Cálculo de Programas
Trabalho Prático
MiEI+LCC — 2020/21

Departamento de Informática
Universidade do Minho

Junho de 2021

Grupo nr.	108 (preencher)
a78914	Ricardo Rodrigues Martins
a93752	Hugo Rafael Lima Pereira
a93785	Ricardo Miguel Santos Gomes

1 Preâmbulo

Cálculo de Programas tem como objectivo principal ensinar a programação de computadores como uma disciplina científica. Para isso parte-se de um repertório de *combinadores* que formam uma álgebra da programação (conjunto de leis universais e seus corolários) e usam-se esses combinadores para construir programas *composicionalmente*, isto é, agregando programas já existentes.

Na sequência pedagógica dos planos de estudo dos dois cursos que têm esta disciplina, opta-se pela aplicação deste método à programação em Haskell (sem prejuízo da sua aplicação a outras linguagens funcionais). Assim, o presente trabalho prático coloca os alunos perante problemas concretos que deverão ser implementados em Haskell. Há ainda um outro objectivo: o de ensinar a documentar programas, a validá-los e a produzir textos técnico-científicos de qualidade.

2 Documentação

Para cumprir de forma integrada os objectivos enunciados acima vamos recorrer a uma técnica de programação dita "literária" [1], cujo princípio base é o seguinte:

> *Um programa e a sua documentação devem coincidir.*

Por outras palavras, o código fonte e a documentação de um programa deverão estar no mesmo ficheiro.

O ficheiro `cp2021t.pdf` que está a ler é já um exemplo de programação literária: foi gerado a partir do texto fonte `cp2021t.lhs`[1] que encontrará no material pedagógico desta disciplina descompactando o ficheiro `cp2021t.zip` e executando:

```
$ lhs2TeX cp2021t.lhs > cp2021t.tex
$ pdflatex cp2021t
```

em que `lhs2tex` é um pre-processador que faz "pretty printing" de código Haskell em LaTeX e que deve desde já instalar executando

```
$ cabal install lhs2tex --lib
```

Por outro lado, o mesmo ficheiro `cp2021t.lhs` é executável e contém o "kit" básico, escrito em Haskell, para realizar o trabalho. Basta executar

```
$ ghci cp2021t.lhs
```

[1] O suffixo 'lhs' quer dizer *literate Haskell*.

Abra o ficheiro `cp2021t.lhs` no seu editor de texto preferido e verifique que assim é: todo o texto que se encontra dentro do ambiente

```
\begin{code}
...
\end{code}
```

é seleccionado pelo GHCi para ser executado.

3 Como realizar o trabalho

Este trabalho teórico-prático deve ser realizado por grupos de 3 (ou 4) alunos. Os detalhes da avaliação (datas para submissão do relatório e sua defesa oral) são os que forem publicados na página da disciplina na *internet*.

Recomenda-se uma abordagem participativa dos membros do grupo de trabalho por forma a poderem responder às questões que serão colocadas na *defesa oral* do relatório.

Em que consiste, então, o *relatório* a que se refere o parágrafo anterior? É a edição do texto que está a ser lido, preenchendo o anexo D com as respostas. O relatório deverá conter ainda a identificação dos membros do grupo de trabalho, no local respectivo da folha de rosto.

Para gerar o PDF integral do relatório deve-se ainda correr os comando seguintes, que actualizam a bibliografia (com BibTEX) e o índice remissivo (com `makeindex`),

```
$ bibtex cp2021t.aux
$ makeindex cp2021t.idx
```

e recompilar o texto como acima se indicou. Dever-se-á ainda instalar o utilitário QuickCheck, que ajuda a validar programas em Haskell e a biblioteca Gloss para geração de gráficos 2D:

```
$ cabal install QuickCheck gloss --lib
```

Para testar uma propriedade QuickCheck *prop*, basta invocá-la com o comando:

```
> quickCheck prop
+++ OK, passed 100 tests.
```

Pode-se ainda controlar o número de casos de teste e sua complexidade, como o seguinte exemplo mostra:

```
> quickCheckWith stdArgs { maxSuccess = 200, maxSize = 10 } prop
+++ OK, passed 200 tests.
```

Qualquer programador tem, na vida real, de ler e analisar (muito!) código escrito por outros. No anexo C disponibiliza-se algum código Haskell relativo aos problemas que se seguem. Esse anexo deverá ser consultado e analisado à medida que isso for necessário.

3.1 Stack

O Stack é um programa útil para criar, gerir e manter projetos em Haskell. Um projeto criado com o Stack possui uma estrutura de pastas muito específica:

- Os módulos auxiliares encontram-se na pasta *src*.
- O módulos principal encontra-se na pasta *app*.
- A lista de depêndencias externas encontra-se no ficheiro *package.yaml*.

Pode aceder ao GHCi utilizando o comando:

```
stack ghci
```

Garanta que se encontra na pasta mais externa **do projeto**. A primeira vez que correr este comando as depêndencias externas serão instaladas automaticamente.

Para gerar o PDF, garanta que se encontra na diretoria *app*.

Problema 1

Os *tipos de dados algébricos* estudados ao longo desta disciplina oferecem uma grande capacidade expressiva ao programador. Graças à sua flexibilidade, torna-se trivial implementar DSLs e até mesmo linguagens de programação.

Paralelamente, um tópico bastante estudado no âmbito de Deep Learning é a derivação automática de expressões matemáticas, por exemplo, de derivadas. Duas técnicas que podem ser utilizadas para o cálculo de derivadas são:

- *Symbolic differentiation*
- *Automatic differentiation*

Symbolic differentiation consiste na aplicação sucessiva de transformações (leia-se: funções) que sejam congruentes com as regras de derivação. O resultado final será a expressão da derivada.

O leitor atento poderá notar um problema desta técnica: a expressão inicial pode crescer de forma descontrolada, levando a um cálculo pouco eficiente. *Automatic differentiation* tenta resolver este problema, calculando **o valor** da derivada da expressão em todos os passos. Para tal, é necessário calcular o valor da expressão **e** o valor da sua derivada.

Vamos de seguida definir uma linguagem de expressões matemáticas simples e implementar as duas técnicas de derivação automática. Para isso, seja dado o seguinte tipo de dados,

$$\begin{aligned}&\textbf{data}\ ExpAr\ a = X\\ &\quad \mid N\ a\\ &\quad \mid Bin\ BinOp\ (ExpAr\ a)\ (ExpAr\ a)\\ &\quad \mid Un\ UnOp\ (ExpAr\ a)\\ &\quad \textbf{deriving}\ (Eq, Show)\end{aligned}$$

onde $BinOp$ e $UnOp$ representam operações binárias e unárias, respectivamente:

$$\begin{aligned}&\textbf{data}\ BinOp = Sum\\ &\quad \mid Product\\ &\quad \textbf{deriving}\ (Eq, Show)\\ &\textbf{data}\ UnOp = Negate\\ &\quad \mid E\\ &\quad \textbf{deriving}\ (Eq, Show)\end{aligned}$$

O construtor E simboliza o exponencial de base e.

Assim, cada expressão pode ser uma variável, um número, uma operação binária aplicada às devidas expressões, ou uma operação unária aplicada a uma expressão. Por exemplo,

$$Bin\ Sum\ X\ (N\ 10)$$

designa $x + 10$ na notação matemática habitual.

1. A definição das funções $inExpAr$ e $baseExpAr$ para este tipo é a seguinte:

$$\begin{aligned}&inExpAr = [\underline{X}, num\_ops]\ \textbf{where}\\ &\quad num\_ops = [N, ops]\\ &\quad ops = [bin, \widehat{Un}]\\ &\quad bin\ (op, (a, b)) = Bin\ op\ a\ b\\ &baseExpAr\ f\ g\ h\ j\ k\ l\ z = f + (g + (h \times (j \times k) + l \times z))\end{aligned}$$

Defina as funções $outExpAr$ e $recExpAr$, e teste as propriedades que se seguem.

Propriedade [QuickCheck] 1 *$inExpAr$ e $outExpAr$ são testemunhas de um isomorfismo, isto é, $inExpAr \cdot outExpAr = id$ e $outExpAr \cdot idExpAr = id$:*

$$\begin{aligned}&prop\_in\_out\_idExpAr :: (Eq\ a) \Rightarrow ExpAr\ a \rightarrow Bool\\ &prop\_in\_out\_idExpAr = inExpAr \cdot outExpAr \equiv id\end{aligned}$$

$$\begin{aligned}&prop\_out\_in\_idExpAr :: (Eq\ a) \Rightarrow OutExpAr\ a \rightarrow Bool\\ &prop\_out\_in\_idExpAr = outExpAr \cdot inExpAr \equiv id\end{aligned}$$

2. Dada uma expressão aritmética e um escalar para substituir o X, a função

$$eval\_exp :: Floating\ a \Rightarrow a \rightarrow (ExpAr\ a) \rightarrow a$$

calcula o resultado da expressão. Na página 12 esta função está expressa como um catamorfismo. Defina o respectivo gene e, de seguida, teste as propriedades:

