Homework 10

Due Wednesday, 4/18/2019

Purpose:

The purpose of this homework assignment is to help you better understand how to create interpreters in Haskell.

Please to turn in a file Hmwk10.pdf with all the answers to the homework, but also pull out your Haskell programs and turn them in a text file named Hmwk10.hs. The first line of that file should be

```
module Hmwk10 where
```

The next line should be a comment with your name. The rest of the file should be your Haskell code. We will take that file and automatically test it by running your code on our test data. As a result, you should be sure that your code compiles properly (i.e., running "ghci Hmwk10.hs" should compile your code without errors and it should be possible to run your code on my test cases (ideally without it blowing up when executed!).

The Hmwk10.hs file will be used for automated testing of your code. Aside from looking for your name in the contents, it is likely that no human will read it. Thus if you don't include that code in the Hmwk10.pdf file, you will likely not get any credit for it. Please remember this!

While your Hmwk10.pdf file will be turned in to gradeScope as usual, you will need to turn in the file Hmwk10.hs via `https://submit.cs.pomona.edu/2019sp/cs101`.

Be sure to test your programs one last time before submitting. I've seen students mess up when commenting code in a way that causes everything to break. Code that doesnt compile will get very little credit. We will be using some automatic testing, and code that doesnt compile brings everything to a halt. With a large class we will not have time to go in to manually tweak your code to make it work.

IMPORTANT: When you write the functions requested below, please make sure that they have the exact names and types specified in the question. If you make an error, your program will crash on our test suite and you will get very little credit.

Be sure that all of my sample code works, and worry about edge cases that might cause your program to give the wrong answer as I will test it more thoroughly.

1. (0 points) **Academic Honesty**
2. (12 points) **Primitive recursive functions**

   We can define predecessor as a primitive recursive function by writing

   ```
   pred (0) = 0
   pred (y+1) = y
   ```

   The predecessor of successors is exactly what as expected, though the predecessor of 0 is 0 (because we are not using negative numbers).

   (a) Please write the primitive recursive definition of the function isZero(x) where isZero(x) = 1 if x is 0, and = 0 otherwise.

(b) Please write the primitive recursive definition of the function monus(m,n) where monus (m,n) = m - n if m $\geq$ n and 0 otherwise.

(c) Please write the primitive recursive definition of the function lessOrEq where lessOrEq (m,n) = 1 if m $\leq$ n and 0 otherwise.

(d) Please write the primitive recursive definition of the function greaterThan where greaterThan (m,n) = 1 if m > n, and 0 otherwise.

3. (20 points) **Interpreters**

The parser for PCF accepts let expressions of the form `let vble = term in body end`, but transforms them into syntax trees for function applications. That is, the above let expression is translated into the syntax tree for

```
((fn vble => body),term)
```

In this problem I would like you to start with a modified parser for PCF that takes let expressions like that above and parses it into an abstract syntax tree of the form `AST_LET(vble,term,body)`.

In this problem you will extend the environment interpreter for PCF to interpret this term correctly.

(a) Begin by writing the computation rule using **environments** for let expressions. This should be similar to the rules expressed in slides 6 – 8 of Lecture 22.

(b) Extend the environment interpreter in `PCFEnvinterpreter.hs` (available on the web page with "Programs from Lecture") to correctly interpret terms of the form `AST_LET(vble,term,body)`. A parser that generates these `AST_LET` terms is available on that same web page. Test your program with let expressions like `let x = 2 in succ x end`.