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

Departamento de Informática
Universidade do Minho

Junho de 2021

Grupo nr.	38
a77667	Luís Manuel Pereira
a73855	José Lopes Ramos
a82529	Carlos Manuel Marques Afonso
a86617	Gonçalo Pinto Nogueira

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 dependências 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,

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

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

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

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,

$$\mathit{Bin}\ \mathit{Sum}\ X\ (N\ 10)$$

designa $x + 10$ na notação matemática habitual.

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

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

Defina as funções $\mathit{outExpAr}$ e $\mathit{recExpAr}$, e teste as propriedades que se seguem.

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

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

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 $eval$.*

$$
\begin{array}{l}
prop\_optimize\_respects\_semantics :: RealFloat\ a \Rightarrow a \rightarrow ExpAr\ a \rightarrow Property \\
prop\_optimize\_respects\_semantics\ a\ exp = not\_NaN\ (eval\_exp\ a\ exp) \Rightarrow (eval\_exp\ a\ exp \stackrel{?}{=} optmize\_eval\ a
\end{array}
$$

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.*

$$
\begin{array}{l}
not\_NaN :: RealFloat\ a \Rightarrow a \rightarrow Bool \\
not\_NaN\ a = \neg\ (isNaN\ a) \\
prop\_congruent :: RealFloat\ a \Rightarrow a \rightarrow ExpAr\ a \rightarrow Property \\
prop\_congruent\ a\ exp = not\_NaN\ (eval\_exp\ a\ (sd\ exp)) \Rightarrow (ad\ a\ exp) \stackrel{?}{=} (eval\_exp\ a\ (sd\ exp))
\end{array}
$$

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

[4] Lei (3.94) em [2], página 98.

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

Obter-se-á de imediato

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

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:

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

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

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

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.

[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.

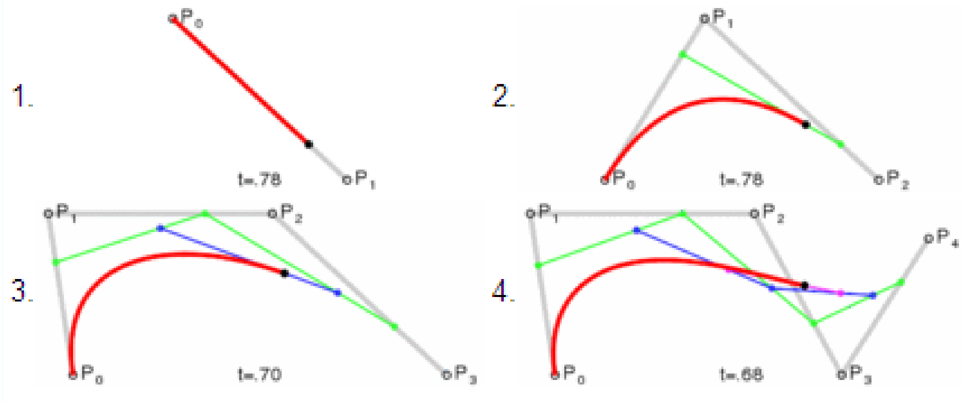

Figura 1: Exemplos de curvas de Bézier retirados da Wikipedia.

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.

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:

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

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.

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

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 $Float$).

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

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.*

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

3. Corra a função $runBezier$ 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,

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

onde $k = length\ x$. 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

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

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 $avg\_aux = (\![[b, q]]\!)$ tal que $avg\_aux = \langle avg, length \rangle$ 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:*

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

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

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

Problema 2

Definição da série de Catalan usando factoriais (1):

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

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

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

Problema 3

Algoritmo:

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

Função auxiliar:

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

2D:

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

Modelo:

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

[10] Fonte: Wikipedia.