Propriedade [QuickCheck] 2 *A função $eval\_exp$ respeita os elementos neutros das operações.*

$$\begin{array}{l}
prop_sum_idr :: (Floating\ a, Real\ a) \Rightarrow a \rightarrow ExpAr\ a \rightarrow Bool \\
prop_sum_idr\ a\ exp = eval_exp\ a\ exp \stackrel{?}{=} sum_idr\ \mathbf{where} \\
\quad sum_idr = eval_exp\ a\ (Bin\ Sum\ exp\ (N\ 0)) \\
prop_sum_idl :: (Floating\ a, Real\ a) \Rightarrow a \rightarrow ExpAr\ a \rightarrow Bool \\
prop_sum_idl\ a\ exp = eval_exp\ a\ exp \stackrel{?}{=} sum_idl\ \mathbf{where} \\
\quad sum_idl = eval_exp\ a\ (Bin\ Sum\ (N\ 0)\ exp) \\
prop_product_idr :: (Floating\ a, Real\ a) \Rightarrow a \rightarrow ExpAr\ a \rightarrow Bool \\
prop_product_idr\ a\ exp = eval_exp\ a\ exp \stackrel{?}{=} prod_idr\ \mathbf{where} \\
\quad prod_idr = eval_exp\ a\ (Bin\ Product\ exp\ (N\ 1)) \\
prop_product_idl :: (Floating\ a, Real\ a) \Rightarrow a \rightarrow ExpAr\ a \rightarrow Bool \\
prop_product_idl\ a\ exp = eval_exp\ a\ exp \stackrel{?}{=} prod_idl\ \mathbf{where} \\
\quad prod_idl = eval_exp\ a\ (Bin\ Product\ (N\ 1)\ exp) \\
prop_e_id :: (Floating\ a, Real\ a) \Rightarrow a \rightarrow Bool \\
prop_e_id\ a = eval_exp\ a\ (Un\ E\ (N\ 1)) \equiv expd\ 1 \\
prop_negate_id :: (Floating\ a, Real\ a) \Rightarrow a \rightarrow Bool \\
prop_negate_id\ a = eval_exp\ a\ (Un\ Negate\ (N\ 0)) \equiv 0
\end{array}$$

Propriedade [QuickCheck] 3 *Negar duas vezes uma expressão tem o mesmo valor que não fazer nada.*

$$\begin{array}{l}
prop_double_negate :: (Floating\ a, Real\ a) \Rightarrow a \rightarrow ExpAr\ a \rightarrow Bool \\
prop_double_negate\ a\ exp = eval_exp\ a\ exp \stackrel{?}{=} eval_exp\ a\ (Un\ Negate\ (Un\ Negate\ exp))
\end{array}$$

3. É possível otimizar o cálculo do valor de uma expressão aritmética tirando proveito dos elementos absorventes de cada operação. Implemente os genes da função

$$optmize\_eval :: (Floating\ a, Eq\ a) \Rightarrow a \rightarrow (ExpAr\ a) \rightarrow a$$

que se encontra na página 12 expressa como um hilomorfismo[2] e teste as propriedades:

Propriedade [QuickCheck] 4 *A função $optimize\_eval$ respeita a semântica da função $eval$.*

$$\begin{array}{l}
prop_optimize_respects_semantics :: (Floating\ a, Real\ a) \Rightarrow a \rightarrow ExpAr\ a \rightarrow Bool \\
prop_optimize_respects_semantics\ a\ exp = eval_exp\ a\ exp \stackrel{?}{=} optmize_eval\ a\ exp
\end{array}$$

4. Para calcular a derivada de uma expressão, é necessário aplicar transformações à expressão original que respeitem as regras das derivadas:[3]

   - Regra da soma:

$$\frac{d}{dx}(f(x) + g(x)) = \frac{d}{dx}(f(x)) + \frac{d}{dx}(g(x))$$

[2] Qual é a vantagem de implementar a função $optimize\_eval$ utilizando um hilomorfismo em vez de utilizar um catamorfismo com um gene "inteligente"?

[3] Apesar da adição e multiplicação gozarem da propriedade comutativa, há que ter em atenção a ordem das operações por causa dos testes.

- Regra do produto:

$$\frac{d}{dx}(f(x)g(x)) = f(x) \cdot \frac{d}{dx}(g(x)) + \frac{d}{dx}(f(x)) \cdot g(x)$$

Defina o gene do catamorfismo que ocorre na função

$$sd :: Floating\ a \Rightarrow ExpAr\ a \rightarrow ExpAr\ a$$

que, dada uma expressão aritmética, calcula a sua derivada. Testes a fazer, de seguida:

Propriedade [QuickCheck] 5 *A função* sd *respeita as regras de derivação.*

$$\begin{array}{l}
prop_const_rule :: (Real\ a, Floating\ a) \Rightarrow a \rightarrow Bool\\
prop_const_rule\ a = sd\ (N\ a) \equiv N\ 0\\
prop_var_rule :: Bool\\
prop_var_rule = sd\ X \equiv N\ 1\\
prop_sum_rule :: (Real\ a, Floating\ a) \Rightarrow ExpAr\ a \rightarrow ExpAr\ a \rightarrow Bool\\
prop_sum_rule\ exp1\ exp2 = sd\ (Bin\ Sum\ exp1\ exp2) \equiv sum_rule\ \mathbf{where}\\
\quad sum_rule = Bin\ Sum\ (sd\ exp1)\ (sd\ exp2)\\
prop_product_rule :: (Real\ a, Floating\ a) \Rightarrow ExpAr\ a \rightarrow ExpAr\ a \rightarrow Bool\\
prop_product_rule\ exp1\ exp2 = sd\ (Bin\ Product\ exp1\ exp2) \equiv prod_rule\ \mathbf{where}\\
\quad prod_rule = Bin\ Sum\ (Bin\ Product\ exp1\ (sd\ exp2))\ (Bin\ Product\ (sd\ exp1)\ exp2)\\
prop_e_rule :: (Real\ a, Floating\ a) \Rightarrow ExpAr\ a \rightarrow Bool\\
prop_e_rule\ exp = sd\ (Un\ E\ exp) \equiv Bin\ Product\ (Un\ E\ exp)\ (sd\ exp)\\
prop_negate_rule :: (Real\ a, Floating\ a) \Rightarrow ExpAr\ a \rightarrow Bool\\
prop_negate_rule\ exp = sd\ (Un\ Negate\ exp) \equiv Un\ Negate\ (sd\ exp)
\end{array}$$

5. Como foi visto, *Symbolic differentiation* não é a técnica mais eficaz para o cálculo do valor da derivada de uma expressão. *Automatic differentiation* resolve este problema cálculando o valor da derivada em vez de manipular a expressão original.

   Defina o gene do catamorfismo que ocorre na função

   $$ad :: Floating\ a \Rightarrow a \rightarrow ExpAr\ a \rightarrow a$$

   que, dada uma expressão aritmética e um ponto, calcula o valor da sua derivada nesse ponto, sem transformar manipular a expressão original. Testes a fazer, de seguida:

   Propriedade [QuickCheck] 6 *Calcular o valor da derivada num ponto* r *via* ad *é equivalente a calcular a derivada da expressão e avalia-la no ponto* r*.*

   $$\begin{array}{l}
prop_congruent :: (Floating\ a, Real\ a) \Rightarrow a \rightarrow ExpAr\ a \rightarrow Bool\\
prop_congruent\ a\ exp = ad\ a\ exp \stackrel{?}{=} eval_exp\ a\ (sd\ exp)
\end{array}$$

Problema 2

Nesta disciplina estudou-se como fazer programação dinâmica por cálculo, recorrendo à lei de recursividade mútua.[4]

Para o caso de funções sobre os números naturais ($\mathbb{N}_0$, com functor $\mathsf{F}\ X = 1 + X$) é fácil derivar-se da lei que foi estudada uma *regra de algibeira* que se pode ensinar a programadores que não tenham estudado Cálculo de Programas. Apresenta-se de seguida essa regra, tomando como exemplo o cálculo do ciclo-for que implementa a função de Fibonacci, recordar o sistema

$$\begin{array}{l}
fib\ 0 = 1\\
fib\ (n+1) = f\ n
\end{array}$$

[4] Lei (3.94) em [2], página 98.

$$
\begin{array}{l}
f\ 0 = 1 \\
f\ (n+1) = \mathit{fib}\ n + f\ n
\end{array}
$$

Obter-se-á de imediato

$$
\begin{array}{l}
\mathit{fib}' = \pi_1 \cdot \mathsf{for}\ \mathit{loop}\ \mathit{init}\ \mathbf{where} \\
\quad \mathit{loop}\ (\mathit{fib}, f) = (f, \mathit{fib} + f) \\
\quad \mathit{init} = (1, 1)
\end{array}
$$

usando as regras seguintes:

- O corpo do ciclo loop terá tantos argumentos quanto o número de funções mutuamente recursivas.
- Para as variáveis escolhem-se os próprios nomes das funções, pela ordem que se achar conveniente.[5]
- Para os resultados vão-se buscar as expressões respectivas, retirando a variável n.
- Em init coleccionam-se os resultados dos casos de base das funções, pela mesma ordem.

Mais um exemplo, envolvendo polinómios do segundo grau $ax^2 + bx + c$ em $\mathbb{N}_0$. Seguindo o método estudado nas aulas[6], de $f\ x = ax^2 + bx + c$ derivam-se duas funções mutuamente recursivas:

$$
\begin{array}{l}
f\ 0 = c \\
f\ (n+1) = f\ n + k\ n \\
k\ 0 = a + b \\
k\ (n+1) = k\ n + 2\ a
\end{array}
$$

Seguindo a regra acima, calcula-se de imediato a seguinte implementação, em Haskell:

$$
\begin{array}{l}
f'\ a\ b\ c = \pi_1 \cdot \mathsf{for}\ \mathit{loop}\ \mathit{init}\ \mathbf{where} \\
\quad \mathit{loop}\ (f, k) = (f + k, k + 2 * a) \\
\quad \mathit{init} = (c, a + b)
\end{array}
$$

O que se pede então, nesta pergunta? Dada a fórmula que dá o n-ésimo número de Catalan,

$$
C_n = \frac{(2n)!}{(n+1)!(n!)} \tag{1}
$$

derivar uma implementação de C_n que não calcule factoriais nenhuns. Isto é, derivar um ciclo-for

$$
\mathit{cat} = \cdots \cdot \mathsf{for}\ \mathit{loop}\ \mathit{init}\ \mathbf{where}\ \cdots
$$

que implemente esta função.

