Programming Paradigms 2023
Session 6: Declaring types and type classes

Problems for solving and discussing

Hans Hüttel

31 October 2023

How you should approach this

- Some problems are meant to be solved in groups. Please work together at the table for those. Screen sharing at your table is best.
- Other problems are meant to be solved in pairs. Please work together two by two, not in groups. Screen sharing at your table is not a good idea for these problems.
- We set a limit on how much time you are going to spend on each of the numbered problems. *If you do not manage to solve a problem* within the time set aside, do not worry about that. We will discuss solutions afterwards. If you finish early, work on one of the "lettered problems" in the problem set instead of working on the next numbered problem.

Problems that we will definitely talk about

1. ***(Everyone at the table together – 15 minutes)***

   Define a Haskell datatype Aexp for arithmetic expressions with addition, multiplication, numerals and variables. The formation rules are

   $$E ::= n \mid x \mid E_1 + E_2 \mid E_1 \cdot E_2$$

   Assume that variables x are strings and that numerals n are integers.

2. ***(Work in pairs– 20 minutes)***

   Use your Haskell datatype from the previous problem to define a function eval that can, when given a term of type Aexp and an assignment ass of variables to numbers compute the value of the expression. *Hint:* Use association lists as described on page 93 to represent assignments.

   As an example, if we have the assignment $[x \mapsto 3, y \mapsto 4]$, eval should tell us that the value of $2 \cdot x + y$ is 10.

3. ***(Everyone at the table together – 15 minutes)***

   A Unix directory contains other directories and also files. Every directory has a name and a finite list of directories, which are the subdirectories (there may be no subdirectories at all). Every file has a name, which is a string, and a size, which is a whole number.

   A famous YouTuber was asked to define an algebraic datatype Dir that describes this and came up with the following.

   ```
   data Dir a String b Int = Empty Null | Mult Dir Dir | Subdir Dir
   ```

   The YouTuber remarked that declarations of data types in Haskell do not allow one to specify that a directory could have any number of subdirectories, so one should therefore assume that there were always two.

   Unfortunately there were problems with the solution. Find out what is wrong and come up with a better solution. It is a good idea to read the problem text that describes Unix directories very carefully – and once you have criticized the existing solution, it is also a good idea *not to try to repair* what the YouTuber proposed but to start from scratch.

4. ***(Everyone at the table together – 15 minutes)***

   On page 98, the book describes search trees; make sure that you understand what important property a search tree has.

   Assume that our type for trees is defined as

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

   This means that trees can now also be empty. Define a function

   ```
   insert :: Ord a => Tree a -> a -> Tree a
   ```

   such that whenever t is a search tree, then insert t x gives us a new search tree that now also contains x.

5. ***(Work in pairs– 20 minutes)***

   From linear algebra we know that a vector space with inner product is one for which the operations of vector sum and dot product are defined. Given two vectors v_1 and v_2, the sum $v_1 + v_2$ is again a vector, and the inner product $v_1 \cdot v_2$ is a number. *Please note:* In linear algebra there is much more to the definition of inner product spaces than these two requirements, but in this problem, please ignore that. Also assume that the inner product is a number of type Int.

   Define a typeclass InVector whose instances are types that can be seen as inner product spaces, where vector sum is called &&& and inner product is called ***. *Hint:* Which section in the text for today do you need here?

   Find out how to declare Bool as an instance of InVector. *Hint:* You have to find a definition of vector sum and inner product for truth values.

More problems to solve at your own pace

In your solutions, remember the learning goals of this session!

a) We say that a binary tree is *balanced* if the number of leaves in every left and right subtree differ by at most one with leaves themselves being trivially balanced. Define a function balanced that will tell us if a binary tree is balanced or not. *Hint:* It is a good idea to also define a function that finds the number of leaves of a tree.

b) This problem refers to Section 8.6 in the book and the types and functions defined there.

   Two Boolean propositions p and q are *equivalent* if they are true for the same substitutions, that is, for every substitution s we have that p is true under s if and only if q is true under s. Define a function equiv where

   ```
   equiv :: Prop -> Prop -> Bool
   ```

   and such that equiv p q returns True if p and q are equivalent and False otherwise.

c) On page 106 there is a definition of the Expr datatype for expressions. Define a higher-order function

   ```
   foldexp :: (Int -> a) -> (a -> a -> a) -> Expr -> a
   ```

   such that foldexp f g replaces each Val constructor in an expression by the function f, and each Add constructor by the function g.

   Then use foldexp f g (with appropriate choices for f and g) to define a function eval :: Expr -> Int that evaluates an expression to an integer value.

d) Complete the following instance declaration:

   ```
   instance Eq a => Eq (Maybe a) where
   ...
   ```

e) Use insert as you have defined it in problem 4 to define a function build that will build a search tree from a given list.

   Use build to define a function sort that can sort any given list.