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

Departamento de Informática
Universidade do Minho

Junho de 2021

Grupo nr. 53	
a93204	José João Gonçalves
a93314	Maria Sofia Rocha Gomes
a93194	Miguel Rodrigues Santa Cruz

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,

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

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

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

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:

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

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 $\mathit{outExpAr} \cdot \mathit{idExpAr} = \mathit{id}$:*

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

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.*

$$prop\_sum\_idr :: (Floating\ a, Real\ a) \Rightarrow a \rightarrow ExpAr\ a \rightarrow Bool$$
$$prop\_sum\_idr\ a\ exp = eval\_exp\ a\ exp \stackrel{?}{=} sum\_idr\ \mathbf{where}$$
$$\quad sum\_idr = eval\_exp\ a\ (Bin\ Sum\ exp\ (N\ 0))$$
$$prop\_sum\_idl :: (Floating\ a, Real\ a) \Rightarrow a \rightarrow ExpAr\ a \rightarrow Bool$$
$$prop\_sum\_idl\ a\ exp = eval\_exp\ a\ exp \stackrel{?}{=} sum\_idl\ \mathbf{where}$$
$$\quad sum\_idl = eval\_exp\ a\ (Bin\ Sum\ (N\ 0)\ exp)$$
$$prop\_product\_idr :: (Floating\ a, Real\ a) \Rightarrow a \rightarrow ExpAr\ a \rightarrow Bool$$
$$prop\_product\_idr\ a\ exp = eval\_exp\ a\ exp \stackrel{?}{=} prod\_idr\ \mathbf{where}$$
$$\quad prod\_idr = eval\_exp\ a\ (Bin\ Product\ exp\ (N\ 1))$$
$$prop\_product\_idl :: (Floating\ a, Real\ a) \Rightarrow a \rightarrow ExpAr\ a \rightarrow Bool$$
$$prop\_product\_idl\ a\ exp = eval\_exp\ a\ exp \stackrel{?}{=} prod\_idl\ \mathbf{where}$$
$$\quad prod\_idl = eval\_exp\ a\ (Bin\ Product\ (N\ 1)\ exp)$$
$$prop\_e\_id :: (Floating\ a, Real\ a) \Rightarrow a \rightarrow Bool$$
$$prop\_e\_id\ a = eval\_exp\ a\ (Un\ E\ (N\ 1)) \equiv expd\ 1$$
$$prop\_negate\_id :: (Floating\ a, Real\ a) \Rightarrow a \rightarrow Bool$$
$$prop\_negate\_id\ a = eval\_exp\ a\ (Un\ Negate\ (N\ 0)) \equiv 0$$

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

$$prop\_double\_negate :: (Floating\ a, Real\ a) \Rightarrow a \rightarrow ExpAr\ a \rightarrow Bool$$
$$prop\_double\_negate\ a\ exp = eval\_exp\ a\ exp \stackrel{?}{=} eval\_exp\ a\ (Un\ Negate\ (Un\ Negate\ exp))$$

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 $eval$.*

$$prop\_optimize\_respects\_semantics :: (Floating\ a, Real\ a) \Rightarrow a \rightarrow ExpAr\ a \rightarrow Bool$$
$$prop\_optimize\_respects\_semantics\ a\ exp = eval\_exp\ a\ exp \stackrel{?}{=} optmize\_eval\ a\ exp$$

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.

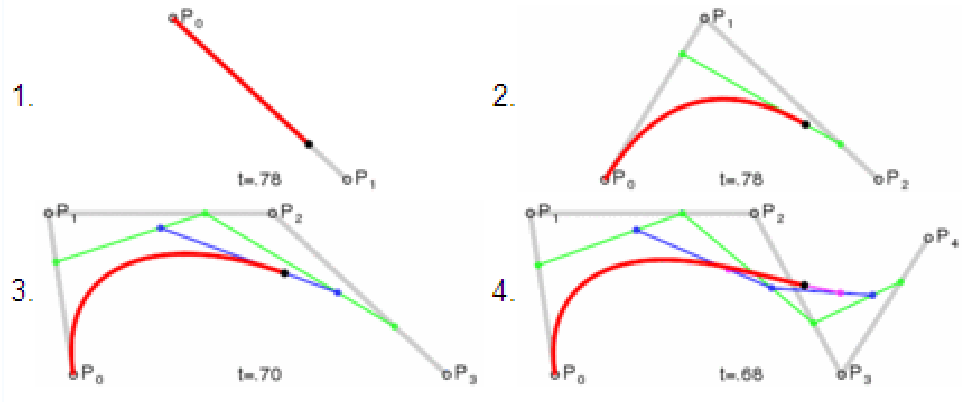

Figura 1: Exemplos de curvas de Bézier retirados da Wikipedia.