Propriedade [QuickCheck] 7 *A função proposta coincidem com a definição dada:*

$$
\mathit{prop\_cat} = (\geqslant 0) \Rightarrow (\mathit{catdef} \equiv \mathit{cat})
$$

Sugestão: Começar por estudar muito bem o processo de cálculo dado no anexo B para o problema (semelhante) da função exponencial.

Problema 3

As curvas de Bézier, designação dada em honra ao engenheiro Pierre Bézier, são curvas ubíquas na área de computação gráfica, animação e modelação. Uma curva de Bézier é uma curva paramétrica, definida por um conjunto $\{P_0, ..., P_N\}$ de pontos de controlo, onde N é a ordem da curva.

O algoritmo de *De Casteljau* é um método recursivo capaz de calcular curvas de Bézier num ponto. Apesar de ser mais lento do que outras abordagens, este algoritmo é numericamente mais estável, trocando velocidade por correção.

[5] Podem obviamente usar-se outros símbolos, mas numa primeira leitura dá jeito usarem-se tais nomes.

[6] Secção 3.17 de [2] e tópico Recursividade mútua nos vídeos das aulas teóricas.

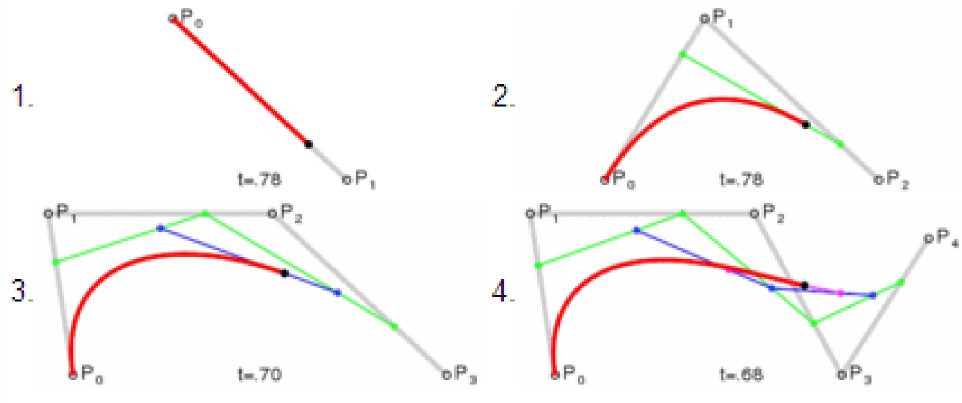

Figura 1: Exemplos de curvas de Bézier retirados da Wikipedia.

De forma sucinta, o valor de uma curva de Bézier de um só ponto $\{P_0\}$ (ordem 0) é o próprio ponto P_0. O valor de uma curva de Bézier de ordem N é calculado através da interpolação linear da curva de Bézier dos primeiros $N-1$ pontos e da curva de Bézier dos últimos $N-1$ pontos.

A interpolação linear entre 2 números, no intervalo $[0,1]$, é dada pela seguinte função:

$$
\begin{array}{l}
linear1d :: \mathbb{Q} \to \mathbb{Q} \to OverTime\ \mathbb{Q} \\
linear1d\ a\ b = formula\ a\ b\ \textbf{where} \\
\quad formula :: \mathbb{Q} \to \mathbb{Q} \to Float \to \mathbb{Q} \\
\quad formula\ x\ y\ t = ((1.0 :: \mathbb{Q}) - (to_{\mathbb{Q}}\ t)) * x + (to_{\mathbb{Q}}\ t) * y
\end{array}
$$

A interpolação linear entre 2 pontos de dimensão N é calculada através da interpolação linear de cada dimensão.

O tipo de dados $NPoint$ representa um ponto com N dimensões.

$$\textbf{type}\ NPoint = [\mathbb{Q}]$$

Por exemplo, um ponto de 2 dimensões e um ponto de 3 dimensões podem ser representados, respetivamente, por:

$$
\begin{array}{l}
p2d = [1.2, 3.4] \\
p3d = [0.2, 10.3, 2.4]
\end{array}
$$

O tipo de dados $OverTime\ a$ representa um termo do tipo a num dado instante (dado por um $Float$).

$$\textbf{type}\ OverTime\ a = Float \to a$$

O anexo C tem definida a função

$$calcLine :: NPoint \to (NPoint \to OverTime\ NPoint)$$

que calcula a interpolação linear entre 2 pontos, e a função

$$deCasteljau :: [NPoint] \to OverTime\ NPoint$$

que implementa o algoritmo respectivo.

1. Implemente $calcLine$ como um catamorfismo de listas, testando a sua definição com a propriedade:

   Propriedade [QuickCheck] 8 *Definição alternativa.*

   $$
   \begin{array}{l}
   prop\_calcLine\_def :: NPoint \to NPoint \to Float \to Bool \\
   prop\_calcLine\_def\ p\ q\ d = calcLine\ p\ q\ d \equiv zipWithM\ linear1d\ p\ q\ d
   \end{array}
   $$

2. Implemente a função $deCasteljau$ como um hilomorfismo, testando agora a propriedade:

Propriedade [QuickCheck] 9 *Curvas de Bézier são simétricas.*

$$
\begin{array}{l}
\mathit{prop\_bezier\_sym} :: [[\mathbb{Q}]] \rightarrow \mathit{Gen\ Bool} \\
\mathit{prop\_bezier\_sym}\ l = \mathit{all}\ ({<}\Delta) \cdot \mathit{calc\_difs} \cdot \mathit{bezs}\ \langle\$\rangle\ \mathit{elements}\ \mathit{ps}\ \mathbf{where} \\
\quad \mathit{calc\_difs} = (\lambda(x,y) \rightarrow \mathit{zipWith}\ (\lambda w\ v \rightarrow \mathbf{if}\ w \geqslant v\ \mathbf{then}\ w - v\ \mathbf{else}\ v - w)\ x\ y) \\
\quad \mathit{bezs}\ t = (\mathit{deCasteljau}\ l\ t, \mathit{deCasteljau}\ (\mathit{reverse}\ l)\ (\mathit{from}_{\mathbb{Q}}\ (1 - (\mathit{to}_{\mathbb{Q}}\ t)))) \\
\quad \Delta = 1\mathrm{e}{-2}
\end{array}
$$

3. Corra a função $\mathit{runBezier}$ e aprecie o seu trabalho[7] clicando na janela que é aberta (que contém, a verde, um ponto inicila) com o botão esquerdo do rato para adicionar mais pontos. A tecla Delete apaga o ponto mais recente.

Problema 4

Seja dada a fórmula que calcula a média de uma lista não vazia x,

$$
\mathit{avg}\ x = \frac{1}{k}\sum_{i=1}^{k} x_i \tag{2}
$$

onde $k = \mathit{length}\ x$. Isto é, para sabermos a média de uma lista precisamos de dois catamorfismos: o que faz o somatório e o que calcula o comprimento a lista. Contudo, é facil de ver que

$$
\begin{array}{l}
\mathit{avg}\ [a] = a \\
\mathit{avg}(a : x) = \frac{1}{k+1}(a + \sum_{i=1}^{k} x_i) = \frac{a + k(\mathit{avg}\ x)}{k+1}\ \text{para}\ k = \mathit{length}\ x
\end{array}
$$

Logo avg está em recursividade mútua com length e o par de funções pode ser expresso por um único catamorfismo, significando que a lista apenas é percorrida uma vez.

1. Recorra à lei de recursividade mútua para derivar a função $\mathit{avg\_aux} = (\!|[b, q]|\!)$ tal que $\mathit{avg\_aux} = \langle \mathit{avg}, \mathit{length} \rangle$ em listas não vazias.
2. Generalize o raciocínio anterior para o cálculo da média de todos os elementos de uma LTree recorrendo a uma única travessia da árvore (i.e. catamorfismo).

Verifique as suas funções testando a propriedade seguinte:

Propriedade [QuickCheck] 10 *A média de uma lista não vazia e de uma LTree com os mesmos elementos coincide, a menos de um erro de 0.1 milésimas:*

$$
\begin{array}{l}
\mathit{prop\_avg} :: [\mathit{Double}] \rightarrow \mathit{Property} \\
\mathit{prop\_avg} = \mathit{nonempty} \Rightarrow \mathit{diff} \leqslant \underline{0.000001}\ \mathbf{where} \\
\quad \mathit{diff}\ l = \mathit{avg}\ l - (\mathit{avgLTree} \cdot \mathit{genLTree})\ l \\
\quad \mathit{genLTree} = [\!(\mathit{lsplit})\!] \\
\quad \mathit{nonempty} = ({>}[\,])
\end{array}
$$

Problema 5

(**NB**: Esta questão é **opcional** e funciona como **valorização** apenas para os alunos que desejarem fazê-la.)

Existem muitas linguagens funcionais para além do Haskell, que é a linguagem usada neste trabalho prático. Uma delas é o F# da Microsoft. Na directoria `fsharp` encontram-se os módulos Cp, Nat e LTree codificados em F#. O que se pede é a biblioteca BTree escrita na mesma linguagem.

Modo de execução: o código que tiverem produzido nesta pergunta deve ser colocado entre o `\begin{verbatim}` e o `\end{verbatim}` da correspondente parte do anexo D. Para além disso, os grupos podem demonstrar o código na oral.

[7] A representação em Gloss é uma adaptação de um projeto de Harold Cooper.

