CPSC 449:
Sample questions: revision for the Midterm Exam

Robin Cockett

March 3, 2021

All the questions from the exercise 1 and 2 and the lab. exam are fair game for the midterm! The first dozen or so questions below come from these sources. The new questions include (a) folding on trees (b) the translation of list comprehension (c) unification (d) α-β-pruning of game trees (e) structural recursion on trees.

1. Show (by hand) how Haskell evaluates:

```
myappend [1,2] [3,4,5,6]
```

   given the code:

```
myappend:: [a] -> [a] -> [a]
myappend [] bs = bs
myappend (a:as) bs = a:(myappend as bs)
```

2. Show (by hand) how Haskell evaluates:

```
sfList [SS 3,SS 4]
```

   Given the code:

```
data SF a = SS a | FF

sfList:: [SF a] ->  SF [a]
sfLIst [] = SS []
sfList (FF:_) = FF
sfList ((SS a):xs) =
        case (sfList xs) of
                SS as -> SS (a:as)
                FF -> FF
```

3. Show (by hand) how Haskell evaluates:

```
myOR [False,True,False]
```

Given the code:

```
foldr:: (a -> b -> b) -> b  -> [a] -> b
foldr f b [] = b
foldr f b (x:xs) = f x (foldr f b xs)

myOR:: [Bool] -> Bool
myOR = foldr myor False

myor:: Bool -> Bool -> Bool
myor  False False = False
myor _ _ = True
```

4. Show (by hand) how Haskell evaluates:

```
unzip [(1,'a'),(2,'b')]
```

given the code:

```
unzip:: [(a,b)] -> ([a],[b])
unzip [] = ([],[])
unzip ((a,b):xs) = case unzip xs of
          (as,bs) ->  (a:as,b:bs)
```

5. Answer the questions concerning Haskell syntax below:

   (a) Which of the types below are the same type:

   i. `a -> b -> c -> d`
   ii. `(a -> b) -> c -> d`
   iii. `a -> (b -> c) -> d`
   iv. `a -> b -> (c -> d)`
   v. `(a -> b) -> (c -> d)`

   (b) In the following terms of type `Integer` what are the types of the functions `f` given that `x,y,z:: Integer`:

   i. `(f x) (y, z)`
   ii. `f x y z`
   iii. `f (x,y,z)`
   iv. `f (x,(y,z))`

(c) Give the types of the following terms (if indeed they type) and indicate which are equal:

i. `"abcd"`
ii. `[('a','b'),('c','d')]`
iii. `('a':['b']):('c':['d'])`
iv. `'a':('b':'c':'d':[])`
v. `["ab","cd"]`

6. Given a list of items and a predicate write code to split the list into two lists: a list of item which satisfies the predicate and a list of items which does not. Show how this may be done this using a `foldr` and using list comprehension. Explain which version is more efficient.

```
split_list:: (a -> Bool) -> [a] -> ([a],[a])
```

7. In a merge sort one step is to merge two ordered lists of items into one ordered list. Write the code for this step

```
merge:: (Ord a) => [a] -> [a] -> [a]
```

8. Write the code to split a list into two lists such that the elements with odd index are in one list while the elements with even index are in the other list:

```
odd_even_split:: [a] -> ([a],[a])
```

Can you write this using a `foldr`?

9. Write a function to determine whether a string is a substring of another string (a substring is a *contiguous* subsequence):

```
substring :: String -> String -> Bool
```

10. Write a function

```
grow :: String -> String
```

which changes a string $a_1a_2a_3...$ to $a_1a_2a_2a_3a_3a_3...$ so grow "now!" == "noowww!!!!".

11. Write a function to find all the substrings of a string.

12. Write a function to produce the list of all the sublists of a list (a sublist is a not necessarily contiguous subsesequence).

13. Why is the following "naive" function for reversing a list $\mathcal{O}(n^2)$:

```
reverse: [a] -> [a]
reverse [] = []
reverse (a:as) = (reverse as) ++ [a]
```

Give a "fast" $\mathcal{O}(n)$ version of reverse.

