Programming Paradigms
Written exam

Aalborg University

8 March 2024
12:30 - 15:30

YOU MUST READ THE FOLLOWING VERY CAREFULLY BEFORE YOU BEGIN!!!

What is this? This problem set consists of 6 problems. The problem set is part of a zip archive together with a file called `solutions.hs`.

How do I begin? *Please do the following immediately!!* Unzip the zip archive – it is not enough to look within it; you *must* unzip it and save the problem set and `solutions.hs` locally on your computer. Otherwise, the answers that you write in `solutions.hs` will not be saved.

Please add your full name, AAU mail address and study number to the top of your local copy of `solutions.hs` saved to your computer where indicated in the file. **DO IT NOW.** If you experience problems with the file extension `.hs`, then rename the file while working and give it the file extension `.hs`, just before you submit it.

Må jeg skrive på dansk? Ja, det må du gerne. You can write your answers in Danish or in English.

How and where should I write my solution? Please write your answers by adding them to the local copy of `solutions.hs` on your computer. You must indicate in comments which problem your text concerns and which subproblem it concerns. All other text that is not runnable Haskell code (such as code that contains syntax errors or type errors) must also be written as comments.

Use the format as exemplified in the snippet shown below.

```
-- Problem 2.2

bingo = 17

-- The solution is to declare a variable called bingo with value 17.
```

Your file must be able to compile. Please ensure that this is the case before you submit your solution. Comment out the parts yourself that cause the file not to compile.

How should I submit my solution? *Submit the local copy of `solutions.hs` that your solutions appear in and nothing else.* **DO NOT SUBMIT A ZIP ARCHIVE.**

What can I consult during the exam? During the exam, you are only allowed to use the textbook *Programming in Haskell* by Graham Hutton, your own notes and your installation of the Haskell programming environment. You are only allowed to use another book or slides if you wrote the book or the slides yourself. GitHub CoPilot, ChatGPT and other AI-based tools are not allowed.

What can I use for my code? You are only allowed to use the Haskell `Prelude` for your Haskell code, unless the text of a specific problem specifically mentions something else. Do not use any special GHCi directives.

Is there anything else I must know? Yes. Please read the text of each problem ***very carefully*** before trying to solve it. **READ IT MORE THAN ONCE.** All the information you will need is in the problem text. Please make sure that you understand what is being asked of you.. **PLEASE TEST YOUR SOLUTIONS TO SEE IF THEY WORK AS INTENDED.** Read the problem text again before you test.

Problem 1 – 16 points

1. A list $[a_1, a_2, \ldots, a_n]$ is *descending* if $a_1 \geq a_2 \geq \ldots \geq a_n$.

   - Write a function `descending` that will return `True` if a list is descending and `False` otherwise. As examples, `descending [6,5,5,1]` should return `True` and `descending ["plip","pli","ppp"]` should return `False`.
   - Is `descending` polymorphic? If yes, is it parametric polymorphic, ad hoc-polymorphic or both?

2. A *segment* of a list xs is a sublist of xs such that all its elements are equal and the neighbouring elements are different from the elements in the sublist.

   As an example, `[2,2,2]` is a segment of `[1,1,2,2,2,3,6]` but `[2,2]` is not a segment of `[1,1,2,2,2,3,6]`, since `2` is a neighbouring number in the list.

   - Define a recursive function `segments` that computes the list of segments of a given list. As an example, `segments [1,1,2,2,2,5,6]` should return `[[1,1],[2,2,2],[5],[6]]` and `segments [True,False,False,Tru` should return `[[True],[False,False],[True,True,True]]`.
   - Is `segments` polymorphic? If yes, is it parametric polymorphic, ad hoc-polymorphic or both?

Problem 2 – 20 points

The authors of a new encyclopedia about values of type `a` represent their information using a hierarchical structure where every entry in the encyclopedia is tagged with a key that is a string and a value of the appropriate type.

Figure 1 shows two encyclopedias.