Anexos

A Como exprimir cálculos e diagramas em LaTeX/lhs2tex

Como primeiro exemplo, estudar o texto fonte deste trabalho para obter o efeito:[8]

$$
\begin{array}{ll}
 & id = \langle f, g \rangle \\
\equiv & \qquad \{ \text{ universal property } \} \\
 & \left\{ \begin{array}{l} \pi_1 \cdot id = f \\ \pi_2 \cdot id = g \end{array} \right. \\
\equiv & \qquad \{ \text{ identity } \} \\
 & \left\{ \begin{array}{l} \pi_1 = f \\ \pi_2 = g \end{array} \right. \\
\Box &
\end{array}
$$

Os diagramas podem ser produzidos recorrendo à *package* LaTeX xymatrix, por exemplo:

$$
\begin{tikzcd}
\mathbb{N}_0 \arrow[d, "(\!|g|\!)"'] & 1 + \mathbb{N}_0 \arrow[l, "\mathsf{in}"'] \arrow[d, "id + (\!|g|\!)"] \\
B & 1 + B \arrow[l, "g"]
\end{tikzcd}
$$

B Programação dinâmica por recursividade múltipla

Neste anexo dão-se os detalhes da resolução do Exercício 3.30 dos apontamentos da disciplina[9], onde se pretende implementar um ciclo que implemente o cálculo da aproximação até $i = n$ da função exponencial $exp\ x = e^x$, via série de Taylor:

$$
exp\ x \quad = \quad \sum_{i=0}^{\infty} \frac{x^i}{i!} \tag{3}
$$

Seja $e\ x\ n = \sum_{i=0}^{n} \frac{x^i}{i!}$ a função que dá essa aproximação. É fácil de ver que $e\ x\ 0 = 1$ e que $e\ x\ (n+1) = e\ x\ n + \frac{x^{n+1}}{(n+1)!}$. Se definirmos $h\ x\ n = \frac{x^{n+1}}{(n+1)!}$ teremos $e\ x$ e $h\ x$ em recursividade mútua. Se repetirmos o processo para $h\ x\ n$ etc obteremos no total três funções nessa mesma situação:

$$
\begin{array}{l}
e\ x\ 0 = 1 \\
e\ x\ (n+1) = h\ x\ n + e\ x\ n \\
h\ x\ 0 = x \\
h\ x\ (n+1) = x\ /\ (s\ n) * h\ x\ n \\
s\ 0 = 2 \\
s\ (n+1) = 1 + s\ n
\end{array}
$$

Segundo a *regra de algibeira* descrita na página 3.1 deste enunciado, ter-se-á, de imediato:

$$
\begin{array}{l}
e'\ x = prj \cdot \mathsf{for}\ loop\ init\ \mathbf{where} \\
\quad init = (1, x, 2) \\
\quad loop\ (e, h, s) = (h + e, x\ /\ s * h, 1 + s) \\
\quad prj\ (e, h, s) = e
\end{array}
$$

[8] Exemplos tirados de [2].

[9] Cf. [2], página 102.

C Código fornecido

Problema 1

$$
\begin{array}{l}
\mathit{expd} :: \mathit{Floating}\ a \Rightarrow a \rightarrow a \\
\mathit{expd} = \mathit{Prelude.exp} \\
\mathbf{type}\ \mathit{OutExpAr}\ a = () + (a + ((\mathit{BinOp}, (\mathit{ExpAr}\ a, \mathit{ExpAr}\ a)) + (\mathit{UnOp}, \mathit{ExpAr}\ a)))
\end{array}
$$

Problema 2

Definição da série de Catalan usando factoriais (1):

$$\mathit{catdef}\ n = (2 * n)! \div ((n + 1)! * n!)$$

Oráculo para inspecção dos primeiros 26 números de Catalan[10]:

$$
\begin{array}{l}
\mathit{oracle} = [\\
\quad 1, 1, 2, 5, 14, 42, 132, 429, 1430, 4862, 16796, 58786, 208012, 742900, 2674440, 9694845, \\
\quad 35357670, 129644790, 477638700, 1767263190, 6564120420, 24466267020, \\
\quad 91482563640, 343059613650, 1289904147324, 4861946401452 \\
\quad]
\end{array}
$$

Problema 3

Algoritmo:

$$
\begin{array}{l}
\mathit{deCasteljau} :: [\mathit{NPoint}] \rightarrow \mathit{OverTime}\ \mathit{NPoint} \\
\mathit{deCasteljau}\ [\,] = \mathit{nil} \\
\mathit{deCasteljau}\ [p] = \underline{p} \\
\mathit{deCasteljau}\ l = \lambda pt \rightarrow (\mathit{calcLine}\ (p\ pt)\ (q\ pt))\ pt\ \mathbf{where} \\
\quad p = \mathit{deCasteljau}\ (\mathit{init}\ l) \\
\quad q = \mathit{deCasteljau}\ (\mathit{tail}\ l)
\end{array}
$$

Função auxiliar:

$$
\begin{array}{l}
\mathit{calcLine} :: \mathit{NPoint} \rightarrow (\mathit{NPoint} \rightarrow \mathit{OverTime}\ \mathit{NPoint}) \\
\mathit{calcLine}\ [\,] = \underline{\mathit{nil}} \\
\mathit{calcLine}\ (p : x) = \overline{g}\ p\ (\mathit{calcLine}\ x)\ \mathbf{where} \\
\quad g :: (\mathbb{Q}, \mathit{NPoint} \rightarrow \mathit{OverTime}\ \mathit{NPoint}) \rightarrow (\mathit{NPoint} \rightarrow \mathit{OverTime}\ \mathit{NPoint}) \\
\quad g\ (d, f)\ l = \mathbf{case}\ l\ \mathbf{of} \\
\qquad [\,] \rightarrow \mathit{nil} \\
\qquad (x : \mathit{xs}) \rightarrow \lambda z \rightarrow \mathit{concat}\ \$\ (\mathit{sequenceA}\ [\mathit{singl} \cdot \mathit{linear1d}\ d\ x, f\ \mathit{xs}])\ z
\end{array}
$$

2D:

$$
\begin{array}{l}
\mathit{bezier2d} :: [\mathit{NPoint}] \rightarrow \mathit{OverTime}\ (\mathit{Float}, \mathit{Float}) \\
\mathit{bezier2d}\ [\,] = \underline{(0, 0)} \\
\mathit{bezier2d}\ l = \lambda z \rightarrow (\mathit{from}_{\mathbb{Q}} \times \mathit{from}_{\mathbb{Q}}) \cdot (\lambda[x, y] \rightarrow (x, y))\ \$\ ((\mathit{deCasteljau}\ l)\ z)
\end{array}
$$

Modelo:

$$
\begin{array}{l}
\mathbf{data}\ \mathit{World} = \mathit{World}\ \{\mathit{points} :: [\mathit{NPoint}] \\
\quad , \mathit{time} :: \mathit{Float} \\
\quad \} \\
\mathit{initW} :: \mathit{World} \\
\mathit{initW} = \mathit{World}\ [\,]\ 0
\end{array}
$$

[10] Fonte: Wikipedia.

$$
\begin{array}{l}
tick :: Float \to World \to World \\
tick\ dt\ world = world\ \{time = (time\ world) + dt\}
\end{array}
$$

$$
\begin{array}{l}
actions :: Event \to World \to World \\
actions\ (EventKey\ (MouseButton\ LeftButton)\ Down\ \_\ p)\ world = \\
\quad world\ \{points = (points\ world) +\!\!+ [(\lambda(x, y) \to \mathsf{map}\ \ to_{\mathbb{Q}}\ [x, y])\ p]\} \\
actions\ (EventKey\ (SpecialKey\ KeyDelete)\ Down\ \_\ \_)\ world = \\
\quad world\ \{points = cond\ (\equiv [\,])\ id\ init\ (points\ world)\} \\
actions\ \_\ world = world
\end{array}
$$

$$
\begin{array}{l}
scaleTime :: World \to Float \\
scaleTime\ w = (1 + cos\ (time\ w))\ /\ 2
\end{array}
$$

$$
\begin{array}{l}
bezier2dAtTime :: World \to (Float, Float) \\
bezier2dAtTime\ w = (bezier2dAt\ w)\ (scaleTime\ w)
\end{array}
$$

$$
\begin{array}{l}
bezier2dAt :: World \to OverTime\ (Float, Float) \\
bezier2dAt\ w = bezier2d\ (points\ w)
\end{array}
$$

$$
\begin{array}{l}
thicCirc :: Picture \\
thicCirc = ThickCircle\ 4\ 10
\end{array}
$$

$$
\begin{array}{l}
ps :: [Float] \\
ps = \mathsf{map}\ \ from_{\mathbb{Q}}\ ps'\ \mathbf{where} \\
\quad ps' :: [\mathbb{Q}] \\
\quad ps' = [0, 0.01\,..\,1] \quad \text{-- interval}
\end{array}
$$

Gloss:

$$
\begin{array}{l}
picture :: World \to Picture \\
picture\ world = Pictures \\
\quad [animateBezier\ (scaleTime\ world)\ (points\ world) \\
\quad , Color\ white \cdot Line \cdot \mathsf{map}\ \ (bezier2dAt\ world)\ \$\ ps \\
\quad , Color\ blue \cdot Pictures\ \$\ [Translate\ (from_{\mathbb{Q}}\ x)\ (from_{\mathbb{Q}}\ y)\ thicCirc\ |\ [x, y] \leftarrow points\ world] \\
\quad , Color\ green\ \$\ Translate\ cx\ cy\ thicCirc \\
\quad]\ \mathbf{where} \\
\quad (cx, cy) = bezier2dAtTime\ world
\end{array}
$$

Animação:

$$
\begin{array}{l}
animateBezier :: Float \to [NPoint] \to Picture \\
animateBezier\ \_\ [\,] = Blank \\
animateBezier\ \_\ [\_] = Blank \\
animateBezier\ t\ l = Pictures \\
\quad [animateBezier\ t\ (init\ l) \\
\quad , animateBezier\ t\ (tail\ l) \\
\quad , Color\ red \cdot Line\ \$\ [a, b] \\
\quad , Color\ orange\ \$\ Translate\ ax\ ay\ thicCirc \\
\quad , Color\ orange\ \$\ Translate\ bx\ by\ thicCirc \\
\quad]\ \mathbf{where} \\
\quad a@(ax, ay) = bezier2d\ (init\ l)\ t \\
\quad b@(bx, by) = bezier2d\ (tail\ l)\ t
\end{array}
$$

Propriedades e *main*:

$$
\begin{array}{l}
runBezier :: \mathsf{IO}\ () \\
runBezier = play\ (InWindow\ \texttt{"Bézier"}\ (600, 600)\ (0, 0)) \\
\quad black\ 50\ initW\ picture\ actions\ tick
\end{array}
$$

$$
\begin{array}{l}
runBezierSym :: \mathsf{IO}\ () \\
runBezierSym = quickCheckWith\ (stdArgs\ \{maxSize = 20, maxSuccess = 200\})\ prop\_bezier\_sym
\end{array}
$$

Compilação e execução dentro do interpretador:[11]

$$
main = runBezier
$$

$$
run = \mathbf{do}\ \{system\ \texttt{"ghc cp2021t"}; system\ \texttt{"./cp2021t"}\}
$$

[11] Pode ser útil em testes envolvendo Gloss. Nesse caso, o teste em causa deve fazer parte de uma função *main*.

