PFL_TP1_G06_03

First Project for the Course Programação Funcional e Lógica (PFL) at Faculdade de Engenharia do Porto (FEUP).

- A brief description of the implemented strategy and each vital functionality
- Real world scenarios that prove the robustness of our program

Authors:

- Manuel Carlos Ramos Alves (up201906910)
- Pedro José Ferreira Moreira (up201905429)

How to Run

To run all you need to do is have a working version of ghc interpreter and run main function that will help you to navigate the functionalities.

> ghci
>
> :load project.hs
>
> main

In order to run the program and the tests, you may need to install the following:

> cabal install --lib split
>
> cabal install --lib QuickCheck

Polynomial Representation

Structure

We've decide not to use a data type once we found a solution to all out problems in the pre-existent types.

As you can see below a polunomial is nothing more than a list of monomials that are in turn an Int (Coef) and a list of tuples containing an Int and a Char (VarExpo);

type Poly = [Mono]

type VarExpo = (Char,Int)

type Coef = Int

type Mono = (Coef, [VarExpo])

- **Poly** : [Mono];
- **Mono** : (Coef, [VarExpo])
- **VarExpo** : (Char,Int);
- **Coef** : Int;

Justification

We've decided to use only tuples and list since it allowed us "more flxibility" and provided us with more built-in functions than we would have access to otherwise.

Every functionality, from parsing to derivation, slipt until atomic operations/functions making our code quite verbose but easier to understand and debug which was much needed.

Every property of our operations was tested and we believe that the code suports pretty much any input that was discussed in classes and that was written on the exercise sheet.

Functionalities

Main - main

> A very simple interface that facilitates the interation with the program.
>
> No need to know our particular syntax or function names. All the user needs to do is input one of three possible operations - sum, mult and deriv.
>
> Following the choice of operation the user is required to input a polynomial in the form of a string (Ex: 2x^5 + 5*y -3)
>
> In the derivation instead of inputing another polynomial followed by ENTER you should input the variable you want to derive in relation to (Ex: x)

Parsing - strToPoly :: String -> Poly

> As stated before we've decide to seperate this functionality into various functions.
>
> strToPoly calls for the removal of spaces then adds a '+' before every '-'. Once this is done we can split the string everytime we find a '+'. Now we have "untreated" strings that represent monomials such as ["2x^2","4x",5]
>
> Now we call strToMono that calls for the extraction of the coefficient and the list of variables and correspondent exponents.

Normalize - normPoly :: Poly -> Poly

> Normalizes a polynomial by removing redudancies such as x^-2 * x^2.
>
> Groups the monomials with the same variables and sums the exponents.

Sum - sumPoly :: Poly -> Poly -> Poly

> Once the normPoly already adds coeficients all we need to do call for the normalization of all of the polynomials using a list comprehension.

Multiplication - multPoly :: Poly -> Poly -> Poly

> Multiply the coeficients of each monomial and concatenate their lists of variables and exponents.
>
> The normPoly takes care of the rest just like in sum

Derivation - derivPoly :: Poly -> Poly -> Poly

> Find a variable and decrease the exponent or eliminate at the variable if the exponent is 1.

Utilization Examples

In the program there is a section of thest where you can observe utilization examples to each funcionality.

Parsing

> strToPoly "1*50 *100 + 100 + 2" = [(5102,[])]
>
> strToPoly "9* x^2 + 5* x + 2* x" = [(9,[('x',2)]),(7,[('x',1)])]
>
> strToPoly "x^2* x^3* y + y* y" = [(1,[('x',5),('y',1)]),(1,[('y',2)])]

Sum

> sum '1* 50* 100 + 100' and '-500*10 + 2' = 102
>
> sum '2* x^2 + 5* x' and 'x^2 + 2* x' = 3x^2 + 7x
>
> sum '2* x* 5* x^6' and 'x^7' = 12*x^7
>
> sum 'x^2* x^3* y' and 'y* y' = x^5y + y^2

Multiplication

> mult '1* 50*100' e '2' = 10000
>
> mult 'x + 2' e 'x - 2' = x^2 - 4
>
> mult '2* x^2* 5' e '5* y^2* 2' = 100* x^2*y^2
>
> mult 'x^2 + y^2' e 'z^2 + x^2' = x^2* y^2 + x^2* z^2 + x^4 + y^2*z^2
>
> mult 'x^20* z^30' e 'z^90* x^12' = x^32*z^120

Derivation

> Derivation '5 + 10*2 + 25' in relation to 'x' = 0
>
> Derivation '2* x^2 + 5* x + 2* x^3' in relation to 'x' = 6x^2 + 4x + 5
>
> Derivation 'x^2* x^3* y + y* y' in relation to 'y' = x^5 + 2*y
>
> Derivation 'x^20* z^30* x^12 -1 *z^200 + 50' in relation to 'z' = 30* x^32* z^29 - 200*z^199

Tests

We decided to test the properties of our operation using quickTest.

We decided not to exclude any test and every property is tested with 100 different inputs attesting our decent performance.

prop_associativity_sum - Test asoociativity in sum

prop_null_element_sum - Test the null element of sum (zero)

prop_comulative_mult - Test comulativity in multiplication

prop_null_element_mult - Test null element in multiplication

prop_coef_deriv - The derivative of 5x should be equal to de derivative of x multiplied by 5

prop_sum_deriv - The sum of derivatives should be equal to the derivative of the sums