1 Real Numbers

1.1 $\sqrt{2}$

Until now, we have looked at *discrete* numbers, that is numbers that are nicely separated from each other so that we can write them down unmistakebly and always know of which number we are currently talking. Now we enter a completely different universe. The universe of continuous numbers that cannot be written down in a finite number of steps. The representation of these numbers consists of infinitely many elements and, therefore, we will never be able to write the number down completely with all its elements. We may give a finite formula that describes how to compute the specific number, but we will never see the whole number written down. Those numbers are known since antiquity and, apparently, their existence came as a great surprise to Greek mathematicians.

The first step of our investigations into this kind of numbers, is to show that they *exist*, *i.e.*, there are contexts where they arise naturally. To start, we will assume that they are not necessary. We assume that all numbers are either natural, integral or fractional. Indeed, any of the fundamental arithmetic operations, $+, -, \times$ and $/$, applied on two rational numbers results always in a rational number, *i.e.* an integer or a fraction. We could therefore suspect that the result of any operation is a rational number, *i.e.* an integer or a fraction.

What about $\sqrt{2}$, the square root of 2? Let us assume that $\sqrt{2}$ is as well a fraction. Then we have two integers n and d, coprime to each other, such that

$$\sqrt{2} = \frac{n}{d} \tag{1.1}$$

and

$$2 = \left(\frac{n}{d}\right)^2 = \frac{n^2}{d^2}. \tag{1.2}$$

Any number can be represented as a product of primes. If $p_1 p_2 \dots p_n$ is the prime factorsiation of n and $q_1 q_2 \dots q_d$ is that of d, we can write:

$$\sqrt{2} = \frac{p_1 p_2 \dots p_n}{q_1 q_2 \dots q_d}. \tag{1.3}$$

It follows that

$$2 = \frac{p_1^2 p_2^2 \dots p_n^2}{q_1^2 q_2^2 \dots q_d^2}. \tag{1.4}$$

As we know from the previous chapter, two different prime numbers p and q squared (or raised to any integer) do not result in two numbers that share factors. The factorisation of p^n is just p^n and that of q^n is just q^n. They are coprime to each other. The fraction in equation 1.4, thus, cannot represent an integer, such as 2. There is only one way for such a fraction to result in an integer, *viz.*, when the numerator is an integer and the denominator is 1, which is obviously not the case for $\sqrt{2}$. It follows that, if the root of an integer is not an integer itself, it is not a rational number either.

But, if $\sqrt{2}$ is not a rational number, a number that can be represented as the fraction of two integers, what the heck is it then?

There are several methods to approximate the number $\sqrt{n}$. The simplest and oldest is the *Babylonian* method, also called *Heron's* method for Heron of Alexandria, a Greek mathematician of the first century who lived in Alexandria.

The idea of Heron's method, basically, is to iteratively approximate the real value starting with a guess. We can start with some arbitrary value. If the first guess, say g, does not equal $\sqrt{n}$, *i.e.* $gg \neq n$, then g is either slightly too big or too small. We either have $gg > n$ or $gg < n$. So, on each step, we improve a bit on the value by taking the average of g and its counterpart n/g. If $gg > n$, then clearly $n/g < g$ and, if $gg < n$, then $n/g > g$. The average of g and a/g is calculated as $(g + a/g)/2$. The result is used as input for the next round. The more iterations of this kind we do, the better is the approximation. In Haskell:

```
heron :: Natural → Natural → Double
heron n s = let a = fromIntegral n in go s a (a / 2)
  where go 0 _ x = x
        go i a x | xx = a     = x
                 | otherwise = go (i − 1) a ((x + a / x) / 2)
```

The function takes two arguments. The first is the number whose square root we want to calculate and the second is the number of iterations we want to do. We then call *go* with s, the number of iterations, a, the *Double* representation of n, and our first guess $a/2$.

In *go*, if we have reached $i = 0$, we yield the result x. Otherwise, we call *go* again with $i - 1$, a and the average of x and a/x.

Let us compute $\sqrt{2}$ following this approach for, say, five iterations. We first have

go 5 2 1 = *go* 4 2 ((1 + 2) / 2)
go 4 2 1.5 = *go* 3 2 ((1.5 + 2 / 1.5) / 2)
go 3 2 1.416666 = *go* 2 2 ((1.416666 + 2 / 1.416666) / 2)
go 2 2 1.414215 = *go* 1 2 ((1.414215 + 2 / 1.414215) / 2)
go 1 2 1.414213 = *go* 0 2 ((1.414213 + 2 / 1.414213) / 2)
go 1 2 1.414213 = 1.414213.

Note that we do not show the complete *Double* value, but only the first six digits. The results of the last two steps, therefore, are identical. They differ, in fact, at the twelfth digit: 1.4142135623746899 (*heron* 2 4) versus 1.414213562373095 (*heron* 2 5). Note that the result of five iterations has one digit less than that of four iterations. This is because that after the last digit, 5, the digit 0 follows and then the *precision* of the Double number type is exhausted. *Irrational* numbers, this is the designation of the type of numbers we are talking about, consist of infinitely many digits. Therefore, the last digit in the number presented above is in fact not the last digit of the number. With slightly higher precision, we would see that the number continues like 03 . . .

The result of (*heron* 2 5) ↑ 2 is fairly close to 2: 1.9999999999999996. It will not get any closer using *Double* representation.

1.2 Φ

Another interesting irrational number is τ or Φ, known as the *divine proportion* or *golden ratio*. The golden ratio is the relation of two quantities, such that the greater relates to the lesser as the sum of both to the greater. This may sound confusing, so here are two symbolic representations:

$$\frac{a}{b} = \frac{a+b}{a}. \tag{1.5}$$

In this equation, a is the greater number and b the lesser. The equation states that the relation of a to b equals the relation of $a + b$ to a. Alternatively we can say

$$\frac{b}{a} = \frac{a}{b-a}. \tag{1.6}$$

Here b is the sum of the two quantities and a is the greater one. The equation states that the relation of b to a is the same as the relation of a to $b - a$, which, of course, is then the smaller quantity.

The golden ratio is known since antiquity. The symbol Φ is an homage to ancient Greek sculptor and painter Phidias (480 – 430 BC). Its first heyday after antiquity was during Renaissance and then it was again extremely popular in the 18^{th} and 19^{th} centuries. Up

to our times, artists, writers and mathematicians have repeatedly called the golden ratio especially pleasing.

From equation 1.6 we can derive the numerical value of Φ in the form $\frac{\Phi}{1}$. We can then calculate b, for any value a, as $a\Phi$. The derivation uses some algebraic methods, which we will study more closely in the next part, especially *completing the square*. As such this section is also an algebra teaser.

We start by setting $\Phi = \frac{b}{a}$ with $a = 1$. We then have on the left side of the equation $\Phi = \frac{b}{1} = b$. On the right-hand side, we have $\frac{a=1}{\Phi-1}$:

$$\Phi = \frac{1}{\Phi - 1}. \tag{1.7}$$

Multiplying both sides by $\Phi - 1$, we get

$$\Phi^2 - \Phi = 1. \tag{1.8}$$

This is a quadratic equation and, with some phantasy, we even recognise the fragment of a binomial formula on the left side. A complete binomial formula would be

$$(a - b)^2 = a^2 - 2ab + b^2. \tag{1.9}$$

We try to pattern match the fragment above such that Φ^2 is a^2 and $-\Phi$ is $-2ab$. That would mean that the last term, b^2 would correspond to the square half of the number that is multiplied by Φ to yield $-\Phi$. $-\Phi$ can be seen as $-1 \times \Phi$. Half of that number is $-\frac{1}{2}$. That squared is $\frac{1}{4}$. So, if we add $\frac{1}{4}$ to both sides of the equation, we would end up with a complete binomial formula on the left side:

$$\Phi^2 - \Phi + \frac{1}{4} = 1 + \frac{1}{4} = \frac{5}{4}. \tag{1.10}$$

We can apply the binomial theorem on the left side and get

$$\left(\Phi - \frac{1}{2}\right)^2 = \frac{5}{4}. \tag{1.11}$$

Now we take the square root:

$$\Phi - \frac{1}{2} = \frac{\pm\sqrt{5}}{2}. \tag{1.12}$$

Note that the $\pm$ in front of $\sqrt{5}$ reflects the fact that the square root of 5 (or any other number) may be positive or negative. Both solutions are possible. However, since we are looking for a positive relation, we only consider the positive solution and ignore the other one. We can therefore simplify to

$$\Phi - \frac{1}{2} = \frac{\sqrt{5}}{2}. \tag{1.13}$$

Finally, we add $\frac{1}{2}$ to both sides:

$$\Phi = \frac{1 + \sqrt{5}}{2} \tag{1.14}$$

and voilá that is Φ. The numerical value is approximately 1.618 033 988 749 895. We can define it as a constant in Haskell as

```
phi :: Double
phi = 0.5 * (1 + sqrt 5)
```

We can now define a function that, for any given a, yields b:

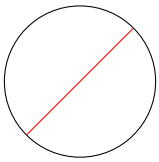
```
golden :: Double → Double
golden a = a * phi
```

golden 1, of course, is just Φ, *i.e.* 1.618 033 988 749 895. *golden* 2 is twice that value, namely 3.236 067 977 499 79. Furthermore, we can state that 2 / (*golden* 2 − 2), which is $\frac{a}{b-a}$, is again an approximation of Φ.

That is very nice. But there is more to come. Φ, in fact, is intimately connected to the Fibonacci sequence, as we will se in the next chapter.

