ and four non-trivial groups with 2 and, respectively, 4 members.

The factorisation of 12 is $\{2, 2, 3\}$, so we apply λ on the numbers 4 and 3, which for both cases is 2. The *lcm* of 2 is just 2 and, therefore, we do not see groups with more than 2 members with the coprimes of 12.

Now, as you may have guessed, *Carmichael's theorem* states that, if a and n are coprime to each other, then

$$a^{\lambda(n)} \equiv 1 \pmod{n}. \tag{1.91}$$

For primes, the theorem is identical to Fermat's little theorem. For powers of odd primes, it reduces to Euler's theorem. The *lcm* of primepowers under the *otherwise*-guard is a consequence of the Chinese Remainder theorem and the very notion of the *lcm*. We know that, if $x \equiv 1 \pmod{n}$, then also $x \equiv 1 \pmod{mn}$. However, mn is not necessarily the first multiple of n and m that establishes the congruence. Any number that is a multiple of both, n and m, would have the same effect and the first number that is a multiple of both is $lcm(m, n)$.

The totient number of twice the power of an odd prime, $2p^k$, is the same as the totient number of that odd prime power, p^k: $\varphi(p^k) = \varphi(2p^k)$. The coprimes of p^k are all numbers from 1 to p^k that are not multiples of p, including all even numbers. Since twice that primepower is an even number, the even numbers are not part of the coprimes of that number. So, the coprimes of $2p^k$ in the range $1 \ldots p^k$, are exactly half of the coprimes of p^k. But now, there are the coprimes in the second half $p^k \ldots 2p^k$. Since the interval is the same in size and we eliminate the same number of numbers in that range as in the first half, namely the even numbers and the multiples of p, we end up with two sequences, each containing half as many numbers as the original sequence of coprimes of p^k. The two halfs together, therefore, make for the same amount of coprimes of p^k and $2p^k$. So, we can handle these cases in the same way.

The general rule, however, would produce the same result. According to the general rule, we would first compute λ for the individual primepowers and then the *lcm* of these values. The factorisation of a number that is twice a primepower contains the factor 2 and the primepower. The value for $\lambda(2)$ is $\varphi(2)$, which is 1. The *lcm* of 1 and another

number is that other number. There, hence, is no difference between this rule and the general rule.

Now, what about the powers of 2 greater 4? To show that the greatest group of a power of 2 is half the totient of that number is quite an interesting exercise in group theory. The coprimes of a power of 2 have a quite peculiar structure, namely

$$1, \ldots, m_1, m_2, \ldots, n-1.$$

Interesting are the middle numbers m_1 and m_2. They both are their own inverses, such that $m_1 m_1 \equiv 1 \pmod{n}$ and $m_2 m_2 \equiv 1 \pmod{n}$. The set of coprimes, therefore, consists of two symmetric halves, each starting and ending with a number that is its own inverse: $1 \ldots m_1$ is the first half, $m_2 \ldots n-1$ is the second half.

The number of coprimes is of course even since they consist of all odd numbers $1 \ldots 2^k - 1$. Therefore, we do not have one central number, but the two middle numbers m_1 and m_2, which are one off the half of 2^k, that is $m_1 = 2^{k-1} - 1$ and $m_2 = 2^{k-1} + 1$. The following calculation shows that both m_1 and m_2 squared are immediately 1 modulo 2^k, for any 2^k with $k > 2$. For m_1 we have:

$$(2^{k-1} - 1)(2^{k-1} - 1).$$

When we multiply this out we get the terms $2^{k-1+k-1}$, which simplifies to 2^{2k-2}, $-2^{k-1} - 2^{k-1}$, which simplifies to -2^k, and 1:

$$2^{2k-2} - 2^k + 1.$$

We can factor 2^k out of the the first two terms:

$$2^k(2^{k-2} - 1) + 1,$$

and see clearly that the first remaining term is divided by 2^k and, thus, disappears modulo 2^k. We are left with 1 and this shows that m_1 is its own inverse.

For m_2, the proof is very similar, with the difference that we are left over with $2^k(2^{k-2}+1)$ for the first term. However, this term is a multiple of 2^k as well, and we are again left with 1.

Now, we can select a random generator of the group, say, a and look what it generates. For explicitness, we consider the case of $2^4 = 16$, whose coprimes are

$$1, 3, 5, 7, 9, 11, 13, 15$$

This group has the form (with arbitrary placement of the inverses of a and b):

$$1, a, b, m_1, m_2, b', a', n-1.$$

We see at once that no generator will create a sequence with 8 elements. Any sequence generated by exponentiation of a can contain only one of the elements m_1, m_2 and $n-1$. Since, if $a^k = m_2$, $a^{2k} = 1$ and, afterwards, the whole cycle repeats. If some a^l, for $k < l < 2k$, was m_1 or $n-1$, then a^{2l} would be 1 again. But that cannot be, since $2l > 2k$ and, therefore, a^{2l} is part of the second cycle, which has to be exactly the same as the first. But, obviously, there was only one 1 in the first cycle, namely at a^{2k} and there must be only one 1 in the second cycle, namely at a^{4k}. Therefore, there can be only one of the elements m_1, m_2 and $n-1$ in the sequence and this reduces the longest possible sequence for this example to 6, for instance:

1	2	3	4	5	6
a	b	m_2	b'	a'	1

Until here it looks fine. But observe that we now have one set with six numbers, the group generated by a, which we will call G, and its complement relative to the set of coprimes, which contains 2 elements. For the group above that containts m_2, the complement consists of m_1 and $n-1$.

Now, we will construct what is called a *coset*. A coset of G, in our context here, is a set of numbers resulting from one element of the complement of G, multiplied by all numbers of G. Let us say, this element is m_1. Then the coset of G created by m_1 denoted m_1G is

$$m_1G = \{m_1a, m_1b, m_1m_2, m_1b', m_1a', m_1\} \tag{1.92}$$

Note that this set contains six numbers. These numbers are necessarily different from all numbers in G, since the numbers in G form a group, the product of two members of which result in another member of it and, for each pair of members of the group c and d, there is one number x in the group, such that $xc = d$. m_1, however, is not member of the group and, if m_1 multiplied by c resulted in another member d, then we would have the impossible case that d is the result of multiplications of c for two different numbers: m_1 and x. Therefore, no number in m_1G can possibly equal any number in G.

But we do not have six numbers! We only have two numbers, namely m_1 and $n-1$. Therefore, no group that we create on a set of coprimes with such a structure can be greater than half of the number of coprimes. In our example that is four. With four numbers in G and four elements in the complement of G, we would have no problem at all. But, definitely, a group with six members does not work. □

A corollary of this simple but important argument is that the order of any subgroup of a group of coprimes must divide the number of coprimes. This extends the proof

of Lagrange's theorem for prime groups to composite groups. We will extend it even further in the future. For the moment, however, we can be satisfied with the result. We have proven Carmichael's theorem.