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:

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

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.

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

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

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

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

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

O anexo C tem definida a função

$$\mathit{calcLine} :: \mathit{NPoint} \to (\mathit{NPoint} \to \mathit{OverTime}\ \mathit{NPoint})$$

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

$$\mathit{deCasteljau} :: [\mathit{NPoint}] \to \mathit{OverTime}\ \mathit{NPoint}$$

que implementa o algoritmo respectivo.

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

   Propriedade [QuickCheck] 8 *Definição alternativa.*

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

2. Implemente a função $\mathit{deCasteljau}$ como um hilomorfismo, testando agora a propriedade:

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

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

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:*

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

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:

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

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

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

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.

$tick :: Float \rightarrow World \rightarrow World$
$tick\ dt\ world = world\ \{time = (time\ world) + dt\}$

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

$scaleTime :: World \rightarrow Float$
$scaleTime\ w = (1 + cos\ (time\ w))\ /\ 2$

$bezier2dAtTime :: World \rightarrow (Float, Float)$
$bezier2dAtTime\ w = (bezier2dAt\ w)\ (scaleTime\ w)$

$bezier2dAt :: World \rightarrow OverTime\ (Float, Float)$
$bezier2dAt\ w = bezier2d\ (points\ w)$

$thicCirc :: Picture$
$thicCirc = ThickCircle\ 4\ 10$

$ps :: [Float]$
$ps = \mathsf{map}\ \ from_{\mathbb{Q}}\ ps'$ **where**
$\quad ps' :: [\mathbb{Q}]$
$\quad ps' = [0, 0.01 \,..\, 1]$ -- interval

Gloss:

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

Animação:

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

Propriedades e *main*:

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

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

Compilação e execução dentro do interpretador:[11]

$main = runBezier$

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

[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:

$$
\begin{array}{l}
\mathbf{instance}\ \mathit{Arbitrary}\ \mathit{UnOp}\ \mathbf{where}\\
\quad \mathit{arbitrary} = \mathit{elements}\ [\mathit{Negate}, E]\\
\mathbf{instance}\ \mathit{Arbitrary}\ \mathit{BinOp}\ \mathbf{where}\\
\quad \mathit{arbitrary} = \mathit{elements}\ [\mathit{Sum}, \mathit{Product}]\\
\mathbf{instance}\ (\mathit{Arbitrary}\ a) \Rightarrow \mathit{Arbitrary}\ (\mathit{ExpAr}\ a)\ \mathbf{where}\\
\quad \mathit{arbitrary} = \mathbf{do}\\
\qquad \mathit{binop} \leftarrow \mathit{arbitrary}\\
\qquad \mathit{unop} \leftarrow \mathit{arbitrary}\\
\qquad \mathit{exp1} \leftarrow \mathit{arbitrary}\\
\qquad \mathit{exp2} \leftarrow \mathit{arbitrary}\\
\qquad a \quad\ \ \leftarrow \mathit{arbitrary}\\
\qquad \mathit{frequency} \cdot \mathsf{map}\ \ (\mathit{id} \times \mathit{pure})\ \$\ [(20, X), (15, N\ a), (35, \mathit{Bin}\ \mathit{binop}\ \mathit{exp1}\ \mathit{exp2}), (30, \mathit{Un}\ \mathit{unop}\ \mathit{exp1})]\\
\mathbf{infixr}\ 5 \stackrel{?}{=}\\
(\stackrel{?}{=}) :: \mathit{Real}\ a \Rightarrow a \rightarrow a \rightarrow \mathit{Bool}\\
(\stackrel{?}{=})\ x\ y = (\mathit{to}_{\mathbb{Q}}\ x) \equiv (\mathit{to}_{\mathbb{Q}}\ y)
\end{array}
$$

Outras funções auxiliares

Lógicas:

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

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:

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

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

Definir:

outExpAr

$$outExpAr\ X = i_1\ \$\ ()$$
$$outExpAr\ (N\ a) = i_2 \cdot i_1\ \$\ a$$
$$outExpAr\ (Bin\ o\ l\ r) = i_2 \cdot i_2 \cdot i_1\ \$\ (o, (l, r))$$
$$outExpAr\ (Un\ o\ e) = i_2 \cdot i_2 \cdot i_2\ \$\ (o, e)$$