1.3 π

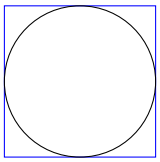
π is probably the most famous irrational number. It emerged in antique mathematics in studying the circle where it expresses the relation between the diameter (depicted in red in the image below) and the circumference (depicted in black):



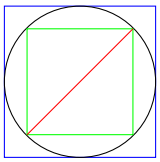
Since, often, the radius, which is half the diameter, is much more important in mathematics, it has been proposed to use $\tau = 2\pi$ where π is used today. But π has survived

through history and, even though slightly suboptimal in some situations, it is still in use today.

The reason why the perimeter instead of the radius was used to define the circle constant is probably because classic approaches to approximate π take the perimeter as basis. They start by drawing a square with side length 1 and inscribe a circle into the square with perimeter 1:



Since the square has side length 1, its perimeter, the sum of all its sides is $1+1+1+1 = 4$ and, as we can see clearly in the picture above, this perimeter is greater than that of the circle. 4, hence, is an upper bound for the circumference of the circle with perimeter 1. A lower bound would then be given by a square inscribed in the circle, such that the distance between its opposing corners (red) is 1, the perimeter of the circle:



We see two right triangles with two green sides on a red basis. The basis is the perimeter of the circle, of which we know that its length is 1. You certainly know the Pythagorean theorem, probably the most famous or notorious theorem of all mathematics, which states that, in a right triangle, one with a right angle, an angle of 90°, the sum of the squares of the sides to the left and right of that angle (the green sides) equals the square of the hypothenuse, the red side, which is opposite to the right angle. This can be stated as:

$$a^2 = b^2 + c^2, \tag{1.15}$$

where a is the red side, whose length we know, namely 1. We further know that the green sides are equal. We hence have:

$$1^2 = 2b^2. \tag{1.16}$$

and further derive

$$1 = \sqrt{2b^2}, \tag{1.17}$$

which is

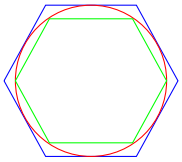
$$1 = \sqrt{2}b. \tag{1.18}$$

Dividing both sides by $\sqrt{2}$, we get

$$b = \frac{1}{\sqrt{2}}, \tag{1.19}$$

the side length of the green square, which is approximately 0.707. The perimeter of the inner square is thus 4×0.707, which is approximately 2.828. Thus π is some value between 2.828 and 4.

That result is not very satifactory, of course. There is room for a lot of numbers between 2.828 and 4. The method was therefore extended by choosing polygons with more than four sides to come closer to the real value of π. The ancient record holder for approximating π is Archimedes who started off with a hexagon, which is easy to construct with compass and ruler:



Then he subsequently doubled the number of sides of the polygon, so that he obtained polygons with 12, 24, 48 and, finally, 96 sides. With this approach he concluded that $\frac{223}{71} < \pi < \frac{22}{7}$, which translates to a number between 3.1408 and 3.1428 and is pretty close to the approximated value 3.14159.

In modern times, mathematicians started to search for approximations by other means than geometry, in particular by infinite series. One of the first series was discovered by Indian mathematician Nilakantha Somayaji (1444 – 1544). It goes like

$$\pi = 3 + \frac{4}{2 \times 3 \times 4} - \frac{4}{4 \times 5 \times 6} + \frac{4}{6 \times 7 \times 8} - \ldots \tag{1.20}$$

We can implement this in Haskell as

```
nilak :: Int → Double
nilak i     | even i      = nilak (i + 1)
            | otherwise = go i 2 3 4
  where go 0 _ _ _ = 3
        go n a b c = let k  | even n = −4
                            | otherwise = 4
                     in (k / (a ∗ b ∗ c)) + go (n − 1) c (c + 1) (c + 2)
```

Here we use a negative term, whenever n, the counter for the step we are performing, is even. Since, with this approach, an even number of steps would produce a bad approximation, we perform, for i even, $i + 1$ and hence an odd number of steps. This way, the series converges to 3.14159 after about 35 steps, *i.e. nilak* 35 is some number that starts with 3.14159.

An even faster convergence is obtained by the beautiful series discovered by French mathematician François Viète (1540 – 1603) in 1593:

$$\frac{2}{\pi} = \frac{\sqrt{2}}{2} \times \frac{\sqrt{2+\sqrt{2}}}{2} \times \frac{\sqrt{2+\sqrt{2+\sqrt{2}}}}{2} \times \ldots \tag{1.21}$$

In Haskell this gives rise to a very nice recursive function:

```
vietep :: Int → Double
vietep i = 2 / (go 0 (sqrt 2))
  where go n t | n ≡ i       = 1
               | otherwise = (t / 2) ∗ go (n + 1) (sqrt (2 + t))
```

The approximation 3.14159 is reached with *vietep* 10.

There are many other series, some focusing on early convergence, others on beauty. An exceptionally beautiful series is that of German polymath Gottfried Wilhelm Leibniz (1646 – 1716), who we will get to know more closely later on:

$$\frac{\pi}{4} = \frac{1}{1} - \frac{1}{3} + \frac{1}{5} - \frac{1}{7} + \frac{1}{9} - \ldots \tag{1.22}$$

In Haskell this is, for instance:

```
leipi :: Int → Double
leipi i = 4 ∗ go 0 1
  where go n d | n ≡ i       = 0
               | otherwise = let x | even n = 1
                                   | otherwise = −1
                             in x / d + go (n + 1) (d + 2)
```

This series converges really slowly. We reach 3.14159 only after about 400 000 steps.

π appears quite often in mathematics, particularly in geometry. But there are also some unexpected entries of this number. The inevitable Leonhard Euler solved a function, which today is called *Riemann zeta function*, for the special case $s = 2$:

$$\zeta(s) = \frac{1}{1^s} + \frac{1}{2^s} + \frac{1}{3^s} + \cdots = \sum_{n=1}^{\infty} \frac{1}{n^s}. \tag{1.23}$$

Euler showed that, for the special case $s = 2$, $\zeta(s)$ converges to $\frac{\pi^2}{6}$; in fact, for any n, n a multiple of 2, $\zeta(n)$ converges to some fraction of a power of π, *e.g.* $\zeta(4)$ approaches $\frac{\pi^4}{90}$, $\zeta(6)$ approaches $\frac{\pi^6}{945}$ and so on.

This is surprising, because the zeta function is not related to circles, but to number theory. It appears for example, when calculating the probability of two numbers being coprime to each other. Two numbers are coprime if they do not share prime factors. The probability of a number being divisible by a given prime p is $\frac{1}{p}$, since every p^{th} number is divisible by p. For two independently chosen numbers, the probability that both are divisible by prime p is therefore $\frac{1}{p} \times \frac{1}{p} = \frac{1}{p^2}$. The reverse probability that both are not divisible by that prime, hence, is $1 - \frac{1}{p^2}$. The probablitiy that there is no prime at all that divides both is then

$$\prod_{p}^{\infty} 1 - \frac{1}{p^2}. \tag{1.24}$$

To cut a long story short, this equation can be transformed into the equation

$$\frac{1}{1 + \frac{1}{2^2} + \frac{1}{3^2} + \ldots} = \frac{1}{\zeta(2)} = \frac{1}{\frac{\pi^2}{6}} = \frac{6}{\pi^2} = 0.607 \approx 61\% \tag{1.25}$$

and with this π appears as a constant in number theory expressing the probability of two randomly chosen numbers being coprime to each other.

1.4 e

The Bernoullis were a family of Huguenots from Antwerp in the Spanish Netherlands from where they fled the repression by the Catholic Spanish authorities, first to Frankfurt am Main, later to Basel in Switzerland. Among the Bernoullis, there is a remarkable number of famous mathematicians who worked in calculus, probability theory, number theory and many areas of applied mathematics. One of the Basel Bernoullis was Johann Bernoulli (1667 – 1748) who worked mainly in calculus and tutored famous mathematicians like Guillaume L'Hôpital, but whose greatest contribution to the history of math

was perhaps to recognise the enormous talent of another of his pupils whose name was Leonhard Euler.

His brother Jacob Bernoulli (1655 – 1705), who worked, as his brother, in calculus, but most prominently in probability theory, is much better known today, partly perhaps because many of Johann's achievements were published under the name of L'Hôpital. Unfortunately, early modern mathematics and science in general was plagued with disputes over priorities in the authorship of contributions, a calamity that authors and authorities later tried to solve by introducing the *droite d'auteur*, better known in the English speaking world as *copyright*.

Among the many problems Jacob studied was the calculation of interests. He started off with a very simple problem. Suppose we have a certain amount of money and a certain interest credited after a given amount of time. To keep it simple, let the amount equal 1 (of any currency of your liking – currencies in Jacob's lifetime were extremely complicated, so we better ignore that detail). After one year 100% interest is paid. After that year, we hence have $1 + \frac{1*100}{100} = 2$ in our account. That is trivial. But what, if the interest is paid in shorter periods during the year? For instance, if the interest is paid twice a year, then the interest for that period would be 50%. After six months we would have $1 + \frac{1*50}{100} = 1.5$ in our account. After one year, the account would then be $1.5 + \frac{1.5*50}{100} = 1.5 + \frac{75}{100} = 1.5 + 0.75 = 2.25$.

Another way to see this is that the initial value is multiplied by 1.5 (the initial value plus the interest) twice: $1 \times 1.5 \times 1.5 = 1 \times 1.5^2 = 2.25$. When we reduce the period even further, say, to three months, then we had $1.25^4 \approx 2.4414$. On a monthly base, we would get $\left(1 + \frac{1}{12}\right)^{12} \approx 2.613$. On a daily basis, we would have $\left(1 + \frac{1}{365}\right)^{365} \approx 2.7145$. With hourly interests and the assumption that one year has $24 \times 365 = 8760$ hours, we would get $\left(1 + \frac{1}{8760}\right)^{8760} \approx 2.71812$. With interest paid per minute we would get $\left(1 + \frac{1}{525600}\right)^{525600} \approx 2.71827$ and on interest paid per second, we would get $\left(1 + \frac{1}{3156000}\right)^{3156000} \approx 2.71828$. In general, for interest on period n, we get:

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

You may have noticed in the examples above that this formula converges with greater and greater ns. For n approaching ∞, it converges to 2.71828, a number that is so beautiful that we should look at more than just the first 5 digits:

$$2.7\ 1828\ 1828\ 4590\ 4523\ldots$$

This is e. It is called Euler's number or, for the first written appearance of concepts related to it in 1618, Napier's number. It is a pity that its first mentioning was not in the year 1828. But who knows – perhaps in some rare Maya calendar the year 1618 actually is the year 1828.

