CPSC 417: Fall Midterm Exam

Robin Cockett

October 27, 2005

This exam is worth 20% of the credit for this course.
It is a take home exam to be returned on Tuesday 1$^{\text{st}}$ of October.
There are 40 marks available.
There are three sections and the last section is worth more than the previous sections.
Answer as many of the questions as you can.

PART I: Haskell

The datatype of Rose trees is `data Rose a = Rs [(a,Rose a)]` The fold function for Rose trees is very simple (but does involve the map function for lists) it has type

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

and one can obtain the map function for Rose trees from it as

```
mapRose::(a -> b) -> (Rose a) -> (Rose b)
mapRose f = foldRose (\k -> Rs (map (\(a,x) -> (f a,x)) k))
```

1. (2 marks) Write the fold function for Rose trees.

2. (2 marks) Write a function which creates the Rose tree of tails for a list:

   ```
   tailRose [1,2,3] = Rs[(1,Rs[(2,Rs[(3,Rs[])]),(3,Rs[])])
                        ,(2,Rs[(3,Rs[])]),(3,Rs[])]
   ```

   Note here that the children of each element in the Rose tree are headed by the elements of the tail of the list after that element.

3. (2 marks) Write a function to determine the maximum of a list of positive numbers (where the maximum of an empty list is taken to be zero) and use this function and the fold over Rose trees to write a function which calculates the maximum height of a Rose tree.

4. (4 marks: more challenging!) There is an obvious function

   ```
   return:: a -> Rose a
   return a = Rs [(a,Rs[])]
   ```

   There is a notion of Rose tree substitution as well. It is defined as a map

   ```
   (>>=)::(Rose a) -> (a -> (Rose b)) -> Rose b
   ```

   which inserts the Rose tree corresponding to each `a`-element into the old tree. This is done by making the `b`-children of a node of the `a`-tree the list of possible values for `b` continuations obtained by taking all the (depth 1) children of the `b`-valued Rose trees associated with the `a`-values of the children of that node. The children of a `b`-node, on the other hand, are simply the children in the `b`-valued tree – while there are some! When they run out one uses the `b`-children of the `a`-node at which one is sitting.

   Implement Rose tree substitution and use it to define a monad on Rose trees: use the monad to implement a "filter" function on Rose trees which removes every subtree whose root value does not satisfy the given predicate.

PART II: Rewriting

1. (4 marks) Which of the following are true:

   (a) If a rewriting system is locally confluent but it is not terminating then it cannot be (globally) confluent.

   (b) If a rewriting system has elements which do not have a normal form it cannot be confluent.

   (c) In a rewriting system which is locally confluent if an element has a normal form it must be unique.

   (d) A rewriting system in which every object has a normal form must be terminating.

   (e) Rewriting systems which are modulo equations are always confluent.

   (f) If a rewriting system is confluent one can decide the "equality" of elements.

2. (6 marks) Consider the terms with one binary operation $+$ and a constant 0 subject to the rewriting rules:

$$\begin{aligned}(x+y)+z &\longrightarrow x+(y+z)\\ x+0 &\longrightarrow x\\ 0+x &\longrightarrow x\end{aligned}$$

   Prove that these rules form a terminating confluent system [hint: first invent a numerical measure (e.g. a weighted size) for the terms which always reduces across a rewrite, then check for local confluence by examining the critical pairs – situations in which the redexes overlap.]

PART III: λ-Calculus

1. (3 marks) Determine whether the following two λ-calculus terms are equal (with respect to α, β equality):

$$\lambda xy.((\lambda f.y(fyx))(\lambda xy.x)) \quad \text{and} \quad \lambda x.(\lambda z.(\lambda x.((\lambda xy.xy)xx)z))$$

   Explain what facts you are using in this determination.

2. (5 marks) In the λ-calculus how do we represent the following datatypes:

   (a) `data BNat = BZ|Bon BNat|Boff BNat`. Provide the definitions of `BZ`, `Bon`, `Boff`, and the fold.

   (b) `data List a = Nil | Cons a (List a)`. Provide the definition of `Nil`, `Cons`, the fold, the case, and the map.

   (c) `Rose a = Rs a (List(Rose a))`. Provide the definition of `Rs`, and of the fold (note this is a slightly different definition of Rose trees from that given in the first section).

3. (6 marks) What is meant by saying that $\mathbf{Y}$ is a fixed point combinator? Prove that Turing's combinator:

$$\mathbf{Y}' = (\lambda xy.y(xxy))(\lambda xy.y(xxy))$$

   is a fixed point combinator. Explain how to implement the recursive function `f n = n:(f (n+1))` using fixed points

4. (6 marks) Which of the following are true? Justify your answers.

   - Every λ-term is α, β-equivalent to a term in normal form.
   - There is only one fixed point combinator up to α, β-equivalence.
   - When reducing a λ-term to normal form using β-reduction one must be careful (normal order reduction) so that one never reaches a term from which infinitely many β-reduction are possible.
   - If a term has a normal form then, because the λ-calculus is confluent no matter what order you apply β-reductions you will reach the normal form.
   - If a λ-calculus term has a normal form there is a fixed equation in the size of the term which bounds the number of β-reductions necessary to reduce it to that normal form.
   - Two λ-terms are α, β-equivalent if and only if they both have β-normal forms which are α-equivalent.
   - There are some functions which can be computed by the λ-calculus which cannot be computed by a Turing Machine.