$$
\begin{aligned}
&tick :: Float \to World \to World \\
&tick\ dt\ world = world\ \{ time = (time\ world) + dt \} \\
&actions :: Event \to World \to World \\
&actions\ (EventKey\ (MouseButton\ LeftButton)\ Down\ \_\ p)\ world = \\
&\quad world\ \{ points = (points\ world) +\!\!+ [(\lambda(x,y) \to \mathsf{map}\ \ to_{\mathbb{Q}}\ [x,y])\ p] \} \\
&actions\ (EventKey\ (SpecialKey\ KeyDelete)\ Down\ \_\ \_)\ world = \\
&\quad world\ \{ points = cond\ (\equiv [\,])\ id\ init\ (points\ world) \} \\
&actions\ \_\ world = world \\
&scaleTime :: World \to Float \\
&scaleTime\ w = (1 + cos\ (time\ w))\ /\ 2 \\
&bezier2dAtTime :: World \to (Float, Float) \\
&bezier2dAtTime\ w = (bezier2dAt\ w)\ (scaleTime\ w) \\
&bezier2dAt :: World \to OverTime\ (Float, Float) \\
&bezier2dAt\ w = bezier2d\ (points\ w) \\
&thicCirc :: Picture \\
&thicCirc = ThickCircle\ 4\ 10 \\
&ps :: [Float] \\
&ps = \mathsf{map}\ \ from_{\mathbb{Q}}\ ps'\ \mathbf{where} \\
&\quad ps' :: [\mathbb{Q}] \\
&\quad ps' = [0, 0.01 \mathinner{..} 1] \quad \text{-- interval}
\end{aligned}
$$

Gloss:

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

Animação:

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

Propriedades e *main*:

$$
\begin{aligned}
&runBezier :: \mathsf{IO}\ () \\
&runBezier = play\ (InWindow\ \texttt{"Bézier"}\ (600, 600)\ (0, 0)) \\
&\quad black\ 50\ initW\ picture\ actions\ tick \\
&runBezierSym :: \mathsf{IO}\ () \\
&runBezierSym = quickCheckWith\ (stdArgs\ \{ maxSize = 20, maxSuccess = 200 \})\ prop\_bezier\_sym
\end{aligned}
$$

Compilação e execução dentro do interpretador:[11]

$$
\begin{aligned}
&main = runBezier \\
&run = \mathbf{do}\ \{ system\ \texttt{"ghc cp2021t"}; system\ \texttt{"./cp2021t"} \}
\end{aligned}
$$

[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:

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

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

Outras funções auxiliares

Lógicas:

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

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:

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

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

Definir:

$$
\begin{aligned}
& out \cdot \mathbf{in} = id \\
\equiv \quad & \{ \text{ def in } \} \\
& out \cdot [\underline{X}, num\_ops] = id \\
\equiv \quad & \{ \text{ Fusão+ (20) } \} \\
& [out \cdot \underline{X}, out \cdot num\_ops] = id \\
\equiv \quad & \{ \text{ Universal+ (17) } \} \\
& \begin{cases} id \cdot i_1 = out \cdot \underline{X} \\ id \cdot i_2 = out \cdot num\_ops \end{cases} \\
\equiv \quad & \{ \text{ Natural-id (1) x2 } \} \\
& \begin{cases} i_1 = out \cdot \underline{X} \\ i_2 = out \cdot num\_ops \end{cases} \\
\equiv \quad & \{ \text{ def num\_ops } \} \\
& \begin{cases} i_1 = out \cdot \underline{X} \\ i_2 = out \cdot [N, ops] \end{cases} \\
\equiv \quad & \{ \text{ Fusão+ (20) } \} \\
& \begin{cases} i_1 = out \cdot \underline{X} \\ i_2 = [out \cdot N, out \cdot ops] \end{cases} \\
\equiv \quad & \{ \text{ Universal+ (17) } \} \\
& \begin{cases} i_1 = out \cdot \underline{X} \\ i_2 \cdot i_1 = out \cdot N \\ i_2 \cdot i_2 = out \cdot ops \end{cases} \\
\equiv \quad & \{ \text{ def ops } \} \\
& \begin{cases} i_1 = out \cdot \underline{X} \\ i_2 \cdot i_1 = out \cdot N \\ i_2 \cdot i_1 = out \cdot [bin, \widehat{Un}] \end{cases} \\
\equiv \quad & \{ \text{ Fusão+ (20) } \} \\
& \begin{cases} i_1 = out \cdot \underline{X} \\ i_2 \cdot i_1 = out \cdot N \\ i_2 \cdot i_2 = [out \cdot bin, out \cdot \widehat{Un}] \end{cases} \\
\equiv \quad & \{ \text{ Universal+ (17) } \} \\
& \begin{cases} i_1 = out \cdot \underline{X} \\ i_2 \cdot i_1 = out \cdot N \\ i_2 \cdot i_2 \cdot i_1 = out \cdot bin \\ i_2 \cdot i_2 \cdot i_2 = out \cdot \widehat{Un} \end{cases} \\
\equiv \quad & \{ \text{ Igualdade Extensional (71) x 4 } \}
\end{aligned}
$$