An alternative way to approach e that converges much faster than the closed form above is the following:

$$1 + \frac{1}{2} + \frac{1}{6} + \frac{1}{24} + \frac{1}{120} + \dots$$

or, in other words:

$$e = \sum_{n=1}^{\infty} \frac{1}{n!}. \tag{1.26}$$

We can implement this equation in Haskell as

```
e_ :: Integer → Double
e_ p = 1 + sum [1 / (dfac n) | n ← [1 .. p]]
  where dfac = fromInteger ∘ fac
```

After some experiments with this function, we see that it converges already after 17 recursions to a value that does not change with greater arguments at *Double* precision, such that $e\_\ 17 \equiv e\_\ 18 \equiv e\_\ 19 \equiv \dots$ We could then implement e as

```
e :: Double
e = e_ 17
```

The fact that e is related to the factorial may led to the suspicion that it also appears directly in a formula dealing with factorials. There, indeed, is a formula derived by James Stirling who we already know for the Stirling numbers. This formula approximates the value of $n!$ without the need to go through all the steps of its recursive definition. Stirling's formula is as follows:

$$n! \approx \sqrt{2\pi n} \left(\frac{n}{e}\right)^n. \tag{1.27}$$

This equation is nice already because of the fact that e and π appear together to compute the result of an important function. But how precise is the approximation? To answer this question, we first implement Stirling's formula:

```
stirfac :: Integer → Integer
stirfac i = ceiling $ (sqrt (2 * pi * n)) * (n / e) ↑ i
  where n = fromIntegral i
```

Note that we *ceil* the value, instead of rounding it just to the next integer value.

Then we define a function to compute the difference $difac\ n = fac\ n - stirfac\ n$. The result for the first 15 numbers is

$$0, 0, 0, 0, 1, 9, 59, 417, 3343, 30104, 301174, 3314113, 39781324, 517289459, 7243645800.$$

For the first numbers, the difference is 0. Indeed:

$$\begin{array}{rcrcr} 1! & = & stirfac(1) & = & 1 \\ 2! & = & stirfac(2) & = & 2 \\ 3! & = & stirfac(3) & = & 6 \\ 4! & = & stirfac(4) & = & 24 \end{array}$$

Then, the functions start to disagree, for instance $5! = 120 \neq stirfac(5) = 119$. The difference grows rapidly and reaches more than 3 million with 12!. But what is the deviation in relation to the real value? We define the function $100 * (\mathit{fromIntegral}\ \$\ \mathit{difac}\ n)\ /\ (\mathit{fromIntegral}\ \$\ \mathit{fac}\ n)$ to obtain the difference in terms of a percentage of the real value. We see starting from 5 (where the first difference occurs):

$$0.8333, 1.25, 1.1706, 1.0342, 0.9212, 0.8295, 0.7545, 0.6918, 0.6388, 0.5933, 0.5539, \ldots$$

For 5, the value jumps up from 0 to 0.8333%, climbs even higher to 1.25% and then starts to descrease slowly. At 42 the deviation falls below 0.2%. At 84, it falls below 0.1% and keeps falling. Even though the difference appears big in terms of absolute numbers, the percentage quickly shrinks and, for some problems, may even be neglible.

A completely different way to approximate e is by *continued fractions*. Continued fractions are infinite fractions, where each denominator is again a fraction. For instance:

$$e = 1 + \cfrac{1}{1 + \cfrac{1}{2 + \cfrac{1}{1 + \cfrac{1}{1 + \cfrac{1}{4 + \cfrac{1}{\ddots}}}}}} \tag{1.28}$$

A more readable representation of continued fractions is by sequences of the denominator like:

$$e = [2; 1, 2, 1, 1, 4, 1, 1, 6, 1, 1, 8, 1, 1, \ldots] \tag{1.29}$$

where the first number is separated by a semicolon to highlight the fact that it is not a denominator, but an integral number added to the fraction that follows. We can capture this very nicely in Haskell, using just a list of integers. However, in some cases we might

have a fraction with numerators other than 1. An elegant way to represent this case is by using fractions instead of integers. We would then represent $\frac{2}{a+...}$ as $\frac{1}{\frac{1}{2}a+...}$. Here is an implementation:

```
contfrac :: [Quoz] → Double
contfrac [] = 1
contfrac [i] = fromQuoz i
contfrac (i : is) = n + 1 / (contfrac is)
  where n = fromQuoz i

fromQuoz :: Quoz → Double
fromQuoz i = case i of
  (Pos (Q nu d)) → fromIntegral nu / fromIntegral d
  (Neg (Q nu d)) → negate (fromIntegral nu / fromIntegral d)
```

For *contfrac* $[2, 1, 2, 1, 1, 4, 1, 1, 6]$ we get 2.7183, which is not bad, but not yet too close to e. With $[2, 1, 2, 1, 1, 4, 1, 1, 6, 1, 1, 8, 1, 1, 10]$ we get 2.718 281 828, which is pretty close.

Examining the sequence a bit further, we see that is has a regular structure. We can generate it by means of the *Engel expansion* named for Friedrich Engel (1861 – 1941), a German mathematician who worked close with the great Norwegian algebraist Sophus Lie (1842 – 1899). The Engel expansion can be implemented as follows:

```
engelexp :: [Integer]
engelexp = 2 : 1 : go 1
  where go n = (2 * n) : 1 : 1 : go (n + 1)
```

The following fraction, however, conoverges must faster than the Engel expansion: $[1, \frac{1}{2}, 12, 5, 28, 9, 44, 13]$. Note that we take advantage of the datatype *Quoz* to represent a numerator that is not 1. This sequence can be generated by means of

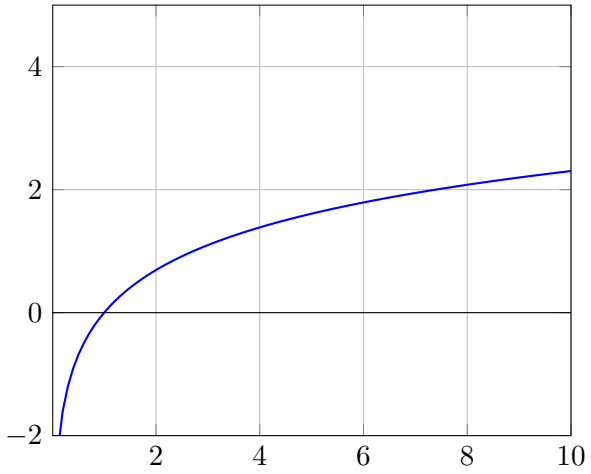
```
fastexp :: [Quoz]
fastexp = 1 : (Pos (1 % 2)) : go 1
  where go n = (16 * n − 4) : (4 * n + 1) : go (n + 1)
```

This fraction converges already after 7 steps to the value 2.718 281 828:

contfrac (*take* 7 *fastexp*).

The area of mathematics where e is really at home is analysis and its vast areas of application, which we will study in the third part of this series. The reason for the prominence of e in analysis stems from the *natural logarithm*, which we already introduced in the first chapter. The natural logarithm of a number n, usually denoted $\ln(n)$, is the exponent x, such that $e^x = n$.

The natural logarithm can be graphed as follows:



The curious fact that earned the natural logarithm its name is that, at $x = 1$, the curve has the slope 1. This might sound strange for the moment. We will investigate that later.

1.5 γ

The *harmonic series* is defined as

$$\sum_{n=1}^{\infty} \frac{1}{n} = 1 + \frac{1}{2} + \frac{1}{3} + \frac{1}{4} + \dots \tag{1.30}$$

A harmonoic series with respect to a given number k, called the *harmonic number* H_k, then is

$$\sum_{n=1}^{k} \frac{1}{n} = 1 + \frac{1}{2} + \dots \frac{1}{k} \tag{1.31}$$

This is easily implemented in Haskell as

```
harmonic :: Natural → Double
harmonic n = sum [1 / d | d ← map fromIntegral [1 .. n]]
```

Some harmonic numbers are (*map harmonic* $[1 \,..\, 10]$):

$1, 1.5, 1.8\overline{3}, 2.08\overline{3}, 2.28\overline{3}, 2.44\overline{9}, 2.5928, 2.7178, 2.8289, 2.9289.$

The harmonic series is an interesting object of study in its own right. Here, however, we are interested in something else. Namely, the difference of the harmonic series and the

natural logarithm:

$$\gamma = \lim_{n\to\infty} H_n - \ln(n). \tag{1.32}$$

We can implement this equation as

```
harmonatural :: Natural → Double
harmonatural n = harmonic n − ln n
  where ln = log ∘ fromIntegral
```

Applied on the first numbers with *map harmonatural* $[1\,..\,10]$, the function does not show interesting results:

$1.0, 0.8068, 0.7347, 0.697, 0.6738, 0.6582, 0.6469, 0.6384, 0.6317, 0.6263, \ldots$

Applied to greater numbers, however, the results approach a constant value:

harmonatural $100 = 0.58220$
harmonatural $1000 = 0.57771$
harmonatural $10000 = 0.57726$
harmonatural $100000 = 0.57722$

With even greater numbers, the difference converges to 0.577 21. This number, γ, was first mentioned by – surprise – Leonhard Euler and some years later by Italian mathematician Lorenzo Mascheroni (1750 – 1800) and is therefore called the Euler-Mascheroni constant.