QuickCheck

Código para geração de testes:

$$
\begin{array}{l}
\textbf{instance}\ \mathit{Arbitrary}\ \mathit{UnOp}\ \textbf{where} \\
\quad \mathit{arbitrary} = \mathit{elements}\ [\mathit{Negate}, E] \\
\textbf{instance}\ \mathit{Arbitrary}\ \mathit{BinOp}\ \textbf{where} \\
\quad \mathit{arbitrary} = \mathit{elements}\ [\mathit{Sum}, \mathit{Product}] \\
\textbf{instance}\ (\mathit{Arbitrary}\ a) \Rightarrow \mathit{Arbitrary}\ (\mathit{ExpAr}\ a)\ \textbf{where} \\
\quad \mathit{arbitrary} = \textbf{do} \\
\qquad \mathit{binop} \leftarrow \mathit{arbitrary} \\
\qquad \mathit{unop} \leftarrow \mathit{arbitrary} \\
\qquad \mathit{exp1} \leftarrow \mathit{arbitrary} \\
\qquad \mathit{exp2} \leftarrow \mathit{arbitrary} \\
\qquad a \quad\ \ \leftarrow \mathit{arbitrary} \\
\qquad \mathit{frequency} \cdot \mathsf{map}\ (\mathit{id} \times \mathit{pure})\ \$\ [(20, X), (15, N\ a), (35, \mathit{Bin}\ \mathit{binop}\ \mathit{exp1}\ \mathit{exp2}), (30, \mathit{Un}\ \mathit{unop}\ \mathit{exp1})]
\end{array}
$$

$$
\begin{array}{l}
\textbf{infixr}\ 5 \stackrel{?}{=} \\
(\stackrel{?}{=}) :: \mathit{Real}\ a \Rightarrow a \to a \to \mathit{Bool} \\
(\stackrel{?}{=})\ x\ y = (\mathit{to}_{\mathbb{Q}}\ x) \equiv (\mathit{to}_{\mathbb{Q}}\ y)
\end{array}
$$

Outras funções auxiliares

Lógicas:

$$
\begin{array}{l}
\textbf{infixr}\ 0 \Rightarrow \\
(\Rightarrow) :: (\mathit{Testable}\ \mathit{prop}) \Rightarrow (a \to \mathit{Bool}) \to (a \to \mathit{prop}) \to a \to \mathit{Property} \\
p \Rightarrow f = \lambda a \to p\ a \Rightarrow f\ a \\
\textbf{infixr}\ 0 \Leftrightarrow \\
(\Leftrightarrow) :: (a \to \mathit{Bool}) \to (a \to \mathit{Bool}) \to a \to \mathit{Property} \\
p \Leftrightarrow f = \lambda a \to (p\ a \Rightarrow \mathit{property}\ (f\ a))\ .\&\&.\ (f\ a \Rightarrow \mathit{property}\ (p\ a)) \\
\textbf{infixr}\ 4 \equiv \\
(\equiv) :: \mathit{Eq}\ b \Rightarrow (a \to b) \to (a \to b) \to (a \to \mathit{Bool}) \\
f \equiv g = \lambda a \to f\ a \equiv g\ a \\
\textbf{infixr}\ 4 \leqslant \\
(\leqslant) :: \mathit{Ord}\ b \Rightarrow (a \to b) \to (a \to b) \to (a \to \mathit{Bool}) \\
f \leqslant g = \lambda a \to f\ a \leqslant g\ a \\
\textbf{infixr}\ 4 \wedge \\
(\wedge) :: (a \to \mathit{Bool}) \to (a \to \mathit{Bool}) \to (a \to \mathit{Bool}) \\
f \wedge g = \lambda a \to ((f\ a) \wedge (g\ a))
\end{array}
$$

D Soluções dos alunos

Os alunos devem colocar neste anexo as suas soluções para os exercícios propostos, de acordo com o "layout"que se fornece. Não podem ser alterados os nomes ou tipos das funções dadas, mas pode ser adicionado texto, disgramas e/ou outras funções auxiliares que sejam necessárias.

Valoriza-se a escrita de *pouco* código que corresponda a soluções simples e elegantes.

Problema 1

São dadas:

$$
\begin{array}{l}
\mathit{cataExpAr}\ g = g \cdot \mathit{recExpAr}\ (\mathit{cataExpAr}\ g) \cdot \mathit{outExpAr} \\
\mathit{anaExpAr}\ g = \mathit{inExpAr} \cdot \mathit{recExpAr}\ (\mathit{anaExpAr}\ g) \cdot g \\
\mathit{hyloExpAr}\ h\ g = \mathit{cataExpAr}\ h \cdot \mathit{anaExpAr}\ g
\end{array}
$$

$$eval\_exp :: Floating\ a \Rightarrow a \rightarrow (ExpAr\ a) \rightarrow a$$
$$eval\_exp\ a = cataExpAr\ (g\_eval\_exp\ a)$$
$$optmize\_eval :: (Floating\ a, Eq\ a) \Rightarrow a \rightarrow (ExpAr\ a) \rightarrow a$$
$$optmize\_eval\ a = hyloExpAr\ (gopt\ a)\ clean$$
$$sd :: Floating\ a \Rightarrow ExpAr\ a \rightarrow ExpAr\ a$$
$$sd = \pi_2 \cdot cataExpAr\ sd\_gen$$
$$ad :: Floating\ a \Rightarrow a \rightarrow ExpAr\ a \rightarrow a$$
$$ad\ v = \pi_2 \cdot cataExpAr\ (ad\_gen\ v)$$

Definir:

Para definir outExpAr, primeiro o grupo observou o tipo da função, fornecido no enunciado. Sabendo isso, definiu-se o ”out”para cada elemento do tipo de dados. Graças à definição do tipo, sabe-se que à entrada da função irá surgir uma estrutura do tipo ExpAr, que contém obrigatoriamente as seguintes sub-estruturas de dados:

- O X da função.
- Um número N seguido do seu valor.
- Uma operação binária Bin, seguida de um operador e duas expressões.
- Uma operação unária Un, seguida de um operador e uma expressão.

Observando o tipo de entrada de inExpAr, sabendo que outExpAr é a sua função inversa, e pela definição do ”out”de um tipo de dados, sabemos que para cada sub-estrutura do tipo ExpAr temos de obter, com o outExpAr, uma das seguintes opções:

- Para o X da função, temos de obter o elemento único do tipo de dados 1.
- Para um número N seguido do seu valor, temos de obter o valor.
- Para uma operação binária Bin, seguida de um operador (Sum ou Product) e duas expressões, temos de obter o operador e as expressões.
- Para uma operação unária Un, seguida de um operador (Negate ou E) e uma expressão, temos de obter o operador e a expressão.

Com isto, foi possível deduzir a seguinte definição de outExpAr:

-- type OutExpAr a = Either () (Either a (Either (BinOp, (ExpAr a, ExpAr a)) (UnOp, EAxpr a))
$$outExpAr\ X = i_1\ ()$$
$$outExpAr\ (N\ n) = (i_2 \cdot i_1)\ n$$
$$outExpAr\ (Bin\ a\ b\ n) = (i_2 \cdot i_2 \cdot i_1)\ (a, (b, n))$$
$$outExpAr\ (Un\ a\ n) = (i_2 \cdot i_2 \cdot i_2)\ (a, n)$$

Para definir a recursividade do tipo de dados, observamos a definição de baseExpAr, e deduzimos os elementos que correspondem a árvores e aos quais queremos aplicar funções recursivas, neste caso, j, k e z da definiçao de baseExpAr.

-- baseExpAr f g h j k l z = f + (g + (h (j k) + l z))
$$recExpAr\ g = baseExpAr\ id\ id\ id\ g\ g\ id\ g$$

Para encontrarmos a definição do gene do catamorfismo, baseamo-nos no seguinte diagrama:

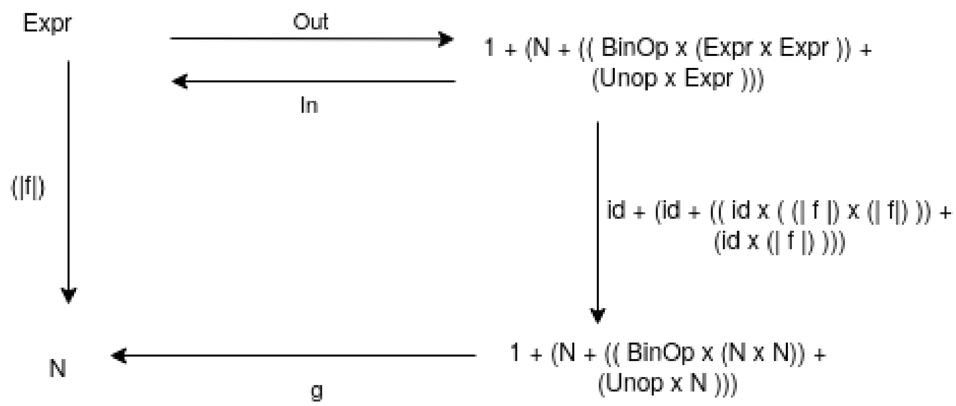

Figura 2: Diagrama do catamorfismo do Problema1.