$$\begin{cases} i_1\ a = (out \cdot \underline{X})\ a \\ (i_2 \cdot i_1)\ a = (out \cdot N)\ a \\ (i_2 \cdot i_2 \cdot i_1)\ (a,(b,c)) = (out \cdot bin)\ (a,(b,c)) \\ (i_2 \cdot i_2 \cdot i_2)\ (a,b) = (out \cdot \widehat{Un})\ (a,b) \end{cases}$$

$\equiv$ $\{$ def-comp (72), uncurry (84) $\}$

$$\begin{cases} i_1\ a = out\ \underline{X}\ a \\ i_2\ (i_1\ a) = out\ (N\ a) \\ i_2\ (i_2\ (i_1\ (a,(b,c)))) = out\ (bin\ (a,(b,c))) \\ i_2\ (i_2\ (i_2\ (a,b))) = out\ (Un\ a\ b) \end{cases}$$

$\square$

$outExpAr\ X = i_1\ ()$
$outExpAr\ (N\ a) = i_2\ (i_1\ a)$
$outExpAr\ (Un\ a\ b) = i_2\ (i_2\ (i_2\ (a,b)))$
$outExpAr\ (Bin\ op\ a\ b) = i_2\ (i_2\ (i_1\ (op,(a,b))))$

--

$recExpAr\ g = baseExpAr\ id\ id\ id\ g\ g\ id\ g$

--

Diagrama g_eval_exp

$$\begin{tikzcd}
Exp \arrow[d, "eval"'] & constX + (N + (bOP \times (Exp \times Exp)) + unOP \times Exp) \arrow[l, "in=[\underline{X},num_ops]"] \arrow[d, "id+(id+(id\times(eval\times eval))+id\times eval)"] \\
\mathbb{A} & 1 + A + (bOP \times A^2) + (unOP \times A) \arrow[l, "g_eval_exp"]
\end{tikzcd}$$

$g\_eval\_exp\ n\ (i_1\ ()) = n$
$g\_eval\_exp\ n\ (i_2\ (i_1\ a)) = a$
$g\_eval\_exp\ n\ (i_2\ (i_2\ (i_1\ (Sum,(a,b))))) = a + b$
$g\_eval\_exp\ n\ (i_2\ (i_2\ (i_1\ (Product,(a,b))))) = a * b$
$g\_eval\_exp\ n\ (i_2\ (i_2\ (i_2\ (Negate,a)))) = -a$
$g\_eval\_exp\ n\ (i_2\ (i_2\ (i_2\ (E,a)))) = expd\ a$

