Homework 1 Solution

CS 440: Programming Languages and Translators, Spring 2020

Problems

1. (Various expressions in ghci)

   1a. Calculates the sine of the (cosine of π).

   ```
   > sin (cos pi)
   -0.8414709848078965
   ```

   1b. Error: We asked for cos of function unary minus. Need parens around - 1.

   ```
   > cos -1
   ```

   1c. Error: Like (b): we asked for sin of function unary cosine. Need parens around `cos pi`.

   ```
   > sin cos pi
   ```

   1d. Calculates 4th root of 16.0: `[sqrt]` is a list containing one element, the square root function. Head of that yields the function itself, and (`sqrt . sqrt`) `16.0` = `sqrt(sqrt(16.0))`.

2. (Fix parentheses)

   2a. `(cos(sqrt 2.5)+(sin pi))*2`. (The parens around * were bad, the others were redundant.)

   2b. `(:) ('a' : "b" ++ "cd") [['c'] ++ "(d)"]`

   2c. `[[[[17]]],[]]`. (Trick: Just typing the original expression into ghci gives you this.)

3. (Use prefix) `(/) ((*) ((+) a b) c) ((^) d e)`. (Trick: substituting numbers for `a`, `b`, …, `e` results in a calculated value you can use to double-check your computation.)

4. (Use infix) ``(x `g` (a `h` b)) `f` (c (e `d` f))``

5. (List comprehension) Calculate the list `[x !! 0, x !!1 , x !! 2,` *etc*`]`, so we want `x !! i` for `i` = `0`, `1`, etc.

   ```
   f x = x == [x !! i | i <- [0..length x - 1]]
   ```

   Note because we're asking `x` == *something*, we need `x` to have type `Eq a => [a]`. If we just return the list comprehension (`f x = [ … ]`), then `x` can be a list of types that aren't instances of `Eq`.

6. (List comprehension) We want `[x, x, x, x, .., x]` where there are `n` `x`'s.

   ```
   stutter n x = [x | i <- [1..n]]
   ```

7. (Use referential transparency to compute part of an infinite list)

   We're given the list `g` and want to calculate `take n g` for `n` = 0, 1, …

   ```
   > g = [1,3,5] : [last x : init x | x <- g]
   ```


Intuitively, the last element of each sublist is going to rotate between `1`, `3`, and `5`, so `g` should have a repeating pattern of length 3. To verify the beginning of the pattern, we can hand-calculate the first few values of `g`. The base case is easy: `take 0 g` = `[]`. Continuing,

```
take 1 g  -- (I'm showing a lot of detail here)
   = head g : [rot x | x <- tail(take 0 g)]
   = [1,3,5] : [rot x | x <- tail []]
   = [1,3,5] : [rot x | x <- []]
   = [1,3,5] : []
   = [[1,3,5]]
take 2 g
   = [1,3,5] : [rot x | x <- take 1 g]
   = [1,3,5] : [rot x | x <- [[1,3,5]]]
   = [1,3,5] : [5:[1,3]]
   = [[1,3,5], [5,1,3]]
```

Let's start using the property `take` (`m`+1) `g` = `take m g` ++ [e_1] where e_1 is the expression for the last element of `take` (`m`+1) `g`. For this particular `g`, e_1 = `[rot x | x <- [last(take m g)]]`. So

```
take 3 g
   = take 2 g ++ [rot x | x <- [[5,1,3]]]
   = [[1,3,5],[5,1,3]] ++ [rot x | x <- [[5,1,3]]]
   = [[1,3,5],[5,1,3]] ++ [[3:[5,1]]]
   = [[1,3,5], [5,1,3], [3,5,1]]
```

And

```
take 4 g  -- (I've cut out a lot of the detail here)
   = [[1,3,5], [5,1,3], [3,5,1]] ++ [rot x | x <- [[3,5,1]]]]
   = [[1,3,5], [5,1,3], [3,5,1], [1,3,5]]
```

So we see explicitly that `last(take 1 g)` = `last(take 4 g)` = `[1,3,5]`. Those values determine the values of `last(take 2 g)` and `last(take 5 g)`, which are therefore equal. More generally, the last elements of `take` 1, 4, 7, 10, … are `[1,3,5]`, the last elements of `take` 2, 5, 8, … are `[5,1,3]`, and the last elements of `take` 3, 6, 9, … are `[3,5,1]`. I.e.,

```
g = [[1,3,5], [5,1,3], [3,5,1]] ++ [[1,3,5], [5,1,3], [3,5,1]] ++ …
```