Para este exercicio, sabemos que $outExpAr \cdot inExpAr = id$, sendo este um isomorfismo. Portanto:

$$outExpAr \cdot inExpAr = id \tag{4}$$

$\equiv$ { Def. inExpAr; Natural-id; Reflexão-+ }

$$outExpAr \cdot [\underline{X}, num\_ops] = [i_1, i_2] \tag{5}$$

$\equiv$ { Fusão-+ }

$$[outExpAr \cdot \underline{X}, outExpAr \cdot num\_ops] = [i_1, i_2] \tag{6}$$

$\equiv$ { Eq-+ }

$$\begin{cases} outExpAr \cdot \underline{X} = i_1 \\ outExpAr \cdot num\_ops = i_2 \end{cases} \tag{7}$$

$\equiv$ { Def. num ops; Natural-id; Reflexão-+ }

$$\begin{cases} outExpAr \cdot \underline{X} = i_1 \\ outExpAr \cdot [N, ops] = i_2 \cdot [i_1, i_2] \end{cases} \tag{8}$$

$\equiv$ { Eq-+ (no segundo ramo) }

$$\begin{cases} outExpAr \cdot \underline{X} = i_1 \\ outExpAr \cdot N = i_2 \cdot i_1 \\ outExpAr \cdot ops = i_2 \cdot i_2 \end{cases} \tag{9}$$

$\equiv$ { Def. ops }

$$\begin{cases} outExpAr \cdot \underline{X} = i_1 \\ outExpAr \cdot N = i_2 \cdot i_1 \\ outExpAr \cdot [bin, \widehat{Un}] = i_2 \cdot i_2 \cdot [i_1, i_2] \end{cases} \tag{10}$$

$\equiv$ { Eq-+ }

$$\begin{cases} outExpAr \cdot \underline{X} = i_1 \\ outExpAr \cdot N = i_2 \cdot i_1 \\ outExpAr \cdot bin = i_2 \cdot i_2 \cdot i_1 \\ outExpAr \cdot \widehat{Un} = i_2 \cdot i_2 \cdot i_2 \end{cases} \tag{11}$$

$\equiv$ { Pointwise }

$$\begin{cases} outExpAr\ \underline{X} = i_1\ () \\ outExpAr\ (N\ a) = i_2 \cdot i_1\ (a) \\ outExpAr\ (Bin\ o\ l\ r) = i_2 \cdot i_2 \cdot i_1\ (o, (l, r)) \\ outExpAr\ (Un\ o\ e) = i_2 \cdot i_2 \cdot i_2\ (o, e) \end{cases} \tag{12}$$

$$\Box \tag{13}$$

Deste modo, chegamos à expressao final de outExpAr.

recExpAr

Aplicando a regra: $baseExpAr\ '\ g\ f = baseExpAr\ id\ g\ id\ f\ f\ id\ f$ em $recExpAr$ fica:

$$recExpAr\ id\ g = baseExpAr'\ id\ g \tag{14}$$

$$\equiv \qquad \{ \text{ equivalente } \}$$

$$recExpAr\ id\ g = baseExpAr\ id\ id\ id\ g\ g\ id\ g \tag{15}$$

$$\Box \tag{16}$$

Chegando a esta definição:

$$recExpAr\ f = id + (id + (id \times (f \times f) + id \times f))$$

g_eval_exp

Tendo a definição de $recExpAr$ é possivel obter o seguinte diagrama do catamorfismo de $eval\_exp$:

$$\begin{tikzcd}
ExpArA \arrow[r, bend left, "out"] \arrow[d, "(\!|g_eval_exp|\!)"'] & \cong & 1 + (A + (BinOp \times (ExpArA \times ExpArA)) + (UnOp \times ExpArA)) \arrow[l, bend left, "\mathbf{in}"] \arrow[d, "id+(id+(id\times(eval_exp\times eval_exp)+id\times eval_exp))"] \\
a & & 1 + (a + ((BinOp \times (a \times a)) + UnOp \times a)) \arrow[ll, "g_eval_exp"]
\end{tikzcd}$$

Para descobrir o gene g_eval_exp temos $k = (\!|g\_eval\_exp|\!)$ sabendo $k \cdot \mathbf{in} = g \cdot (id + (k \times k))$.

