DM552 exercises

Department of Mathematics and Computer Science
University of Southern Denmark

October 11, 2017

1. Prove the following statements (function definitions of *length*, *map*, $(+\!\!+)$ etc. are given on the last page of this document)

   (a) For any list xs,

   $$length\ (map\ f\ xs) \equiv length\ xs$$

   (b) For any lists xs and ys:

   $$map\ f\ (xs +\!\!+ ys) \equiv map\ f\ xs +\!\!+ map\ f\ ys$$

   (c) For any $n \in \mathbb{N}_0$ and any list xs,

   $$take\ n\ xs +\!\!+ drop\ n\ xs \equiv xs$$

2. Given the type declaration

   $$\textbf{data}\ Tree = Leaf\ Int \mid Node\ Tree\ Tree$$

   show that the number of leaves in a such a tree is always one greater than the number of nodes, by induction on trees. Hint: start by defining functions that count the number of leaves and nodes in a tree.

3. Consider the following *Tree* data type:

$$\textbf{data}\ \mathit{Tree}\ v = \mathit{Node}\ v\ [\mathit{Tree}\ v]$$

$$\begin{aligned}
&\mathit{testTree} :: \mathit{Tree}\ \mathit{Int}\\
&\mathit{testTree} = \mathit{Node}\ 3\ [\mathit{Node}\ 4\\
&\qquad [\mathit{Node}\ 5\ [\,]\\
&\qquad ,\mathit{Node}\ 6\ [\,]\\
&\qquad ,\mathit{Node}\ 7\ [\,]\\
&\qquad]\\
&\quad ,\mathit{Node}\ 9\\
&\qquad [\mathit{Node}\ 10\ [\,]\\
&\qquad]\\
&\quad]
\end{aligned}$$

(a) Write a function *showTree* which returns the lines of a printed tree:

```
*Main> putStrLn $ unlines $ showTree testTree
3
+- 4
|  +- 5
|  +- 6
|  +- 7
+- 9
   +- 10
```

$$\begin{aligned}
&\mathit{showTree} :: \mathit{Show}\ v \Rightarrow \mathit{Tree}\ v \to [\mathit{String}]\\
&\mathit{showTree}\ (\mathit{Node}\ v\ \mathit{ts}) = (...)\\
&\textbf{instance}\ \mathit{Show}\ v \Rightarrow \mathit{Show}\ (\mathit{Tree}\ v)\ \textbf{where}\\
&\quad \mathit{show}\ t = \mathit{unlines}\ \$\ \mathit{showTree}\ t
\end{aligned}$$

(b) Given a list of tuples containing title and level number,

$$
\begin{aligned}
&\mathit{outline} :: [(\mathit{String}, \mathit{Int})]\\
&\mathit{outline} = [\\
&\quad (\texttt{"Haskell"}, 1),\\
&\quad (\texttt{"Introduction"}, 2),\\
&\quad (\texttt{"Expressions"}, 3),\\
&\quad (\texttt{"Types"}, 3),\\
&\quad (\texttt{"Patterns"}, 3),\\
&\quad (\texttt{"List algorithms"}, 2),\\
&\quad (\texttt{"Structural recursion"}, 3),\\
&\quad (\texttt{"List comprehensions"}, 3),\\
&\quad (\texttt{"Prolog"}, 1),\\
&\quad (\texttt{"Predicate logic"}, 3)\\
&\quad]
\end{aligned}
$$

Write a function with type

$$\mathit{outlineToTree} :: a \rightarrow \mathit{Int} \rightarrow [(a, \mathit{Int})] \rightarrow \mathit{Tree}\ (\mathit{Maybe}\ a)$$

which takes a title, a level number and a list of children, and returns a *Tree* -

```
*Main> outlineToTree "Programming Languages" 1 outline
Just "Programming Languages"
+- Just "Haskell"
|  +- Just "Introduction"
|  |  +- Just "Expressions"
|  |  +- Just "Types"
|  |  +- Just "Patterns"
|  +- Just "List algorithms"
|  |  +- Just "Structural recursion"
|  |  +- Just "List comprehensions"
+- Just "Prolog"
   +- Nothing
      +- Just "Predicate logic"
```

4. A type t is a Monoid, if there exists an operation $(+) :: t \to t \to t$ and an identity element $e :: t$, s.t.

   - for all $x :: t$, $x + e = x = e + x$
   - for all $x, y, z :: t$, $x + (y + z) = (x + y) + z$

   In Haskell, there is the type class **Data.Monoid**. Here,

   - the operation is called *mappend*,
   - the identity element is called *mempty*
   - the fold is called *mconcat*

   Complete the following instance declarations:

```
import Data.Monoid (Monoid, mappend, mempty, mconcat)
newtype Product a = Product a deriving Show
newtype Sum a = Sum a deriving Show
newtype All = All Bool deriving Show
newtype Any = Any Bool deriving Show
newtype First a = First (Maybe a) deriving Show
newtype Last a = Last (Maybe a) deriving Show
instance Num a => Monoid (Sum a) where
  (Sum a) `mappend` (Sum b) = ...
  mempty                    = ...
instance Num a => Monoid (Product a) where
  ...
instance Monoid All where
  ...
instance Monoid Any where
  ...
instance Monoid (First a) where
  ...
instance Monoid (Last a) where
  ...
```

Function definitions

$$
\begin{aligned}
&map :: (a \to b) \to [\,a\,] \to [\,b\,]\\
&map\ f\ [\,] \qquad\quad = [\,]\\
&map\ f\ (x : xs) = (f\ x) : map\ f\ xs
\end{aligned}
$$

$$
\begin{aligned}
&take :: Int \to [\,a\,] \to [\,a\,]\\
&take\ n\ \_ \qquad\quad \mid n \leqslant 0 = [\,]\\
&take\ \_\ [\,] \qquad\qquad\qquad = [\,]\\
&take\ n\ (x : xs) \qquad\quad\ = x : take\ (n-1)\ xs
\end{aligned}
$$

$$
\begin{aligned}
&drop :: Int \to [\,a\,] \to [\,a\,]\\
&drop\ n\ xs \qquad \mid n \leqslant 0 = xs\\
&drop\ \_\ [\,] \qquad\qquad\quad\ = [\,]\\
&drop\ n\ (\_ : xs) \qquad\quad = drop\ (n-1)\ xs
\end{aligned}
$$

$$
\begin{aligned}
&(+\!\!+) :: [\,a\,] \to [\,a\,] \to [\,a\,]\\
&[\,] +\!\!+ ys \qquad\quad = ys\\
&(x : xs) +\!\!+ ys = x : (xs +\!\!+ ys)
\end{aligned}
$$

$$
\begin{aligned}
&length :: [\,a\,] \to Int\\
&length\ [\,] = 0\\
&length\ (x : xs) = 1 + length\ xs
\end{aligned}
$$