This mysterious number appears in different contexts and, apparently, quite often as a difference or average. An ingenious investigation was carried out by Belgian mathematician Charles Jean de la Vallée-Poussin (1866 – 1962) who is famous for his proof of the Prime number theorem. Vallée-Poussin studied the quotients of a number n and the primes up to that number. If n is not prime itself, then there are some prime numbers p, namely those of the prime factorisation of n, such that $\frac{n}{p}$ is an integer. For others, this quotient is a rational number, which falls short of the next natural number. For instance, there are four prime numbers less than 10: 2, 3, 5 and 7. The quotients are

$5, 3.\overline{3}, 2, 1.\overline{428571}$.

5 and 2, the quotients of 2 and 5, respectively, are integers and there, hence, is no difference. The quotient $\frac{10}{3} = 3.\overline{3}$, however, falls short of 4 by $0.\overline{6}$ and $\frac{10}{7} = 1.\overline{428571}$ falls short of 2 by $0.\overline{57142828}$.

Vallée-Poussin asked what the average of this difference is. For the example 10, the average is about $0.3\overline{095238}$. One might think that this average, computed for many numbers or for very big numbers, is about 0.5, so that the probability for the quotient of n and a random prime number to fall into the first or the second half of the rational numbers between two integers is equal, *i.e.* 50% for both cases. It turns out it is not. For

huge numbers, de la Vallée-Poussin's average converges to 0.577 21, the Euler-Mascheroni constant.

It converges quite slow, however. If we implement the prime quotient as

$$
\begin{aligned}
&pquoz :: Natural \to [Double] \\
&pquoz\ n = [\, d\ /\ p \mid p \leftarrow ps\,] \\
&\quad \textbf{where}\ ps = map\ fromIntegral\ (takeWhile\ (<n)\ allprimes) \\
&\qquad\quad\ \ d\ \ = fromIntegral\ n
\end{aligned}
$$

and its average as

$$
\begin{aligned}
&pquozavg :: Integer \to Double \\
&pquozavg\ n = (sum\ ds)\ /\ (fromIntegral\ \$\ length\ ds) \\
&\quad \textbf{where}\ qs\ = pquoz\ n \\
&\qquad\quad\ \ ns = map\ (fromIntegral \circ ceiling)\ qs \\
&\qquad\quad\ \ ds = [\, n - q \mid (n, q) \leftarrow zip\ ns\ qs\,]
\end{aligned}
$$

we can experiment with some numbers like

$pquozavg\ 10 = 0.3\overline{095238}$
$pquozavg\ 100 = 0.548731$
$pquozavg\ 1000 = 0.5590468$
$pquozavg\ 10000 = 0.5666399$
$pquozavg\ 100000 = 0.5695143$
...

With greater and greater numbers, this value approaches γ. Restricting n to prime numbers produces good approximations of γ much earlier. From 7 on, *pquozavg* with primes results in numbers of the form $0.5\ldots$ $pquozavg\ 43 = 0.57416$ is already very close to γ. It may be mentioned that 43 is suspiciously close to 42.

With de la Vallée-Pussin's result in mind, it is not too surprising that γ is related to divisors and Euler's totient number. A result of Gauss' immediate successor in Göttingen, Peter Gustav Lejeune-Dirichlet (1805 – 1859), is related to the average number of divisors of the numbers $1\ldots n$. We have already defined a function to generate the divisors of a number n, namely *divs*. Now we map this function on all numbers up to n:

$$
\begin{aligned}
&divsupn :: Natural \to [[Natural]] \\
&divsupn\ n = map\ divs\ [1\,..\,n]
\end{aligned}
$$

Applied to 10, this function yields:

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

For modelling Lejeune-Dirichlet's result, we further need to count the numbers of divisors of each number:

$$
\begin{aligned}
&ndivs :: Integer \to [Int] \\
&ndivs = map\ length \circ divsupn
\end{aligned}
$$

Applied again to 10, *ndivs* produces:

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

Now we compute the average of this list using

```
dirichlet :: Integer → Double
dirichlet n   = s / l
  where ds = ndivs n
        l  = fromIntegral $ length ds
        s  = fromIntegral $ sum ds
```

For *dirichlet* 10 we see 2.7. This does not appear too spectacular. Greater numbers show:

dirichlet $100 = 4.759$
dirichlet $250 = 5.684$
dirichlet $500 = 6.38$
dirichelt $1000 = 7.069$

As we can see, the number is slowly increasing resembling a log function or, more specifically, the natural log. When we compare the natural log, we indeed see that the results are close:

$\ln 100 = 4.605$
$\ln 250 = 5.521$
$\ln 500 = 6.214$
$\ln 1000 = 6.907$

For greater and greater numbers, the difference of the *dirichlet* function and the natural logarithm approaches

$$0.154435 \approx 2\gamma - 1. \tag{1.33}$$

For the five examples above, the difference is still significantly away from that number:

$\Delta 100 = 0.2148$
$\Delta 250 = 0.1625$
$\Delta 500 = 0.1635$
$\Delta 1000 = 0.1612,$

but already $\Delta 2000 = 0.158$ comes close and $\Delta 4000 = 0.1572$ approaches the value even further.

An important constant derived from γ is e^{γ}, which is a limit often seen in number theory. One instance is the lower bound of the totient function. There is a clear upper bound, namely $n - 1$. Indeed, $\varphi(n)$ can never yield a value greater $n - 1$ and this upper bound is reached exclusively by prime numbers. There is no such linear lower bound. That is,

$\varphi(n)$ can assume values that are much smaller than the value seen for $n-1$ or other numbers less than n. But there is a lower bound that slowly grows with n. This lower bound is often given as $\frac{n}{\ln\ln n}$. This lower bound, however, is too big. There are some values that are still below that border. $\varphi(40)$, for instance, is 16. $\frac{40}{\ln\ln 40}$, however, is around 30. A better, even still not perfect approximation, is

$$\frac{n}{e^{\gamma}\ln\ln n}.$$

For the $n = 40$ again, $\frac{40}{e^{\gamma}\ln\ln 40}$ is around 17 and, hence, very close to the real value.

We see that γ is really a quite mysterious number that appears in different contexts, sometimes in quite a subtle manner. The greatest mystery, however, is that it is not so clear that this number belongs here in the first place. Indeed, it has not yet been shown that γ is irrational. In the approximations, we have studied in this section, we actually have not seen the typical techniques to create irrational numbers like roots, continuous fractions and infinite series. If γ is indeed rational, then it must be the fraction of two really large numbers. In 2003, it has been shown that the denominator of such a fraction must be greater than 10^{242080}. A number big enough, for my taste, to speak of *irrational*.

1.6 Representation of Real Numbers

The set of real numbers $\mathbb{R}$ is the union of the rational and the irrational numbers. When we write real numbers on paper, we use the decimal notation. A number in decimal notation corresponds to an ordinary integer terminated by a dot called the decimal point; this integer corresponds to the part of the real number greater 1 or 0 of course. After the dot a stream of digits follows, which is not necessarily a number in the common sense, since it may start with zeros, *e.g.* 0.0001. In fact, one could say that the part after the dot corresponds to a a reversed integer, since the zero following this number have no impact on the value of the whole expression, *i.e.* $0.10 = 0.1$.

Any rational number can be expressed in this system. An integer corresponds just to the part before the dot: $1.0 = 1$. A fraction like $\frac{1}{2}$ is written as 0.5. We will later look at how this is computed concretely. Rationals in decimal notation can be easily identified: all numbers in decimal notation with a finite part after the dot are rational: 0.25 is $\frac{1}{4}$, 0.75 is $\frac{3}{4}$, 0.2 is $\frac{1}{5}$ and so on.

There are some rational numbers that are infinite. For example, $\frac{1}{3}$ is $0.333333\ldots$, which we encode as $0.\overline{3}$. Such periodic decimals are easy to convert to fractions. We just have to multiply them by a power of 10, such that there is a part greater 0 before the decimal point and that the first number of the the repeating period is aligned to it. For $0.\overline{3}$, this is just $10 \times 0.\overline{3} = 3.\overline{3}$. For $0.1\overline{6}$, it would be $10 \times 0.1\overline{6} = 1.\overline{6}$. For $0.\overline{09}$, it would be $10^2 \times 0.\overline{09} = 9.\overline{09}$. We then subtract the original number from the result. If the original

number is x, we now have $10^n x - x = (10^n - 1)x$. For $x = 0.\overline{3}$ this is $9x$; for $x = 0.1\overline{6}$ this, too, is $9x$ and for $x = 0.\overline{09}$ this is $99x$. The results are 3, 1.5 and 9 respectively. We now build a fraction of this result as numerator and the factor (9 or 99) in the denominator. Hence, $0.\overline{3} = \frac{3}{9} = \frac{1}{3}$, $0.1\overline{6} = \frac{1.5}{9} = \frac{3}{18} = \frac{1}{6}$ and $0.\overline{09} = \frac{9}{99} = \frac{1}{11}$.

A curiosity resulting from this calculations is that $9 \times 1/3 = 3$ and $9 \times 0.\overline{3} = 2.\overline{9}$ are actually the same! A correct implementation must take this into account. Try it with Haskell, you will see that $9 * 0.3333333333333333$ is indeed 3. It will not work if you forget a three. The precision of the *Double* number type is 16 digits after the decimal point. With only 15 threes, the result will be 2.9999999999999997.

Irrational numbers in the decimal notation have infinite many digits after the decimal point. With this said, it is obvious that we cannot represent irrational numbers in this system. We can of course represent any number by some kind of formula like $\sqrt{5}$ and do some math in this way such as $\frac{1+\sqrt{5}}{2}$, *etc.* But often, when we are dealing with applied mathematics, such formulas are not very useful. We need an explicit number. But, unfortunately or not, we have only limited resources in paper, brainpower and time. That is, at some point we have to abandon the calculations and work with what can be achieved with the limited resources we have at our disposal.