14. Why is the complexity of the following program $\mathcal{O}(\mathsf{Fib}(n))$ (assuming a positive input!):

```
fib 0 = 0
fib 1 = 1
fib n = (fib n) + (fib (n+1))
```

Give a $\mathcal{O}(n)$ version of `fib`.

15. A programmer writes the following code but fails to provide types:

```
data SF a = SS a | FF
    deriving (Show,Eq)

myhead [] = FF
myhead (a:as) = SS a

mytail [] = []
mytail (_:xs) = xs

myzip [] _ = []
myzip _ [] = []
myzip (a:as) (b:bs) = (a,b):(myzip as bs)

mystery xs =
   myhead [x|(x,y) <- myzip xs (mytail xs),x==y]
```

(a) Provide the types for `myhead`, `mytail`, `myzip`, and `mystery`.

(b) Explain what `mystery` does: what is the result of `mystery "abccdeffghii"`?

(c) Rewrite the code using a `foldr`?

16. A programmer writes the following code but fails to provide a type:

```
mycode f g [] = ([],[])
mycode f g (a:as) = case (mycode f g as) of
          (xs,ys) -> ((f a):xs, (g a):yx)
```

(a) What is the most general type of the code?

(b) What is the result of evaluating the following

```
mycode (\a -> a+6)
          (\b -> b `mod` 2 \= 0)
             [3,7,10,2,9,17]
```

(c) Rewrite `mycode` using a `foldr`.

17. Given the following list comprehension:

```
graph:: (Eq b) => (a ->b) -> [a] -> [b] -> [(a,b)]
graph f as bs = [ (x,y) | x <- as,y <- bs, f x == y]
```

(a) What does `graph` do?

(b) Translate the list comprehension into basic Haskell using the translation scheme, $[\![\_]\!]$:

$$\begin{aligned} [\![[s|\,]]\!] &= [s] \\ [\![[s|x \leftarrow t, r]]\!] &= \mathsf{concat}(\mathsf{map}(\backslash x \rightarrow [\![[s|r]]\!])t) \\ [\![[s|p, r]]\!] &= \mathsf{if}\ p\ \mathsf{then}\ [\![[s|r]]\!]\ \mathsf{else}\ [] \end{aligned}$$

18. Given the following "search tree" datatype

```
data STree a = Sn ( STree a) a (Stree a)
             | Tp
           deriving (Show, Eq)
```

The fold for search trees is:

```
foldST g t (Tp) = t
foldST g t (Sn s1 a s2) = g (foldST g t s1) a (foldST g t s2)
```

(a) What is the most general type of `foldST`:

(b) Use `foldST` to write a program, `whgt::STree Int -> Int`, to find the "weighted height" of a search tree of integers. This is the greatest sum of integers on any path from the root to a tip of such a tree. Thus, for example:

```
whgt (Sn(Sn Tp 5 Tp) 6 (Sn(Sn Tp 3 Tp) 1 (Sn Tp 2 Tp))) = 11
```

19. The `foldr` for lists (as above) is defined as:

```
foldr f g [] = g
foldr f g (x:xs) = f x (foldr f g xs)
```

(a) What is the most general type of `foldr`?

(b) Using `foldr` write the function

```
group :: (a ->a -> Bool) -> [a] -> [[a]]
```

which, given a predicate `p::a -> a -> Bool` and a list, breaks the list into a series of (maximal) sublists in which any two consecutive elements satisfy the predicate. For example, suppose that the predicate `nbr` determines whether two integers differ by at most one, then

```
group nbr [2,1,3,4,5,5,4,7,4,3,3]=[[2,1],[3,4,5,5,4],[7],[4,3,3]]
```

20. Given the datatype of expressions:

```
data Exp f  v = Opn f [Exp f v]
              | Var v
```

Expressions are trees with two sorts of nodes: a variable node and an operation node. The operation node has a function "value" `f` which is paired with a list of arguments.

