Funtional and Logic Programming
Exercise Set 2

Tom Schrijvers Steven Keuchel
{tom.schrijvers,steven.keuchel}@ugent.be

Deadline: Tuesday, October 9th, 2012, 11:59pm

Higher-order functions

1. Write the following functions in terms of *foldr*

(a) *reverse*
(b) *append*
(c) *concat*
(d) *filter*

and the following functions in terms of *foldl*

(e) *reverse*

2. We can represent polynoms as a list of coefficients.

$$\textbf{data}\ \mathit{Polynom}\ a = \mathit{PN}\ [a]$$
$$\quad \textbf{deriving}\ \mathit{Show}$$

where

$$\mathit{PN}\ [a_n, ..., a_0]$$

represents the polynom

$$a_n x^n + ... + a_0 x^0.$$

Write an evaluation function *evalPN* that evaluates a polynom at a given value. Use Horner's rule and a fold.

Zips

Another often used higher-order function on lists is *zipWith*

$$\begin{array}{llll}
\mathit{zipWith} :: (a \to b \to c) \to [a] \to [b] \to [c] \\
\mathit{zipWith}\ f\ [\,] & bs & = [\,] \\
\mathit{zipWith}\ f\ as & [\,] & = [\,] \\
\mathit{zipWith}\ f\ (a : as) & (b : bs) & = f\ a\ b : \mathit{zipWith}\ f\ as\ bs
\end{array}$$

It takes a binary function and two lists and produces a list where the function is applied element-wise. If one of the lists is longer than the other, excess elements of the longer list are discarded.

3. A vector of $\mathbb{R}^n$ can be represented as a list of floating-point numbers.

$$\textbf{type}\ Vector = [Float]$$

Implement the following operations on vectors using higher-order functions

(a) Vector addition

$$add :: Vector \to Vector \to Vector$$

(b) Scalar multiplication

$$scale :: Float \to Vector \to Vector$$

(c) Dot product

$$dot :: Vector \to Vector \to Float$$

Note: Vector addition and the dot product are only defined for vectors of the same dimension. However, the focus of this exercise is on higher-order functions so you can ignore checking for this and assume that the input vectors always have the same dimension.

4. (Challenging) A matrix can be represented as a list of rows, where rows are represented as vectors of same length.

$$\textbf{type}\ Matrix = [Vector]$$

The following matrix

$$\begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix}$$

is thus represented by the following value.

$$m :: Matrix$$
$$m = [[1, 2], [3, 4]]$$

Implement the following operations using higher-order functions.

(a) Matrix-vector multiplication.

$$multMatVec :: Matrix \to Vector \to Vector$$

(b) Adding a column on the left-side of a matrix.

$$addColumn :: Vector \to Matrix \to Matrix$$

(c) Transposing a matrix.

$$transpose :: Matrix \to Matrix$$

(d) Matrix-matrix multiplication.

$$multMatMat :: Matrix \to Matrix \to Matrix$$

List comprehensions with and without the sugar

5. Rewrite the following functions as list comprehensions

(a) *map*,
(b) *filter*,
(c) *concat*.

6. Rewrite the following functions using *concat*, *map* and *filter*

(a) $lc1 :: (a \to b) \to (a \to Bool) \to [a] \to [b]$
$lc1\ f\ p\ as = [f\ a \mid a \leftarrow as, p\ a]$

(b) $lc2 :: [a] \to (a \to [b]) \to (b \to Bool) \to [(a, b)]$
$lc2\ as\ bf\ p = [(a, b) \mid a \leftarrow as, b \leftarrow bf\ a, p\ b]$

(c) $lc3 :: Int \to [(Int, Int, Int)]$

$$\begin{aligned} lc3\ n = [(a, b, c) \mid\ & a \leftarrow [1 \mathinner{..} n], \\ & b \leftarrow [a \mathinner{..} n], even\ a, \\ & c \leftarrow [b \mathinner{..} n], a * a + b * b \equiv c * c] \end{aligned}$$

Type classes

The following code defines datatypes for representing structured (X)HTML markup.

```haskell
data Attr = Attr String String
  deriving (Eq, Show)
data HtmlElement
   = HtmlString String
   | HtmlTag String [Attr] HtmlElements
  deriving (Eq, Show)
type HtmlElements = [HtmlElement]
```

A piece of HTML code is either plain text *HtmlString* or is a tagged node *HtmlTag* with attributes. In case of a node, other elements can be nested under it. The following HTML code

```html
<a href="http://www.ugent.be/">
 Universiteit Gent
</a>
```

is represented by the following value

```haskell
example :: HtmlElement
example =
  HtmlTag "a" [Attr "href" "http://www.ugent.be/"]
    [HtmlString "Universiteit Gent"]
```

We can group all types that can be rendered as HTML in a type class.

```haskell
class HTML a where
  toHtml :: a -> HtmlElement
```

The above code is available in Minerva so you do not have to type it yourself.

7. Write an *HTML* instance that creates an anchor for the following datatype

```haskell
data Link = Link
              String   -- Link target.
              String   -- Text to show.
```

8. Write an *HTML* instance for *Either*.

9. Write an *HTML* instance for Haskell lists using unordered HTML lists. The following code is an example of an unordered HTML list.

```html
<ul>
<li>Appels</li>
<li>Bananas</li>
<li>Oranges</li>
</ul>
```

10. Model datatypes for an address book. You should store at least the following information about your contacts

- First and last name.
- A list of email addresses.
- For each email address you should store if it is a work or private email address.

Define an example address book with at least two entries.

11. Define *HTML* instances for the types of your address book.

12. (Optional) Define a function *renderElement* :: *HtmlElement* → *String* that converts the structured markup representation into the concrete plain text representation of the HTML code. Now you can write the HTML code of your address book into a file and view it in a browser. For this you can call the function *writeFile* :: *FilePath* → *String* → *IO* () in GHCi like this:

```haskell
Prelude> writeFile "AddressBook.html" (renderElement (toHtml myAddressBook))
```

More on I/O will be covered later in the course.