Portanto:

$$k \cdot [\underline{X}, num\_ops] = [g \cdot i_1, g \cdot i_2 \cdot (k \times k)] \tag{17}$$

$$\equiv \qquad \{ \text{ Eq-+ } \}$$

$$\begin{cases} k \cdot \underline{X} = g \cdot i_1 \\ k \cdot num\_ops = g \cdot i_2 \end{cases} \tag{18}$$

$$\equiv \qquad \{ \text{ Def. num ops; Natural-id; Reflexão-+ } \}$$

$$\begin{cases} k \cdot \underline{X} = g \cdot i_1 \\ k \cdot [N, ops] = g \cdot i_2 \cdot [i_1, i_2] \end{cases} \tag{19}$$

$$\equiv \qquad \{ \text{ Eq-+ (no segundo ramo) } \}$$

$$\begin{cases} k \cdot \underline{X} = i_1 \\ k \cdot N = g \cdot i_2 \cdot i_1 \\ k \cdot ops = g \cdot i_2 \cdot i_2 \end{cases} \tag{20}$$

$$\equiv \qquad \{ \text{ Def. ops } \}$$

$$\begin{cases} k \cdot \underline{X} = i_1 \\ k \cdot N = g \cdot i_2 \cdot i_1 \\ k \cdot [bin, \widehat{Un}] = g \cdot i_2 \cdot i_2 \cdot [i_1, i_2] \end{cases} \tag{21}$$

$\equiv$ $\{$ Eq-+ $\}$

$$\begin{cases} k \cdot \underline{X} = i_1 \\ k \cdot N = g \cdot i_2 \cdot i_1 \\ k \cdot bin = g \cdot i_2 \cdot i_2 \cdot i_1 \\ k \cdot \widehat{Un} = g \cdot i_2 \cdot i_2 \cdot i_2 \end{cases} \tag{22}$$

$\equiv$ $\{$ Pointwise $\}$

$$\begin{cases} k\ \underline{X} = i_1\ () \\ k\ (N\ a) = g \cdot i_2 \cdot i_1\ (a) \\ k\ (Bin\ o\ l\ r) = g \cdot i_2 \cdot i_2 \cdot i_1\ (o, (l, r)) \\ k\ (Un\ o\ e) = g \cdot i_2 \cdot i_2 \cdot i_2\ (o, e) \end{cases} \tag{23}$$

$$\Box \tag{24}$$

A partir disto e com as regras da matemática pode definir-se o gene da seguinte forma:

$g\_eval\_exp\ n\ (i_1\ a) = n;$
$g\_eval\_exp\ n\ (i_2\ (i_1\ (a))) = a;$
$g\_eval\_exp\ n\ (i_2\ (i_2\ (i_1\ (Product, (a, b))))) = a * b;$
$g\_eval\_exp\ n\ (i_2\ (i_2\ (i_1\ (Sum, (a, b))))) = a + b;$
$g\_eval\_exp\ n\ (i_2\ (i_2\ (i_2\ (Negate, b)))) = (-1) * b;$
$g\_eval\_exp\ n\ (i_2\ (i_2\ (i_2\ (E, b)))) = expd\ b;$

optimize_eval
Para tirar proveito dos elementos absorventes de cada operação, definimos um anamorfismo que não altera a estrutura de dados o que permite que o catamorfimo permaneça igual ao definido anteriormente.

$clean\ (Bin\ Product\ \_\ (N\ 0)) = outExpAr\ \$\ N\ 0$
$clean\ (Bin\ Product\ (N\ 0)\ \_) = outExpAr\ \$\ N\ 0$
$clean\ x = outExpAr\ x$
--
$gopt\ a = g\_eval\_exp\ a$

sd_gen
Tendo a definição de $recExpAr$ é possivel obter o seguinte diagrama do catamorfismo de sd:

$$\begin{tikzcd} ExpArA \arrow[rr, bend left, "out"] \arrow[d, "(|sd\_gene|)"'] \arrow[r, phantom, "\cong"] & & 1 + (A + (BinOp \times (ExpArA \times ExpArA)) + (UnOp \times ExpArA)) \arrow[ll, bend left, "\mathbf{in}"] \arrow[d, "id+(id+(id\times(sd\times sd)+id\times sd))"] \\ ExpAr\ A \times ExpAr\ A & & 1 + (A + ((BinOp \times ((ExpArA)^2 \times (ExpArA)^2)) + UnOp \times (ExpArA)^2)) \arrow[ll, "sd\_gen"'] \end{tikzcd}$$

Para descobrir o gene sd_gene temos $k = (|sd\_gene|)$ sabendo $k \cdot \mathbf{in} = g \cdot (id + (k \times k))$.
Portanto:

$$k \cdot [\underline{X}, num\_ops] = [g \cdot i_1, g \cdot i_2 \cdot (k \times k)] \tag{25}$$