1.9 How many Fractions?

How many negative numbers are there? That is a strange question! How do you want me to answer? I can tell you how many numbers of a specific kind are there only in relation to another kind of numbers. The words "how many" clearly indicate that the answer is again a number.

Let us try to state the question more pecisely: are there more or fewer negative numbers than natural numbers or are there exactly as many negative numbers as natural numbers?

To answer the question, I would like to suggest a way to compare two sets. To compare the size of two sets, A and B, we create a third set, which consists of pairs (a, b), such that $a \in A$ and $b \in B$. The sets are equal, if and only if every $a \in A$ appears exactly once in the new set and every $b \in B$ appears exactly once in the new set. If there is an $a \in A$ that does not appear in the new set, but all $b \in B$ appear exactly once, then B is greater than A. If there is a $b \in B$ that does not appear, but all $a \in A$ appear, then A is greater than B.

Furthermore, I suggest a way of counting a set A. We count a set by creating a new set that consists of pairs (a, n), such that $a \in A$ and $n \in \mathbb{N}$. For n, we start with 0 and, for each element in A, we increase n by 1 before we put it into the new set, like this:

$$
\begin{aligned}
&\mathit{count} :: [a] \to [(a, \mathit{Natural})] \\
&\mathit{count} = \mathit{go}\ 0 \\
&\quad \textbf{where}\ \mathit{go}\ \_\ [\,] = [\,] \\
&\qquad\quad \mathit{go}\ n\ (x : xs) = (x, n+1) : \mathit{go}\ (n+1)\ xs
\end{aligned}
$$

The greatest number n, we find in the pairs of this set is the number of elements in A. Let us see if we can count the negative numbers in this manner. We count them by creating the set $\{(-1, 1), (-2, 2), (-3, 3), \dots\}$. Do we ever run out of negative or natural numbers? I don't think so. Should we ever feel that we run out of negative numbers, then we just take the current natural number and put a minus sign before it. Should we ever feel that we run out of natural numbers, then we simply take the current negative number and remove the negative sign. This proves, I guess, that there is a way to assign each negative number to exactly one natural number and vice versa. There are hence as many negative numbers as natural numbers.

Well, how many numbers are that? That are $|\mathbb{N}|$ numbers. If you want a word for that, call it *aleph-zero* and write it like this: $\aleph_0$. A set with this cardinality is infinite, but countable. Calling *count* on it, we will never get a final answer. But we will have a partial result at any given step.

What about all the integers? There should be twice as much as natural numbers, right? Let us see. We first create a set to count the natural numbers:

$$\{(1,1),(2,2),(3,3),\dots\}$$

Then we insert a negative number before or behind each positive number:

$$\{(1,1),(-1,2),(2,3),(-2,4),(3,5),(-3,6),\dots\}$$

Again, it appears that we do not run out of natural numbers to count all the integers. The set of integers, hence, has cardinality $\aleph_0$ too.

What about fractions? On the first sight, fractions look very different. There are infinitely many of them between any two natural numbers as we have seen with Zeno's paradox. But now comes Cantor and his first diagonal argument to show that fractions are countable and, therefore, that the set of fractions has cardinality $\aleph_0$.

Cantor's proof, his first diagonal argument, goes as follows. He arranged the fractions in a table, such that the first column contained the integers starting from 1 in the first row and counting up advancing row by row. The integers correspond to fractions with 1 as denominator. So, we could say, the first column of this table is dedicated to denominator 1. The second column, correspondingly, is dedicated to denominator 2; the third to denominator 3 and so on. Then, the rows are dedicated likewise to numerators. The first row contains numerator 1, the second contains numerator 2, the third numerator 3 and so on. Like this:

$\frac{1}{1}$	$\frac{1}{2}$	$\frac{1}{3}$	$\frac{1}{4}$	$\frac{1}{5}$	$\frac{1}{6}$	$\dots$
$\frac{2}{1}$	$\frac{2}{2}$	$\frac{2}{3}$	$\frac{2}{4}$	$\frac{2}{5}$	$\frac{2}{6}$	$\dots$
$\frac{3}{1}$	$\frac{3}{2}$	$\frac{3}{3}$	$\frac{3}{4}$	$\frac{3}{5}$	$\frac{3}{6}$	$\dots$
$\frac{4}{1}$	$\frac{4}{2}$	$\frac{4}{3}$	$\frac{4}{4}$	$\frac{4}{5}$	$\frac{4}{6}$	$\dots$
$\frac{5}{1}$	$\frac{5}{2}$	$\frac{5}{3}$	$\frac{5}{4}$	$\frac{5}{5}$	$\frac{5}{6}$	$\dots$
$\frac{6}{1}$	$\frac{6}{2}$	$\frac{6}{3}$	$\frac{6}{4}$	$\frac{6}{5}$	$\frac{6}{6}$	$\dots$
$\dots$	$\dots$	$\dots$	$\dots$	$\dots$	$\dots$	$\dots$

This table is a brute-force approach to list all fractions in two dimensions. Obviously, this table contains all possible fractions, since, for any possible pair of numbers, it contains a fraction, which, however, is not necessarily in canonical form, so the table contains duplicates.

Then, using this table, he created a sequence. He started with the first cell containing $\frac{1}{1}$. From there he went to the next row $\frac{2}{1}$. Then he applied the rule *up*, that is he went up following a diagonal line to the first row, so he would eventually reach $\frac{1}{2}$. He went one to the right and then applied rule *down*, that is he went down following a diagonal line to the first column, so he would eventually reach $\frac{3}{1}$. Now he continued to the next row and repeated the process of going *up* and *down* in diagonal lines infitely adding the number of each cell he crossed to the sequence.

The sequence evolves as follows: Cantor starts with $\frac{1}{1}$, adds $\frac{2}{1}$, applies rule *up* and adds $\frac{1}{2}$, goes to the right, adds $\frac{1}{3}$ and applies rule *down* adding $\frac{2}{2}$; then he goes to the next row adding $\frac{4}{1}$ and goes *up* again adding $\frac{3}{2}$, $\frac{2}{3}$ and $\frac{1}{4}$ and so he goes on forever.

