CPSC 521: midterm exam

Robin Cockett

November 9, 2010

This exam is worth 20%. Each question is marked out of 25 points making a total of 100 points available:

1. (a) (10 points) Give the description of the `Monad` class and provide as instances the list monad and the exception monad.

   (b) (10 points) Describe the translation from "do" syntax to core Haskell and translate the following function:

   ```
   image:: Eq b => (a -> b) -> [a] -> [b] -> [b]
   image f xs ys = do x <- xs
                      y <- ys
                      if y == f x then return y else []
   ```

   (c) (5 points) Rewrite this function using list comprehension: what does it do?

2. (a) (12 points) Demonstrate the leftmost outermost reduction strategy on the following λ-terms:

   i. $(\lambda zx.z(xz))(\lambda y.xy)z$
   ii. $(\lambda xy.x)(\lambda xy.x)((\lambda x.xx)(\lambda x.xx))$
   iii. $(\lambda xy.xy(xx))(\lambda x.y)(\lambda x.xx)(\lambda x.xx)$

   What are the advantages and disadvantages of this reduction strategy? What is lazy reduction and in what regard is it a better reduction strategy?

   (b) (8 points) How do you represent lists in the λ-calculus? What are the λ-terms for the constructors and the associated map and fold function?

   (c) (5 points) Explain what it means to say that β-reduction is confluent. Explain why this means that $\mathsf{true} \neq \mathsf{false}$ (i.e. the λ-calculus is consistent).

3. (a) (6 points) Explain how "pairs" are represented in the lambda calculus. How are the two projection functions programmed?

   (b) (6 points) Explain what a fixed point combinator is. Prove that

$$\mathbf{Y} := (\lambda f x. f(x x f))(\lambda f x. f(x x f))$$

   is a fixed point combinator.

   (c) (8 points) Explain how the recursive `gcd` function

```
gcd x y = if x<y then gcd x (y-x)
          else if x>y then gcd (x-y) y
               else x
```

   is programmed in the λ-calculus (assume the If combinator and arithmetic functions).

   (d) (5 points) Explain briefly why all computable functions can be programmed in the λ-calculus.

4. The aim is to write a function to determine the height and width of a tree with edges weighted by integers. The height is the maximum weighted length of a path from the root to a leaf. The width is the maximum weighted length of path between any two leaves.

   We shall hold the tree as `Rose Int`, where this is the following sort of rose tree:

```
data Rose a = Rose [(a,Rose a)]
```

   Here is an example of such a tree

```
Rose [(1,t1),(2,Rose [])] where
    t1 = Rose [(4,Rose [(2,Rose []),(3,Rose []),(4,Rose [])])
              ,(2,Rose [])
              ,(1,Rose [(2,Rose []),(1,Rose []),(3,Rose [])])
              ]
```

   (a) (3 marks) Draw the tree above and give its height and width.

   (b) (6 marks) Write a `foldRose` function for this data type. Its type should be:

```
foldRose:: ([(a,c)] -> c) -> (Rose a) -> c
```

   (c) (8 marks) Write a height function (do not forget the edge lengths!):

```
hgt: Rose Int -> Int
```

   (d) (8 marks) Use the fold to write a width function:

```
width: Rose Int -> Int
```

   Here are extensive hints!!

   The "node width" at a node of such a rose tree is the sum of the two largest heights of the subtrees below it (plus the edge lengths to reach those subtrees, of course) – if there are no subtrees this is zero, and if there is one it is that height (plus edge length).

   However, to be the "best-width" it must also exceed any width that any subtree has seen!

   The first step is to process a list of edge lengths and best-width, height pairs to produce a new best-width and height:

```
best_width_hgt: [(Int,(Int,Int))] -> (Int,Int)
```

   To do this one needs to `foldr` over this list to produce at each stage the best height and best width seen so far (starting at $(0,0)$ of course). To calculate the new best-width take the maximum of:

   i. The node-width produced by the current subtree (i.e. best height so far summed with the current edge length and subtree height);
   ii. The best-width of the current tree;
   iii. The best-width seen so far.

   Now use a rose-fold to produce the tree width.