$\equiv$ { Eq-+ }

$$\begin{cases} k \cdot \underline{X} = g \cdot i_1 \\ k \cdot num\_ops = g \cdot i_2 \end{cases} \tag{26}$$

$\equiv$ { Def. num ops; Natural-id; Reflexão-+ }

$$\begin{cases} k \cdot \underline{X} = g \cdot i_1 \\ k \cdot [N, ops] = g \cdot i_2 \cdot [i_1, i_2] \end{cases} \tag{27}$$

$\equiv$ { Eq-+ (no segundo ramo) }

$$\begin{cases} k \cdot \underline{X} = i_1 \\ k \cdot N = g \cdot i_2 \cdot i_1 \\ k \cdot ops = g \cdot i_2 \cdot i_2 \end{cases} \tag{28}$$

$\equiv$ { Def. ops }

$$\begin{cases} k \cdot \underline{X} = i_1 \\ k \cdot N = g \cdot i_2 \cdot i_1 \\ k \cdot [bin, \widehat{Un}] = g \cdot i_2 \cdot i_2 \cdot [i_1, i_2] \end{cases} \tag{29}$$

$\equiv$ { Eq-+ }

$$\begin{cases} k \cdot \underline{X} = i_1 \\ k \cdot N = g \cdot i_2 \cdot i_1 \\ k \cdot bin = g \cdot i_2 \cdot i_2 \cdot i_1 \\ k \cdot \widehat{Un} = g \cdot i_2 \cdot i_2 \cdot i_2 \end{cases} \tag{30}$$

$\equiv$ { Pointwise }

$$\begin{cases} k\ \underline{X} = i_1\ () \\ k\ (N\ a) = g \cdot i_2 \cdot i_1\ (a) \\ k\ (Bin\ o\ l\ r) = g \cdot i_2 \cdot i_2 \cdot i_1\ (o, (l, r)) \\ k\ (Un\ o\ e) = g \cdot i_2 \cdot i_2 \cdot i_2\ (o, e) \end{cases} \tag{31}$$

$$\square \tag{32}$$

A partir dos cálculos e das regras da derivação é possivel chegar à seguinte definição:

$$\begin{array}{l} sd\_gen :: Floating\ a \Rightarrow \\ \quad () + (a + ((BinOp, ((ExpAr\ a, ExpAr\ a), (ExpAr\ a, ExpAr\ a))) + (UnOp, (ExpAr\ a, ExpAr\ a)))) \rightarrow (ExpAr\ a \\ sd\_gen\ (i_1\ ()) = (X, N\ 1) \\ sd\_gen\ (i_2\ (i_1\ a)) = (N\ a, N\ 0) \\ sd\_gen\ (i_2\ (i_2\ (i_1\ (Sum, ((a, b), (c, d)))))) = (Bin\ Sum\ a\ c, Bin\ Sum\ b\ d) \\ sd\_gen\ (i_2\ (i_2\ (i_1\ (Product, ((a, b), (c, d)))))) = (Bin\ Product\ a\ c, Bin\ Sum\ (Bin\ Product\ a\ d)\ (Bin\ Product\ b\ c) \\ sd\_gen\ (i_2\ (i_2\ (i_2\ (Negate, (a, b))))) = (Un\ Negate\ a, Un\ Negate\ b) \\ sd\_gen\ (i_2\ (i_2\ (i_2\ (E, (a, b))))) = (Un\ E\ a, Bin\ Product\ (Un\ E\ a)\ b) \end{array}$$

ad_gen

$$
\begin{tikzcd}
ExpArA \arrow[r, phantom, "\cong"] \arrow[rr, bend left, "out"] \arrow[d, "(\!|ad\_gen|\!)"'] & & 1 + (A + (BinOp \times (ExpArA \times ExpArA)) + (UnOp \times ExpArA)) \arrow[ll, bend left, "\mathbf{in}"] \arrow[d, "id+(id+(id\times(ad\times ad)+id\times ad))"] \\
A \times A & & 1 + (A + ((BinOp \times ((ExpArA)^2 \times (ExpArA)^2)) + UnOp \times (ExpArA)^2)) \arrow[ll, "ad\_gen"]
\end{tikzcd}
$$

