CMSC 161: Honors Introduction to Computer Programming

Austin Clyde
Professor Chugh

1 Class 1

1.1 House Keeping

- Daily assignments are to be turned in at the beginning of each lecture, on paper. Assignments marked by an astrik need to be done
- Something else goes here. $x + \sigma$

1.2 Introduction

Haskell is:

- functional
- pure
- lazy

...and thats about all he told us.

1.3 Online Notes

Functions in haskell are by curried (though tuples can be used as well. So a function that in Java would like `rate (distance, time)` will actually be written as `rate distance time`. Haskell has built in `curry` and `uncurry` functions.

2 Class 2

Functions in Haskell only take one argument. So haskell left associates parenthesies $(((e_1e_2)e_3)e_4)$, so e_1 takes one single argument and returns another function that takes e_3. –This is a curried function. We could easily define the function to just take a single 3-tuple, which is just a

single argument. Every function in Haskell is unary.

If we wanted to right associate parenthesies, we could use infix `$` to represent right associaitive parenthesies.
Since all functions take one argument, we can create a constant function `one _ = 1`, rather than using a variable that isn't referenced later `one x = 1` .

2.1 Lists

The most basic list is `[]`. We can on the fly define lists with commas, `[1, 2, 3, 4]`. All values within the list must have the same type. One difference between a tuple and list is in management. Tuples are hard coded by size; lists are hard coded by type.

Lists can also be created by cons, which is right associtives. We can construct lists like `[1, 2, 3]` by `1 : 2 : 3 : []` .
Effectivly this makes lists have a base case of `[]` and everything else is an inductive case.

The type of a list is defined once the first 'typed' element is cons onto the empty list.

Ranges can be generated by `..` , which is includisve. If you include a comma it might look for a pattern? Look this up or something.

You can remove n elements from a list with `take n list`.

3 Class 3

(op m) == flip (op) m (m op) == (op) m

3.1 η reduction

For instance,

$$fx = gx$$
$$f = g.$$

Since g is a fucntion already, we can smimply bind f to that function. This is mostly a matter of style. Either fx or f work, but be consistent. Different oppertunites lend intself to different styles.

3.2 Types

For instance, the list [] is a polymoprhic type. So [char] = [] char.

Numbers are intersting because they are a part of a type class **Num**. Hence, 10 is type Num but can also be type Int, Integer, or float. Types in the subset of the larger class can

be casted, but Haskell will not allow you to check types.

The type expressions are basically the same and SML. Prenthesises for type expressions assoc to the right.

4 Class 4 & 5

4.1 Algebraic data tpypes

Using the keyword `data`, you can define your own type. i.e.
`data Person = Student | Teacher`.
Just as functions are the name.

4.2 Foldr]

Take the code: `1:2:3:[] = f 1 (f 2 (f 3 init)) = 1‘f‘(2 ‘f‘ (3‘f‘ init))`.

You can also write `foldl` such that

```
-- where acc is the inital value with the function

foldr f acc []     = acc
foldr f acc (x:xs) = f x (foldr f acc xs)

foldl f acc [] = acc
foldl f acc (x:xs) = foldl f (f x acc) xs
```