Programming Paradigms 2025
Session 12: Catching up once more

Problems

Hans Hüttel

25 November 2025

1 Problems that we will definitely talk about

1. The goal of this problem is to define a function frequencies that, given a string s, creates a list of pairs [(x1,f1),....(xk,fk)] such that if the character xi occurs a total number of fi times throughout the list s, then the list of pairs will contain the pair (xi,fi).

   As an example of this,

   ```
   frequencies "regninger"
   ```

   should return the list

   ```
   [('r',2),('e',2),('g',2),('n',2),('i',1)]
   ```

   a) What should the type of the function be?

   b) Use *recursion* to give a definition of frequencies.

2. Here is a type declaration for simple expressions.

   ```
   data Exp a = Var a | Val Integer | Add (Exp a) (Exp a) | Mult (Exp a) (Exp a) deriving
   Show
   ```

   Show how do make this type into an instance of Functor.

   When would it be useful to think of Exp a as a functor? Think of a good example!

3. Consider trees whose elements are values of some type in the type class Ord. The type Tree a is defined by

   ```
   data Tree a = Leaf a | Node (Tree a) (Tree a)
   ```

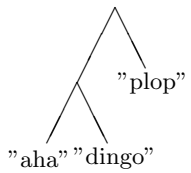

Figure 1: The ordered mytree1

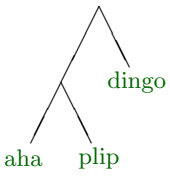

Figure 2: The unordered mytree2

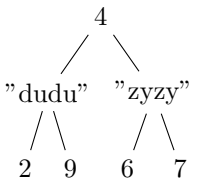

Figure 3: A mixed Int, String-tree that is layered

Use a monad to write a function minorder that takes such a tree and checks if the numbers in the structure are in non-decreasing order when read from left to right. If it is, the function should return Just k, where k is the smallest number in the tree, otherwise it should return Nothing. The tree mytree shown in Figure 1 is ordered, so minorder mytree1 should return Just "aha", but the tree in Figure 2 is not ordered, so minorder mytree2 should return Nothing.

First define another function minmax that finds the minimal and the maximal element in a tree under the assumption that the tree is ordered. Then use minmax to define minorder.

Hints: First, find some good test cases. Then find out which monad you should use.

Warning! Only use monadic notation!

Additional problems

1. Define a foldM function whose type should be

```
foldM :: Monad m => (t1 -> t2 -> m t2) -> [t1] -> t2 -> m t2
```

The idea is that the function works like foldl but folds over a monad.

Here is an example that shows what will happen if we fold over the IO monad. If we let

```
dingo x = do
          putStrLn (show x)
          return x
```

then we should see the following behaviour.

```
*Main> foldM (\x y -> (dingo (x+y))) [1,2,3,4] 0
1
3
6
10
```

2. This goal of this problem is to define a data structure and a function over it.

   We consider a data structure that we call mixed a, b-trees. Let a and b be any types. Then we say that a tree is a mixed a, b-tree if it is has one of these forms given below.

   - It can be an empty leaf
   - It can be a leaf labelled with an element of type a
   - It can be a leaf labelled with an element of type b
   - It can consist of a node labelled with an element of type a and a left and right subtree that are also mixed a, b-trees.
   - It can consist of a node labelled with an element of type b and a left and right subtree that are also mixed a, b-trees.

   Figures 3 and 4 show two mixed Int, String-trees; empty leaves are not shown.

   a) Define a datatype MixedTree for mixed a, b-trees in Haskell.
   b) Represent the tree in Figure 3 as a term of type MixedTree.

   We say that a mixed a, b-tree t is *layered* if the following holds:

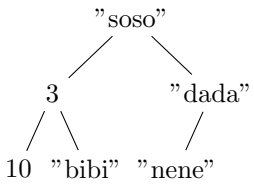

Figure 4: A mixed Int, String-tree that is not layered

- Every node in t labelled with an element of type a has all its children labelled with elements of type b.
- Every node in t labelled with an element of type b has all its children labelled with elements of type a.

The tree in Figure 3 is layered, the tree in Figure 4 is not.

Define a function islayered which takes a tree as argument and tells us if the tree is layered.