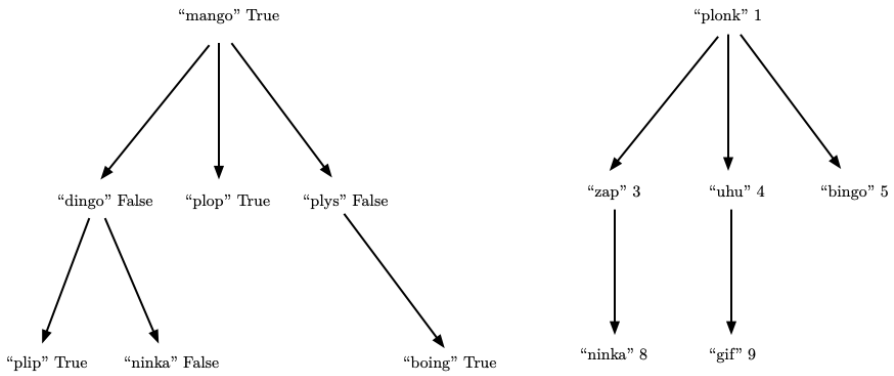

Figure 1: Two encyclopedias t_1 (left) and t_2 (right)

1. Define a type `Encyclopedia` for encyclopedias. Show how you represent t_1 and t_2 in Figure 1 as values `t1` and `t2` of your type.

2. An encyclopedia has a search function `containskey` that will tell us if an element with a given key is found in the encyclopedia. As an example, `containskey "ninka" t1` should return `True` and `containskey "bob" t2` should return `False`.

   Define `containskey`.

3. An encyclopedia is *layered* if it holds that all values at the same level of the encyclopedia are larger than the values in the levels above. As an example, t_2 is layered, since 8 and 9 at level 3 are greater than the values 3, 4 and 5 at level 2 – which are greater than the value 1 at level 1.

   Define a function `layered` that can tell us if an encyclopedia is layered.

Problem 3 – 14 points

The `interleave` function takes two lists xs and ys and builds a new list in which every element of xs appears immediately before the element at the same position in ys. If the two lists do not have the same length, the shortest list has priority.

As examples, `interleave [1,2,3,17] [4,5,6]` should return `[1,4,2,5,3,6]`, and `interleave [1,2] [4,5,6]` should return `[1,4,2,5]`.

1. Define `interleave` using recursion.
2. Define `interleave` using `map` and `zip`.

Problem 4 – 18 points

Here are the formation rules for the set of plus-expressions.

$$e ::= x \mid n \mid e_1 + e_2$$

where x represents a variable which is a string and n represents an integer numeral.

1. Define a type `PExp` for plus-expressions.
2. A state is a function that maps variables to integer values. Not all variables need to have a value assigned to them. An example of a state is s_0 where $s_0\ x \mapsto 4$ and $s_0\ y \mapsto 5$ and the value of all other variables is undefined.

   Define a type `State` that is the type of states.
3. The valuation `eval` is a function that takes a plus-expression e and a state s and returns the value of e if the value of the plus-expression is defined. As an example, the value of $x + y$ in the state s_0 defined above is 9, whereas the value of $4 + z$ in the state s_0 is not defined, since z is not in the domain of s_0.

   Use the `Maybe` monad to define the function `eval`.

Problem 5 – 16 points

Here are four types. For each of them find a declaration or expression that has this type as its most general type. In other words, no type annotations are to be used. For each type, explain if there is polymorphism involved, if there is, if this is parametric or ad hoc-polymorphism.

1. `Num b => a -> b -> ([a], b)`
2. `(t1 -> t2 -> t1 -> t3) -> t1 -> t2 -> t3`
3. `Maybe [p -> Bool]`
4. `(t1 -> a) -> t1 -> (t2 -> a) -> t2 -> Int`

Problem 6 – 16 points

The *harmonic sequence* is the infinite sequence

$$1, \frac{1}{2}, \frac{1}{3}, \frac{1}{4}, \frac{1}{5}, \frac{1}{6} \cdots$$

1. Define the harmonic sequence using recursion.
2. Define the harmonic sequence using `map`
3. A partial sum to level n of the harmonic sequence is the sum

$$\sum_{i=1}^{n} \frac{1}{i}$$

   Give two different ways of defining the function `psum`, one using recursion and one not using recursion, that will both, given an n, compute the partial sum to level n of the harmonic sequence. As an example, both implementations of `psum 5` should return `2.283333333333333`.