CPSC 417: Fall Midterm Exam

Robin Cockett

March 5, 2002

This exam is worth 30% of the credit for this course.
There are 40 marks available: answer as many of the questions below as you can.

1. (15 points) Given a binary search tree:

$$\texttt{data STree a} = \texttt{Tip}|\texttt{Node}(\texttt{STree a}, \texttt{a}, \texttt{STree a})$$

the fold function is defined as:

```
foldST:: c -> ((c,a,c) -> c) -> STree a -> c
foldST tip node Tip = tip
foldST tip node (Node(t1,a,t2))
        = node(foldST tip node t1,a,foldST tip node t2)
```

Recall that one can sum the nodes of a search tree of integers using this fold function which works from the "bottom up":

```
sumST :: STree Int -> Int
sumST t = foldST 0 (\(n,a,m) -> n+a+m) t
```

(a) Define a "zipper" function which takes in a pair of list of lists of the same type and produces such a list of lists:

$$\texttt{zipper} : [[\texttt{a}]]- > [[\texttt{a}]]- > [[\texttt{a}]]$$

The function must append the lists which are at the same index in the two (outer) lists. When one (outer) list is shorter than the other then the result must have length the same as the *longest* of the two lists. The value at of an index which only occurs in one list is just the value at that index in the longer list. Thus:

$$\texttt{zipper}\ \ [[1,2],[5]]\ \ [[3,4],[],[6,7]] = [[1,2,3,4],[5],[6,7]]$$

(b) Define a function "pather" which given a pair of lists of lists and an element produces a list of lists:

$$\texttt{pather} :: ([[\texttt{a}]], \texttt{a}, [[\texttt{a}]])- > [[\texttt{a}]].$$

The output list of lists should contain the singleton list of the element and all the inner lists with that element appended on the front:

```
pather ([[1,2],[1,3,4]],0,[[2,5],[2],[2,6,7]])
           = [[0,1,2],[0,1,3,4],[0],[0,2,5],[0,2],[0,2,6,7]]
```

(hint: use the `map` function for lists).

(c) The paths of binary search tree are defined by

- The search tree `tip` has no paths
- The search tree `node(t1,a,t2)` has paths `a:p` where `p` is either a path of `t1`, a path of `t2`, or `p = Nil`.

Write a program which returns all the paths of a binary search tree:

$$\texttt{pathST} : \texttt{STree a}-> [[\texttt{a}]]$$

Thus, for example,

```
pathST (Node(Node(Tip,1,Tip),2,Node(Node(Tip,3,Tip),4,Tip)))
      = [[2,1],[2],[2,4,3],[2,4]]
```

(Hint: the idea is that you should use the fold and a function defined above!)

(d) The n^{th} level of a binary search tree (which exist only if the tree has depth at least n) is the list of values in the nodes of the tree which are at depth n read from left to right across the tree. Write a program which returns the list of levels of a binary search tree:

$$\texttt{levelST} : \texttt{STree a}-> [[\texttt{a}]]$$

Thus, for example,

```
levelST (Node(Node(Tip,1,Tip),2,Node(Node(Tip,3,Tip),4,Tip)))
      = [[2],[1,4],[3]]
```

(Hint: think fold!)

2. (15 points)

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

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

Explain what fact you are using in this determination.

(b) What is meant by saying that $\mathbf{Y}$ is a fixed point combinator? Prove that

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

is a fixed point combinator.

(c) How can we represent the booleans `True` and `False`, and lists in the λ-calculus? Provide the definitions of `Nil` and `Cons`.

Use the fixed point combinator $\mathbf{Y}$ to define the infinite list

`[True,False,True,False,True,False,...]`

which alternates between `True` and `False`.

(d) Which of the following are true? Justify your answers.

- Every λ-term has a normal form.
- Every λ-term has a head-normal form.
- Every λ-term with a head normal form has a normal form.
- If a term has a normal form then no matter what order you apply β-reductions you will reach the normal form.
- If a λ-calculus term has a normal form there are at most exponential (in the size of the term) number of β-reductions necessary to reduce it to that normal form.

$$\frac{x : P \vdash x : P}{x : P, z : \Gamma \vdash x : P}\ \text{proj}$$

$$\frac{x : P, z : \Gamma \vdash t : Q}{z : \Gamma \vdash \lambda x.t : P \to Q}\ \text{abst}$$

$$\frac{z : \Gamma \vdash f : P \to Q \quad z : \Gamma \vdash t : P}{z : \Gamma \vdash (ft) : Q}\ \text{app}$$

Table 1: Annotate type system

$$\frac{x : P \vdash x : Q}{x : P, z : \Gamma \vdash x : Q}\ P = Q$$

$$\frac{x : P, z : \Gamma \vdash t : Q}{z : \Gamma \vdash \lambda x.t : R}\ R = P \to Q$$

$$\frac{z : \Gamma \vdash f : P \to Q \quad z : \Gamma \vdash t : P}{z : \Gamma \vdash (ft) : Q}\ \text{fresh } P$$

Table 2: Annotate type system with type equations

3. (10 points)

   (a) Given the annotated type system in table 1 give the lambda term which corresponds to the following proof:

$$\dfrac{\dfrac{\dfrac{\dfrac{R \to Q \vdash R \to Q}{P, P \to R, R \to Q \vdash R \to Q}\ \text{proj} \quad \dfrac{\dfrac{P \vdash P}{P, P \to R, R \to Q \vdash P}\ \text{proj} \quad \dfrac{P \to R \vdash P \to R}{P, P \to R, R \to Q \vdash P \to R}\ \text{proj}}{P, P \to R, R \to Q \vdash R}\ \text{app}}{P, P \to R, R \to Q \vdash Q}\ \text{app}}{P \to R, R \to Q \vdash P \to Q}\ \text{abst}}{}$$

   (b) Using the type inference system in table 2 to derive the type equations which are induced by the following lambda term:

$$\lambda x.(gx)f$$

   where g and f are free variables (so you start with the sequent $f : F, g : G \vdash \lambda x.(gx)f : T$ and your aim is to infer the variable types F, G, and T).

   (c) Solve these type equations using the unification algorithm to infer the types F, G, and T.