The fold for the expression type is:

```
foldExp:: (f -> [c] -> c) -> (v -> c) -> (Exp f v) -> c
foldExp g h (Var v) = h v
foldExp g h (Opn f args) = g f (map (foldExp g h) args)
```

(a) Using `foldr` on list write a function to calculate the minimum of a list of integers: `mins:  [Int] -> Int`.

(b) Similarly, using `foldr` on list write a function to calculate the maximum of a list of integers: `maxs:  [Int] -> Int`.

(c) Now, using the fold for expressions write a program `minmax:: Exp Bool Int -> Int`, which takes the maximum of the arguments when function value is `True` and takes the minimum of the arguments when the function value is `False`.

21. (Harder!) A rose tree with values at the arguments of the nodes,

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

has its "tips" given by `Rs []` and may be regarded as a weighted tree (the weights are of type `a`) with the weights on the edges. The fold for this rose tree is:

```
foldRose g (Rs args)
       = g (map (\(a,arg) -> (a,foldRose g arg)) args)
```

(a) What is the type of `foldRose`?

(b) Use the fold on rose trees to find the weighted width of a tree

$$\texttt{width} :: \texttt{RoseInt} -> \texttt{Int}.$$

The weighted width is the maximum sum of the edges on any path from tip to tip in the tree. For example:

$$\texttt{width}(\texttt{Rs}[(4,\texttt{Rs}[]),(1,\texttt{Rs}[]),(6,\texttt{Rs}[]),(3,\texttt{Rs}[])]) = 10.$$

22. Given a rose tree with values at the nodes

```
data Rose a = RS a [Rose a]
```

(a) Write down the fold for this rose tree providing its type.

(b) Use the fold to generate the list of all the paths (labelled by the values at the nodes) from root to a leaf in the tree:

```
paths::Rose a->[[a]]
```

(c) The weighted height of a rose tree is given by the maximum of the sum of the weights on any path from root to leaf. Write a function to calculate the weight of a rose tree given a weighting function:

```
wgt:: (a->Int)->(Rose a)->Int
```

23. (Harder) Given a rose tree with values at the nodes

```
data Rose a = RS a [Rose a]
```

write a function to calculate the level sets of the rose tree. This a sequence of lists of values which occur at the same depths starting with the least deep level sets and ending with the deepest level sets.

24. Give the most general unifier for the following pairs of terms:

(a) $f(x, y)$ and $g(y, z)$

(b) $f(x, y, z)$ and $f(x, y)$

(c) $f(x, y, y)$ and $f(x, x, z)$

(d) $f(g(x, y), z)$ and $f(w, g(w, z))$

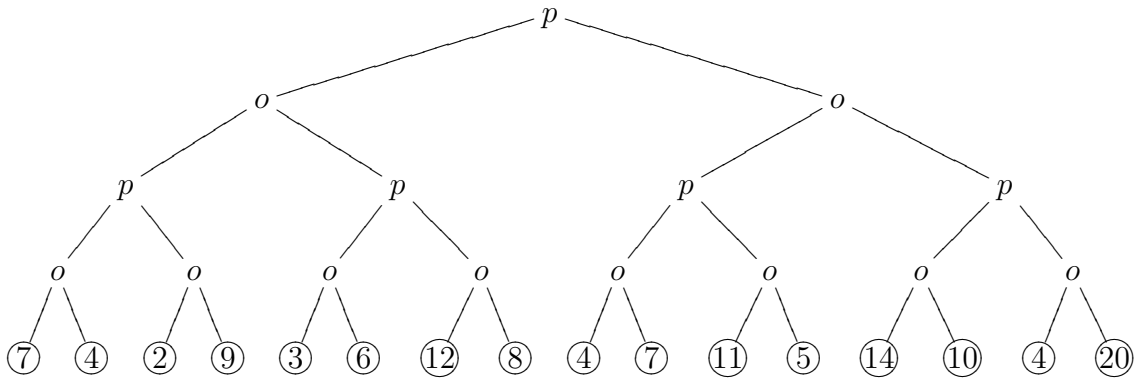
(e) $f(h(x, y), w, h(y, x))$ and $f(x, x, y)$