We can reformulate this rule in Haskell, which will make the process clearer:

$$
\begin{aligned}
&\mathit{cantor1} :: [\mathit{Ratio}] \\
&\mathit{cantor1} = (1 \,\%\, 1) : \mathit{go}\ 2\ 1 \\
&\quad \textbf{where}\ \mathit{go}\ \ n\ 1 = \mathit{up}\ \ n\ 1 +\!\!+ \mathit{go}\ 1\ (n+1) \\
&\qquad\quad \mathit{go}\ \ 1\ d = \mathit{down}\ 1\ d +\!\!+ \mathit{go}\ (d+1)\ 1 \\
&\qquad\quad \mathit{down}\ n\ 1 = [n \,\%\, 1] \\
&\qquad\quad \mathit{down}\ n\ d = (n \,\%\, d) : \mathit{down}\ (n+1)\ (d-1) \\
&\qquad\quad \mathit{up}\ \ 1\ d = [1 \,\%\, d] \\
&\qquad\quad \mathit{up}\ \ n\ d = (n \,\%\, d) : \mathit{up}\ \ (n-1)\ (d+1)
\end{aligned}
$$

When we look at rule *up*, starting at the bottom of the code, we see the base case where the numerator is 1. In this case, we just yield $[1 \,\%\, d]$. Otherwise, we call *up* again with the numerator n decremented by 1 and the denominator incremented by 1. $\mathit{up}\ 4\ 1$, thus, is processed as follows:

$$
\begin{aligned}
&\mathit{up}\ 4\ 1 = (4 \,\%\, 1) : \mathit{up}\ 3\ 2 \\
&\mathit{up}\ 3\ 2 = (3 \,\%\, 2) : \mathit{up}\ 2\ 3 \\
&\mathit{up}\ 2\ 3 = (2 \,\%\, 3) : \mathit{up}\ 1\ 4 \\
&\mathit{up}\ 1\ 4 = [1 \,\%\, 4]
\end{aligned}
$$

yielding the sequence $\frac{4}{1}, \frac{3}{2}, \frac{2}{3}, \frac{1}{4}$. *go*, after calling *up*, proceeds with $\mathit{go}\ 1\ (n+1)$. We, hence, would continue with $\mathit{go}\ 1\ 5$, which calls *down*. The base case of *down* is the case where the denominator is 1. Otherwise, we increment the numerator by 1 and decrement the denominator by 1. $\mathit{down}\ 1\ 5$, hence, is processed as follows:

$$
\begin{aligned}
&\mathit{down}\ 1\ 5 = (1 \,\%\, 5) : \mathit{down}\ 2\ 4 \\
&\mathit{down}\ 2\ 4 = (2 \,\%\, 4) : \mathit{down}\ 3\ 3 \\
&\mathit{down}\ 3\ 3 = (3 \,\%\, 3) : \mathit{down}\ 4\ 2 \\
&\mathit{down}\ 4\ 2 = (4 \,\%\, 2) : \mathit{down}\ 5\ 1 \\
&\mathit{down}\ 5\ 1 = [5 \,\%\, 1]
\end{aligned}
$$

yielding the sequence $\frac{1}{5}, \frac{2}{4}, \frac{3}{3}, \frac{4}{2}, \frac{5}{1}$. When we put the sequence together, including the first steps, we see

$$\frac{1}{1}, \frac{2}{1}, \frac{1}{2}, \frac{1}{3}, \frac{2}{2}, \frac{3}{1}, \frac{4}{1}, \frac{3}{2}, \frac{2}{3}, \frac{1}{4}, \frac{1}{5}, \frac{2}{4}, \frac{3}{3}, \frac{4}{2}, \frac{5}{1}, \ldots$$

When we reduce all fractions to canonical form, we see a lot of repetitions:

$$\frac{1}{1}, \frac{2}{1}, \frac{1}{2}, \frac{1}{3}, \frac{1}{1}, \frac{3}{1}, \frac{4}{1}, \frac{3}{2}, \frac{2}{3}, \frac{1}{4}, \frac{1}{5}, \frac{1}{2}, \frac{1}{1}, \frac{2}{1}, \frac{5}{1}, \ldots$$

We see the numbers $\frac{1}{1} = 1$, $\frac{2}{1} = 2$ and $\frac{1}{2}$ repeated several times. They will continue to appear over and over again and, even worse, other numbers will start to reappear too. That is, before we can use this sequence to count fractions, we need to filter duplicates out leading to the sequence

$$\frac{1}{1}, \frac{2}{1}, \frac{1}{2}, \frac{1}{3}, \frac{3}{1}, \frac{4}{1}, \frac{3}{2}, \frac{2}{3}, \frac{1}{4}, \frac{1}{5}, \frac{5}{1}, \ldots$$

But now we see that we can enumerate, that is count, the fractions creating the sequence

$$\left(\frac{1}{1}, 1\right), \left(\frac{2}{1}, 2\right), \left(\frac{1}{2}, 3\right), \left(\frac{1}{3}, 4\right), \left(\frac{3}{1}, 5\right), \left(\frac{4}{1}, 6\right), \left(\frac{3}{2}, 7\right), \left(\frac{2}{3}, 8\right), \left(\frac{1}{4}, 9\right), \ldots$$

This clearly shows that the cardinality of the set of fractions is $\aleph_0$. □

This result may feel a bit odd on the first sight. We clearly have the feeling that there must be more fractions than integers, because, between any two integers, there are infinitely many fractions. When we think of a visualisation with a pair of balances, with the fractions being in one balance and the integers in the other, then, what we would see at any given instance, clearly indicates that there must be more fractions than integers. However, our feeling betrays us, when it comes to infinity. Indeed, our feeling was not made for infinity. Therefore, at least if we accept the notions of comparison and counting outlined above, then we have to accept the result of Cantor's argument. Even further, I would say that the fact that this argument shows things in a way that contradicts our spontaneous way to see these things, underlines the extraordinary quality of this argument. Cantor lets us see things that are usually hidden from our perception. This makes Cantor, who was seen by his contemporary opponents as a kind of sorcerer, a true magus.

