Numbers, Numerals & Polynomials Homework

WU CS 465—Fritz Ruehr—Spring 2019

For this homework assignment you will need to think about Peano's definition of the natural numbers (ℕ), which include 0, 1, 2, etc.—but no negative numbers or fractions. Recall that the Haskell definition of naturals uses the constructors `Succ` and `Zero`, and that we will (of course!) also endow it with a fold function (here `foldn`):

```
data Nat = Zero | Succ Nat

foldn s z Zero     = z
foldn s z (Succ n) = s (foldn s z n)
```

(I switched the argument order from lecture because this way seemed more … natural 😉 .)
For example, a function to convert `Nats` to Haskell's `Int` or `Integer` types can be written as follows (basically just replacing `Zero` and `Succ` with their "true integer" equivalents):

```
n2i = foldn (+1) 0
```

This leads to some very concise definitions of the addition, multiplication, and exponentiation functions, as follows:

```
add m = foldn Succ m
mul m = foldn (add m) Zero
exp m = foldn (mul m) one
```

You can get all the relevant definitions for Nats and polynomials in the files from lecture at:

```
http://www.willamette.edu/~fruehr/465/code/Nats.html
http://www.willamette.edu/~fruehr/465/code/Poly.html
```

Now on to the exercises!

1. Use the `foldn` function to convert Peano `Nats` to **tally strings** using the 'pipe' character, so that (for example) the tally for `five` is `"|||||"`; note that `Zero` will be represented by the empty string. (So, define `tally :: Nat -> String`.)

2. We can also represent `Nats` as lists of units, where units are similar to the FAST type **1**, with value ● … but in Haskell terms, that's `()::()`. I.e., the Haskell value and type are both written as "`()`". Define versions of `add` and `mul` (say `add'` and `mul'`) that work on this alternative type, so that they operate on lists of units:

   ```
   add', mul' :: [()] -> [()]
   ```

   (Hint: imagine those units `()` are little stones in piles: how do you add them? And how did we discuss in class you could think of multiplication?)

3. As we discussed in lecture, time can be expressed as mixed-radix numerals, using bases 2, 12, 60, and 60 for am/pm, hours, minutes and seconds. Using this

representation, how can you easily compute the number of seconds are in a day? Can you convert between “raw seconds” and a mixed radix time numeral (say as a list of `Ints`)?

4. If we allow only a single occurrence of x in an algebraic term that otherwise contains only addition and multiplication operators, it will “evaluate” to a polynomial that is in fact *linear,* i.e., of the form “mx+b” from middle school algebra. How could you prove this, even if just informally? (Hint: think in terms of induction!)

5. In beginning calculus, the derivative (∂) of a function is a fundamental notion. For polynomials, we can compute the derivative as follows: for each term, multiply the power of the term by the coefficient and then subtract one from the power, finally dropping any constant term (where the power of x is 0). So, for example, we have

$$\partial\ (\ 4x^3 + 7x^2 + 3x + 1\ ) = 12x^2 + 14x + 3$$

since 3×4 = 12, 2×7 = 14 and 1×3 = 3. Write a derivative function in Haskell that finds the derivative of a polynomial represented (as in lecture and the `Poly.html` file) as a **reversed** list of coefficients. So, continuing the example from above:

```
deriv [1,3,7,4] == [3,14,12]
```