PFL - Projeto 1

Grupo T02G3

Este trabalho foi realizado por:

- Ana Beatriz Cruz Fontão (up202003574)
- Ana Rita Baptista de Oliveira (up202004155)

Como compilar e correr

- Instalar o ghci 9.0.1
- Abrir um terminal ghci
- Compilar usando :load Proj.hs
- Usar as funções normalize, add, multiply e findDerivative de acordo com as instruções descritas no resto do README

Representação Interna de Polinómios- descrição e justificação

Monómios

Para a representação dos Monómios (Mon), nós decidimos usar a estrutura seguinte: (Int,[(Char,Int)]). O primeiro elemento do tuplo representa o coeficiente do monómio. O segundo elemento do tuplo é uma lista de tuplos em que cada um desses tuplos representa uma variável e o respetivo expoente. Nesta implementação, consideramos que cada variável apenas aparece uma vez na lista de tuplos.

Ex.: (2,[('x',1),('y',5)]) = 2xy^5

Quando o monómio é uma constante decidimos que seria representado com apenas o tuplo ('x',0) na lista de variáveis.

Ex.: (1,[('x',0)]) = 1; (0,[('x',0)]) = 0

Polinómios

Para a representação dos Polinómios (Pol), decidimos que seria uma lista de Monómios: [Mon].

Nós consideramos estas as representações mais adequadas uma vez que se tratam de monómios com multiplas variáveis. Inicialmente consideramos a representação para um Monómio (Int, String, [Int]), em que a string seria uma lista das variáveis e a lista de inteiros os respetivos expoentes. Acabamos por não optar por esta representação já que seria necessário fazer a operação zip entre variáveis e expoentes múltiplas vez. Com a nossa representação, a repetição da operação deixa de ser necessária.

Estratégia de Implementação das Funcionalidades

Parsing

A função **createPol** permite o parsing de string para polinómio, enquanto a função **printPolResult** faz o contrário.

As nossas funções principais (normalize, add, multiply, findDerivative) aceitam todas os polinómios na forma de strings. O parsing passa por criar o polinómio analisando a string e adicionando monómio a monómio.

Normalização

A função principal para esta funcionalidade é a função **normalize**. Ela recebe uma string com o polinómio e retorna também uma string com o resultado.

A implementação da normalização segues os seguintes passos:

- Criar um polinómio a partir da string recebida
- Ordenar as variáveis de cada monómio a alfabéticamente
- Começando numa lista com apenas 0, adicionar todos os monómios um de cada vez seguindo as regras de adição de monómios
- Ordenar o polinómio resultante por dois critérios: monómio com maior expoente e monómio com maior número de variáveis (descrescente em ambos)
- Remover todas as parcelas que têm valor 0
- Converter o polinómio final de novo em string

Adição

A função principal para esta funcionalidade é a função **add**. Ela recebe duas strings com os polinómios e retorna uma string com o resultado.

A implementação da adição segues os seguintes passos:

- Criar dois polinómios a partir das strings recebidas
- Concatenar os dois polinómios
- Aplicar a normalização ao polinómio obtido no ponto anterior
- Converter o polinómio final em string

Multiplicação

A função principal para esta funcionalidade é a função **multiply**. Ela recebe duas strings com os polinómios e retorna uma string com o resultado.

A implementação da multiplicação segues os seguintes passos:

- Criar dois polinómios a partir das strings recebidas
- Multiplicar cada monómio do 1º polinómio pelos monómios do segundo polinómio
- Na multiplicação de dois monómios, multiplicamos os coeficientes normalmente e usamos foldr para as variáveis
- Normalizar o resultado
- Converter o polinómio final em string

Derivação

A função principal para esta funcionalidade é a função **findDerivative**. Ela recebe um caracter com a variável à ordem que queremos derivar e uma string com o polinómio para derivar. Retorna uma string com o resultado.

A implementação da derivação segues os seguintes passos:

- Criar um polinómio a partir da string recebida
- Derivar monómio a monómio o polinómio de acordo com a variável recebida
- Normalizar o resultado
- Converter o polinómio final de novo em string

Exemplos de utilização

Normalização

normalize :: String -> String

Input	Output
normalize ""	"0"
normalize "0"	"0"
normalize "1"	"1"
normalize "x^2+x^2"	"2x^2"
normalize "xy + yx"	"2xy"
normalize "x+y^2"	"y^2 + x"
normalize "x^2 - z^3 + y^2x"	"- z^3 + xy^2 + x^2"
normalize "a y + 0 - y"	"ay - y"

Adição

add :: String -> String -> String

Input	Output
add "" ""	"0"
add "" "1"	"1"
add "x" ""	"x"
add "20" "82"	"102"
add "x^2" "y+z"	"x^2 + z + y"
add "f^3 + 5" "f -4f^3"	"- 3f^3 + f + 5"
add "z^0 - x^3 + 5" "zy -5yz + 0"	"- x^3 - 4yz + 5 + 1"

Multiplicação

multiply :: String -> String -> String

Input	Output

Input	Output
multiply "" ""	"0"
multiply "1" ""	"0"
multiply "" "x^2"	"0"
multiply "2" "4"	"8"
multiply "x" "x+y+z"	"x^2 + xz + xy"
multiply "x - 3 - 3" "0 - y^5x^2"	"6x^2y^5 - x^3y^5"

Derivação

findDerivative :: Char -> String -> String

Input	Output
findDerivative ' ' ""	"0"
findDerivative ' ' "x^2"	"x^2"
findDerivative 'x' ""	"0"
findDerivative 'x' "x^20"	"20x^19"
findDerivative 'y' "x + 2y + 1"	"2"
findDerivative 'z' "xz^2 + 2z^3 + 3z^0"	"6z^2 + 2xz"

Exemplos mais complexo

É possível compor as funções de diversas formas, permitindo operações mais complexas

Input	Output
findDerivative 'z' (add "50 - zx + xy" (multiply "2x^2z^3 - 1" "5xy - 2yx"))	"18x^3yz^2 - x"
multiply (findDerivative 'y' "5y - z^30 -y^4") (add "3xy +z^3 " " 1yx - 3zy")	"- 16xy^4 + 12y^4z + 5z^3 - 4y^3z^3 - 15yz + 20xy"