It is, by the way, quite simple to extend the argument to negative fractions. We just have to insert behind each number its additive inverse, resulting in the sequence:

$$\left(\frac{1}{1}, 1\right), \left(-\frac{1}{1}, 2\right), \left(\frac{2}{1}, 3\right), \left(-\frac{2}{1}, 4\right), \left(\frac{1}{2}, 5\right), \left(-\frac{1}{2}, 6\right), \left(\frac{1}{3}, 7\right), \left(-\frac{1}{3}, 8\right), \ldots$$

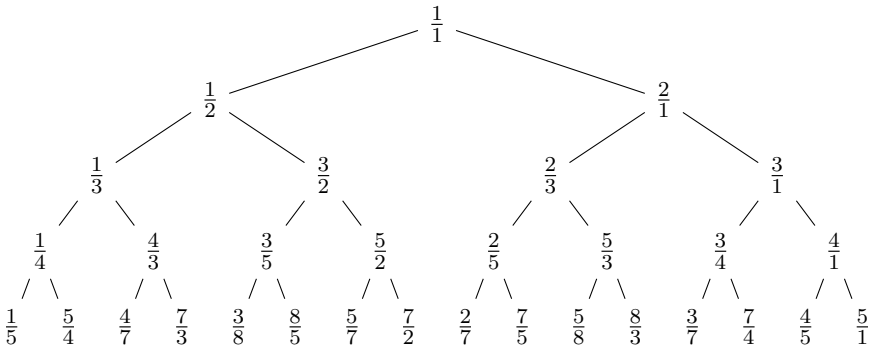
Indeed, there appears to be a lot of room for new numbers, once we are dealing with infinity. This led the great David Hilbert to the analogy of the hotel, which is today named after him *Hilbert's Hotel*. In this analogy, there is a hotel with the uncommon property of having infinitely many rooms. This hotel will never run out of rooms for new guests. If a new guest arrives and there are already infinitely many guests, the manager just asks the guests to move one room up, *i.e.* the guest currently in room 1 moves to room 2, the guest in room 2 moves to room 3 and so on. At the end of the process, room 1 is free for the new arriving guest. Since there are infinitely many rooms, there is no guest, even though there are infinitely many of them already, who would not find a room with a room number one greater than his previous room number.

This approach works for any number of finitely many guests arriving. But even when infinitely many new guests arrive, the manager, still, has resources. In this case, he could ask the guests to move to a room with a number twice the number of his current room, *e.g.* the guest in room 1 would move to room 2, the guest in room 2 would move to room 4, the guest in room 3 would move to room 6 and so on leaving infinitely many rooms with odd room numbers unoccupied and making room for infinitely many new guests.

But let us go back to more technical stuff. In spite of its ingenuity, Cantor's argument is not perfect. In particular, the sequence and how it is created is quite uggly, but, apparently, nobody, for more than hundred years, cared too much about that. Then, in 2000, Neil Calkin and Herbert Wilf published a paper with a new sequence with a bunch of interesting properties that make this sequence for enumerating the fractions much more attractive than Cantor's original sequence. The beginning of the sequence is

$$\frac{1}{1}, \frac{1}{2}, \frac{2}{1}, \frac{1}{3}, \frac{3}{2}, \frac{2}{3}, \frac{3}{1}, \frac{1}{4}, \frac{4}{3}, \frac{3}{5}, \frac{5}{2}, \frac{2}{5}, \frac{5}{3}, \frac{3}{4}, \frac{4}{1}, \ldots$$

The sequence, as we will show in a minute, corresponds to a *binary tree* of the form



When you have a closer look at the tree, you see that the kids of each node are created by a simple formula. If the current node has the form $\frac{n}{d}$, then the left kid corresponds

to $\frac{n}{n+d}$ and the right kid corresponds to $\frac{n+d}{d}$. For instance, the kids of $\frac{1}{1}$ are $\frac{1}{1+1}$ and $\frac{1+1}{1}$. The kids of $\frac{3}{2}$ are $\frac{3}{3+2}$ and $\frac{3+2}{2}$.

We can easily create this tree in Haskell. First, we need a data type to represent the tree:

```
type CalWiTree = Tree Ratio
```

The *Tree* data type is in fact not a binary tree, but a generic tree with an arbitrary number of nodes. But it is simple to implement and serves our purpose. The data type is parametrised, so we define a specialised data type *Tree Ratio* and the type synonym *CalWiTree* referring to this type. Now we create the tree with:

```
calWiTree :: Zahl → Ratio → CalWiTree
calWiTree 1 r             = Node r []
calWiTree i r@(Q n d) = Node r [calWiTree (i − 1) (n % (n + d)),
                                calWiTree (i − 1) ((n + d) % d)]
```

The function takes two arguments, a *Zahl* and a *Ratio*. The *Ratio* is the starting point and the *Zahl* is the number of generations we want to create. Often we do not want the function to create the entire sequence – for that a lot of patience and memory resources would be necessary – but only a tiny part of it. In this case, we set the *Zahl* to $i \geq 1$. If i reaches 1, we create the current node without kids. If we want the function to create the entire infinite tree, we just assign a value $i < 1$. If $i \neq 1$, we create a node with r as data and the kids resulting from calling *calWiTree* on $\frac{n}{n+d}$ and $\frac{n+d}{d}$ with i decremented by 1.

This shows that there is a very simple algorithm to generate the tree. We will now show that Calkin-Wilf tree and Calkin-Wilf sequence are equivalent. We do so by creating an algorithm that converts the tree to the sequence.

We may be tempted to do this with a typical recursive function that does something with the current node and then adds the result of the operation to the partial sequences that result from recursively calling the function on the left and the right kid. This approach, however, is *depth-first*. The resulting sequences would follow the branches of the tree. It would create partial sequences like, for instance, $\frac{1}{1}, \frac{1}{2}, \frac{1}{3}, \frac{1}{4}, \ldots$ But what we need is partial sequences that cover generation by generation, *i.e.* $\frac{1}{1}, \frac{1}{2}, \frac{2}{1}, \frac{1}{3}, \frac{3}{2}, \frac{2}{3}, \frac{3}{1}$ and so on. In other words, we need a *breadth-first* approach.

Since this is a generic problem, we can define a function on the level of the generic *Tree* data type that creates a sequence composed of subsequences corresponding to tree generations:

$$
\begin{aligned}
&\mathit{getKids} :: \mathit{Natural} \to \mathit{Tree}\ a \to [a] \\
&\mathit{getKids}\ 1\ (\mathit{Node}\ r\ \_) \qquad\quad = [r] \\
&\mathit{getKids}\ n\ (\mathit{Node}\ r\ [\,]) \qquad\quad = [\,] \\
&\mathit{getKids}\ n\ (\mathit{Node}\ r\ (x : xs)) = \mathit{getKids}\ (n-1)\ x +\!\!+ \\
&\qquad\qquad\qquad\qquad\qquad\quad \mathit{getKids}\ n\ (\mathit{Node}\ r\ xs)
\end{aligned}
$$

The function receives two arguments, a *Natural*, which determines the generation we want to obtain, and the tree on which we are operating. If the generation is 1, we just give back the data of the current node. Otherwise, we distinguish two cases: If the current node has no kids, then the result is the empty list. This indicates that we have exhausted the current (finite) tree. Otherwise, we advance recursively on the head and tail of the list of kids. Decisive is that we do not add anything to the resulting sequence, before we have reached the intended depth n. This way, the function produces a sequence containing all kids on level n. We now just apply *getKids* to all generations in the tree:

$$
\begin{aligned}
&\mathit{calWiTree2Seq} :: \mathit{CalWiTree} \to [\mathit{Ratio}] \\
&\mathit{calWiTree2Seq}\ t = \mathit{go}\ 1 \\
&\quad \textbf{where}\ \mathit{go}\ n = \textbf{case}\ \mathit{getKids}\ n\ t\ \textbf{of} \\
&\qquad\qquad\qquad\quad [\,] \to [\,] \\
&\qquad\qquad\qquad\quad \mathit{sq} \to \mathit{sq} +\!\!+ \mathit{go}\ (n+1)
\end{aligned}
$$

We have shown that there is a simple algorithm to generate the tree and that there is a simple algorithm to convert the tree into the sequence. The latter is quite useful, since it means that tree and sequence are equivalent. This allows us to prove some crucial properties of the sequence using the tree, which is much simpler than proving them on the sequence directly.