--

$clean :: (Eq\ a, Num\ a) \Rightarrow ExpAr\ a \rightarrow () + (a + ((BinOp,(ExpAr\ a, ExpAr\ a)) + (UnOp, ExpAr\ a)))$
$clean\ (N\ a) = outExpAr\ (N\ a)$
$clean\ (Bin\ Product\ \_\ (N\ 0)) = outExpAr\ (N\ 0)$
$clean\ (Bin\ Product\ (N\ 0)\ \_) = outExpAr\ (N\ 0)$
$clean\ (Bin\ Product\ a\ (N\ 1)) = outExpAr\ (a)$
$clean\ (Bin\ Product\ (N\ 1)\ a) = outExpAr\ (a)$
$clean\ (Bin\ Sum\ a\ (N\ 0)) = outExpAr\ (a)$
$clean\ (Bin\ Sum\ (N\ 0)\ a) = outExpAr\ (a)$
$clean\ (Un\ E\ (N\ 0)) = outExpAr\ (N\ 1)$
$clean\ (Bin\ op\ a\ b) = outExpAr\ (Bin\ op\ a\ b)$
$clean\ (Un\ op\ a) = outExpAr\ (Un\ op\ a)$
$clean\ (X) = outExpAr\ (X)$

--

$gopt\ n = g\_eval\_exp\ n$

$sd\_gen :: Floating\ a \Rightarrow$
$() + (a + ((BinOp,((ExpAr\ a, ExpAr\ a),(ExpAr\ a, ExpAr\ a))) + (UnOp,(ExpAr\ a, ExpAr\ a)))) \rightarrow (ExpAr\ a$
$sd\_gen\ (i_2\ (i_1\ a)) = (N\ a, N\ 0)$
$sd\_gen\ (i_2\ (i_2\ (i_1\ (Sum,((a,b),(c,d)))))) = (Bin\ Sum\ (a)\ (c), (Bin\ Sum\ b\ d))$
$sd\_gen\ (i_2\ (i_2\ (i_1\ (Product,((a,b),(c,d)))))) = (Bin\ Product\ a\ c, (Bin\ Sum\ (Bin\ Product\ a\ d)\ (Bin\ Product\ b\ c$

$sd\_gen\ (i_2\ (i_2\ (i_2\ (Negate, (a, b))))) = (Un\ Negate\ a, Un\ Negate\ b)$
$sd\_gen\ (i_2\ (i_2\ (i_2\ (E, (a, b))))) = ((Un\ E\ a), Bin\ Product\ (Un\ E\ a)\ b)$
$sd\_gen\ (i_1\ ()) = (X, N\ 1)$

$ad\_gen :: (Floating\ a, Eq\ a) \Rightarrow$
$\quad a \to () + (a + ((BinOp, ((ExpAr\ a, a), (ExpAr\ a, a))) + (UnOp, (ExpAr\ a, a)))) \to (ExpAr\ a, a)$
$ad\_gen\ x\ (i_2\ (i_1\ a)) = (N\ a, 0)$
$ad\_gen\ x\ (i_2\ (i_2\ (i_1\ (Sum, ((a, b), (c, d)))))) = (Bin\ Sum\ (a)\ (c), b + d)$
$ad\_gen\ x\ (i_2\ (i_2\ (i_1\ (Product, ((a, b), (c, d)))))) = (Bin\ Product\ a\ c, d * (optmize\_eval\ x\ a) + b * (optmize\_eval\ x$
$ad\_gen\ x\ (i_2\ (i_2\ (i_2\ (Negate, (a, b))))) = (Un\ Negate\ a, (-b))$
$ad\_gen\ x\ (i_2\ (i_2\ (i_2\ (E, (a, b))))) = ((Un\ E\ a), (Prelude.exp\ (optmize\_eval\ x\ a)) * b)$
$ad\_gen\ x\ (i_1\ ()) = (X, 1)$

