Lund University
Department of Computer Science

EDAN40: Functional Programming
25th April 2022, 14–19

Exam

1. **Type derivation** (1p)

   (a) Assume that the type of `reduce` is

   ```
   reduce :: a -> a
   ```

   Find the type of

   ```
   prepare = reduce . words . map toLower . filter
                                            (not . flip
                                                   elem ".,:;*!#%&|")
   ```

   (b) Given that

   ```
   map2 :: (a -> b, c -> d) -> (a, c) -> (b, d)
   ```

   find the destination type `b` of the following function:

   ```
   rulesCompile :: [(String, [String])] -> b
   rulesCompile = (map . map2) (words . map toLower, map words)
   ```

   (c) Given that

   ```
   transformationApply :: Eq a => a -> ([a] -> [a]) -> [a] -> ([a], [a])
                                                             -> Maybe [a]
   orElse :: Maybe a -> Maybe a -> Maybe a
   ```

   find the type of

   ```
   foldr1 orElse (map (transformationApply wildcard f x) pats)
   ```

2. **Proving program properties** (2p)

   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)
   ```

   Let `Either` be defined as follows:

   ```
   data Either a b = Left a | Right b
   ```

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

   ```
   instance Functor (Either a) where
       fmap f (Right x) = Right (f x)
       fmap f (Left x) = Left x
   ```

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

3. **Sparks** (1p)

   Explain what a *spark* is, and where does it occur in Haskell. What is it good for?

4. **Programming** (1p)

   Assume we are developing a library for image processing. We might then represent an image as a function from the unit square `[0,1]x[0,1]` to some color type `a`. Slightly generalized this may be expressed as:

   ```
   type Image a = Position -> a
   type Position = (Float, Float)
   ```

   We may now for example define:

   ```
   type Region = Image Boolean
   type ColourImage = Image Colour
   ```

   (a) Write a function

   ```
   paste :: Region -> Image a -> Image a -> Image a
   paste reg im1 im2
   ```

   which pastes `im1` into `im2` wherever `reg` is true. Pasting `a` into `b` replaces values of `b` by the corresponding values of `a`.

   (b) Implement the following functions which convert ordinary functions to functions on images:

   ```
   lift0 :: a -> Image a
   lift1 :: (a -> b) -> Image a -> Image b
   lift2 :: (a -> b -> c) -> Image a -> Image b -> Image c
   ```

   so that it, for example, is possible to express the difference between two images as:

   ```
   im1 `lift2 (-)` im2
   ```

   (c) Describe what you need to do in order to be able to write this difference as:

   ```
   im1 - im2
   ```

5. **Monadic computations** (1p)

   What is the type and value of `e` defined below? Motivate your answer.

```
e k = do
     x <- k
     Nothing
     return 42
```

Good Luck!