PFL - 1º Trabalho Prático

Representação Interna

A representação interna de um polinómio no nosso trabalho é declarada da seguinte forma:

```
data Polynomial = Polynomial {variable :: Variable, coefficients ::
[Coefficient]} deriving (Show)

type Coefficient = Int
type Exponent = Int
type Variable = String
```

e consiste na divisão semântica de cada parcela importante de um polinómio de maneira concisa: um parâmetro *variable* que guarda os variáveis literais de um polinómio/monómio e uma lista *coefficients* que guarda coeficientes associados à variável usando o index na lista como representação do expoente, de forma ascendente e começando no primeiro grau. Tendo um polinómio que consta com mais do que uma variável, cada polinómio associado a uma variável específica é guardada numa super-lista de polinómios.

Qualquer constante é guardada numa estrutura com o mesmo nome cuja variável é uma string vazia ("") e cuja lista de *coefficients* conserva apenas as constantes que aparecem no polinómio.

Nos edge cases em que um polinómio apresenta variáveis compostas (ex.: "a*x^2"), esta é tratada de forma diferente: a representação em polinómio agrega tanto variáveis e expoentes e a lista de coeficientes resume-se a apenas o mesmo relativo a essa variável. Isto foi feito de forma a simplificar operações de multiplicação.

Exemplificando:

```
"2*x^3 + 5*x^2" => ("x", [2, 5, 0, 0])
-
"x^4 + 5*y + 10 + 2 => [("", [10, 2]), ("x", [4, 0, 0, 0]), [("y", [5])]]
-
"4*x*y^2" => ("x*y", [4])
```

Esta representação interna foi escolhida com base na simplicidade e eficiência da estrutura de dados: para quê guardar expoentes em separado quando se pode usar tamanhos de listas para os coeficientes? Para além disso, devido a esta escolha, a adição de polinómios foi completamente trivializada, embora a multiplicação/derivação não tenha sido tão simples, como será explicado em seguida. Não existe também separação de monómios e polinómios na estrutura de dados: No fundo, um monómio é um polinómio com um único termo e uma lista de polinómios é tratada como um polinómio.

No que toca às funcionalidades principais, os outputs para string encontram-se separados numa função diferente da principal, de forma a facilitar a análise da representação interna, por exemplo: `multiplyPolynomial` mostra o resultado numa lista de polinómios representados pela estrutura de

dados personalizada, enquanto que `showMultiplyPolynomial` invoca a função anterior e normaliza o resultado, representando-o em formato de string.

Normalização

A normalização é feita organizando o polinómio por ordem alfabética crescente de variáveis, seguida de ordem decrescente de expoentes. No caso de variáveis compostas, só é considerada a primeira variável e o primeiro expoente (e as próprias variáveis compostas são construídas pela ordem alfabética mencionada anteriormente). Assim, na normalização, pode-se apenas considerar o polinómio atual e o seguinte, somando os seus coeficientes conforme seja necessário num novo polinómio e continuado a percorrer a lista até ao fim.

Adição

A adição é simplesmente uma junção de listas (duas de cada vez) de polinómios seguidas da sua normalização, dado que tudo o que é feito na adição é também feito na normalização.

Multiplicação

Quanto à multiplicação, tentou-se separar ao máximo cada passo da mesma em diferentes funções para facilitar testes e compreensão do código. Essencialmente, existem 3 casos: multiplicação de polinómios de variável simples e igual entre eles, multiplicação de polinómios de variável simples e diferente entre eles e multiplicação de polinómios onde pelo menos um deles possui variável composta.

- Variáveis iguais: Cria-se uma nova lista de coeficientes de tamanho igual à soma dos tamanhos das duas listas de coeficientes dos polinómios a serem multiplicados (selecionados das duas listas iniciais de polinómios), preenchida com zeros. À medida que se percorre recursivamente as listas de coeficientes, atualiza-se a nova lista com o produto dos coeficientes no índice igual a `tamanho da nova lista de coeficientes - tamanho da lista de coeficientes 1 - tamanho da lista de coeficientes 2`. O módulo Data.Sequence foi indispensável para esta parte.

- Variáveis diferentes: Junta-se o literal e o expoente do primeiro polinómio com o literal e o expoente do segundo polinómio numa string, e multiplicam-se os coeficientes. O novo polinómio guarda a string como literal e o coeficiente na sua lista, garantindo assim a sua apresentação correta quando chamadas as funções de print.

- Variáveis compostas: Semelhante ao caso anterior, mas faz-se o parsing das variáveis compostas para ver quais variáveis individuais existem e fazer a adição dos expoentes sempre que for adequado.