Problema 2

$C\ n = (2\ n)\,!/\,((n + 1)\,! * (n)!)$
$C\ 0 = 1$
$C\ (n + 1) = cima\ n\ /\ baixo\ n$
--
$cima\ n = (2\ n)\,!$
$cima\ (n + 1) = (2\ n)\,! * (2\ n + 2) * (2\ n + 1)$
$cima\ 0 = 1$
$cima\ (n + 1) = cima\ n * d\ n * f\ n$
--
$d\ n = (2\ n + 2)$
$d\ (n + 1) = (2\ n + 2) + 2$
$d\ 0 = 2$
$d\ (n + 1) = 2 + d\ n$
--
$f\ n = (2\ n + 1)$
$f\ (n + 1) = (2\ n + 1) + 2$
$f\ 0 = 1$
$f\ (n + 1) = 2 + f\ n$
--
$baixo\ n = (n + 1)\,! * (n!)$
$baixo\ (n + 1) = (n + 1)\,! * (n!) * (n + 2) * (n + 1)$
$baixo\ 0 = 1$
$baixo\ (n + 1) = baixo\ n * t\ n * next\ n$
--
$t\ n = (n + 2)$
$t\ (n + 1) = (n + 2) + 1$
$t\ 0 = 2$
$t\ (n + 1) = 1 + t\ n$
--
$next\ n = (n + 1)$
$next\ (n + 1) = (n + 1) + 1$
$next\ 0 = 1$
$next\ (n + 1) = 1 + next\ n$

Definir

$loop\ (c, cima, baixo, t, next, d, f) = (cima \div baixo, (mul\ ((mul\ (cima, d)), f)), (mul\ ((mul\ (baixo, t)), next)), (add$
$inic = (1, 1, 1, 2, 1, 2, 1)$
$prj\ (c, cima, baixo, t, next, d, f) = cima \div baixo$

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.

Problema 3

O primeiro passo na resolução deste problema foi a leitura cuidada da documentação. De seguida o grupo procurou assumir calclLne como um cata morfismo de listas, para tal, teriamos de usar cataList. Assumimos calcLine como $(|h|)$ e o seguinte diagrama representa o pretendido:

$$\begin{tikzcd}
NPoint \arrow[rr, bend left, "out"] \arrow[d, "cataList\ h"'] & \cong & 1 + \mathbb{Q} \times NPoint \arrow[ll, bend left, "in"] \arrow[d, "id+id\times calcLine"] \\
NPoint- > OverTimeNPoint & & 1 + \mathbb{Q} \times (NPoint- > OverTimeNPoint) \arrow[ll, "h"]
\end{tikzcd}$$

$$\begin{array}{l}
calcLine :: NPoint \to (NPoint \to OverTime\ NPoint) \\
calcLine = cataList\ h\ \mathbf{where} \\
\quad h = [aux1, aux2] \\
\quad aux1\ _\ _ = id\ nil \\
\quad aux2\ (d, f)\ l = \mathbf{case}\ l\ \mathbf{of} \\
\qquad [\,] \to nil \\
\qquad (x : xs) \to \lambda z \to concat\ \$\ (sequenceA\ [singl \cdot linear1d\ d\ x, f\ xs])\ z
\end{array}$$

De seguida revisitamos a teoria relativa à técnica de Divide and Conquer utilizada no trabalho com hilomorfismos e procuramos uma adaptação ao nosso desafio, atendendo a importância de init e tail.

Assim para definirmos os anamorfismos e o catamorfismos avaliamos a situações iniciais, obtendo o anamorfismo. Sendo a sua saída um NPoint ou um par de listas de NPoint.

Uma vez que o catamorfismo recebe o resultado da chamada recursiva tivemos que adaptar a utilização de calcLine para obtermos o resultado pretendido.

$$\begin{array}{l}
deCasteljau :: [NPoint] \to OverTime\ NPoint \\
deCasteljau = hyloAlgForm\ alg\ coalg\ \mathbf{where} \\
\quad coalg\ [\,] = i_1\ () \\
\quad coalg\ [p] = i_2\ (i_1\ p) \\
\quad coalg\ l = i_2\ (i_2\ (init\ l, tail\ l)) \\
\quad alg = [\underline{nil}, [\underline{\cdot}, aux]] \\
\quad aux\ (p, q) = \lambda pt \to (calcLine\ (p\ pt)\ (q\ pt))\ pt \\
hyloAlgForm\ g\ h = g \cdot (id + (id + ((hyloAlgForm\ g\ h) \times (hyloAlgForm\ g\ h)))) \cdot h
\end{array}$$

Problema 4

Solução para listas não vazias:

Diagramas Lista não vazia.

