FUNCTIONAL PROGRAMMING MT2020

Sheet 4

6.7 A variation of insertion sort uses a carefully ordered tree to keep the partially sorted values. In this question a *binary search tree* is an element of

```
> data Tree a = Fork (Tree a) a (Tree a) | Empty
```

which is ordered so that all values that appear in the left tree of a fork are smaller than the value at the fork, and all values that appear in forks in the right subtree are bigger.

In order to deal with repetitions in this question use a $\mathit{Tree}\ [a]$ to contain the elements of a list of type $[a]$.

Write a function $\mathit{insert} :: \mathit{Ord}\ a \Rightarrow a \rightarrow \mathit{Tree}\ [a] \rightarrow \mathit{Tree}\ [a]$ which inserts a value into a tree, keeping its ordering property.

Write a function $\mathit{flatten} :: \mathit{Tree}\ [a] \rightarrow [a]$ which takes a binary search tree and produces a list of its elements in order.

Use these to write

```
> bsort :: Ord a => [a] -> [a]
> bsort = flatten . foldr insert Empty
```

7.1 Express the Cartesian product function

```
> cp :: [[a]] -> [[a]]
> cp []       = [[]]
> cp (xs:xss) = [ x:ys | x <- xs, ys <- cp xss ]
```

from the lectures as an instance of *fold* (or the standard function *foldr*).

7.2 The function

```
> cols :: [[a]] -> [[a]]
> cols    [xs]   = [ [x] | x <- xs ]
> cols (xs:xss) = zipWith (:) xs (cols xss)
```

is not quite in the form of a fold (on lists) because there is a special case for singelton lists. Define a function cols' which agrees with cols wherever that function is defined, and for which $\mathit{cols}'\ []$ has a value that makes the equation for $\mathit{cols}\ [xs]$ redundant. This should give the definition of cols' the form of a fold. Finally, write cols' as an instance of *fold*.

8.1 Aligning text in columns involves *justification*: perhaps to the right or left. One way of doing involves padding strings to a given length:

```
*Main> rjustify 10 "word"
"      word"
*Main> ljustify 10 "word"
"word      "
```

Define functions

```
> rjustify :: Int -> String -> String
> ljustify :: Int -> String -> String
```

to do this. What do your functions do if the string is wider than the target length? Is that what you would want, and if not how would you do it differently?

8.2 Suppose we represent an $n \times m$ matrix by a list of n rows, each of which is a list of m elements. These matrices will be elements of

```
> type Matrix a = [[a]]
```

that are, additionally to what the type says, rectangular and non-empty. Without writing any new recursive definitions:

1. define $\mathit{scale} :: \mathit{Num}\ a \Rightarrow a \rightarrow \mathit{Matrix}\ a \rightarrow \mathit{Matrix}\ a$ which multiplies each element of a matrix by a scalar (the qualification $\mathit{Num}\ a$ in the type means that *scale* can use arithmetic on values of type a);
2. define a function $\mathit{dot} :: \mathit{Num}\ a \Rightarrow [a] \rightarrow [a] \rightarrow a$ which calculates the dot-product of two vectors of the same length;
3. define $\mathit{add} :: \mathit{Num}\ a \Rightarrow \mathit{Matrix}\ a \rightarrow \mathit{Matrix}\ a \rightarrow \mathit{Matrix}\ a$ which adds to matrices (of the same size as each other);
4. define $\mathit{mul} :: \mathit{Num}\ a \Rightarrow \mathit{Matrix}\ a \rightarrow \mathit{Matrix}\ a \rightarrow \mathit{Matrix}\ a$ which multiplies matrices of appropriately matching sizes;
5. define $\mathit{table} :: \mathit{Show}\ a \Rightarrow \mathit{Matrix}\ a \rightarrow \mathit{String}$ that translates a matrix (of printable elements) into a string that can be printed to show the matrix with each element right-justified in a column just wide enough to contain each of its elements. You may want to use the predefined *unwords* and *unlines*.

```
*Main> putStr (table [[1,-500,-4], [100,15043,6], [5,3,10]])
  1  -500 -4
100 15043  6
  5     3 10
```

Geraint Jones, 2020