CS4450 Lecture 17 Notes

Wednesday, September 26, 2018 3:03 PM

HW3 is out, due Tuesday 10/2 by 3pm.

Next week: Monday, we'll have a HW3 help session with the TAs like we had for HW1. It will be in the usual classroom instead of the TA's office.

Wednesday: Review session with Tom Reynolds in the usual classroom.

Friday: Midterm covering Chapters 1-6 of LYAHGG, HWs 1-3, and all slides up to, but not including, these slides/notes.

TODAY: We're starting trees. THIS MATERIAL WILL NOT BE ON THE MIDTERM, BUT WILL BE ON THE FINAL!

What will be on the midterm? TYPES! They're easy to use as multiple choice questions.

Example: Here's a function. Which types best describe how it calculates? (choose a,b,c, or d)

To answer these, go through the slides carefully to understand what's going on.

There will be no trick questions on the midterm!

You can't memorize the answers of past midterms, but they will be similar.

Let's finish up talking about folds.

Folds

foldl folds from the left.

Why two folds? Sometimes, using one can be more efficient than using the other. Two

recursion patterns => two folds (every recursive pattern has a fold).

Reversing a list, revisited

Obvious, but ineffecient:

```
reverse [] = []
reverse (x:xs) = reverse xs ++ [x]
--Much better:
reverse xs = rev [] xs
where rev acc [] = acc
rev acc (x:xs) = rev (x:acc) xs
-- We can redefine reverse as:
reverse xs = foldl f [] xs
where f xs x = x : xs
```

Function compostion:

We learned this in algebra, "g composed with f" is an example of function composition.

Slide 48 illustrates a great visual example of function composition.

f:g is also an appropriate way to denote function composition in mathematics.

Haskell code for function composition:

```
-- We'll do two versions, one with lamba expressions and one without.
compose :: (a -> b) -> (b -> c) -> (a -> c)
compose f g a = g (f a)
compose f g = \a -> g (f a)
--The above two definitions are equivalent, they just perform slightly
differently.
```

compose is a built-in function in Haskell. Denoted as **f . g** as well (ensure you leave spaces between the dot and **f** and **g**, otherwise a syntax error will occur.

How do you write a program that filters out negatives and maps sqrt? There are several ways of writing this function, and we want to use composition.

```
-- Can do either of the below two.
```

```
filter (\x -> x >= 0)
_ :: [Float] -> [Float]
```

```
filter (>=0)
```

We won't be asked a harder question about **foldl** on the midterm because we haven't covered it in depth in the homeworks yet. HOWEVER, you WILL need to know **foldr** for the midterm because we've used it in the homeworks!

Trees

Binary Trees

What is a binary tree? It's either a leaf with a single node, or it is a branch with two binary tree subtrees. These are the two ways to create a binary tree.

What a binary tree looks like in Haskell:

```
data Btree a = Leaf a | Fork (Btree a) (Btree b)
```

The slides on Binary Trees show a great visual representation on how a binary tree is represented in Haskell.

You can also represent the tree as:

```
(Leaf1) `Fork` (Leaf2)
```

You can also define an "infix" operator as:

```
data Btree a = Leaf a | (Btree a) :<x> (Btree a)
```

The :<x> is an infix operator. We don't have to know this for the midterm.

How do we get the height of a tree? It really depends on whether you start with a zero or one for the height of a tree.

How do we write **height**? These are in the slides for Haskell trees.

What is the size of a tree? It's the number of leaves that a tree has.

What should the type of **size** be?

```
size :: Btree -> Int
```

How about **flatten**? Given a binary tree**, flatten** returns a list with all of the items at the leaves of the tree.

Friday, we'll be shown the fold.