The point in time at which we take the decision that we now have calculated enough is the measure for the precision of the real number type in question. On paper, we would hence say that we write only a limited number of digits after the decimal point. In most day-to-day situations where real numbers play a role, like in dealing with money, cooking, medication or travelling distances, we calculate up to one or two decimal places. Prices, for instances are often given as 4.99 or something, but hardly 4.998. Recipes would tell that we need 2.5 pounds or whatever of something, using one decimal place. One would say that it is about 1.5km to somewhere, but hardly that it is 1.49km. In other areas, especially in science much more precision is needed. We therefore need a flexible datatype.

A nice and clean format to represent real numbers uses two integers or, as the following definition, two natural numbers:

data *RealN* = *R Natural Natural*

The first number represents the integral part. You will remember that a number is a list of digits where every digit is multiplied by a power of ten according to the place of the digit. The digit, counted from the right, is multiplied by $10^0 = 1$. The second is multiplied by 10^1, the third by 10^2 and so on. The digit multiplied by 10^0, is the last digit before the decimal point. If we wanted to push it to the right of the decimal point, we would need to reduce the exponent. So, we multiplied it not by 10^0, but by 10^{-1} to push it to the first decimal place. This is the function of the second number in the datatype above. It represents the value of the least significant bit in terms of the exponent to which we have to raise 10 to obtain the number represented by this datatype. Since our datatype uses a natural number, we have to negate it to find the

exponent we need.

For instance, the number $R\ 25\ 2$ corresponds to 25×10^{-2}, which we can reduce stepwise to 2.5×10^{-1} and 0.25×10^{0}. A meaningful way to show this datatype would therefore be:

```
instance Show RealN where
  show (R a e) = show a ++ "*10^(-" ++ show e ++ ")"
```

There are obviously many ways to represent the same number with this number type. 1, for instance, can be represented as

```
one = R 1 0
one = R 10 1
one = R 100 2
one = R 1000 3
...
```

To keap numbers as concise as possible, we define a function to simplify numbers with redundant zeros:

```
simplify :: RealN → RealN
simplify (R 0 _) = R 0 0
simplify (R a e) | e > 0 ∧
                   a `rem` 10 ≡ 0 = simplify (R (a `div` 10) (e − 1))
                 | otherwise      = R a e
```

As long as the exponent is greater 0 and the base a is divisible by 10, we reduce the exponent by one and divide a by 10. In other words, we remove unnecessary zeros. The following constructor uses *simplify* to create clean real numbers:

```
real :: Natural → Natural → RealN
real i e = simplify (R i e)
```

1.7 $\mathbb{R}$

We now define how to check two real numbers for equality:

```
instance Eq RealN where
  r1@(R a e1) ≡ r2@(R b e2) | e1 ≡ e2 = a ≡ b
                            | e1 > e2 = r1 ≡ blowup e1 r2
                            | e1 < e2 = blowup e2 r1 ≡ r2
```

If the exponents are equal, then we trivially compare the coefficients. Otherwise, we first expand the number with the smaller exponent using *blowup*:

$$
\begin{aligned}
&\mathit{blowup} :: \mathit{Natural} \to \mathit{RealN} \to \mathit{RealN} \\
&\mathit{blowup}\ i\ (R\ r\ e) \mid i \leqslant e \quad\ = R\ r\ e \\
&\qquad\qquad\qquad\quad \mid \mathit{otherwise} = R\ (r * 10 \uparrow (i - e))\ i
\end{aligned}
$$

That is simple! If the target i is greater than the current exponent of the number, we just multiply the coefficient by 10 raised to the difference of the target exponent and the current exponent and make the target the new exponent. Otherwise, nothing changes.

We continue with comparison, which follows exactly the same logic:

$$
\begin{aligned}
&\textbf{instance}\ \mathit{Ord}\ \mathit{RealN}\ \textbf{where} \\
&\quad \mathit{compare}\ \mathit{r1}@(R\ a\ \mathit{e1})\ \mathit{r2}@(R\ b\ \mathit{e2}) \mid \mathit{e1} \equiv \mathit{e2} = \mathit{compare}\ a\ b \\
&\qquad\qquad\qquad\qquad\qquad\qquad\qquad\quad \mid \mathit{e1} > \mathit{e2}\ = \mathit{compare}\ \mathit{r1}\ (\mathit{blowup}\ \mathit{e1}\ \mathit{r2}) \\
&\qquad\qquad\qquad\qquad\qquad\qquad\qquad\quad \mid \mathit{e1} < \mathit{e2}\ = \mathit{compare}\ (\mathit{blowup}\ \mathit{e2}\ \mathit{r1})\ \mathit{r2}
\end{aligned}
$$

Now we make *RealN* instance of *Num*:

$$
\begin{aligned}
&\textbf{instance}\ \mathit{Num}\ \mathit{RealN}\ \textbf{where} \\
&\quad (R\ a\ \mathit{e1}) + (R\ b\ \mathit{e2}) \mid \mathit{e1} \equiv \mathit{e2}\ \ = \mathit{simplify}\ \$\ R\ (a + b)\ \mathit{e1} \\
&\qquad\qquad\qquad\qquad\qquad\quad \mid \mathit{e1} > \mathit{e2}\ \ = \mathit{simplify}\ \$\ R\ (a + b * 10 \uparrow (\mathit{e1} - \mathit{e2}))\ \mathit{e1} \\
&\qquad\qquad\qquad\qquad\qquad\quad \mid \mathit{otherwise} = \mathit{simplify}\ \$\ R\ (a * 10 \uparrow (\mathit{e2} - \mathit{e1}) + b)\ \mathit{e2} \\
&\quad (R\ a\ \mathit{e1}) - (R\ b\ \mathit{e2}) \mid \mathit{e1} \equiv \mathit{e2} \wedge \\
&\qquad\qquad\qquad\qquad\qquad\qquad\ a \geqslant b \quad\ = \mathit{simplify}\ \$\ R\ (a - b)\ \mathit{e1} \\
&\qquad\qquad\qquad\qquad\qquad\quad \mid \mathit{e1} \equiv \mathit{e2}\ \ = \mathit{error}\ \texttt{"subtraction beyond zero!"} \\
&\qquad\qquad\qquad\qquad\qquad\quad \mid \mathit{e1} > \mathit{e2}\ \ = \mathit{simplify}\ \$\ (R\ a\ \mathit{e1}) - (R\ (b * 10 \uparrow (\mathit{e1} - \mathit{e2}))\ \mathit{e1}) \\
&\qquad\qquad\qquad\qquad\qquad\quad \mid \mathit{otherwise} = \mathit{simplify}\ \$\ (R\ (a * 10 \uparrow (\mathit{e2} - \mathit{e1}))\ \mathit{e2}) - (R\ b\ \mathit{e2}) \\
&\quad (R\ a\ \mathit{e1}) * (R\ b\ \mathit{e2}) = \mathit{real}\ (a * b)\ (\mathit{e1} + \mathit{e2}) \\
&\quad \mathit{negate}\ \ r \quad\ = r \quad \text{-- we cannot negate natural numbers} \\
&\quad \mathit{abs}\ \ \ r \quad\ = r \\
&\quad \mathit{signum}\ r \quad\ = r \\
&\quad \mathit{fromInteger}\ i = R\ (\mathit{fromIntegral}\ i)\ 0
\end{aligned}
$$

Addition is again the same logic. For two numbers with equal exponents, we just add the coefficients. If the exponents differ, we first convert the smaller number to the greater exponent.

For subtraction, note that we define *RealN* like numbers before without negatives. To consider signedness, we still have to use the datatype *Signed RealN*. Consequently, we have to rule out the case where the first number is smaller than the second one.

Multiplication is interesting. We multiply two real numbers by multiplying the coefficients and adding the exponents. We have already seen this logic, when defining the natural number type. Some simple examples may convince you that this is the right way to go. 1×0.1, for instance, is 0.1. In terms of our *RealN* type, this corresponds to $(R\ 1\ 0) * (R\ 1\ 1) \equiv (R\ (1 * 1)\ (0 + 1)$.

The next task is to make *RealN* instance of *Fractional*:

```
instance Fractional RealN where
  (/) = rdiv 17
  fromRational r = (R (fromIntegral $ R.numerator r) 0) /
                   (R (fromIntegral $ R.denominator r) 0)
```

The method *fromRational* is quite simple. We just create two real numbers, the numerator of the original fraction and its denominator, and the we divide them. What we need to do this, of course, is division. Division, as usual, is a bit more complicated than the other arithmetic operations. We define it as follows:

```
rdiv :: Natural → RealN → RealN → RealN
rdiv n r1@(R a e1) r2@(R b e2) | e1 < e2 =
                                 rdiv n (blowup e2 r1) r2
                               | a < b ∧ e1 ≡ e2 =
                                 rdiv n (blowup (e2 + 1) r1) r2
                               | otherwise =
                                 simplify (R (go n a b) (e1 − e2 + n))
  where go i x y | i ⩽ 0 = 0
                 | otherwise =
                   case x `quotRem` y of
                     (q, 0) → 10 ↑ i ∗ q
                     (q, r) → let (r′, e)      = borrow r y
                                  q′           = 10 ↑ i ∗ q
                              in if e > i then q′
                                          else q′ + go (i − e) r′ y
```

rdiv has one more argument than the arithmetic operations seen before. This additional argument, n, defines the precision of the result. This is necessary, because, as we will see, the number of iterations the function has to perform depends on the precision the result is expected to have.

