CPSC 449: Assignment 4

Fall 2020

See the D2L site for due date/time.

1. [20%] A *polynomial in one variable* (or "polynomial" in short) is defined inductively as follows:

   - a constant of type **`Int`** is a polynomial,
   - *the* variable is a polynomial,
   - if P and Q are polynomials, then $P + Q$ is a polynomial, and
   - if P and Q are polynomials, then $P \times Q$ is a polynomial.

   Except for the above there is no other polynomial. Recall that such a polynomial can be represented by the algebraic type below:

   ```
   data Poly = PConst Int |
               PVar |
               PAdd Poly Poly |
               PMul Poly Poly
   ```

   A function f of type **`Int->Int`** is the ***denotation***[1] of a **`Poly`** P iff f is a single-argument function that evaluates P at the argument. For example, **`(\x -> x + x)`** is the denotation of **`(PAdd PVar PVar)`**. Develop a ***denotational compiler*** for **`Poly`**:

   ```
   compilePoly :: Poly -> (Int->Int)
   ```

   such that **`(compilePoly`** P**`)`** returns the denotation of P. The function **`compilePoly`** must be primitively recursive. An implementation that violates this requirement is considered a *non-solution*!

2. [25%] **[Thompson]** exercise 11.34. Use the following definition of the function **`concat`**:

   ```
   concat = foldr (++) []
   ```

   You may also use the axiom **`(map++)`** on page 261 (under Exercise 11.31).

[1] For the mathematically minded, we assume that function equality is defined in an extensional manner, as described in §11.6 of **[Thompson]**.

3. [30%] Complete the implementation of the **`Expr`** parser as presented in the lecture slides. Your code should be based on **`parser.hs`**, which has been posted at the course web site. **Important:** *You are not allowed to modify the code that is already given in* ***`parser.hs`***. Specifically, you need to provide the implementation for the following functions:

   - **`isOp`** and **`charToOp`** (see **[Thompson]** exercise 17.12)
   - **`makeExpr`**
   - **`optional`** and **`neList`** (see **[Thompson]** exercise 17.10)
   - **`stringToExpr`** (see **[Thompson]** exercise 17.14)

4. [25%] **[Thompson]** exercise 17.25.