Already a quick glance at the tree reveales some interesting properties, for instance, that, in all cases, the left kid is smaller and the right kid is greater than the parent node. This, of course, is just a consquence of the generating algorithm. We also see that the integers are all in the right-most branch, which equals the first column in Cantor's table. The left-most branch equals the first row in Cantor's table: it contains all fractions with 1 in the numerator. We also see that all fractions are in canonical form, different from Cantor's table. Also, no fraction repeats and, as far as we can tell, the fractions appear to be complete.

The crucial properties, those that we need to show that the Calkin-Wilf sequence contains exactly all fractions and, thus, can be used for Cantor's argument in place of the old sequence, are:

1. All fractions in the tree are in canonical form;
2. Every possible fraction (in canoncial form) appears in the tree;
3. Every fraction appears exactly once.

The first property is simple to prove. We observe that the first fractions appearing in the tree that do not have 1 in numerator or denominator have there either 2 or 3. 2 and 3 a coprime to each other. Summing two coprimes, a and b, will lead to a number that again is coprime to both a and b. Therefore, if we have at a node n a fraction whose numerator and denominator are coprime to each other, the whole subtree below n will contain fractions whose numerator and denominator are coprime to each other. Therefore, the subtrees below $\frac{3}{2}$ and $\frac{2}{3}$ will only contain fractions in canonical form.

The fractions that actually have 1 in numerator or denominator, however, will necessarily lead to a fraction whose numerator and denominator are coprime to each other, since no number n, except $n = 1$, shares any divisor with $n + 1$. Since we start the tree with $\frac{1}{1}$, the whole tree can only contain fractions in canonical form. □

We prove the second property by contradiction. Let us assume that there are fractions that do not appear in the tree. Then, there is a number of fractions that have the smallest denominator and, among those, there is one with the smallest numerator. Let $\frac{n}{d}$ be this fraction. If $n > d$, then $\frac{n-d}{d}$ cannot appear either, since $\frac{n}{d}$ would be its right kid. But that is a contradiction to our assumption that $\frac{n}{d}$ was the nonappearing fraction with the smallest numerator among those fractions with denominator d, but, obviously, $n - d$ is an even smaller numerator. If $n < d$, then $\frac{n}{d-n}$ cannot appear either, since, $\frac{n}{d}$ would be its left kid. But that again is a contradiction, since the denominator is smaller than the denominator of one of the fractions we assumed to be those with the smallest denominator. The only way to solve this dilemma is to assume that n and d are equal. Then, indeed, $\frac{n-d}{d} = 0$ and $\frac{n}{n-d} = \perp$ would not be in the tree. But such a fraction with $n = d$ is irrelevant, since it reduces to $\frac{1}{1}$, which already is in the tree. □

To prove the third property, we first observe that 1 is the root of the tree. Since any fraction below 1 is either $\frac{n}{n+d}$ or $\frac{n+d}{d}$, there cannot be a fraction with $n = d$. With this out of the way, we can argue as we did for the second property: we assume there are fractions that appear more than once. From all these fractions, there is a group that shares the smallest denominator and, among this group, one with the smallest numerator. But this fraction is then either the left kid of two fractions of the form $\frac{n}{d-n}$, making the denominator of these fractions even smaller, or the right kid of two fractions of the form $\frac{n-d}{d}$, making the numerator of these fractions even smaller. In both cases we arrive at a contradiction. □

We can also prove directly – but the argument may be more subtle or, which is the same, almost self-evident. We know that all nodes are of either of the forms $\frac{n}{n+d}$ or $\frac{n+d}{d}$. Since, except for the root node, we never have $n = d$, no node derived from one of those fractions can ever equal one derived from the other. To say that two fractions derived from such nodes are equal, would mean that we could have two numbers n and d, such that $n = n + d$ and $d = n + d$. That would only work if $n = 0$ and $d = 0$. But that case cannot occur. □

That taken all together shows that the Calkin-Wilf sequence contains all fractions exactly

once. Since we can enumerate this sequence, we can enumerate all fractions and, hence, $|\mathbb{Q}| = \aleph_0$. □

There is still another advantage of this sequence over Cantor's original one. There is a simple, yet quite exciting way to compute which is the n^{th} fraction. The key is to realise that we are dealing with a structure, namely a binary tree, that stores sequences of binary decisions. At any given node in the tree, we can go either right or left. We could, therefore, describe the n^{th} position as a trajectory through the tree, where at each node, we take a binary decision: going right or going left. An efficient way to encode a sequence of binary decisions is a binary number, and, indeed, the position in the sequence, represented as a binary number leads to the fraction at that position. Here is a function that, given a natural number n, returns the fraction in the Calkin-Wilf sequence at position n:

```
calwiR :: Natural → Ratio
calwiR = go (0 % 1) ∘ toBinary
  where go r []                 = r
        go r@(Q n d) (0 : xs) = go (n % (n + d)) xs
        go r@(Q n d) (1 : xs) = go ((n + d) % d) xs
```

We start by converting n to binary format (*i.e.* a list of 0s and 1s). Then we call *go* starting with 0, since, for the first number 1, we want position 1. If we have exhausted the binary number, the result is just r, the rational number we pass to *go*. Otherwise, we distinguish two cases: the head of the binary number being 0 or 1. If it is 0, we follow the left branch, which has the fraction $\frac{n}{n+d}$ at its top; if it is 1, we follow the right branch with the fraction $\frac{n+d}{d}$.