If the first number is smaller than the second, either because its exponent or its coefficient is smaller, we blow it up so that it is at least the same size. Then we calculate the new coefficient by means of *go* and the new exponent as the difference of the first and the second exponent plus the expected precision. Note that division has the inverse effect on the size of the exponents as multiplication. When we look again at the example 1 and 0.1, we have $1/0.1 = 10$, which translates to $(R\ 1\ 0)\ /\ (R\ 1\ 1) = R\ (1\ /\ 1)\ (0 - 1)$, which of course is the same as $R\ 10\ 0$.

The inner function *go* proceeds until i, which initially is n, becomes 0 or smaller. In each step, we divide x, initially the coefficient of the first number, and y, the coefficient of the second number. If the result leaves no remainder, we are done. We just raise q to the power of the step in question. Otherwise, we continue dividing the remainder r by y. But before we continue, we borrow from y, that is, we increase r until it is at least y. i is then decremented by the number of zeros we borrowed this way. If we run out of i, so to speak, that is if $e > i$, then we terminate with q raised to the current step.

borrow is just the same as *blowup* applied to two natural numbers:

```
borrow :: Natural → Natural → (Natural, Natural)
borrow a b | a ⩾ b      = (a, 0)
           | otherwise = let (x, e) = borrow (10 * a) b in (x, e + 1)
```

We now make *RealN* instance of *Real*. We need to define just one method, namely how to convert *RealN* to *Rational*, which we do just by creating a fraction with the coefficient of the real number in the numerator and 10 raised to the exponent in the denominator. The rest is done by the *Rational* number type:

```
instance Real RealN where
  toRational (R r e) = i % (10 ↑ x)
    where i = fromIntegral r :: Integer
          x = fromIntegral e :: Integer
```

We further add a function to convert real numbers to our *Ratio* type:

```
r2R :: RealN → Ratio
r2R (R a e) = ratio a (10 ↑ e)
```

We also add a function to convert our real number type to the standard *Double* type:

```
r2d :: RealN → Double
r2d r@(R a e) | e > 16      = r2d (roundr 16 r)
              | otherwise = (fromIntegral a) / 10 ↑ e
```

An inconvenience is that we have to round a number given in our number type so it fits into a *Double*. For this, we assume that the *Double* has a precision of 16 decimal digits. This is not quite true. The *Double* type has room for 16 digits. But if the first digits after the decimal point are zeros, the *Double* type will present this as a number raised to a negative exponent, just as we do with our real type. In this case, the *Double* type may have a much higher precision than 16. For our purpose, however, this is not too relevant.

So, here is how we round:

```
roundr :: Natural → RealN → RealN
roundr n (R a e) | n ⩾ e      = R a e
                 | otherwise = let b = a `div` 10
                                   l = a − 10 * b
                                   d | l < 5 = 0
                                     | otherwise = 1
                               in roundr n (R (b + d) (e − 1))
```

That is, we first get the least significant digit l as $l = a - 10 * (div\ a\ 10)$, where, if *div* is the Euclidian division, the last digit of a remains. If the last digit is less than 5, we just drop it off. Otherwise, we add 1 to the whole number. If we now define

$one = R\ 1\ 0$
$three = R\ 3\ 0$
$third = one\ /\ three$,

then $roundr\ 16\ (three * third)$ yields 1, as desired.

Finally, we define

type *SReal* = *Signed RealN*

and have a fulfledged real number datatype.

1.8 The Stern-Brocot Tree

Achille Brocot (1817 – 1878) was part of a clockmaker dynasty in Paris started by his father and continuing after his death. The Brocots had a strong emphasis on engineering and, under the pressure of cheap low-quality imports mainly from the USA, innovated clockmaking with the aim to reduce production cost without equivalent degradation in quality. The constant engineering work manifested in a considerable number of patents hold by family members. The most productive, in terms of engineering, however, was Achille who improved many of his father's inventions and developed new ones. He also introduced a novelty to mathematics, which, surprisingly, has not only practical, but also theoretical value.

In clockmaking, as in machine construction in general, determining the ratio of components to each other, for instance, gear ratios, is a very frequent task. As often in practice, those ratios are not nice and clean, but very odd numbers with many decimal digits. Brocot developed a way to easily approximate such numbers with arbitrary precision and, in consequence, to approximate any real number with arbitrary precision. In the process, he developed yet another way to list all rational numbers.

Brocot's method can be described in terms of finite continued fractions. Recall that we can use lists of the form

$$[n; a, b, c, \dots]$$

to encode continued fractions like

$$n + \frac{1}{a + \frac{1}{b + \frac{1}{c + \dots}}}.$$

In contrast to continued fractions we have seen so far, we now look at finite continued fractions that actually result in rational numbers. The process to compute such a continued fraction can be captured in Haskell as:

$$
\begin{aligned}
&\mathit{contfracr} :: [\mathit{Ratio}] \to \mathit{Ratio}\\
&\mathit{contfracr}\ [\,] \quad\ = 0\\
&\mathit{contfracr}\ [i] \quad\ = i\\
&\mathit{contfracr}\ (i : \mathit{is}) = i + (\mathit{invert}\ \$\ \mathit{contfracr}\ \mathit{is})
\end{aligned}
$$

Here, *invert* is a function to create the multiplicative invert of a fraction, *i.e.* $\mathit{invert}(\frac{n}{d}) = \frac{d}{n}$ or in Haskell:

$$
\begin{aligned}
&\mathit{invert} :: \mathit{Ratio} \to \mathit{Ratio}\\
&\mathit{invert}\ (Q\ n\ d) = Q\ d\ n
\end{aligned}
$$

As you will see immediately, the expression $(\mathit{invert}\ \$\ \mathit{contfracr}\ \mathit{is})$, corresponds to

$$\frac{1}{\mathit{contfracr}\ \mathit{is}}.$$

The definition above, hence, creates a continued fraction that terminates with the last element in the list.

Now we introduce a simple rule to create from any continued fraction given in list notation two new continued fractions:

$$
\begin{aligned}
&\mathit{brocotkids} :: [\mathit{Ratio}] \to ([\mathit{Ratio}], [\mathit{Ratio}])\\
&\mathit{brocotkids}\ r = \mathbf{let}\ h \ \ = \mathit{init}\ r\\
&\qquad\qquad\qquad\qquad\ \ l \ \ = \mathit{last}\ r\\
&\qquad\qquad\qquad\qquad\ \ s \ \ = \mathit{length}\ r\\
&\qquad\qquad\qquad\qquad\ \ \mathit{k1} = h +\!\!+ [l + 1]\\
&\qquad\qquad\qquad\qquad\ \ \mathit{k2} = h +\!\!+ [l - 1, 2]\\
&\qquad\qquad\quad \mathbf{in\ if}\ \mathit{even}\ s\ \mathbf{then}\ (\mathit{k1}, \mathit{k2})\ \mathbf{else}\ (\mathit{k2}, \mathit{k1})
\end{aligned}
$$

This function yields two lists, k_1 and k_2. They are computed as the initial part of the input list, to which, in the case of k_1, one number is appended, namely the last element of the input list plus 1, or, in the case of k_2, two numbers are appended, namely the last element minus 1 and 2. For the input list $[0, 1]$, which is just 1, for instance, k_1 is $[0, 2]$, which is $\frac{1}{2}$, and k_2 is $[0, 0, 2]$, which is $\frac{1}{\frac{1}{2}} = 2$.

When we compare the paritiy of the length of the lists, we see that k_1 has the same parity as the input list and k_2 has the opposite parity. In particular, if the input list is even, then k_1 is even and k_2 is odd; if it is odd, then k_1 is odd and k_2 is even.

Now, we see for an even list like $[a, b]$ that there is an integer, a, to which the inverse of the second number, b is added. If b grows, then the overall result shrinks. The structure of an odd list is like $[a, b, c]$. Again, the integer a is added to the inverse of what follows in the list. But this time, if c grows, the inverse of c, $\frac{1}{c}$, shrinks and, as such, the value of $\frac{1}{b+1/c}$ grows. Therefore, if the number of elements is even, the value of k_1 is less than the value of the input list and, if it is odd, then the value is greater. You can easily convince yourself that for k_2 this is exactly the other way round. In consequence, the

numerical value of the left list returned by *brocotkids* is always smaller than that of the input list and that of the right one is greater.

Using this function, we can now approximate any real number. The idea is that, if the current continued fraction results in a number greater than the number in question, we continue with the left kid; if it is smaller, we continue with the right kid. Here is an implementation:

```
approx :: Natural → RealN → [Ratio]
approx i d = go i [0, 1]
  where go :: Natural → [Ratio] → [Ratio]
        go 0 _ = []
        go j r = let k@(Q a b) = contfracr r
                     d' = (fromIntegral a) / (fromIntegral b)
                     (k1, k2) = brocotkids r
                 in if d' ≡ d then [Q a b]
                              else if d' < d then k : go (j − 1) k2
                                             else  k : go (j − 1) k1
```

This function takes two arguments. The first, a natural number, defines the number of iterations we want to do. The second is the real number we want to approximate. We start the internal *go* with i and the list $[0, 1]$. In *go*, as long as $j > 0$, we compute the rational number that corresponds to the input list; then we compute the corresponding real number. If the number we computed this way equals the input (*i.e.* the input is rational), we are done. Otherwise, if it is less than the input, we continue with k_2; if it is greater, we continue with k_1.

The function yields the whole trajectory whose last number is the best approximation with n iterations. The result of *approx* 10 *pi*, for instance is:

$$1, 2, 3, 4, \frac{7}{2}, \frac{10}{3}, \frac{13}{4}, \frac{16}{5}, \frac{19}{6}, \frac{22}{7}.$$

The last fraction $\frac{22}{7}$ is approximately 3.142857, which still is a bit away from 3.141592. We reach 3.1415 with *approx* 25 *pi*, for which the last fraction is $\frac{333}{106} = 3.141509$. This way, we can come as close to π as we wish.