Com isto, o grupo pôde concluir que a função eval_exp é uma função que, dado um valor Floating para X e uma expressão ExpAr, permite passar do tipo de dados ExpAr para um valor numérico do tipo Floating, e o gene deve receber um dos seguintes casos:

- Um valor (), perante o qual o gene deve retornar o valor fornecido para o X da função (const a)
- Um valor Floating, perante o qual o gene deve retornar esse mesmo valor (id)
- As operações binárias Sum ou Product seguidas de dois valores Floating, perante as quais o gene deve calcular a soma ou o produto dos valores (a + b ou a * b).
- As operações unárias Negate ou E seguidas de um valor Floating, perante as quais o gene deve calcular a negação ou a exponencial do valor (-x ou Prelude.exp x)

Com isto, foi possível chegar à seguinte solução:

$g\_eval\_exp\ a = [\underline{a}, num\_ops]$ **where**
$\quad num\_ops = [id, ops]$
$\quad ops = [bin, uno]$
$\quad bin\ (Sum, (a, b)) = a + b$
$\quad bin\ (Product, (a, b)) = a * b$
$\quad uno\ (Negate, x) = -x$
$\quad uno\ (E, x) = Prelude.exp\ x$

Como optmize_eval está definido como um hilomorfismo, dada uma expressão ExpAr e um valor Floating, primeiro irá ser aplicado um anamorfismo, responsável por realizar as otimizações necessárias à expressão, e de seguida um catamorfismo, que calculará o valor da expressão resultante. Para este problema, o grupo conseguiu definir o gene do catamorfismo "clean"que aplica as leis do elemento absorvente da multiplicação, de modo a simplificar a expressão e evitar cálculos desnecessários, e como gene do catamorfismo "gopt", escolheu utilizar ao gene já definido anteriormente para cálculo de expresões ExpAr, ou seja, "g_eval_exp". Adicionalmente, o grupo calculou também os elementos neutros de algumas operações, que embora marginalmente, podem optimizar o cálculo das expressões.

$clean\ (Bin\ Product\ \_\ (N\ 0)) = outExpAr\ (N\ 0)$
$clean\ (Bin\ Product\ (N\ 0)\ \_) = outExpAr\ (N\ 0)$
$\quad$ -- elementos neutros
$clean\ (Bin\ Product\ exp\ (N\ 1)) = outExpAr\ exp$
$clean\ (Bin\ Product\ (N\ 1)\ exp) = outExpAr\ exp$
$clean\ (Bin\ Sum\ exp\ (N\ 0)) = outExpAr\ exp$
$clean\ (Bin\ Sum\ (N\ 0)\ exp) = outExpAr\ exp$
$clean\ exp = outExpAr\ exp$

$gopt\ val = g\_eval\_exp\ val$

Para definir a symbolic differentiation, o grupo inspirou-se no gene do catamorfismo já definido anteriormente, g_eval_exp, e aplicou os mesmos princípios de raciocínio de modo a criar uma alternativa para cada tipo de operação identificada, e assim produzir a derivada da expressão. O raciocínio passou também por devolver em todas as alternativas um par, em que no primeiro elemento se produz a expressão do cálculo da operação em questão e no segundo elemento a derivada da expressão. Isto é necessário, visto que para calcular a derivada do produto e da exponencial, são necessárias não só as derivadas das expressões envolvidas, como também as próprias expressões. Note-se que esta função não calcula o valor da derivada num ponto por si, mas produz uma expressão que pode ser avaliada com eval_exp para se descobrir a derivada num ponto X.

Alguma inspiração foi também retirada de uma página online [3], onde se encontra um pequeno artigo muito interessante sobre symbolic differentiation e uma definição funcional e recursiva desta.

$$
\begin{array}{l}
sd\_gen :: Floating\ a \Rightarrow \\
\quad () + (a + ((BinOp, ((ExpAr\ a, ExpAr\ a), (ExpAr\ a, ExpAr\ a))) + (UnOp, (ExpAr\ a, ExpAr\ a)))) \rightarrow (ExpAr\ a \\
sd\_gen = [sd\_gen\_var, num\_ops]\ \mathbf{where} \\
\quad num\_ops = [sd\_gen\_const, ops] \\
\quad ops = [sd\_bin, sd\_uno] \\
sd\_gen\_var\ () = (X, N\ 1) \\
sd\_gen\_const\ a = (N\ a, N\ 0) \\
sd\_bin\ (Sum, ((a, a'), (b, b'))) = (Bin\ Sum\ a\ b, Bin\ Sum\ a'\ b') \\
sd\_bin\ (Product, ((a, a'), (b, b'))) = (Bin\ Product\ a\ b, Bin\ Sum\ (Bin\ Product\ a\ b')\ (Bin\ Product\ a'\ b)) \\
sd\_uno\ (Negate, (a, a')) = (Un\ Negate\ a, Un\ Negate\ a') \\
sd\_uno\ (E, (a, a')) = (Un\ E\ a, Bin\ Product\ (Un\ E\ a)\ a')
\end{array}
$$

Para a automatic differentiation temos um raciocínio semelhante, mas no segundo elemento do par resultante do gene, calculamos o valor da derivada da operação conforme as expressões dadas à entrada, em vez de calcularmos a expressão da derivada. Isto resulta numa função que embora não seja muito diferente da symbolic differentiation, em termos de raciocínio, é mais eficiente, e permite calcular diretamente o valor da derivada num ponto X.

Foi também definido um tipo AExpAr para permitir escrever o tipo de ad_gen de forma mais sucinta.

$$
\begin{array}{l}
\mathbf{type}\ AExpAr\ a = (ExpAr\ a, a) \\
ad\_gen :: Floating\ a \Rightarrow a \rightarrow () + (a + ((BinOp, (AExpAr\ a, AExpAr\ a)) + (UnOp, AExpAr\ a))) \rightarrow AExpAr\ a \\
ad\_gen\ p = [ad\_gen\_var, num\_ops]\ \mathbf{where} \\
\quad num\_ops = [ad\_gen\_const, ops] \\
\quad ops = [ad\_bin\ p, ad\_uno\ p] \\
ad\_gen\_var\ () = (X, 1) \\
ad\_gen\_const\ a = (N\ a, 0) \\
ad\_bin\ p\ (Sum, ((a, a'), (b, b'))) = (Bin\ Sum\ a\ b, a' + b') \\
ad\_bin\ p\ (Product, ((a, a'), (b, b'))) = (Bin\ Product\ a\ b, ((eval\_exp\ p\ a) * b') + (a' * (eval\_exp\ p\ b))) \\
ad\_uno\ p\ (Negate, (a, a')) = (Un\ Negate\ a, (-a')) \\
ad\_uno\ p\ (E, (a, a')) = (Un\ E\ a, (eval\_exp\ p\ (Un\ E\ a)) * a')
\end{array}
$$

Todas as alíneas deste problema foram testadas e passaram em todos os testes do quickCheck.

Problema 2

Definir

$$
\begin{array}{l}
loop\ (c, t, b) = ((t * c)\ `div`\ b, t + 4, b + 1) \\
inic = (1, 2, 2) \\
prj\ (c, t, b) = c
\end{array}
$$

por forma a que

$$cat = prj \cdot \mathsf{for}\ loop\ inic$$

seja a função pretendida. **NB**: usar divisão inteira. Apresentar de seguida a justificação da solução encontrada.

O problema identificado diz respeito à aplicação da recursividade mútua numa função sobre números naturais. Para encontrar a solução deste problema, o grupo tentou definir recursivamente a fórmula da função de Catalan, c, resolvendo matematicamente a seguinte equação, removendo a variável n do lado direito da equação, e definindo chamadas recursivas das funções auxiliares que se consideraram necessárias:

$$c\ 0 = 1$$
$$c\ (n+1) = \frac{(2n+2)!}{(n+2)!(n+1)!}$$

Pela aplicação sucessiva da simplificação do fatorial:

$$(n+1)! = (n+1) * n!$$

Obtém-se:

$$c\ (n+1) = \frac{(2n+2)(2n+1)!}{(n+2)(n+1)!(n+1)(n!)}$$
$$c\ (n+1) = \frac{(2n+2)(2n+1)(2n)!}{(n+2)(n+1)!(n+1)(n!)}$$
$$c\ (n+1) = \frac{2(n+1)(2n+1)(2n)!}{(n+2)(n+1)!(n+1)(n!)}$$
$$c\ (n+1) = \frac{2(2n+1)(2n)!}{(n+2)(n+1)!(n!)}$$
$$c\ (n+1) = \frac{2(2n+1)}{n+2} * \frac{(2n)!}{(n+1)!(n!)}$$
$$c\ (n+1) = \frac{4n+2)}{n+2} * c\ n$$

Obtida c em recursividade mútua, resta ainda definir a fração da equação, que ainda possui a variável n. Neste caso, considere-se t (top) como numerador, e b (bottom) como denominador. Temos então, a seguinte definição de t:

$$t = 4n + 2$$
$$t\ 0 = 2$$
$$t\ (n+1) = 4n + 6$$
$$t\ (n+1) = 4n + 2 + 4$$
$$t\ (n+1) = t\ n + 2$$

Descoberto t, segue-se a definição de b:

$$b = n + 2$$
$$b\ 0 = 2$$
$$b\ (n+1) = n + 3$$
$$b\ (n+1) = n + 2 + 1$$
$$b\ (n+1) = b\ n + 1$$

Tal como no Anexo B, o grupo obteve três funções. Seguindo a *regra de algibeira* da página 3.1, obtiveram-se c, t, e b como as três funções mutuamente recursivas, juntamente com as suas definições, os números inteiros 1, 2 e 2 como resultados dos casos base de c, t, e b, respetivamente, e ainda se fez a projeção do resultado em c. Foi também utilizada a divisão inteira através de *div*.

