Programming Paradigms
Written exam (test case)

Aalborg University

November 2022

You must read the following before you begin

1. This problem set is part of a zip archive together with a file `solutions.hs`.
2. Please add your full name, AAU mail address and study number to the top of `solutions.hs` where indicated in the file.
3. Please write your answers by adding them to `solution.hs`. You must indicate in comments which problem your text concerns and which sub-problem it concerns. All text that is not runnable Haskell code must be commented out.

   Use the format as exemplified in the snippet shown below.

   ```
   -- Problem 2.2

   bingo = 17

   -- The solution is to declare the value bingo with the value 17.
   ```

4. Submit the resulting version of `solutions.hs` and nothing else to Digital Eksamen when you are done.
5. During the exam, you are allowed to use the textbook *Programming in Haskell* by Graham Hutton, your own notes and your installation of the Haskell programming environment.
6. You are only allowed to use the Haskell `Prelude` for your solutions, unless the text of a specific problem mentions that you should also use another specific module. No special GHCi directives are to be used.
7. Please read the text of each problem carefully before trying to solve it.

Problem 1 – 18 points

Somebody wrote the function allAnswers given below.

The intention is to apply a function to every element of a list and return a list of the function values obtained. However, if one of the values is nothing, then the result must be Nothing.

```
allAnswers f [] = Just []
allAnswers f (x:xs) = let fun = f x
                      in if (fun == Nothing) then Nothing
                      else x : (allAnswers f xs)
                           where x=x
```

1. Explain why this piece of code does not work. (There may be more than one mistake in it!)
2. Give a new definition of allAnswers that uses higher-order functions (such as all, map etc.) and works as intended.
3. Give another new definition of allAnswers that uses the Maybe monad and works as intended.

Problem 2 – 14 points

1. Here are four types. For each of them, define a Haskell value (which may be a function) that has this particular type.

   a) `Eq a1 => a2 -> (a1, a1) -> [a2]`

   b) `Eq a => (a -> a -> Bool) -> Bool -> a -> a -> Bool`

   c) `Show a => a -> IO b -> IO b`

   d) `(a -> b) -> a -> a -> [b]`

2. Which of these four values involve

   - parametric polymorphism only
   - overloading (ad hoc-polymorphism) only
   - both forms of polymorphism

   ?

Problem 3 – 16 points

A binary leaf-labelled a-tree is a binary tree whose leaves are labelled with elements that have type a.

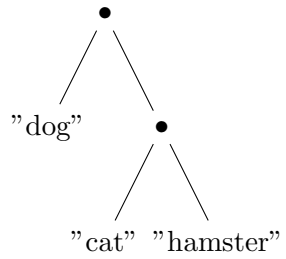

Figure 1: A binary leaf-labelled String-tree

1. Define a datatype Tree a for binary leaf-labelled a-trees and show how the tree of Figure 1 is represented as a term of this datatype.

2. Define a function minimax

   ```
   minimax :: Ord a => Tree a -> (a, a)
   ```

   that finds the minimal and maximal elements of a binary leaf-labelled tree. In the tree shown in Figure 1, the minimal element is "cat" and the maximal element is "hamster" – according to the lexicographic ordering on strings.

Problem 4 – 14 points

1. Convert the following do-block into an equivalent expression that does not use the do-notation but uses monadic binds.

```
echo = do
  putStr "Please type a word: "
  s <- getLine
  putStrLn ("You typed " ++ s)
```

2. Write a do block that defines a value

```
seconds = ....
```

that

- from the console gets an input string from the user that is a string corresponding to a list of pairs of truth values
- outputs a string to the console that is a list of the second components of these pairs

As an example, we expect to see the following behaviour:

```
*Main> seconds
[(True,False),(False,True),(False,False)]
[False,True,False]
*Main>
```

Problem 5 – 18 points

An alternating (t_1, t_2)-list is a list in which every other element has type t_1 and all other elements have type t_2.

The list

[5,True,6,False,7,True]

is a (Integer,Bool)-list. However, such lists are not allowed in Haskell so this list cannot be written using normal Haskell syntax for lists.

1. Explain why this is not allowed in Haskell.

2. Define a datatype Alternating (that has a suitable number of parameters) that defines terms that correspond to alternating lists. Show how you represent the list shown in the above using your datatype as a value that we shall call myalt.

3. Using your definition of the Alternating datatype, define a function separate that separates an alternating list into two ordinary lists. If we apply separate to a representation of the above, we expect the following:

```
*Main> separate myalt
([5,6,7],[True,False,True])
*Main>
```

4. Using your definition of the Alternating datatype, give a definition of the infinite alternating list that consists of each natural number from 1 upwards followed by a string of the same number of a's. That is, define what corresponds to

   [1, "a", 2, "aa", 3, "aaa", 4, "aaaa"....]

Problem 6 – 18 points

Let us consider the type constructor ToPairs defined by

```
newtype ToPairs a = TP (a,a)
```

1. Give example of values that have types

   ```
   ToPairs Bool
   ```

   and

   ```
   ToPairs ((Maybe Int) -> Int)
   ```

2. Show how to define ToPairs to be an instance of Functor.
3. Show how to define ToPairs to be an instance of Applicative.