Since, with the *brocotkids* function, we always create two follow-ups for the input list, we can easily define a binary tree where, for each node, k_1 is the left subtree and k_2 is the right subtree. This tree, in fact, is well-known and is called *Stern-Brocot tree* in honour of Achille Brocot and Moritz Stern, the German number theorist we already know from the discussion of the Calkin-Wilf tree.

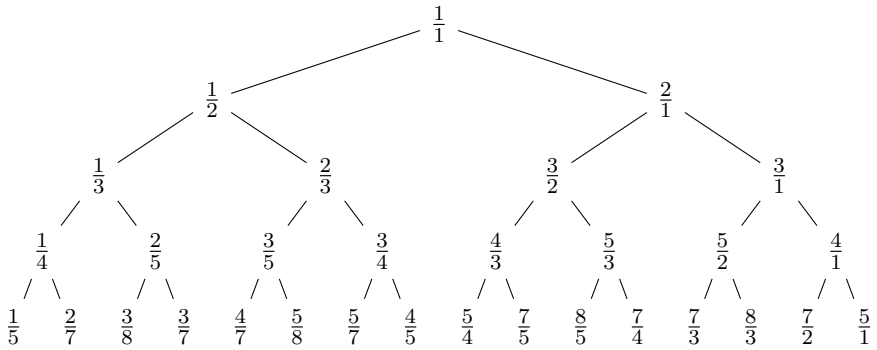
The Stern-Brocot tree can be defined as

```
type SterBroc = Tree [Ratio]
sterbroc :: Zahl → [Ratio] → SterBroc
sterbroc i r | i ≡ 0       = Node r []
             | otherwise = let (k1, k2) = brocotkids r
                           in Node r [sterbroc (i − 1) k1,
                                      sterbroc (i − 1) k2]
```

The function *sterbroc* takes an integer argument to define the number of generations we want to create and an initial list of *Ratio*. If we have exhausted the number of generations, we create the final *Node* without kids. If we start with a negative number, we will generate infinitely many generations. Otherwise, we create the *brocotkids* and continue with *sterbroc* on k_1 and k_2. We can now convert the continued fractions in the nodes to fractions by *fmap*ing *contfracr* on them. Here is a function that creates the Stern-Brocot Tree from root node $[0, 1]$ labled with fractions:

```
sterbrocTree :: Zahl → Tree Ratio
sterbrocTree i = fmap contfracr (sterbroc i [0, 1])
```

For the first five generations of this tree are



As you can see at once, this tree has many properties in common with the Calkin-Wilf tree. First and trivially, the left kid of a node k is less than k and the right kid of the same node is greater than k. The left-most branch of the tree contains all fractions with 1 in the numerator like $\frac{1}{1}, \frac{1}{2}, \frac{1}{3}, \frac{1}{4}$ and so on. The right-most branch contains the integers $\frac{1}{1}, \frac{2}{1}, \frac{3}{1}, \frac{4}{1}$ and so on.

Furthermore, the product of each generation is 1. For instance, $\frac{1}{1} = 1$, $\frac{1}{2} \times \frac{2}{1} = 1$, $\frac{1}{3} \times \frac{2}{3} \times \frac{3}{2} \times \frac{3}{1} = 1$ and so on. In fact, we see in each generation the same fractions we would also see in the Calkin-Wilf tree. The order of the fraction, however, is different. More precisely, the order of the inner fractions differs, since, as we have seen, the left-most and right-most numbers are the same.

We could hence ask the obvious question: how can we permute the generations of the Stern-Brocot tree to obtain the generations of the Calkin-Wilf tree and vice versa? Let

us look at an example. The 4^{th} generation of the Calkin-Wilf tree is
getKids 4 (*calWiTree* 4 (Q 1 1)):

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

The 4^{th} generation of the Stern-Brocot tree is
getKids 4 (*sterbroctree* 4):

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

We see that only some fractions changed their places and the changes are all direct swaps, such that the second position in the Calkin-Wilf tree changed with the fifth position and the fourth position changed with the seventh position. The other positions, the first, third, sixth and eighth, remain in their place. We could describe this in cyclic notation, using indexes from 0 – 7 for the eight positions:

$$(1, 4)(3, 6).$$

In other words, we represent the generations as arrays with indexes 0 – 7:

	0	1	2	3	4	5	6	7
Calkin-Wilf	$\frac{1}{4}$	$\frac{4}{3}$	$\frac{3}{5}$	$\frac{5}{2}$	$\frac{2}{5}$	$\frac{5}{3}$	$\frac{3}{4}$	$\frac{4}{1}$
Stern-Brocot	$\frac{1}{4}$	$\frac{2}{5}$	$\frac{3}{5}$	$\frac{3}{4}$	$\frac{4}{3}$	$\frac{5}{3}$	$\frac{5}{2}$	$\frac{4}{1}$

So, what is so special about the indexes 1, 3, 4 and 6 that distinguishes them from the indexes 0, 2, 5 and 7? When we represent these numbers in binary format with leading zeros, so that all binary numbers have the same length, we have

0	1	2	3	4	5	6	7
000	001	010	011	100	101	110	111

When we look at the indexes whose fractions do not change, we see one property that they all have in common: they are all symmetric. That is, when we reverse the bit strings, we still have the same number. 0 = 000 reversed is still 000 = 0; 2 = 010 reversed is still 010 = 2; 5 = 101 reversed is still 101 = 5 and 7 = 111 reversed is still 111 = 7. 1 = 001 reversed, however, is 100 = 4 and vice versa and 3 = 011 reversed is 110 = 6. This corresponds exactly to the permutation $(1, 4)(3, 6)$ and is an instance of a bit-reversal permutation.

Let us try to implement the bit-reversal permutation. First we implement the bit-reverse of the indexes. To do so, we first need to convert the decimal index into a binary number; then we add zeros in front of all binary numbers that are shorter than the greatest number; then we simply reverse the lists of binary digits, remove the leading zeros and convert back to decimal numbers. This can be nicely expressed by the function

$$
\begin{aligned}
&\mathit{bitrev} :: \mathit{Int} \to \mathit{Int} \to \mathit{Int} \\
&\mathit{bitrev}\ x = \mathit{fromIntegral} \circ \mathit{fromBinary} \circ \\
&\qquad\qquad \mathit{cleanz} \circ \mathit{reverse} \circ \mathit{fillup}\ x\ 0 \circ \\
&\qquad\qquad \mathit{toBinary} \circ \mathit{fromIntegral}
\end{aligned}
$$

where *fillup* is defined as

$$
\begin{aligned}
&\mathit{fillup} :: \mathit{Int} \to \mathit{Int} \to [\mathit{Int}] \to [\mathit{Int}] \\
&\mathit{fillup}\ i\ z\ \mathit{is} \mid \mathit{length}\ \mathit{is} \equiv i = \mathit{is} \\
&\qquad\qquad\qquad \mid \mathit{otherwise} \quad\ = \mathit{fillup}\ i\ z\ (z : \mathit{is})
\end{aligned}
$$

and *cleanz* as

$$
\begin{aligned}
&\mathit{cleanz} :: [\mathit{Int}] \to [\mathit{Int}] \\
&\mathit{cleanz}\ [\,] \quad\ = [\,] \\
&\mathit{cleanz}\ [0] \quad\ = [0] \\
&\mathit{cleanz}\ (0 : \mathit{is}) = \mathit{cleanz}\ \mathit{is} \\
&\mathit{cleanz}\ \mathit{is} \quad\ = \mathit{is}
\end{aligned}
$$

To apply this, we first have to calculate the size of the greatest number in our set in binary format. If we assume that we have a list of consecutive numbers from $0 \ldots n-1$, then the size of the greatest number is just $\log_2 n$, the binary logarithm of n. For $n = 8$, for instance, this is 3. With this out of the way, we can define a bit reversal of the indexes of any set $[a]$ as:

$$
\begin{aligned}
&\mathit{idxbitrev} :: [a] \to [\mathit{Int}] \\
&\mathit{idxbitrev}\ \mathit{xs} = \mathbf{let}\ l\ = \mathit{fromIntegral}\ \$\ \mathit{length}\ \mathit{xs} \\
&\qquad\qquad\qquad\qquad\ \ x = \mathit{round}\ \$\ \mathit{logBase}\ 2\ (\mathit{fromIntegral}\ l) \\
&\qquad\qquad\qquad\ \ \mathbf{in}\ [\mathit{bitrev}\ x\ i \mid i \leftarrow [0\,..\,l-1]]
\end{aligned}
$$

and use this function to permute the original input list:

$$
\begin{aligned}
&\mathit{bitreverse} :: [a] \to [a] \\
&\mathit{bitreverse}\ \mathit{xs} = \mathit{go}\ \mathit{xs}\ (\mathit{idxbitrev}\ \mathit{xs}) \\
&\quad \mathbf{where}\ \mathit{go}\ \_\ [\,] \quad\ = [\,] \\
&\qquad\qquad \mathit{go}\ \mathit{zs}\ (p : \mathit{ps}) = \mathit{zs}\ !!\ p : \mathit{go}\ \mathit{zs}\ \mathit{ps}
\end{aligned}
$$

Applied on the list $[0, 1, 2, 3, 4, 5, 6]$, we see exactly the $(1, 4)(3, 6)$ permutation we saw above, namely $[0, 4, 2, 6, 1, 5, 3]$. Applied on a generation from the Calkin-Wilf tree, we see the corresponding generation from the Stern-Brocot tree. Let $t = \mathit{calWiTree}\ (-1)\ (1\%\ 1)$, we see for