Derivação

A derivação faz-se através da separação de constantes e literais: Trata-se primeiro destes últimos, alterando a lista de coeficientes multiplicando o expoente do monómio em questão ao coeficiente e reduzindo esse expoente por um através da manipulação da posição do coeficiente na lista (no caso de variáveis simples) ou faz-se parsing da variável composta para a reconstruir segundo as regras explícitas anteriormente. Após os literais estarem processados, utiliza-se uma função para detetar que novas constantes irão surgir da derivação e adiciona-se um polinómio à lista com esse valor no coeficiente. Todos os polinómios que não estejam relacionados com a variável em questão são tratados como constantes e, por isso, apagados.

Exemplos de execução

No módulo Tests encontram-se funções que testam todas as funcionalidades principais e algumas funções intermédias que expõem o funcionamento interno do programa. De qualquer das formas, seguem-se algumas imagens de execuções das funções do programa:

> Normalização

```
ghci> showPolynomialTrimmed (testPolynomial)
"3*x^3 + 6*x^2 + 5*x + 5*x^2 + 4*x + 8 + 6"
ghci> showPolynomialTrimmed (sortAndNormalize testPolynomial)
"3*x^3 + 11*x^2 + 9*x + 14"
```

> Adição

```
ghci> showPolynomialTrimmed (sortAndNormalize testPolynomial)
"3*x^3 + 6*x^2 + 5*x + 5*x*y^2 + 2*y^3 + 7*y^2 + 5*y + 14"
ghci> showPolynomialTrimmed (sortAndNormalize testPolynomial2)
"2*y^2 + 3*y + 3*z*w + 5"
ghci> showPolynomialTrimmed (sortAndNormalize (addPolynomial testPolynomial testPolynomial2))
"3*x^3 + 6*x^2 + 5*x + 5*x*y^2 + 2*y^3 + 9*y^2 + 8*y + 3*z*w + 19"
```

> Multiplicação

```
ghci> showPolynomialTrimmed createPolynomial
"2*z*w + 8*z^2 + 4*z + 4"
ghci> showPolynomialTrimmed createPolynomial2
"3*z*w + 2*y^2 + 3*y + 5"
ghci> showPolynomialTrimmed (sortAndNormalize (multiplyPolynomials createPolynomial cr
eatePolynomial2))
"6*w^2*z^2 + 24*w*z^3 + 12*w*z^2 + 4*w*y^2*z + 6*w*y*z + 8*y^2 + 12*y + 8*y^2*z + 16*y
^2*z^2 + 24*y*z^2 + 12*y*z + 40*z^2 + 20*z + 22*z*w + 20"
```

> Derivada

```
ghci> showPolynomialTrimmed (sortAndNormalize testPolynomial)
"3*x^3 + 6*x^2 + 5*x + 5*y^2 + 4*y + 14"
ghci> showPolynomialTrimmed (derivePartial "x" (sortAndNormalize testPolynomial))
"9*x^2 + 12*x + 5"
```

> Conjunto de operações: Soma do polinómio 1 com o polinómio 2, seguida de uma nova soma do novo polinómio com o polinómio 1, finalizada com uma derivada parcial em ordem a x.

```
ghci> showPolynomialTrimmed (sortAndNormalize testPolynomial)
"3*x^3 + 6*x^2 + 5*x + 5*y^2 + 4*y + 14"
ghci> showPolynomialTrimmed (sortAndNormalize testPolynomial2)
"2*y^2 + 3*y + 3*z*w + 5"
ghci> showPolynomialTrimmed (sortAndNormalize (addPolynomial testPolynomial (addPolynomial testPolynomial testPolynomial2)))
"6*x^3 + 12*x^2 + 10*x + 12*y^2 + 11*y + 3*z*w + 33"
ghci> showPolynomialTrimmed (sortAndNormalize (derivePartial "x" (addPolynomial testPolynomial (addPolynomial testPolynomial testPolynomial2))))
"18*x^2 + 24*x + 10"
```

Nota sobre parsing

Infelizmente, o parsing de polinómios completos escritos no terminal pelo utilizador não foi implementado neste trabalho, apenas o parsing de polinómios compostos em strings feitas no código. No entanto, existe uma função main que serve de alternativa mais amigável ao utilizador para testar as funções principais e aceita input do utilizador, tanto na escolha de opções como na escolha de variável a derivar nas funções de derivação. Embora não sirva como substituto, foi uma forma de tornar mais acessível o teste do código em geral. De qualquer das formas, existem funções separadas de teste fora da função main no módulo Tests e, claro, a chamada de funções diretamente também é uma possibilidade.