(f) $f(x, g(y, z))$ and $f(g(z, y), x)$

(g) $h(u, g(v, w), u)$ and $h(z, z, f(x, y))$

(h) $f(g(x, y), z)$ and $f(z, h(x, y))$

25. Given the following minmax tree indicate where the $\alpha - \beta$ cut-offs occur for the maximizing player:



26. In all these exercises you may assume that the datatypes are finite and the functions all terminate. Consider the following functions:

$$
\begin{aligned}
&(.) :: (b \to c) \to (a \to b) \to (a \to c)\\
&\ g.f = x \to g(fx)
\end{aligned}
$$

$$
\begin{aligned}
&\mathsf{map} :: (a \to b) \to ([a] \to [b])\\
&\mathsf{map}\ f\ [] = []\\
&\mathsf{map}\ f\ (x : xs) = (f\ a) : (\mathsf{map}\ f\ xs)
\end{aligned}
$$

$$
\begin{aligned}
&\mathsf{filter} :: (a \to \mathsf{Bool}) \to ([a] \to [a])\\
&\mathsf{filter}\ p\ [] = []\\
&\mathsf{filter}\ p\ (x : xs) | p\ x \qquad\quad = x : (\mathsf{filter}\ p\ xs)\\
&\qquad\qquad\qquad\quad | \mathsf{otherwise} = (\mathsf{filter}\ p\ xs)
\end{aligned}
$$

$$\mathsf{concat} = \mathsf{fold}\ \mathsf{append}\ []$$

(a) Prove that : $\mathsf{fold}\ f\ z\ (\mathsf{append}\ xs\ ys) = \mathsf{fold}\ f\ (\mathsf{fold}\ f\ z\ ys)\ xs$. Which form is more efficient? Which form is more readable?

(b) Prove that: $\mathsf{filter}\ p\ (\mathsf{append}\ xs\ ys) = (\mathsf{append}\ (\mathsf{filter}\ p\ xs)\ (\mathsf{filter}\ p\ ys)$. Which form is more efficient? Which form is more readable?

(c) Prove that: $\mathsf{map}\ (f.g)\ xs = \mathsf{map}\ f\ (\mathsf{map}\ g\ xs)$.Which form is more efficient? Which form is more readable?

(d) Prove that: $(\mathsf{filter}\ p).(\mathsf{map}\ f\ xs) = \mathsf{filter}\ (p.f)\ xs$. Which form is more efficient? Which form is more readable?

(e) Prove that: $\mathsf{rev}(\mathsf{map}\ f\ xs) = \mathsf{map}\ f\ (\mathsf{rev}\ xs)$. Is this still true for infinite list? Which form is more efficient? Which form is more readable?

(f) Prove that: $\mathsf{reverse}\ (\mathsf{reverse}\ x) = x$. Is this still true for infinite lists? Is this true for lists with elements which do not terminate?

(g) Prove that $\mathsf{filter}\ p\ (\mathsf{filter}\ q\ x) = \mathsf{filter}\ (p\ \&\&\ q)\ x$. Which form is more efficient? Which form is more readable?

(h) Define multiplication on the natural numbers, mult, and show that it is associative and commutative and distributes over the addition, add.

(i) Prove that the higher-order reverse is equal to the first-order reverse.

(j) Prove that $\mathsf{concat}(\mathsf{map}\ (\mathsf{map}\ f)\ xss) = \mathsf{map}\ f(\mathsf{concat}\ xss)$.

(k) Define an efficient first order collect function and prove it equal to the naive collect.

27. Mostly for fun! Who are the following people? When did they live? Where did they live? What did they do?

(a) Basile Bouchon
(b) Jean-Baptiste Falcon
(c) Jacques de Vaucanson
(d) Joseph Marie Jacquard
(e) Napoleon Bonaparte
(f) Charles Babbage
(g) Ada Lovelace
(h) Herman Hollerith
(i) Paul Otlet
(j) George Stibitz
(k) Konrad Zuse
(l) Alan Turing
(m) Alonzo Church
(n) Haskell Curry
(o) Greta Thunberg