$getKids\ 3\ t$	$\frac{1}{3}, \frac{3}{2}, \frac{2}{3}, \frac{3}{1}$
$bitreverse\ (getKids\ 3\ t)$	$\frac{1}{3}, \frac{2}{3}, \frac{3}{2}, \frac{3}{1}$
$getKids\ 4\ t$	$\frac{1}{4}, \frac{4}{3}, \frac{3}{5}, \frac{5}{2}, \frac{2}{5}, \frac{5}{3}, \frac{3}{4}, \frac{4}{1}$
$bitreverse\ (getKids\ 4\ t)$	$\frac{1}{4}, \frac{2}{5}, \frac{3}{5}, \frac{3}{4}, \frac{4}{3}, \frac{5}{3}, \frac{5}{2}, \frac{4}{1}$

Since the generations of the Stern-Brocot tree are nothing but permutations of the Calkin-Wilf tree, we can derive a sequence from the Stern-Brocot tree that lists all rational numbers. We create this sequence in exaclty the same way we did for the CalkinWilf tree, namely

```
enumQsb :: [Ratio]
enumQsb = go 1 $ sterbrocTree (−1)
  where go i t = getKids i t ++ go (i + 1) t
```

The numerators of the sequence derived in this way from the Calkin-Wilf tree equal the well-known Stern sequence. Is there another well-known sequence that is equivalent to the numerators of the Stern-Brocot tree sequence? Let us ask the On-line Encyclopedia with the first segment of that sequence generated by *map numerator* (*take* 20 *enumQsb*):

$$1, 1, 2, 1, 2, 3, 3, 1, 2, 3, 3, 4, 5, 5, 4, 1, 2, 3, 3, 4, 5, 5, 4, 5, 7.$$

The Encyclopedia tells us that this is the numerators of the *Farey sequence*. This sequence, named for British geologist John Farey (1766 – 1826), has a lot of remarkable properties. The Farey sequence of n lists all fractions in canoncial form between 0 and 1, usually included, with a denominator less or equal than n. For instance, the Farey sequence of 1, designated F_1 just contains $0, 1$; F_2 contains $0, \frac{1}{2}, 1$; F_3 contains $0, \frac{1}{3}, \frac{2}{3}, 1$ and so on.

A direct way to implement this could be to combine all numbers from $0 \ldots n$ in the numerator with all numbers $1 \ldots n$ in the denominator that are smaller than 1 and to sort and *nub* the resulting list, like this:

```
farey2 :: Natural → [Ratio]
farey2 n = sort (nub $ filter (⩽ 1) $
                    concatMap (λx → map (x%) [1 . . n]) [0 . . n])
```

With this approach, we create a lot of fractions that we do not need and that we filter out again afterwards. A more interesting approach, also in the light of the topic of this section, is the following:

$$
\begin{aligned}
&\mathit{farey} :: \mathit{Natural} \to [\mathit{Ratio}] \\
&\mathit{farey}\ n = 0 : \mathit{sort}\ (\mathit{go}\ 1\ \$\ \mathit{sterbrocTree}\ (-1)) \\
&\quad \textbf{where}\ \mathit{go}\ k\ t = \textbf{let}\ g = \mathit{getKids}\ k\ t \\
&\qquad\qquad\qquad\qquad\quad l = \mathit{filter}\ \mathit{fltr}\ g \\
&\qquad\qquad\qquad\quad \textbf{in if}\ \mathit{null}\ l\ \textbf{then}\ l\ \textbf{else}\ l +\!\!+ \mathit{go}\ (k+1)\ t \\
&\qquad\quad \mathit{fltr}\ k\ = k \leqslant 1 \wedge n \geqslant \mathit{denominator}\ k
\end{aligned}
$$

Here, we iterate over the generations of the Stern-Brocot tree removing the fractions that are greater than 1 or have a denominator greater n. When we do not get results anymore, *i.e.* all denominators are greater than n, we are done.

Let us try this algorithm on some numbers:

$$F_4 = \left\{0, \frac{1}{4}, \frac{1}{3}, \frac{1}{2}, \frac{2}{3}, \frac{3}{4}, 1\right\} \tag{1.34}$$

$$F_5 = \left\{0, \frac{1}{5}, \frac{1}{4}, \frac{1}{3}, \frac{2}{5}, \frac{1}{2}, \frac{3}{5}, \frac{2}{3}, \frac{3}{4}, \frac{4}{5}, 1\right\} \tag{1.35}$$

$$F_6 = \left\{0, \frac{1}{6}, \frac{1}{5}, \frac{1}{4}, \frac{1}{3}, \frac{2}{5}, \frac{1}{2}, \frac{3}{5}, \frac{2}{3}, \frac{3}{4}, \frac{4}{5}, \frac{5}{6}, 1\right\} \tag{1.36}$$

We see some interesting properties. First and this should be obvious, we see n as a denominator in sequence F_n exaclty $\varphi(n)$ times. For F_6, for instance, we could create the fractions $\frac{1}{6}$, $\frac{2}{6}$, $\frac{3}{6}$, $\frac{4}{6}$ and $\frac{5}{6}$. The fractions $\frac{2}{6} \ldots \frac{4}{6}$, however, are not in canonical form, since the numerators $2 \ldots 4$ all share divisors with 6. Since there are $\varphi(n)$ numerators that do not share divisors with n, there are only $\varphi(n)$ fractions less than 1 whith n in the denominator.

Another property is that, for two consecutive fractions in the Farey sequence, $\frac{a}{b}$ and $\frac{c}{d}$, the cross products ad and cb are consecutive integers. In again F_6, for the fractions $\frac{1}{6}$ and $\frac{1}{5}$, the cross products, trivially, are 5 and 6. More interesting are the fractions $\frac{3}{5}$ and $\frac{2}{3}$ whose cross products are $3 \times 3 = 9$ and $5 \times 2 = 10$.

Even further, for any three consecutive fractions in a Farey sequence, the middle one, called the mediant fraction, can be calculated from the outer ones as $\frac{a}{b}, \frac{a+c}{b+d}, \frac{c}{d}$. For instance in F_6:

$$\frac{1+1}{6+4} = \frac{2}{10} = \frac{1}{5}, \tag{1.37}$$

$$\frac{2+3}{5+5} = \frac{5}{10} = \frac{1}{2} \tag{1.38}$$

and

$$\frac{3+5}{4+6} = \frac{8}{10} = \frac{4}{5}. \tag{1.39}$$

This property can be used to compute F_{n+1} from F_n. We just have to insert those mediant fractions of two consecutive fractions in F_n, for which the denominator is $n+1$. In F_6 we would insert

$$\frac{0+1}{1+6}, \frac{1+1}{4+3}, \frac{2+1}{5+2}, \frac{1+3}{2+5}, \frac{2+3}{3+4}, \frac{5+1}{6+1}$$

resulting in

$$F_7 = \left\{0, \frac{1}{7}, \frac{1}{6}, \frac{1}{5}, \frac{1}{4}, \frac{2}{7}, \frac{1}{3}, \frac{2}{5}, \frac{3}{7}, \frac{1}{2}, \frac{4}{7}, \frac{3}{5}, \frac{2}{3}, \frac{5}{7}, \frac{3}{4}, \frac{4}{5}, \frac{5}{6}, \frac{6}{7}, 1\right\} \tag{1.40}$$

We can implement this as

```
nxtFarey :: Natural → [Ratio] → [Ratio]
nxtFarey n []    = []
nxtFarey n [r]   = [r]
nxtFarey n (a : b : rs) | denominator a +
                          denominator b ≡ n = nxtFarey n (a : x : b : rs)
                        | otherwise         = a : nxtFarey n (b : rs)
  where x = let n1 = numerator a
                n2 = numerator b
                d1 = denominator a
                d2 = denominator b
            in (n1 + n2) % (d1 + d2)
```

In fact, we can construct the Stern-Brocot tree by means of mediant fractions. The outer fractions, in this algorithm are the predecessors of the current node, namely the direct predecessor and either the predecessor of the predecessor or the sibling of that node. For instance, the second node in the third generation is $\frac{2}{3}$. Its kids are $\frac{3}{5}$ and $\frac{3}{4}$. $\frac{3}{5}$ is $\frac{2+1}{3+2}$ and, thus, the sum of $\frac{2}{3}$ and its predecessor; $\frac{4}{3}$, however, is $\frac{2+1}{3+1}$ and, hence, the sum of the $\frac{2}{3}$ and the predecessor of its predecessor.

The question now is how to bootstrap this algorithm, since the root node does not have predecessors. For this case, we imagine two predecessors, namely the fractions $\frac{0}{1}$ and $\frac{1}{0}$, the latter of which, of course, is not a proper fraction. The assumption of such nodes, however, helps us derive the outer branches, where, on the left side, the numerator does not change, hence is constructed by addition with 0, and, on the right side, the denominator does not change and is likewise constructed by addition with 0.

We implement this as

```
mSterbroctree :: Zahl → Natural → Natural →
                 Natural → Natural → Ratio → Tree Ratio
mSterbroctree 0 _ _ _ _ r = Node r []
mSterbroctree n a b c d r = let rn = numerator r
                               rd = denominator r
                               k1 = (a + rn) % (b + rd)
                               k2 = (c + rn) % (d + rd)
                           in  if k1 < k2
                               then Node r [mSterbroctree (n − 1) a b rn rd k1,
                                            mSterbroctree (n − 1) c d rn rd k2]
                               else  Node r [mSterbroctree (n − 1) c d rn rd k2,
                                            mSterbroctree (n − 1) a b rn rd k1]
```

Note that we have to use two pairs of natural numbers instead of two fractions to encode the predecessors. This is because we have to represent the imagined predecessor $\frac{1}{0}$, which is not a proper fraction. Finally, we check for the smaller of the resulting numbers k_1 and k_2 to make sure that the smaller one always goes to the left and the greater to the right. This implementation now gives exactly the same tree as the implementation using continued fractions introduced at the beginning of the section.

1.9 Field Extension

1.10 p-adic Numbers

1.11 Real Factorials

1.12 The Continuum

1.13 Review of the Number Zoo