Consider $n = 25$ as an example. 25 in binary format is 11001. We go through the steps:

```
go (Q 0 1) [1,1,0,0,1] = go (1 % 1) [1,0,0,1]
go (Q 1 1) [1,0,0,1] = go (2 % 1) [0,0,1]
go (Q 2 1) [0,0,1] = go (2 % 3) [0,1]
go (Q 2 3) [0,1] = go (2 % 5) [1]
go (Q 2 5) [1] = go (7 % 5) []
go (Q 7 5) [] = Q 7 5
```

Let us check if this is true; *take* 1 $ *drop* 24 $ *calWiTree2Seq* (*calWiTree* 5 (1 % 1)) gives

[*Q* 7 5],

which corresponds to the correct result $\frac{7}{5}$. With this we can create the sequence much simpler without deviating through the tree:

```
calwis :: [Ratio]
calwis = map calwiR [1..]
```

We can also do the opposite: compute the position of any given fraction. For this, we just have to turn the logic described above around:

$$
\begin{aligned}
&\mathit{calwiP} :: \mathit{Ratio} \to \mathit{Natural}\\
&\mathit{calwiP} = \mathit{fromBinary} \circ \mathit{reverse} \circ \mathit{go}\\
&\quad \textbf{where } \mathit{go}\ (Q\ 0\ \_) = [\,]\\
&\qquad\quad \mathit{go}\ (Q\ n\ d) \mid n \geqslant d \quad\ = 1 : \mathit{go}\ ((n-d)\ \%\ d)\\
&\qquad\qquad\qquad\quad \mid \mathit{otherwise} = 0 : \mathit{go}\ (n\ \%\ (d-n))
\end{aligned}
$$

We look at the fraction and, if $n \geq d$, then we add 1 to the digits of the resulting binary number, otherwise, we add 0. Note that there is only one case where $n = d$ (as we have proven above), namely the root node of the tree. In all other cases, we either have $n > d$ or $n < d$. In the first case, we know that the fraction is a right kid and in the second case, we know it is a left kid. When we call this function on $\frac{7}{5}$, we see the steps:

$$
\begin{aligned}
&\mathit{go}\ (Q\ 7\ 5) = 1 : \mathit{go}\ (2\ \%\ 5)\\
&\mathit{go}\ (Q\ 2\ 5) = 0 : \mathit{go}\ (2\ \%\ 3)\\
&\mathit{go}\ (Q\ 2\ 3) = 0 : \mathit{go}\ (2\ \%\ 1)\\
&\mathit{go}\ (Q\ 2\ 1) = 1 : \mathit{go}\ (1\ \%\ 1)\\
&\mathit{go}\ (Q\ 1\ 1) = 1 : \mathit{go}\ (0\ \%\ 1)\\
&\mathit{go}\ (Q\ 0\ 1) = [\,],
\end{aligned}
$$

which leads to the list $1 : 0 : 0 : 1 : 1 : [\,]$. This evaluated and reversed is $1, 1, 0, 0, 1$, which, converted to decimal representation, is 25.

Surprisingly or not, the Calkin-Wilf sequence is not completely new. A part of it was already studied in the 19^{th} century by German mathematician Moritz Stern (1807 – 1894), successor of the early deceased sucessor of Gauss at the University of Göttingen, Lejeune-Dirichlet, and professor of Bernhard Riemann. The numerators of the Calkin-Wilf sequence correspond to *Stern's diatomic sequence*. Using the Calkin-Wilf sequence, we can produce Stern's sequence with the function

$$
\begin{aligned}
&\mathit{stern} :: [\mathit{Natural}]\\
&\mathit{stern} = \mathit{map}\ \mathit{numerator}\ \mathit{calwis}\\
&\quad \textbf{where } \mathit{numerator}\ (Q\ n\ \_) = n
\end{aligned}
$$

take 32 *stern* shows:

$$1, 1, 2, 1, 3, 2, 3, 1, 4, 3, 5, 2, 5, 3, 4, 1, 5, 4, 7, 3, 8, 5, 7, 2, 7, 5, 8, 3, 7, 4, 5, 1.$$

Mapping *denominator* defined as $\mathit{denominator}\ (Q\ \_\ d) = d$ on the Calkin-Wilf sequence would give a very similar result: the Stern sequence one ahead, *i.e.*:

$$1, 2, 1, 3, 2, 3, 1, 4, 3, 5, 2, 5, 3, 4, 1, 5, 4, 7, 3, 8, 5, 7, 2, 7, 5, 8, 3, 7, 4, 5, 1, 6.$$

Edsgar Dijkstra, the great pioneer of the art of computer programming, studied this sequence not knowing that it had already been studied before. He called it the *fusc* sequence and generated it with the function

$$
\begin{aligned}
&\mathit{fusc} :: \mathit{Natural} \to \mathit{Natural}\\
&\mathit{fusc}\ 0 = 0\\
&\mathit{fusc}\ 1 = 1\\
&\mathit{fusc}\ n \mid \mathit{even}\ n \quad\ = \mathit{fusc}\ (n\ `\mathit{div}`\ 2)\\
&\qquad\quad \mid \mathit{otherwise} = \mathbf{let}\ k = (n-1)\ `\mathit{div}`\ 2\\
&\qquad\qquad\qquad\quad\ \mathbf{in}\ \mathit{fusc}\ k + \mathit{fusc}\ (k+1)
\end{aligned}
$$

From the definition of the *fusc* function, we can read some of the many properties of Stern's sequence (and the Calkin-Wilf tree). First, an even number has the same value as half of that number, for instance *fusc* 3 is 2 and so is *fusc* 6, *fusc* 12, *fusc* 24 and so on. Even numbers in binary format end on zeros. For instance, 3 in binary notation is 11. 2×3 is 110, 2×6 is 1100, 2×12 is 11000 and so on. The binary format clearly indicates that, after having reached the number before the trail of zeros at the end, we go down in a straight line following the left branch of that node in the Calkin-Wilf tree. Since, in the left path, the numerator never changes, the result of $fusc(n)$ equals the result of $fusc(2n)$.

We also see that for any power of 2, *fusc* equals *fusc* 1, which is 1; *map fusc* $[2, 4, 8, 16, 32, 64, 128, 256]$, hence, gives $[1, 1, 1, 1, 1, 1, 1, 1]$. Note that, looking at the Calkin-Wilf tree, this is immediate obvious, since powers of 2 in binary representation are numbers of the form $1, 10, 100, 1000, \ldots$ Those numbers indicate that we navigate through the tree in a straight line following the left branch of the root node $\frac{1}{1}$.

The *fusc* results of powers of two minus one $(1, 11, 111, 1111, \ldots)$ equal the number of digits of this number in binary form. This is the right outer branch of the tree with the integers.