Diagrama avg

$$\begin{tikzcd}
A^+ \arrow[d, "avg"'] & A + A \times A^+ \arrow[l, "in={[single,cons]}"'] \arrow[d, "id+id\times\langle avg,length\rangle"] \\
\mathbb{N}_0 & A + A \times (\mathbb{N}_0 \times \mathbb{N}_0) \arrow[l, "g{[id,alpha]}"']
\end{tikzcd}$$

diagrama length

$$
\begin{tikzcd}
A^+ \arrow[d, "length"'] & & A + A \times A^+ \arrow[ll, "{in=[single,cons]}"'] \arrow[d, "id+id\times\langle avg,length\rangle"] \\
\mathbb{N}_0 & & A + A \times (\mathbb{N}_0 \times \mathbb{N}_0) \arrow[ll, "{g[\underline{1},succ\cdot\pi_2\cdot\pi_2]}"]
\end{tikzcd}
$$

$$\langle avg, length\rangle = cataL\ \langle [id, alpha], [\underline{1}, \mathsf{succ}\ \cdot (\pi_2 \cdot \pi_2)]\rangle$$

$\equiv$ $\{$ Lei da Troca (28) $\}$

$$\langle avg, length\rangle = cataL\ [\langle id, \underline{1}\rangle, \langle alpha, \mathsf{succ}\ \cdot (\pi_2 \cdot \pi_2)\rangle]$$

$\square$

$avg :: [Double] \rightarrow Double$
$avg = \pi_1 \cdot avg\_aux$

$fL\ f = id + id \times f$
$outL\ [a] = i_1\ a; outL\ (a : l) = i_2\ (a, l)$
$intL = [singl, cons]$
$cataL\ g = g \cdot fL\ (cataL\ g) \cdot outL$
$alpha :: (Fractional\ a, Enum\ a, Ord\ a) \Rightarrow (a, (a, a)) \rightarrow a$
$alpha\ (a, (aver, leng)) = (/)\ ((+)\ ((*)\ aver\ leng)\ a)\ (\mathsf{succ}\ \ leng)$
$avg\_aux = cataL\ [\langle id, \underline{1}\rangle, \langle alpha, \mathsf{succ}\ \cdot (\pi_2 \cdot \pi_2)\rangle]$

Solução para árvores de tipo LTree:
Diagramas LTree.
Diagrama LTree avg

$$
\begin{tikzcd}
LtreeA \arrow[d, "avg"'] & & A + (LtreeA)^2 \arrow[ll, "inLtree"'] \arrow[d, "id+\langle avg,length\rangle"] \\
\mathbb{N}_0 & & A + (\mathbb{N}_0 \times \mathbb{N}_0)^2 \arrow[ll, "{g[id,beta]}"]
\end{tikzcd}
$$

diagrama LTree length

$$
\begin{tikzcd}
LtreeA \arrow[d, "length"'] & & A + (LtreeA)^2 \arrow[ll, "inLtree"'] \arrow[d, "id+\langle avg,length\rangle"] \\
\mathbb{N}_0 & & A + (\mathbb{N}_0 \times \mathbb{N}_0)^2 \arrow[ll, "{g[\underline{1},mylength]}"]
\end{tikzcd}
$$

$$\langle avg, length\rangle = (\!|\ \langle [id, beta], [\underline{1}, mylength]\rangle\ |\!)$$

$\equiv$ $\{$ Lei da Troca (28) $\}$

$$\langle avg, length\rangle = (\!|\ [\langle id, \underline{1}\rangle, \langle beta, mylength\rangle]\ |\!)$$