Para calcular o valor da derivada de uma expressão nesse ponto, sem manipular a expressão original é necessário que o gene do catamorfismo crie um par com o valor original e o valor da derivada. Utilizando os cálculos anteriores e as regras matemáticas chega-se à seguinte definição:

$$
\begin{array}{l}
ad\_gen\ n\ (i_1\ ()) = (n, 1);\\
ad\_gen\ n\ (i_2\ (i_1\ (a))) = (a, 0);\\
ad\_gen\ n\ (i_2\ (i_2\ (i_1\ (Product, ((a, b), (c, d)))))) = ((a * c), (a * d) + (b * c));\\
ad\_gen\ n\ (i_2\ (i_2\ (i_1\ (Sum, ((a, b), (c, d)))))) = (a + c, b + d);\\
ad\_gen\ n\ (i_2\ (i_2\ (i_2\ (Negate, (a, b))))) = ((-1) * (a), (-1) * (b));\\
ad\_gen\ n\ (i_2\ (i_2\ (i_2\ (E, (a, b))))) = ((expd\ a), (expd\ a) * b);
\end{array}
$$

Problema 2

Definir

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

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.

A função pode ser definida recursivamente como se deriva a seguir:

$$C_{n+1} = \frac{(2(n+1))!}{(n+2)!(n+1)!} \iff C_{n+1} = \frac{(2n+2)(2n+1)2n!}{(n+2)(n+1)!(n+1)n!} \iff C_{n+1} = \frac{(2n+2)(2n+1)}{(n+2)(n+1)} \times C_n \iff C_{n+1} = \frac{(2(n+1)(2n+1)}{(n+2)(n+1)} \times C_n \iff C_{n+1} = \frac{4n+2}{n+2} \times C_n$$

A partir daqui podemos definir 3 funções: c, k e s tal que:

$$
\begin{array}{l}
c\ 0 = 1\\
c_{n+1} = c_n * \frac{k_n}{s_n}
\end{array}
$$

$$
\begin{array}{l}
k\ 0 = 2\\
k_n = 4n + 2 \iff k_{n+1} = 4n + 4 + 2k_{n+1} = k_n + 4
\end{array}
$$

$$
\begin{array}{l}
s\ 0 = 2\\
s_n = n + 2 \iff s_{n+1} = n + 2 + 1 \iff s_{n+1} = s_n + 1
\end{array}
$$

Sendo a *inic* igual á inicialização das variáveis, ou seja, *inic = (1,2,2)*, *loop* igual ao corpo do ciclo, *loop (c,k,s) = (c*k 'div' s,k+4,s+1)* e *prj* a função que dado um triplo devolve o primeiro elemento.

Problema 3

$$
\begin{array}{l}
calcLine :: NPoint \rightarrow (NPoint \rightarrow OverTime\ NPoint) \\
calcLine = cataList\ h\ \mathbf{where} \\
\quad h = \bot \\
deCasteljau :: [NPoint] \rightarrow OverTime\ NPoint \\
deCasteljau = hyloAlgForm\ alg\ coalg\ \mathbf{where} \\
\quad coalg = \bot \\
\quad alg = \bot \\
hyloAlgForm = \bot
\end{array}
$$

Problema 4

<u>Solução para listas não vazias:</u>

$$avg = \pi_1 \cdot avg\_aux$$

$$
\begin{array}{l}
avg\_aux = cataList\ [\underline{b}, q]\ \mathbf{where} \\
\quad b = (1, 0) \\
\quad q\ (h, (avg, lng)) = ((h + (avg * lng))\ /\ (\mathsf{succ}\ \ lng), (\mathsf{succ}\ \ lng))
\end{array}
$$

NOTA: Neste caso, uma vez que estamos a trabalhar com listas não vazias, utilizamos $\mathbf{in} = [singl, cons]$ em vez de $\mathbf{in} = [nil, cons]$.

O diagrama correspondente a este exercício traduz-se no seguinte:

$$
\begin{tikzcd}
A+ \arrow[d, "(\![g]\!)"'] & A + A \times A \arrow[l, "inList"'] \arrow[d, "id + id \times (\![g]\!)"] \\
A \times N & A + A\ x\ (N \times N) \arrow[l, "g"]
\end{tikzcd}
$$

Pelo enunciado sabemos que:
$avg\_aux = (\![[b, q]]\!)$ e que $avg\_aux = \langle avg, length \rangle$.
Deste modo, podemos inferir que $\langle avg, length \rangle = (\![[b, q]]\!)$.
Portanto torna-se necessário traduzir este either num split, para que seja possivel aplicar a Lei da Recursividade Mútua (Fokkinga).

