COMPSCI 1JC3

Introduction to Computational Thinking

Fall 2017

Assignment 3

Dr. William M. Farmer

McMaster University

Revised: October 27, 2017

The purpose of Assignment 3 is to write a module in Haskell that defines an algebraic data type of polynomials and implements various functions on the members of the type including symbolic differentiation. The requirements for Assignment 3 and for Assignment 3 Extra Credit are given below. You are required to do Assignment 3, but Assignment 3 Extra Credit is optional. Please submit Assignment 3 as a single `.hs` file to the Assignment 3 folder on Avenue under Assessments/Assignments. If you choose to do Assignment 3 Extra Credit for extra marks, please submit it also as a single `.hs` file to the Assignment 3 Extra Credit folder on Avenue in the same place. Both Assignment 3 and Assignment 3 Extra Credit are due **November 3, 2017 before midnight.** Assignment 3 is worth 3% of your final grade, while Assignment 3 Extra Credit is worth 2 extra percentage points.

Although you are allowed to receive help from the instructional staff and other students, your submitted program must be your own work. Copying will be treated as academic dishonesty!

1 Background

Let C be a set of *coefficients* closed under addition and multiplication such as the integers $\mathbb{Z}$, the rational numbers $\mathbb{Q}$, or the real numbers $\mathbb{R}$. A *polynomial over* C is a mathematical expression that is constructed from an *indeterminant* x and members of C by applying addition ($+$) and multiplication ($*$) operators. Let P be the set of polynomials over some C. The value of a polynomial $p \in P$ at $c \in C$ is the result of replacing the indeterminant x with c. For example, the value of $(2 * x) + 4$ at 3 is $(2 * 3) + 4 = 12$.

Every polynomial p represents a *polynomial function* $f_p : C \to C$ that maps $c \in C$ to the value of p at c. For every $p \in P$, there is a $q \in P$ that has the form

$$a_0 + a_1 * x^1 + a_2 * x^2 + \cdots + a_m * x^m,$$

where $a_0, a_1, \ldots \in C$, x^i is an abbreviation for $x * \cdots * x$ (i times), and parentheses have been depressed, such that f_p and f_q are the same function. q is called the *standard form* of p. The *degree* of p is m, the largest exponent appearing in q. For example, the standard form of $(x + 1) * (x + 2)$ is

$$2 + 3 * x^1 + x^2$$

and the degree of $(x + 1) * (x + 2)$ is 2.

2 Assignment 3

The purpose of this assignment is to create a Haskell module for polynomials over the integers.

2.1 Requirements

1. The name of your Haskell file is `Assign_3_`*`YourMacID`*`.hs` where *YourMacID* is your actual MacID.

2. Your name, MacID, the date, and "Assignment 3" are given in comments at the top of your file.

3. The first uncommented line of the file should be

```
module Polynomial where
```

4. The file contains the algebraic data type definition

```
data Poly =
    X
  | Coef Integer
  | Sum Poly Poly
  | Prod Poly Poly
  deriving Show
```

5. The file includes a function named `polyValue` of type

```
Poly -> Integer -> Integer
```

   such that `polyValue p n` is the value of `p` at `n`.

6. The file includes a function named `polyDegree` of type

```
Poly -> Integer
```

   such that `polyDegree p` is the degree of `p`.

7. The file includes a function named `polyDeriv` of type

   ```
   Poly -> Poly
   ```

   such that `polyDeriv p` represents the derivative of the polynomial function represented by `p`. `polyDeriv p` thus symbolically differentiates a polynomial `p`.

8. Your file can be imported into GHCi and all of your functions perform correctly.

2.2 Testing

Include in your file a test plan for the functions `polyValue`, `polyDegree`, and `polyDeriv`. The test plan must include at least three test cases for each function. Each test case should have following form:

`Function:` Name of the function being tested.
`Test Case Number:` The number of the test case.
`Input:` Inputs for function.
`Expected Output:` Expected output for the function.
`Actual Output:` Actual output for the function.

The test plan should be at the bottom of your file in a comment region beginning with a {- line and ending with a -} line.

3 Assignment 3 Extra Credit

The purpose of this assignment is to create a Haskell module for polynomials over an arbitrary type.

3.1 Requirements

1. The name of your Haskell file is `Assign_3_ExtraCredit_`*`YourMacID`*`.hs` where *YourMacID* is your actual MacID.

2. Your name, MacID, the date, and "Assignment 3 Extra Credit" are given in comments at the top of your file.

3. The first uncommented line of the file should be

   ```
   module Polynomial where
   ```

4. The file contains the algebraic data type definition

```
data Poly a =
    X
  | Coef a
  | Sum (Poly a) (Poly a)
  | Prod (Poly a) (Poly a)
  deriving Show
```

5. The file includes a function named `polyValue` of type

   ```
   Num a => Poly a -> a -> a
   ```

   such that `polyValue p n` is the value of `p` at `n`.

6. The file includes a function named `polyDegree` of type

   ```
   Poly a -> Integer
   ```

   such that `polyDegree p` is the degree of `p`.

7. The file includes a function named `polyDeriv` of type

   ```
   Num a => Poly a -> Poly a
   ```

   such that `polyDeriv p` represents the derivative of the polynomial function represented by `p`. `polyDeriv p` thus symbolically differentiates a polynomial `p`.

8. The file includes a function named `polyNewton` of type

   ```
   (Fractional a, Ord a) => Poly a -> a -> a
   ```

   such that `polyNewton p s` computes, using Newton's method with the seed `s`, a number `n` such that `polyValue p n` is approximately 0. `polyNewton p` thus solves the polynomial equation $\mathtt{p} = 0$.

9. The file includes a function named `polyAsList` of type

   ```
   (Num a, Eq a) => Poly a -> [a]
   ```

   such that `polyDeriv p` returns the list $[a_0, a_1, \ldots, a_m]$ where

   $$a_0 + a_1 * x^1 + a_2 * x^2 + \cdots + a_m * x^m$$

   is the standard form of `p`. `polyAsList p` thus transforms a polynomial `p` into standard form.

10. Your file successfully loads into GHCi and all of your functions perform correctly.

3.2 Testing

Include in your file a test plan for the functions `polyValue`, `polyDegree`, `polyDeriv`, `polyNewton`, and `polyAsList`. The test plan must include at least three test cases for each function.