FP 2009-2010, Eindtoets
2010, Mar 17 , 14.00-17.00, EDUC Gamma

Hand in the separate sheet with the 5 solutions. Don't forget to fill out your name! Do not forget necessary parentheses! Each question has a value of 2 points (total 10).

1. **Induction Proof**
   Prove by induction that $\mathit{foldr}\ f\ e\ (\mathit{reverse}\ xs) = \mathit{foldl}\ (\mathit{flip}\ f)\ e\ xs$.

2. **Huffman Trees**

   (a) Give the data type definition for a Huffman tree, i.e. $\mathit{Huff}\ a$ which encodes for values of some type a.

   (b) Write a function $\mathit{mkHuff} :: \mathit{Ord}\ a \Rightarrow [(a, \mathit{Weight})] \rightarrow \mathit{Huff}\ a$ which constructs the Huffman tree that is used in the encoding for the argument of mkHuff. Note that **type** $\mathit{Weight} = \mathit{Int}$.

3. **Generalised Trees** were defined as:

   $$\textbf{data}\ \mathit{GTree}\ f\ a = \mathit{GLeaf} \mid \mathit{GNode}\ a\ (f\ (\mathit{GTree}\ f\ a))$$

   If we ask Hoogle what it knows about $f\ \ b \rightarrow [b]$ it tells us something about the module $\mathit{Data.Foldable}$:

   $$\begin{array}{l} \textbf{class}\ \mathit{Foldable}\ t\ \textbf{where} \\ \quad \ldots \quad \text{-- left out and not important for this exercise} \\ \quad \mathit{foldr} :: (a \rightarrow b \rightarrow b) \rightarrow b \rightarrow t\ a \rightarrow b \quad \text{-- generalise the function foldr} \\ \mathit{toList} :: \mathit{Foldable}\ t \Rightarrow t\ b \rightarrow [b] \quad \text{-- Lists the elements of a structure.} \end{array}$$

   (a) Define the type of a function $\mathit{enumPrefix}$ which flattens a generalised tree (i.e. produces a list of all the a values occurring in the generalised tree instance) in such a way that a value in a node precedes all the node values of the children of that node (hence we return the prefix order)

   (b) Give the definition for **instance** $\mathit{Foldable}\ [\,]$ and define the function toList.

   (c) Give an (efficient) definition of $\mathit{enumPrefix}$, using an accumulating parameter or equivalent approach.

4. **Enumerating Trees** In the context of the following definitions:

   $$\begin{array}{l} \textbf{data}\ \mathit{Tree} = \mathit{Leaf} \mid \mathit{Bin}\ \mathit{Tree}\ \mathit{Tree} \\ h\ \mathit{Leaf} = 0 \\ h\ (\mathit{Bin}\ l\ r) = 1 + \mathit{max}\ (h\ l)\ (h\ r) \end{array}$$

   we call $h\ t$ the height of a tree t.

   (a) Assuming that a function $ex :: \mathit{Int} \rightarrow [\mathit{Tree}]$ exists which returns all trees with height exactly equal to the parameter, write a function $lt :: \mathit{Int} \rightarrow [\mathit{Tree}]$ that returns all trees with height less than its parameter.

   (b) Use the function lt to define the function ex.

   (c) If you have not done so yet write an efficient version of the function ex which has costs which are linear in the size of the result (hint: tuple the computations of lt and ex).

5. **Graphs**
   We may represent labelled graphs for which all nodes are reachable from a designated root node by the data type:

   $$\textbf{data}\ \mathit{Graph}\ \mathit{label} = \mathit{Graph}\ \mathit{label}\ [\mathit{Graph}\ \mathit{label}]$$

(a) Write a function $build :: Eq\ a \Rightarrow [(a, [a])] \rightarrow a \rightarrow Graph\ a$, which converts a description of a graph to a $Graph\ a$. The first element of each pair (in the first argument) is the label of a node, and the associated list contains the labels of the nodes at the other end of the outgoing edges of that node. The second parameter is the designated root node by which we can refer to the graph. Hint: use lazy evaluation and build a table which for each *label* contains its *Graph label*. You may assume that the parameters to the function *build* actually describe a proper structure, and especially that all nodes are actually occurring in the description. You may complete the following code fragment:

```
build nodes root = locate root
  where allgraphs = ...
        locate n  = ...
```

(b) Write a function $isCircular :: Eq\ a \Rightarrow Graph\ a \rightarrow Bool$ which checks whether the argument graph contains (reachable) cycles. Hint: maintain a list of visited nodes during the traversal of the graph.