$$
\begin{array}{ll}
 & [b, q] \quad (33) \\
\equiv & \quad \{\ \text{Reflexão-x}\ \} \\
 & \langle \pi_1, \pi_2 \rangle \cdot [b, q] \quad (34) \\
\equiv & \quad \{\ \text{Fusão-x}\ \} \\
 & \langle \pi_1 \cdot [b, q], \pi_2 \cdot [b, q], \cdot \rangle \quad (35) \\
\equiv & \quad \{\ \text{Fusão-+}\ \} \\
 & \langle [\pi_1 \cdot b, \pi_1 \cdot q], [\pi_2 \cdot b, \pi_2 \cdot q] \rangle \quad (36) \\
\Box & \quad (37)
\end{array}
$$

Deste modo, conseguimos aplicar diretamente a Lei da Recursividade Mútua (tal como sugerido), uma vez que sabemos que:

$avg\_aux = \langle avg, length \rangle$ é equivalente a $\langle avg, length \rangle = (\!|([\pi_1 \cdot b, \pi_1 \cdot q]\ [\pi_2 \cdot b, \pi_2 \cdot q])|\!)$

$$\tag{38}$$

$$\equiv \qquad \{ \text{ Fokkinga } \}$$

$$\begin{cases} avg \cdot \mathbf{in} = [\pi_1 \cdot b, \pi_1 \cdot q] \cdot F\ \langle avg, length \rangle \\ length \cdot \mathbf{in} = [\pi_2 \cdot b, \pi_2 \cdot q] \cdot F\ \langle avg, length \rangle \end{cases} \tag{39}$$

$$\equiv \qquad \{ \text{ Def. in; Ff = id + id x f } \}$$

$$\begin{cases} avg \cdot [singl, cons] = [\pi_1 \cdot b, \pi_1 \cdot q] \cdot (id + id \times \langle avg, length \rangle) \\ length \cdot [singl, cons] = [\pi_2 \cdot b, \pi_2 \cdot q] \cdot (id + id \times \langle avg, length \rangle) \end{cases} \tag{40}$$

$$\equiv \qquad \{ \text{ Fusão-+; Absorção-+ } \}$$

$$\begin{cases} [avg \cdot singl, avg \cdot cons] = [\pi_1 \cdot b, \pi_1 \cdot q \cdot (id \times \langle avg, length \rangle)] \\ [length \cdot singl, length \cdot cons] = [\pi_2 \cdot b, \pi_2 \cdot q \cdot (id \times \langle avg, length \rangle)] \end{cases} \tag{41}$$

$$\equiv \qquad \{ \text{ Eq-+ } \}$$

$$\begin{cases} avg \cdot singl = \pi_1 \cdot b \\ avg \cdot cons = \pi_1 \cdot q \cdot (id \times \langle avg, length \rangle) \\ length \cdot singl = \pi_2 \cdot b \\ lenght \cdot cons = \pi_2 \cdot q \cdot (id \times \langle avg, length \rangle) \end{cases} \tag{42}$$

$$\equiv \qquad \{ \text{ Pointwise } \}$$

$$\begin{cases} avg \cdot singl\ l = (\pi_1 \cdot b)\ l \\ avg \cdot cons\ (h, t) = \pi_1 \cdot q \cdot (id \times \langle avg, length \rangle)\ (h, t) \\ length \cdot singl\ l = (\pi_2 \cdot b)\ l \\ lenght \cdot cons\ (h, t) = \pi_2 \cdot q \cdot (id \times \langle avg, length \rangle)\ (h, t) \end{cases} \tag{43}$$

$$\equiv \qquad \{ \text{ Assoc-comp; Def. cons } \}$$

$$\begin{cases} avg \cdot singl\ l = \pi_1 \cdot (b\ l) \\ avg\ (h : t) = \pi_1 \cdot q \cdot (id\ h, \langle avg, length \rangle\ t) \\ length \cdot singl\ l = \pi_2 \cdot (b\ l) \\ lenght\ (h : t) = \pi_2 \cdot q \cdot (id\ h, \langle avg, length \rangle\ t) \end{cases} \tag{44}$$

$$\equiv \qquad \{ \text{ Def-split } \}$$

$$\begin{cases} avg \cdot singl\ l = \pi_1 \cdot (b\ l) \\ avg \cdot cons\ (h : t) = \pi_1\ (q\ (h, (avg\ t, length\ t))) \\ length \cdot singl\ l = \pi_2 \cdot (b\ l) \\ lenght \cdot cons\ (h : t) = \pi_2\ (q\ (h, (avg\ t, length\ t))) \end{cases} \tag{45}$$

$$\square \tag{46}$$

Após chegar a este ponto, é possível inferir que temos que tomar b e q como pares. Assim, temos $b = (1, 1)$ e $q\ (h, (avg, lng)) = ((h + (avg * lng)) \ / \ (\mathsf{succ}\ \ lng), (\mathsf{succ}\ \ lng))$.