Problema 3

Para este problema, primeiro definiu-se a o gene da função calcLine, que permite calcular o ponto na reta entre dois NPoints num determinado instante dado por um valor Float. Como se trata de um catamorfismo de listas, foram definidos casos para a lista vazia e para a lista não vazia. A função foi fortemente inspirada na definição de calcLine fornecida em anexo. Note-se o tipo do gene, h.

```
h :: Either () (Rational, NPoint -> OverTime NPoint) -> NPoint -> OverTime NPoint
```

$$
\begin{aligned}
&calcLine :: NPoint \rightarrow (NPoint \rightarrow OverTime\ NPoint)\\
&calcLine = cataList\ h\ \textbf{where}\\
&\quad h = [\underline{\cdot}\ \$\ \underline{nil}, points]\\
&\quad points\ (q, fun)\ [\,] = nil\\
&\quad points\ (q, fun)\ (x : xs) = concat \cdot sequenceA\ [singl \cdot linear1d\ q\ x, fun\ xs]
\end{aligned}
$$

Para definir a função de deCasteljau, primeiro o grupo teve de primeiro observar a função original em anexo. Com isto, foi possível observar que o comportamento da função se assemelha a um hilomorfismo de LTrees, com duas chamadas recursivas a deCasteljau a cada iteração, mais uma vez, semelhante ao comportamento do hilomorfismo em LTrees. Como base, podemos observar que temos dois casos, a lista vazia, ou a lista singular. Isto leva-nos a deduzir que a função ou recebe o vazio ou um NPoint.

$$hyloAlgForm = hyloLTree$$

Seguindo a linha de pensamento do hilomorfismo anteriormente definido, continuamos a resolução do problema, agora definindo a álgebra e a coalgebra do catamorfismo e anamorfismo, respetivamente, que compõem o hilomorfismo. Com o seguinte diagrama, podemos observar o comportamento genérico do hilomorfismo:

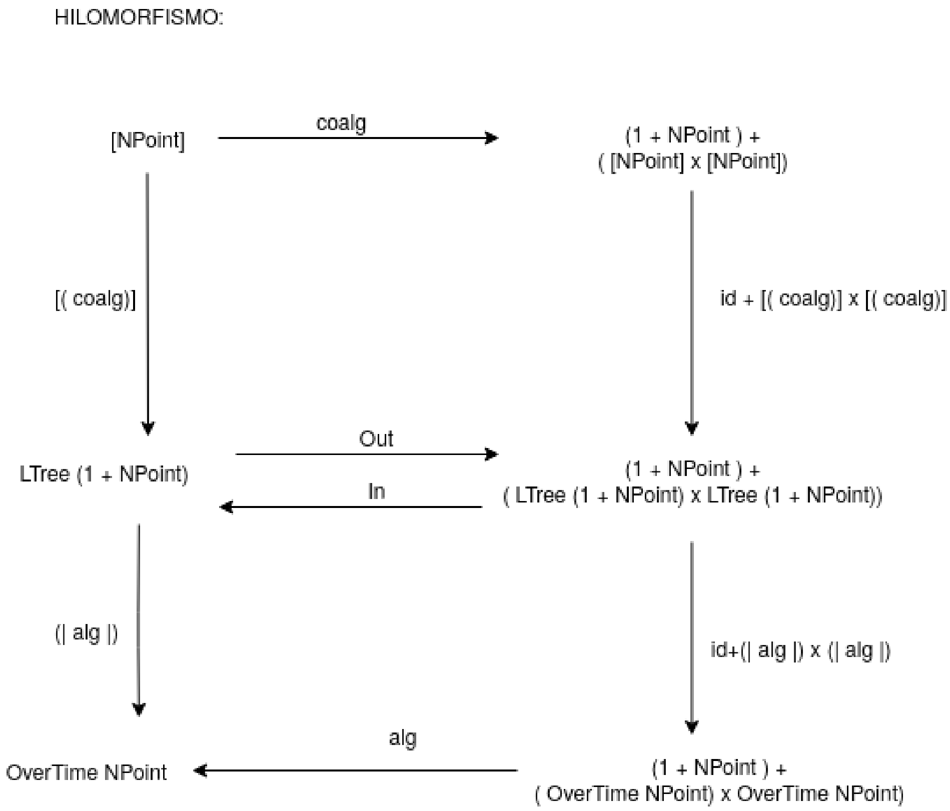

Figura 3: Diagrama do hilomorfismo do Problema3.

Em coalg, inspirando-nos na definição fornecida em anexo, queremos fazer a parte que do init e do tail da lista de NPoints que formarão a curva. Em alg, inspirando-nos mais uma vez na definição fornecida em anexo, queremos fazer a recursividade e o calcLine entre os NPoints recebidos por recursividade da esquerda e da direita da árvore, dando também o Float para o instante em que o cálculo é efetuado. Com isto, completamos a definição de deCasteljau.

$$
\begin{aligned}
&deCasteljau :: [NPoint] \rightarrow OverTime\ NPoint\\
&deCasteljau = hyloAlgForm\ alg\ coalg\ \textbf{where}\\
&\quad coalg :: [NPoint] \rightarrow (() + NPoint) + ([NPoint], [NPoint])\\
&\quad coalg\ [\,] = i_1\ \$\ i_1\ ()\\
&\quad coalg\ [a] = i_1\ \$\ i_2\ a\\
&\quad coalg\ l = i_2\ (init\ l, tail\ l)
\end{aligned}
$$

$alg :: (() + NPoint) + (OverTime\ NPoint, OverTime\ NPoint) \rightarrow OverTime\ NPoint$
$alg = [[\underline{nil}, \underline{\cdot}], pairn]$
$pairn\ (l, r)\ pointtime = calcLine\ (l\ pointtime)\ (r\ pointtime)\ pointtime$

As funções passam nos testes do quickCheck, e a função runBezier corre como esperado.

Problema 4

Solução para listas não vazias:

$avg = \pi_1 \cdot avg\_aux$

Para resolver este problema, o grupo começou por definir as operações sobre o tipo de dados "list not empty", ou ListNE. Assim, assegurou-se que as funções operam sobre listas não vazias apenas. Assim, foi possível definir, partindo da lei da recursividade mútua, definir avg_aux. Para tal, dá-se destaque à utilização da lei de Fokkinga, da lei de Leibniz, da lei da troca, e da passagem da definição de length para pointfree Haskell.

$$\equiv \qquad \{ \text{ Recursividade mútua } \}$$

$$\begin{cases} f \cdot \mathbf{in} = h \cdot F\ \langle f, g \rangle \\ g \cdot \mathbf{in} = k \cdot F\ \langle f, g \rangle \end{cases}$$

$$\equiv \qquad \{ \text{ Fokkinga com f := avg, g := length, h := c, k := d } \}$$

$$\begin{cases} avg \cdot \mathbf{in} = c \cdot F\ \langle avg, length \rangle \\ length \cdot \mathbf{in} = d \cdot F\ \langle avg, length \rangle \end{cases}$$

$$\equiv \qquad \{ \text{ Def. in para listas } \}$$

$$\begin{cases} avg \cdot [singl, cons] = c \cdot F\ \langle avg, length \rangle \\ length \cdot [singl, cons] = d \cdot F\ \langle avg, length \rangle \end{cases}$$

$$\equiv \qquad \{ \text{ Base-cata } \}$$

$$\begin{cases} avg \cdot [singl, cons] = c \cdot B\ (id, \langle avg, length \rangle) \\ length \cdot [singl, cons] = d \cdot B\ (id, \langle avg, length \rangle) \end{cases}$$

$$\equiv \qquad \{ \text{ baseListNE = f + f x g } \}$$

$$\begin{cases} avg \cdot [singl, cons] = c \cdot (id + id \times \langle avg, lenth \rangle) \\ length \cdot [singl, cons] = d \cdot (id + id \times \langle avg, lenth \rangle) \end{cases}$$

$$\equiv \qquad \{ \text{ c := [c1, c2], d := [d1, d2] } \}$$

$$\begin{cases} avg \cdot [singl, cons] = [c1, c2] \cdot (id + id \times \langle avg, lenth \rangle) \\ length \cdot [singl, cons] = [d1, d2] \cdot (id + id \times \langle avg, lenth \rangle) \end{cases}$$

$$\equiv \qquad \{ \text{ Fusão + } \}$$

$$\begin{cases} [avg \cdot singl, avg \cdot cons] = [c1, c2] \cdot (id + id \times \langle avg, lenth \rangle) \\ [length \cdot singl, length \cdot cons] = [d1, d2] \cdot (id + id \times \langle avg, lenth \rangle) \end{cases}$$

$$\equiv \qquad \{ \text{ Universal +, 2x } \}$$

$$\begin{cases} \begin{cases} avg \cdot singl = [c1, c2] \cdot (id + id \times \langle avg, lenth \rangle) \cdot i_1 \\ avg \cdot cons = [c1, c2] \cdot (id + id \times \langle avg, lenth \rangle) \cdot i_2 \end{cases} \\ \begin{cases} length \cdot singl = [c1, c2] \cdot (id + id \times \langle avg, lenth \rangle) \cdot i_1 \\ length \cdot cons = [d1, d2] \cdot (id + id \times \langle avg, lenth \rangle) \cdot i_2 \end{cases} \end{cases}$$

$$\equiv \qquad \{ \text{ Natural i1, Natural i2 } \}$$

$$\begin{cases} \begin{cases} avg \cdot singl = [c1, c2] \cdot i_1 \cdot id \\ avg \cdot cons = [c1, c2] \cdot i_2 \cdot id \times \langle avg, length \rangle \end{cases} \\ \begin{cases} length \cdot singl = [c1, c2] \cdot i_1 \cdot id \\ length \cdot cons = [d1, d2] \cdot i_2 \cdot id \times \langle avg, length \rangle \end{cases} \end{cases}$$