The *fusc* results of powers of two plus one $(1, 11, 101, 1001, \ldots)$ also equal the number of digits in the binary representation of that number. These numerators appear in the immediate neighbours of the powers of two in the left outer branch of the tree, for instance $\frac{3}{2}, \frac{4}{3}, \frac{5}{4}, \frac{6}{5}, \ldots$

What about numbers with an alternating sequence of 1s and 0s, like 101010101? Those numbers are not in the outer branches and not even close to them. Indeed, they tend to the horizontal centre of the tree. The first 1 leads to node $\frac{1}{1}$. We now go left, that is, we add the numerator to the denominator leading to $\frac{1}{2}$; we then add the denominator to the numerator leading to $\frac{3}{2}$; then we add the numerator to the denominator again leading to $\frac{3}{5}$ and so we go on and obtain the fractions $\frac{8}{5}, \frac{8}{13}, \frac{21}{13}, \frac{21}{34}, \frac{55}{34}, \ldots$ Do you see the point? All the numerators and denominators are Fibonacci numbers! Well, what we did above, adding the two numbers we obtained before starting with the pair $(1, 1)$, is just the recipe to create the Fibonacci numbers.

An amazing property of *fusc*, found by Dijkstra, is the fact that two numbers whose binary representations are the reverse of each other have the same *fusc* result. 25, for instance, is 11001. The reverse, 10011, is 19, and $\mathit{fusc}\ 19 = \mathit{fusc}\ 25 = 7$.

For the Calwin-Wilf tree this means that, when we have two trajectories through the

tree, where each step after the root node is the opposite of the simultaneous step of the other one, we arrive at two fractions with the same numerator. The trajectory defined by the binary sequence $1, 1, 0, 0, 1$ leads, first, to the root node $\frac{1}{1}$, then through $\frac{2}{1}$, $\frac{2}{3}$ and $\frac{2}{5}$ to $\frac{7}{5}$. The reverse of this sequence, $1, 0, 0, 1, 1$ leads, first, to the root node $\frac{1}{1}$ and then through $\frac{1}{2}$, $\frac{1}{3}$ and $\frac{4}{3}$ to $\frac{7}{3}$.

A similar is true for two numbers, whose binary sequences can be transformed into one another by inverting the inner bits. For instance, 11001, 25, can be transformed into 10111, inverting all bits, but the first and the last one. 10111 is 23 and *fusc* 19 = *fusc* 23 = *fusc* 25. What about the bit inverse of 19? That is 11101, the reverse of 10111 and 29 in decimal notation. Therefore *fusc* 19 = *fusc* 23 = *fusc* 25 = *fusc* 29.

We can reformulate this result in terms of group theory. We have three basic transformations i, the identity, ρ, the reverse, and β, the bit inverse. We add one more transformation, the composition of ρ and β and call it $\sigma = \rho \cdot \beta$. The operation defined over this set is composition. We see that the identity is part of the set; for each transformation, its inverse is in the set, too, because $\rho \cdot \rho = i$, $\beta \cdot \beta = i$ and $\sigma \cdot \sigma = i$. To illustrate the logic of this group with the numbers above, we define it on the base string 19, which is 10011:

$i = 10011$
$\rho = 11001$
$\beta = 11101$
$\sigma = 10111$.

Now we can play around and see that we will never generate a string that is not already in the group:

$\rho \cdot i = 11001$
$\rho \cdot \rho = i = 10011$
$\beta \cdot \beta = i = 10011$
$\sigma \cdot \sigma = i = 10011$
$\rho \cdot \beta = \sigma = 10111$
$\beta \cdot \rho = \sigma = 10111$
$\sigma \cdot \rho = \beta = 11101$
$\sigma \cdot \beta = \rho = 11001$
...

All elements of one group are in the same generation of the Calkin-Wilf tree, since they all have the same number of digits. Numbers with a symmetric binary representation, such that $\rho = i$, lead to groups with only two distinguishable members, for instance $fusc(2^n{+}1) = fusc(2^{n+1}{-}1)$. The same is true for numbers with a binary representation such that $\rho = \beta$, for instance, 101011 (43) = 110101 (53) and $fusc(43) = fusc(53) = 13$.

There are infinitely many numbers with the same *fusc* result. Most of these numbers have trailing zeros and, as such, are in the long shadow thrown by one of the original odd

numbers with the same result. One example of such a shadow is the outer left branch, which maintains the numerator of the root node $\frac{1}{1}$ and also maintains the leading 1 in the binary representation of its positions, merely adding more and more 0s to it.

How many "original" numbers in this sense are there for a given *fusc* result n? The answer is simple if we consider two facts: 1. The fractions in the Calkin-Wilf tree are in canonical form, *i.e.* numerator and denominator do not share divisors, and 2. Any position number whose binary representation ends with 1, points to a right kid and, for all fractions $\frac{n}{d}$ that are right kids: $n > d$. Binary numbers, however, that end with 0, point to a left kid and, therefore, $n < d$. In other words, the number of original numbers for a given numerator n is $\varphi(n)$, the totient number of n. The denominators of the original fractions are the coprimes of n.

The numerator 7, for instance, appears in six positions: 19, 23, 25, 29, 65 and 127. The denominators of the fractions at those positions are 3, 2, 5, 4, 6 and 1. For numerator 8, there are only four such numbers: 21, 27, 129 and 255. The denominators at those positions are 5, 3, 7 and 1. Note that 21 in binary format is 10101, which is its own reverse, and 27 is the bit inverse of 21, namely 11011, which also is its own reverse.

Furthermore, those numbers appear in groups with 2 or 4 members, depending on the properties of the binary representation. The number of such groups, hence, is $\frac{\varphi(n)}{k}$, where k is some integer that divides $\varphi(n)$. For 7: $k = 3$, since there are two groups, one containing 4 elements, the other containing 2. For 8: $k = 2$, since there are two groups, both containing 2 elements.

The last group is the one consisting of binary numbers with n bits, *i.e.* $2^{n-1}+1$ and 2^n-1. The other groups appear in generations of the Calkin-Wilf tree before the generation with that final group. For 7, the generation of the group with four members is the fifth generation and the generation with the final group is of course the seventh generation. In other cases, the groups can be many generations apart. The numerator 55, for instance, appears for the first time in generation 10, namely in the fraction $\frac{55}{34}$ (both Fibonacci numbers). This is far off from generation 55 with the group consisting of $\frac{55}{1}$ and $\frac{55}{54}$.

Interestingly, Dijkstra was not aware of the relation of the *fusc* algorithm to the Stern sequence, and the Calkin-Wilf tree was not even around at that time. Dijkstra describes *fusc* as a state automaton that parses strings consisting of 1s and 0s. The parsing result would be a number, namely the result of *fusc*. We could now say that the Calkin-Wilf tree is a model that gives meaning to the strings in terms of trajectories through the tree.

