Lund University
Department of Computer Science

EDAN40: Functional Programming
1st June 2023, 8–13

Exam

1. **Point-free notation**

   Rewrite the following two definitions into a point-free form (i.e., `f = ...`, `g = ...`), using neither lambda-expressions nor list comprehensions nor enumeration nor `where` clause nor `let` clause:

   ```
   f x y = 5 / (x + y)
   g x y = [y z | z <- [x..]]
   ```

2. **Type derivation**

   Find the types of the following expressions:

   ```
   ($ ($))
   (. (.))
   (: (:))
   (== (==))
   (|| (||))
   ```

3. **Proving program properties**

   The `Functor` class is defined as follows:

   ```
   class Functor f where
     fmap :: (a -> b) -> f a -> f b
   ```

   It is mandatory that all instances of `Functor` should obey:

   ```
   fmap id      = id
   fmap (p . q) = (fmap p) . (fmap q)
   ```

   Assume the following definition of `Maybe` types as a functor instance:

   ```
   instance Functor Maybe where
       fmap f (Just x) = Nothing
       fmap f Nothing = Nothing
   ```

   Is this a correct definition of a functor instance? Why or why not? **Prove your claim**.

4. **Function composition** Below you will find a list of seven equations: at least one of them is false. Which are the true ones and which are false?

   (a) `map f . take n = take n . map f`

   (b) `map f . reverse = reverse . map f`

   (c) `map f . sort = sort . map f`

   (d) `reverse . concat = concat . reverse . map reverse`

   (e) `filter p . concat = concat . map (filter p)`

   (f) `filter (p . g) = map (invertg) . filter p . map g`
   where `invertg` is defined in such way that
   `invertg . g = id`

   (g) `map f . filter p = map fst . filter snd . map (fork (f,p))`
   where

   ```
   fork :: (a -> b, a -> c) -> a -> (b, c)
   fork (f, g) x = (f x, g x)
   ```

5. **Monadic computations**

   What is the type and value of the following expression?

   ```
   do "edan40"; [1, 10, 100]
   ```

6. **Language**

   What does it mean that all functions in HASKELL are *curried*?

Good Luck!