DM552 Exercises 2

Department of Mathematics and Computer Science
University of Southern Denmark

September 8, 2017

Please bring your laptop to this exercise class! You will get time to solve the exercises in class, and in the end, the solutions will be discussed.

Boolean functions

The concept of truth tables for a boolean function is well-known. Below, we see the truth table of $\wedge$ (AND).

a	b	$a \wedge b$
F	F	F
F	T	F
T	F	F
T	T	T

In Haskell, we can make our own implementation of $\wedge$ using pattern matching

$$
\begin{aligned}
&\mathit{andImpl} :: \mathit{Bool} \to \mathit{Bool} \to \mathit{Bool} \\
&\mathit{andImpl}\ \mathit{False}\ \mathit{False} = \mathit{False} \\
&\mathit{andImpl}\ \mathit{False}\ \mathit{True}\ = \mathit{False} \\
&\mathit{andImpl}\ \mathit{True}\ \mathit{False}\ = \mathit{False} \\
&\mathit{andImpl}\ \mathit{True}\ \mathit{True}\ \ = \mathit{True}
\end{aligned}
$$

We can shorten it using wildcards

$$
\begin{aligned}
&\mathit{andImpl} :: \mathit{Bool} \to \mathit{Bool} \to \mathit{Bool} \\
&\mathit{andImpl}\ \mathit{True}\ \mathit{True} = \mathit{True} \\
&\mathit{andImpl}\ \_\ \_ = \mathit{False}
\end{aligned}
$$

or

$$\begin{aligned}
&\mathit{andImpl} :: \mathit{Bool} \to \mathit{Bool} \to \mathit{Bool}\\
&\mathit{andImpl}\ \mathit{True}\ a = a\\
&\mathit{andImpl}\ _\ _ = \mathit{False}
\end{aligned}$$

1. Implement $\neg$ (NOT), $\vee$ (OR), $\oplus$ (XOR) and NAND using pattern matching and wildcards:

$$\begin{aligned}
&\mathit{notImpl} :: \mathit{Bool} \to \mathit{Bool}\\
&\mathit{orImpl}\ \ :: \mathit{Bool} \to \mathit{Bool} \to \mathit{Bool}\\
&\mathit{xorImpl} :: \mathit{Bool} \to \mathit{Bool} \to \mathit{Bool}\\
&\mathit{nandImpl} :: \mathit{Bool} \to \mathit{Bool} \to \mathit{Bool}
\end{aligned}$$

A few simple functions

1. Define a function *eeny* that returns the string `"eeny"` for even inputs – and `"meeny"` for odd inputs.

$$\mathit{eeny} :: \mathit{Integer} \to \mathit{String}$$

2. Define a function *fizzbuzz* that returns `"Fizz"` for numbers divisible by 3, `"Buzz"` for numbers divisible by 5, and `"FizzBuzz"` for numbers divisible by both. For other numbers it returns the empty string.

$$\mathit{fizzbuzz} :: \mathit{Integer} \to \mathit{String}$$

   You can use the function *mod* to compute modulo.

Recursive functions on integers

The mathematical function

$$gcd(a, b) = \begin{cases} a, & b = 0 \\ gcd(b, a \bmod b), & b > 0 \end{cases}$$

can be implemented in Haskell by:

$$\begin{aligned}
&\mathit{gcd} :: \mathit{Integer} \to \mathit{Integer} \to \mathit{Integer}\\
&\mathit{gcd}\ a\ 0 = a\\
&\mathit{gcd}\ a\ b = \mathit{gcd}\ b\ (a\ `\mathit{mod}`\ b)
\end{aligned}$$

1. The sum of all numbers up to n can be defined using recursion:

$$S(n) = \begin{cases} 0, & n = 0 \\ n + S(n-1), & n > 0 \end{cases}$$

   Implement a function $\mathit{sumTo} :: \mathit{Integer} \to \mathit{Integer}$ in Haskell, which implements this definition.