Note-se que para q utilizámos o cálculo da média ponderada para o primeiro elemento do par e o incremento do length para o segundo.

Solução para árvores de tipo LTree:

$$\begin{array}{l} avgLTree = \widehat{(/)} \cdot \langle \pi_1 \cdot (\!|\ gene\ |\!), \pi_2 \cdot (\!|\ gene\ |\!) \rangle\ \mathbf{where} \\ \quad h1 = id \\ \quad h2 = \widehat{(+)} \cdot (\pi_1 \times \pi_1) \\ \quad k1 = \underline{1} \\ \quad k2 = \widehat{(+)} \cdot (\pi_2 \times \pi_2) \\ \quad b = \langle h1, k1 \rangle \end{array}$$

$$q = \langle h2, k2 \rangle$$
$$gene = [b, q]$$

Para este exercicio, resolvemos utilizar um $\langle sum, length \rangle$, uma vez que nos facilita o cálculo e, deste modo apenas temos que efetuar a divisão entre os membros deste split para obter a média.

O diagrama correspondente a este exercício traduz-se no seguinte:

$$\begin{tikzcd}
\mathsf{LTree}\ A \arrow[d, "(\!|\, g\,|\!)"'] & A + (\mathsf{LTree}\ A\ x\ \mathsf{LTree}\ A) \arrow[l, "\mathbf{in}"'] \arrow[d, "id + (\!|\, g\,|\!) \times (\!|\, g\,|\!)"] \\
A \times Nat & A + (A \times N)\ x\ (A \times N) \arrow[l, "g"]
\end{tikzcd}$$

Assim, passamos a demonstrar os cálculos:

$$\left\{ \begin{array}{l} sum \cdot \mathbf{in} = h \cdot F\ \langle sum, length \rangle \\ len \cdot \mathbf{in} = k \cdot F\ \langle sum, length \rangle \end{array} \right. \tag{47}$$

$$\equiv \qquad \{\ h = [h1, h2];\ k = [k1, k2];\ \text{Ff = id + f x f}\ \}$$

$$\left\{ \begin{array}{l} sum \cdot \mathbf{in} = [h1, h2] \cdot (id + \langle sum, length \rangle\ x\ \langle sum, length \rangle) \\ len \cdot \mathbf{in} = [k1, k2] \cdot (id + \langle sum, length \rangle\ x\ \langle sum, length \rangle\ 0) \end{array} \right. \tag{48}$$

$$\equiv \qquad \{\ \text{Absorção-+; Def. in; Eq-+}\ \}$$

$$\left\{ \begin{array}{l} sum \cdot Leaf = h1 \\ sum \cdot Fork = h2 \cdot (\langle sum, length \rangle\ x\ \langle sum, length \rangle) \\ length \cdot Leaf = k1 \\ lenght \cdot Fork = k2 \cdot (\langle sum, length \rangle\ x\ \langle sum, length \rangle) \end{array} \right. \tag{49}$$

$$\square \tag{50}$$

Neste ponto, é preciso ter em conta o objetivo de cada ramo.
- Para o primeiro ramo, sabemos que o somatório de uma folha é o proprio valor (dividir por 1).
- Para o segundo ramo, temos que somar os valores dos somatórios decorrentes de cada ramo.
- Para o terceiro ramo, sabemos que o tamanho de uma arvore com apenas uma folha é 1.
- Finalmente, para o quarto ramo, queremos atualizar o length, somando os length's das duas arvores descendentes do nodo em questão.

Deste modo, conseguimos obter os valores para h1, h2, k1 e k2.
Posteriormente, temos que ter em conta que no final do cálculo temos $\langle [h1, h2], [k1, k2] \rangle$, mas sendo o gene um either, temos que aplicar a Lei da Troca como meio de obter um either de splits, ficando então $[\langle h1, k1 \rangle, \langle h2, k2 \rangle]$, sendo $b = \langle h1, k1 \rangle$ e $q = \langle h2, k2 \rangle$.

Por fim, para obter o avg, temos apenas que efetuar a divisão do somatório pelo tamanho, recorrendo ao $\widehat{(/)}$.

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.