A final remark relates to the product of one generation in the tree. Each generation consists of fractions whose numerators and denominators were created by adding the numerators and denominators of the fractions of the previous generation. We start with the fraction $\frac{1}{1}$. In consequence, in any generation, there is for any fraction $\frac{n}{d}$ a fraction $\frac{d}{n}$. The fractions in the fifth generation for example, the one containing the fractions at positions 19, 23, 25 and 29 in the Calkin-Wilf sequence, can be paired up in the following

way:

$$\left(\frac{1}{5},\frac{5}{1}\right),\left(\frac{5}{4},\frac{4}{5}\right),\left(\frac{4}{7},\frac{7}{4}\right),\left(\frac{7}{3},\frac{3}{7}\right),\left(\frac{3}{8},\frac{8}{3}\right),\left(\frac{8}{5},\frac{5}{8}\right),\left(\frac{5}{7},\frac{7}{5}\right),\left(\frac{7}{2},\frac{2}{7}\right).$$

The product of each pair is 1. The product of all fractions in one generation is therefore 1 as well. You can try this out with the simple function

$$\begin{aligned}&\mathit{genprod} :: \mathit{Natural} \to \mathit{CalWiTree} \to \mathit{Ratio}\\&\mathit{genprod}\ n\ t = \mathit{product}\ (\mathit{getKids}\ n\ t)\end{aligned}$$

A sequence with so many nice properties, one might feel to say with some poetic fervour, cannot be meaningless. Isn't there anything in the (more or less) real world that these numbers would actually count? It turns out there is. There are in fact many things the Stern sequence actually counts. Just to mention two things: It counts odd binomial coefficients of the form $\binom{n-r}{r}, 0 \leq 2r \leq n$. That is the odd numbers in the first half of the lines $n - r$ in Pascal's triangle. This would translate to a function creating a sequence of numbers of the form

$$\begin{aligned}&\mathit{oddCos} :: \mathit{Natural} \to [\mathit{Natural}]\\&\mathit{oddCos}\ n = \mathit{filter}\ \mathit{odd}\ [\mathit{choose}\ (n-r)\ r \mid r \leftarrow [0\,..\,(n\ `\mathit{div}`\ 2)]]\end{aligned}$$

fusc $(n + 1)$ is exactly the size of that sequence. For instance: *oddCos* 5 is $[1, 3]$ and *fusc* 6 is 2; *oddCos* 6 is $[1, 5, 1]$ and *fusc* 7 is 3; *oddCos* 7 is $[1]$ and *fusc* 8, which is a power of 2, is 1; *oddCos* 8 is $[1, 7, 15, 1]$ and *fusc* 9 is 4; *oddCos* 9 is $[1, 21, 5]$ and *fusc* 10 is 3.

Moritz Stern arrived at his sequence, when studying ways to represent numbers as powers of 2. Any number can be written as such a sum and most numbers even in various ways. For instance, $2 = 2^0 + 2^0$, $3 = 2^0 + 2^1 = 2^0 + 2^0 + 2^0$, $4 = 2^1 + 2^1 = 2^0 + 2^0 + 2^1$, $5 = 2^0 + 2^2 = 2^0 + 2^1 + 2^1$ and so on. Stern focussed on so called *hyperbinary* systems, that is sums of powers of 2, where any power appears at most twice. For instance, $3 = 2^0 + 2^1$ is such a system, but $3 = 2^0 + 2^0 + 2^0$ is not. Stern's sequence counts the number of ways this is possible for any number $n - 1$. In other words, *fusc* $(n + 1)$ is exactly the number of hyperbinary systems for n. For 3, as an example, there is only one way and *fusc* 4 is 1; for 4, there are 3 such systems: $2^0 + 2^0 + 2^1$, $2^1 + 2^1$ and 2^2 and *fusc* 5 is 3; for 5, there are only 2 such systems: $2^0 + 2^1 + 2^1$ and $2^0 + 2^2$ and *fusc* 6 is 2.

Finding all hyperbinary systems for a number n is quite an interesting problem in its own right. A brute-force and, hence, inefficient algorithm could apply the following logic. We first find all powers of 2 less than or equal to n:

$$\begin{aligned}&\mathit{pows2} :: \mathit{Natural} \to [\mathit{Natural}]\\&\mathit{pows2}\ n = \mathit{takeWhile}\ (\leqslant n)\ [2 \uparrow x \mid x \leftarrow [0\,..\,]]\end{aligned}$$

We then create all permutations of this set and try to build sums that equal n:

$$
\begin{aligned}
&\mathit{hyperbin} :: \mathit{Natural} \to [[\mathit{Natural}]] \\
&\mathit{hyperbin}\ n = \mathit{nub}\ (\mathit{go}\ \$\ \mathit{perms}\ \$\ \mathit{pows2}\ n) \\
&\quad \textbf{where}\ \mathit{go}\ \mathit{pss} = \mathit{filter}\ (\lambda k \to \mathit{sum}\ k \equiv n) \\
&\qquad\qquad\qquad\qquad [\mathit{sort}\ (\mathit{sums}\ 0\ \mathit{ps}\ \mathit{ps}) \mid \mathit{ps} \leftarrow \mathit{pss}] \\
&\qquad\quad \mathit{sums}\ \_\ [\,]\ [\,] \qquad\quad = [\,] \\
&\qquad\quad \mathit{sums}\ s\ [\,]\ \mathit{ds} \qquad\quad = \mathit{sums}\ s\ \mathit{ds}\ [\,] \\
&\qquad\quad \mathit{sums}\ s\ (p : \mathit{ps})\ \mathit{ds} \mid s + p > n = \mathit{sums}\ s\ \mathit{ds}\ \mathit{ps} \\
&\qquad\qquad\qquad\qquad\qquad\quad \mid s + p \equiv n = [p] \\
&\qquad\qquad\qquad\qquad\qquad\quad \mid \mathit{otherwise} = p : \mathit{sums}\ (s + p)\ \mathit{ps}\ \mathit{ds}
\end{aligned}
$$

Note that we pass the available pool of powers of $2 \leq n$ twice to *sums*. When the first instance is exhausted or $s + p > n$, we start to use the second instance of the pool. This reflects the fact that we are allowed to use every number twice. If the sum $s+p$ equals n, we have found a valid hyperbinary system. Otherwise, if $s + p < n$, we continue adding the current power to the result set. In *go*, we try *sums* on all permutations of the powers filtering the resulting sets for which *sum* equals n. We sort each list to make lists with equal elements equal and to, thus, be able to recognise duplicates and to remove them with *nub*.