2. Binomial coefficients can be defined using recursion:

$$B(n,k) = \begin{cases} B(n-1,k) + B(n-1,k-1), & 1 \leqslant k \leqslant n-1 \\ 1, & k = 0 \lor k = n \\ 0, & k < 0 \lor k > n \end{cases}$$

   Implement a function $\mathit{binomial} :: \mathit{Integer} \to \mathit{Integer} \to \mathit{Integer}$ in Haskell, which implements this definition.

3. Use recursion to define a function $\mathit{power} :: \mathit{Integer} \to \mathit{Integer} \to \mathit{Integer}$. $\mathit{power}\ n\ k$ should compute n^k.

4. Use recursion to define a function $\mathit{ilog2} :: \mathit{Integer} \to \mathit{Integer}$. $\mathit{ilog2}\ n$ should be the number of times you can halve the integer n (rounding down) before you get 1.

Recursive functions on lists

A list in Haskell is constructed using the following primitives

$$\begin{array}{ll} [\,] :: [\,a\,] & \text{-- the empty list} \\ (:) :: a \to [\,a\,] \to [\,a\,] & \text{-- the cons operator} \end{array}$$

A non-empty list is of the form $(x : \mathit{xs})$ where x is the first element of the list, and xs is the tail of the list.

As an example of a recursive function on lists, see

$$\begin{array}{ll} \mathit{flattenMaybe} :: [\,\mathit{Maybe}\ a\,] \to [\,a\,] & \\ \mathit{flattenMaybe}\ [\,] & = [\,] \\ \mathit{flattenMaybe}\ (\mathit{Just}\ x : \mathit{xs}) & = x : \mathit{flatten}\ \mathit{xs} \\ \mathit{flattenMaybe}\ (\mathit{Nothing} : \mathit{xs}) & = \mathit{flatten}\ \mathit{xs} \end{array}$$

This function returns a list of all values which are wrapped in Just - e.g. $\mathit{flatten}\ [\,\mathit{Just}\ 1, \mathit{Nothing}, \mathit{Just}\ 2\,] \equiv [1,2]$.

Please try to implement the following functions using only recursion, pattern matching and the list constructors $[\,]$ and $(:)$. You should not use list functions from Haskell's standard library. If time permits, you can afterwards find ways to solve the exercises using helper functions from the standard library.

1. Create a function which drops all the zeros from a list:

$$\begin{aligned}&\mathit{dropZeros} :: [\mathit{Int}] \to [\mathit{Int}]\\&\mathit{dropZeros}\ [-1,0,1,2,0] \equiv [-1,1,2]\end{aligned}$$

2. Create a function which flattens a list of lists, i.e.

$$\begin{aligned}&\mathit{flatten} :: [[a]] \to [a]\\&\mathit{flatten}\ [[1],[\,],[1,2,3],[2]] \equiv [1,1,2,3,2]\end{aligned}$$

3. Create function which repeats each element of the list:

$$\begin{aligned}&\mathit{twiceAll} :: [a] \to [a]\\&\mathit{twiceAll}\ [1,2,3,4] \equiv [1,1,2,2,3,3,4,4]\end{aligned}$$

4. Create a function which flips the sign of every other element of the list.

$$\begin{aligned}&\mathit{alternate} :: [\mathit{Int}] \to [\mathit{Int}]\\&\mathit{alternate}\ [1,2,3,4,5] \equiv [1,-2,3,-4,5]\end{aligned}$$

5. Create a function which repeats each element of the list a specified number of times:

$$\begin{aligned}&\mathit{replicateAll} :: \mathit{Int} \to [a] \to [a]\\&\mathit{replicateAll}\ 2\ [1,2,3,4] \equiv [1,1,2,2,3,3,4,4]\\&\mathit{replicateAll}\ 3\ [1,2,3,4] \equiv [1,1,1,2,2,2,3,3,3,4,4,4]\end{aligned}$$

6. Create function which computes the cumulative sum of a list.

$$\begin{aligned}&\mathit{cumulativeSum} :: [\mathit{Int}] \to [\mathit{Int}]\\&\mathit{cumulativeSum}\ [1,2,3,4] \equiv [1,1+2,1+2+3,1+2+3+4] \equiv [1,3,6,10]\end{aligned}$$