$\square$

$mylength :: Num\ a1 \Rightarrow ((a2, a1), (a3, a1)) \rightarrow a1$
$mylength\ ((a, b), (c, d)) = (+)\ (b)\ (d)$
$beta :: Fractional\ a \Rightarrow ((a, a), (a, a)) \rightarrow a$
$beta\ ((a, b), (c, d)) = (/)\ ((+)\ ((*)\ a\ b)\ ((*)\ c\ d))\ ((+)\ b\ d)$
$avgLTree :: Fractional\ a \Rightarrow \mathsf{LTree}\ a \rightarrow a$
$avgLTree = \pi_1 \cdot (\!|\ gene\ |\!)$ **where**
$\quad gene = [\langle id, \underline{1}\rangle, \langle beta, mylength\rangle]$

Problema 5

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

```
// (1) Datatype definition -----------------------------------------------------

type BTree<'a> = Empty | Node of 'a * (BTree<'a> * BTree<'a>)


let inBTree x = either (konst Empty) Node x

let outBTree x =
     match x with
     | Empty -> i1()
     | Node (a,(t1,t2)) -> i2(a,(t1,t2))

// (2) Ana + cata + hylo -------------------------------------------------------


let baseBTree f g x = (id -|- (f >< (g >< g))) x

let recBTree f = baseBTree id f

let rec cataBTree g = g << (recBTree (cataBTree g)) << outBTree

let rec anaBTree g = inBTree << (recBTree (anaBTree g) ) << g

let hyloBTree a c = cataBTree a << anaBTree c


// (3) Map ---------------------------------------------------------------------

//instance Functor BTree
//         where fmap f = cataBTree ( inBTree . baseBTree f id )
let fmap f x = (cataBTree ( inBTree << baseBTree f id )) x


// (4) Examples ----------------------------------------------------------------

// (4.1) Inversion (mirror) ----------------------------------------------------

let invBTree x = cataBTree (inBTree << (id -|- (id >< swap))) x

// (4.2) Counting --------------------------------------------------------------

let countBTree x = cataBTree (either zero (succ << add << p2)) x


// (4.3) Serialization ---------------------------------------------------------


// (4.3) Serialization ---------------------------------------------------------

let inord a =
     let join(x,(l,r))=l@[x]@r
     in (either nil join) a

let inordt x = cataBTree inord x
```

```
let preord a =
          let f(x,(l,r))=x::l@r
          in (either nil f) a

let preordt x = cataBTree preord x


let postordt a =
          let f(x,(l,r))=l@r@[x]
          in cataBTree (either nil f) a



// (4.4) QuickSort -------------------------------------------------------


let rec part p l =
     match l with
      | [] -> ([],[])
      | h::t -> if p h then let (s,l) = part p t in (h::s,l) else let s,l = part p t



//let qsep l =
//     match l with
//      | [] -> Left()
//      | h::t -> let s,l = part (<h) t in Right (h,(s,l))


//let qSort x = (hyloBTree inord qsep) x



// (4.5) Traces ------------------------------------------------

//let tunion(a,(l,r)) x = union (map a:: l) (map a:: r) x

//let traces x = cataBTree (either (konst [[]]) tunion) x



// (4.6) Towers of Hanoi ------------------------------------------------------------

let present x = inord x

let strategy(d,l) =
     match l with
      | 0 -> i1()
      | n -> i2((n-1,d),((not d,n-1),(not d,n-1)))

let hanoi x = hyloBTree present strategy x


// (5) Depth and balancing (using mutual recursion) --------------------------


let h(a,((b1,b2),(d1,d2))) = (b1 && b2 && abs(d1-d2)<=1,1+max d1 d2)
```

```
let f((b1,d1),(b2,d2)) = ((b1,b2),(d1,d2))

let baldepth x = cataBTree (either (konst(true,1)) (h<<(id><f))) x

let balBTree x = (p1 << baldepth) x

let depthBTree x = (p2 << baldepth) x
```

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.