$\equiv$ { Natural id, Cancelamento + }

$$\left\{\begin{array}{l}\left\{\begin{array}{l} avg \cdot singl = c1 \\ avg \cdot cons = c2 \cdot (id \times \langle avg, length\rangle) \end{array}\right. \\ \left\{\begin{array}{l} length \cdot singl = d1 \\ length \cdot cons = d2 \cdot (id \times \langle avg, length\rangle) \end{array}\right.\end{array}\right.$$

$\equiv$ { Natural id, Cancelamento + }

$$\left\{\begin{array}{l}\left\{\begin{array}{l} avg \cdot singl = c1 \\ avg \cdot cons = c2 \cdot (id \times \langle avg, length\rangle) \end{array}\right. \\ \left\{\begin{array}{l} length \cdot singl = d1 \\ length \cdot cons = d2 \cdot (id \times \langle avg, length\rangle) \end{array}\right.\end{array}\right.$$

$\equiv$ { Pela definição pointfree de avg, avg.cons := (uncurry (/)) . ¡(uncurry (+) . (id x (uncurry (*)))), succ . p2 . p2¿ . (id x

$$\left\{\begin{array}{l}\left\{\begin{array}{l} avg \cdot singl = c1 \\ \widehat{(/)} \cdot \langle \widehat{(+)} \cdot (id\ x\ \widehat{(*)}), \mathsf{succ}\ \cdot \pi_2 \cdot \pi_2\rangle \end{array}\right. \\ \left\{\begin{array}{l} length \cdot singl = d1 \\ length \cdot cons = d2 \cdot (id \times \langle avg, length\rangle) \end{array}\right.\end{array}\right.$$

$\equiv$ { Def. length }

$$\left\{\begin{array}{l}\left\{\begin{array}{l} avg \cdot singl = c1 \\ \widehat{(/)} \cdot \langle \widehat{(+)} \cdot (id\ x\ \widehat{(*)}), \mathsf{succ}\ \cdot \pi_2 \cdot \pi_2\rangle \end{array}\right. \\ \left\{\begin{array}{l} length \cdot singl = d1 \\ \mathsf{succ}\ \cdot length \cdot \pi_2 = d2 \cdot (id \times \langle avg, length\rangle) \end{array}\right.\end{array}\right.$$

$\equiv$ { Natural p2 }

$$\left\{\begin{array}{l}\left\{\begin{array}{l} avg \cdot singl = c1 \\ \widehat{(/)} \cdot \langle \widehat{(+)} \cdot (id\ x\ \widehat{(*)}), \mathsf{succ}\ \cdot \pi_2 \cdot \pi_2\rangle \end{array}\right. \\ \left\{\begin{array}{l} length \cdot singl = d1 \\ \mathsf{succ}\ \cdot \pi_2 \cdot (id \times length) = d2 \cdot (id \times \langle avg, length\rangle) \end{array}\right.\end{array}\right.$$

$\equiv$ { Cancelamento x }

$$\left\{\begin{array}{l}\left\{\begin{array}{l} avg \cdot singl = c1 \\ \widehat{(/)} \cdot \langle \widehat{(+)} \cdot (id\ x\ \widehat{(*)}), \mathsf{succ}\ \cdot \pi_2 \cdot \pi_2\rangle \end{array}\right. \\ \left\{\begin{array}{l} length \cdot singl = d1 \\ \mathsf{succ}\ \cdot \pi_2 \cdot (id \times (\pi_2 \cdot \langle avg, length\rangle)) = d2 \cdot (id \times \langle avg, length\rangle) \end{array}\right.\end{array}\right.$$

$\equiv$ { Natural id }

$$\left\{\begin{array}{l}\left\{\begin{array}{l} avg \cdot singl = c1 \\ \widehat{(/)} \cdot \langle \widehat{(+)} \cdot (id\ x\ \widehat{(*)}), \mathsf{succ}\ \cdot \pi_2 \cdot \pi_2\rangle \end{array}\right. \\ \left\{\begin{array}{l} length \cdot singl = d1 \\ \mathsf{succ}\ \cdot \pi_2 \cdot (id \cdot id \times (\pi_2 \cdot \langle avg, length\rangle)) = d2 \cdot (id \times \langle avg, length\rangle) \end{array}\right.\end{array}\right.$$

$\equiv$ { Functor x }

$$\left\{\begin{array}{l}\left\{\begin{array}{l} avg \cdot singl = c1 \\ \widehat{(/)} \cdot \langle \widehat{(+)} \cdot (id\ x\ \widehat{(*)}), \mathsf{succ}\ \cdot \pi_2 \cdot \pi_2\rangle \end{array}\right. \\ \left\{\begin{array}{l} length \cdot singl = d1 \\ \mathsf{succ}\ \cdot \pi_2 \cdot (id \times \pi_2) \cdot (id \times \langle avg, length\rangle) = d2 \cdot (id \times \langle avg, length\rangle) \end{array}\right.\end{array}\right.$$

$\equiv$ { Leibniz }

$$\left\{\begin{array}{l}\left\{\begin{array}{l} avg \cdot singl = c1 \\ \widehat{(/)} \cdot \langle \widehat{(+)} \cdot (id\ x\ \widehat{(*)}), \mathsf{succ}\ \cdot \pi_2 \cdot \pi_2\rangle \end{array}\right. \\ \left\{\begin{array}{l} length \cdot singl = d1 \\ \mathsf{succ}\ \cdot \pi_2 \cdot (id \times \pi_2) = d2 \end{array}\right.\end{array}\right.$$

□

Daqui concluí-se:

$$\langle avg, length \rangle = (\!|\langle c, d \rangle|\!)$$

$$\equiv \quad \{ \text{ Expandindo c e d para [c1,c2] e [d1,d2] } \}$$

$$\langle avg, length \rangle = (\!|\langle [c1, c2], [d1, d2] \rangle|\!)$$

$$\equiv \quad \{ \text{ Lei da Troca } \}$$

$$\langle avg, length \rangle = (\!|[\langle c1, d1 \rangle, \langle c2, d2 \rangle]|\!)$$

□

Uma vez concluído este raciocínio, podemos escrever a seguinte definição de ListNE e de avg_aux, que não é mais do que uma simplificação da derivação obtida com a lei de recursividade mútua.

$$cataListNE\ g = g \cdot recListNE\ (cataListNE\ g) \cdot outListNE$$
$$baseListNE\ f\ g = f + f \times g$$
$$recListNE\ f \quad = id + id \times f$$
$$inListNE = [singl, cons]$$
$$outListNE\ [a] = i_1\ a$$
$$outListNE\ (a : x) = i_2\ (a, x)$$
$$avg\_aux = cataListNE\ [\langle c1, d1 \rangle, \langle c2, d2 \rangle]\ \textbf{where}$$
-- c1 = average . singl, mas simplificando...
$$c1 = id$$
-- vamos receber Either [Double] ([Double], (Double, Double))
$$c2 = \widehat{(/)} \cdot \langle \widehat{(+)} \cdot (id \times \widehat{(*)}), \mathsf{succ} \cdot \pi_2 \cdot \pi_2 \rangle$$
-- d1 = genericLength . singl, mas simplificando...
$$d1 = \underline{(1 :: Double)}$$
$$d2 = \mathsf{succ} \cdot \pi_2 \cdot (id \times \pi_2)$$

Solução para árvores de tipo LTree:

Tendo já feito a definição para listas não vazias, o mesmo raciocínio foi generalizado de modo a obter a definição da média para LTrees. Note-se que a definição de média em cada fork é agora (averageLeft x lengthLeft + averageRight x lengthRight) / (lengthLeft + lengthRight).

$$avgLTree = \pi_1 \cdot (\!|\ gene\ |\!)\ \textbf{where}$$
$$gene = [\langle c1, d1 \rangle, \langle c2, d2 \rangle]$$
-- c1 = averageLTree . Leaf, mas simplificando...
$$c1 = id$$
-- vamos receber Either Double ((Double,Double), (Double, Double))
$$c2 = \widehat{(/)} \cdot \langle \widehat{(+)} \cdot \langle \widehat{(*)} \cdot \langle \pi_2 \cdot \pi_1, \pi_1 \cdot \pi_1 \rangle, \widehat{(*)} \cdot \langle \pi_2 \cdot \pi_2, \pi_1 \cdot \pi_2 \rangle \rangle, \widehat{(+)} \cdot \langle \pi_2 \cdot \pi_1, \pi_2 \cdot \pi_2 \rangle \rangle$$
-- d1 = lengthLTree . Leaf, mas simplificando...
$$d1 = \underline{(1 :: Double)}$$
$$d2 = \widehat{(+)} \cdot \langle \pi_2 \cdot \pi_1, \pi_2 \cdot \pi_2 \rangle$$

As funções passam nos testes do quickCheck e correm como esperado.

Problema 5

Inserir em baixo o código F# desenvolvido, entre `\begin{verbatim}` e `\end{verbatim}`:

Índice

Referências

[1] D.E. Knuth. *Literate Programming*. CSLI Lecture Notes Number 27. Stanford University Center for the Study of Language and Information, Stanford, CA, USA, 1992.

[2] J.N. Oliveira. *Program Design by Calculation*, 2018. Draft of textbook in preparation. viii+297 pages. Informatics Department, University of Minho.

[3] Jared Tobin. Yo dawg we heard you like derivatives, 2021. License: https://creativecommons.org/licenses/by/4.0/.