Exercise Class 5

1 Binary Representation

First we want to be able to calculate the binary representation of non-integer decimal numbers by hand. Before we start, notice what a digit 1 after the decimal point in the binary representation actually means. Here is a little overview:

binary:	1	1	1	1	.	1	1	1
decimal:	8	4	2	1		$\frac{1}{2}$	$\frac{1}{4}$	$\frac{1}{8}$

Next we calculate the binary representation of the decimal number 1.9 as an example. In the lecture we have seen an algorithm for that:

x	b_i	$x - b_i$	$2 \cdot (x - b_i)$
1.9	1	0.9	1.8
1.8	1	0.8	1.6
1.6	1	0.6	1.2
1.2	1	0.2	0.4
0.4	0	0.4	0.8
0.8	0	0.8	1.6
1.6	1	0.6	1.2

Notice the bold **1.6** above? This is the beginning of the repeated part. So the number evaluates to $1.1\overline{1100}$.

Why the decimal number 1.9 or even 0.1 has no finite binary representation, but a finite decimal one? The reason is that displaying 0.1 or $\frac{1}{10}$ in a finite fashion requires the prime numbers 2 and 5 (because $10 = 2 \cdot 5$). The decimal system is based on those two prime numbers, the binary system (and the hexadecimal system) is based on the prime number 2 ($16 = 2^4$). Hence there is no finite representation of 0.1 in the binary (and hexadecimal) system, but in the decimal system there is one.

1.1 Exercise: Binary Representation

1.1.1 Task

Compute the binary expansions of the following decimal numbers.

1. 0.25
2. 11.1

1.1.2 Solution

1. 0.01
2. $1011.0\overline{0011}$

To see the calculations in more detail, see Script Exercise 59.

2 Normalized FP-systems $F^*(b, p, e_{min}, e_{max})$

$F^*(b, p, e_{min}, e_{max})$ describes the set of numbers of the form

$$\pm\ b_0.b_1b_2...b_{p-1} \cdot b^e$$

where $b_i \in \{0, ..., b-1\}$ and $b_0 \neq 0$, and $e \in \{e_{min}, e_{min}+1,\ ...,\ e_{max}-1, e_{max}\}$.

For example, $F^*(2, 4, -2, 2)$ contains the following numbers:

$1.000 \cdot 2^1$ (i.e. 2 in decimal)
$1.001 \cdot 2^{-1}$ (i.e. 0.5625 in decimal)
$1.111 \cdot 2^{-2}$ (i.e. 0.46875 in decimal)
...

But, for example, the following numbers are **not** included:

$0.000 \cdot 2^1$ (i.e. 0 is not "normalizable")
$1.0001 \cdot 2^{-1}$ (since too many places after the decimal point)
$1.111 \cdot 2^5$ (since exponent $5 \notin \{-2, -1, 0, 1, 2\}$)

2.1 Exercise: Some numbers in $F^*(2, 4, -2, 2)$

2.1.1 Task

State the following numbers in $F^*(2, 4, -2, 2)$:

1. the largest number;
2. the smallest number;
3. the smallest non-negative number.

Compute how many numbers are in the set $F^*(2, 4, -2, 2)$.

2.1.2 Solution

1. The largest number is $1.111 \cdot 2^2$ which is 7.5 in decimal.
2. The smallest number is $-1.111 \cdot 2^2$ which is -7.5 in decimal.
3. The smallest non-negative number is $1.000 \cdot 2^{-2}$ which is 0.25 in decimal.

The set has 80 numbers in it. This can be seen as follows. For a fixed exponent there are three digits we can vary freely, and for each number also the negative number is in the set, thus resulting in $2 \cdot 2^3 = 16$ numbers per exponent. On the other hand, there are 5 possible exponents, thus resulting in $5 \cdot 16 = 80$ numbers. Notice that in normalized number systems we cannot "count some numbers twice" as we've seen in the lecture that the representation of a number is unique.

2.2 Exercise: Computations in $F^*(2, 4, -2, 2)$

2.2.1 Task

Add $1.001 \cdot 2^{-1}$ (i.e. 0.5625) and $1.111 \cdot 2^{-2}$ (i.e. 0.46875).

2.2.2 Solution

The two numbers $1.001 \cdot 2^{-1}$ (i.e. 0.5625) and $1.111 \cdot 2^{-2}$ (i.e. 0.46875) are added as follows:

1. Bring both to say exponent -1: $1.001 \cdot 2^{-1}$ and $0.1111 \cdot 2^{-1}$
2. Add as binary numbers:
$$\begin{array}{rr} & 1.001 \cdot 2^{-1} \\ + & 0.1111 \cdot 2^{-1} \\ \hline & 10.0001 \cdot 2^{-1} \end{array}$$
3. Re-normalize: $1.00001 \cdot 2^0$
4. Round: $1.000 \cdot 2^0$

Notice that this does not coincide with their exact sum 1.03125 as the precision 4 is not high enough to represent this number.

3 Floating Point Guidelines

In the lecture, you have seen guidelines how to correctly use floating point numbers. On the course website, you can find a slide deck that provides additional examples for these rules.

4 Comparing Floating Point Numbers

In the lecture, you have seen a guideline that you should never check floating point numbers for strict equality. Instead, you should test that the absolute difference between them is smaller than some small value. On the course website you can find slides that explain this idea in more detail.