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

Departamento de Informática
Universidade do Minho

Junho de 2021

Grupo nr.	93
a71452	Diogo Filipe Ferreira Pereira Monteiro
a80292	Joao Aniceto Rodrigues Rocha
a82726	Matias Abreu Capitão
a77457	Rafael Antunes Simões

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" [**?**], cujo princípio base é o seguinte:

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

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

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

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

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

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

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

```
$ ghci cp2021t.lhs
```

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

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

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

é seleccionado pelo GHCi para ser executado.

3 Como realizar o trabalho

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

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

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

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

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

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

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

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

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

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

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

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

3.1 Stack

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

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

Pode aceder ao GHCi utilizando o comando:

```
stack ghci
```

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

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

Problema 1

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

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

- *Symbolic differentiation*
- *Automatic differentiation*

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

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

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

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

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.*

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

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

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

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 $eval$.*

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

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.*

$$
\begin{array}{l}
prop\_const\_rule :: (Real\ a, Floating\ a) \Rightarrow a \rightarrow Bool \\
prop\_const\_rule\ a = sd\ (N\ a) \equiv N\ 0 \\
prop\_var\_rule :: Bool \\
prop\_var\_rule = sd\ X \equiv N\ 1 \\
prop\_sum\_rule :: (Real\ a, Floating\ a) \Rightarrow ExpAr\ a \rightarrow ExpAr\ a \rightarrow Bool \\
prop\_sum\_rule\ exp1\ exp2 = sd\ (Bin\ Sum\ exp1\ exp2) \equiv sum\_rule\ \textbf{where} \\
\quad sum\_rule = Bin\ Sum\ (sd\ exp1)\ (sd\ exp2) \\
prop\_product\_rule :: (Real\ a, Floating\ a) \Rightarrow ExpAr\ a \rightarrow ExpAr\ a \rightarrow Bool \\
prop\_product\_rule\ exp1\ exp2 = sd\ (Bin\ Product\ exp1\ exp2) \equiv prod\_rule\ \textbf{where} \\
\quad prod\_rule = Bin\ Sum\ (Bin\ Product\ exp1\ (sd\ exp2))\ (Bin\ Product\ (sd\ exp1)\ exp2) \\
prop\_e\_rule :: (Real\ a, Floating\ a) \Rightarrow ExpAr\ a \rightarrow Bool \\
prop\_e\_rule\ exp = sd\ (Un\ E\ exp) \equiv Bin\ Product\ (Un\ E\ exp)\ (sd\ exp) \\
prop\_negate\_rule :: (Real\ a, Floating\ a) \Rightarrow ExpAr\ a \rightarrow Bool \\
prop\_negate\_rule\ exp = sd\ (Un\ Negate\ exp) \equiv Un\ Negate\ (sd\ exp)
\end{array}
$$

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 [?], página 98.

$$f\ 0 = 1$$
$$f\ (n+1) = \mathit{fib}\ n + f\ n$$

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:

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

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 [?] e tópico Recursividade mútua nos vídeos das aulas teóricas.

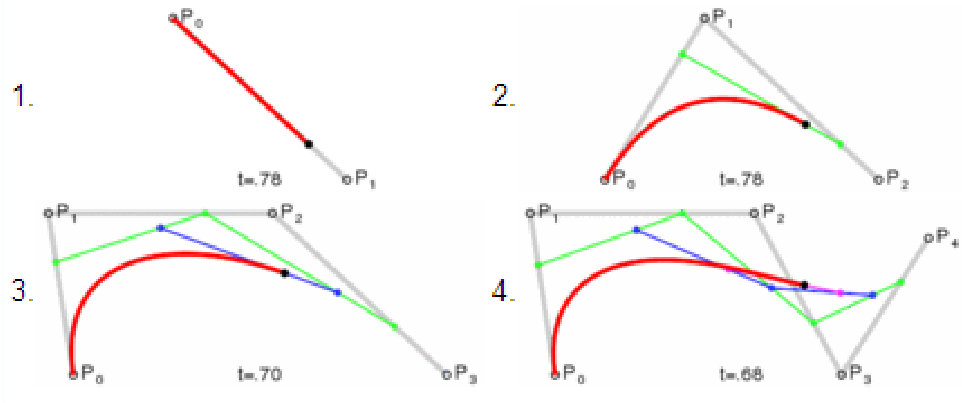


Figura 1: Exemplos de curvas de Bézier retirados da Wikipedia.

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:

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

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:

$$p2d = [1.2, 3.4]$$
$$p3d = [0.2, 10.3, 2.4]$$

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.*

   $$prop\_calcLine\_def :: NPoint \to NPoint \to Float \to Bool$$
   $$prop\_calcLine\_def\ p\ q\ d = calcLine\ p\ q\ d \equiv zipWithM\ linear1d\ p\ q\ d$$

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

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

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

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:

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

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:

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

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 [?].
[9]Cf. [?], página 102.

C Código fornecido

Problema 1

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

Problema 2

Definição da série de Catalan usando factoriais (4):

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

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

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

Problema 3

Algoritmo:

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

Função auxiliar:

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

2D:

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

Modelo:

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

[10] Fonte: Wikipedia.

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

Gloss:

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

Animação:

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

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:

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

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

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:

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

$$
\begin{array}{l}
eval\_exp :: Floating\ a \Rightarrow a \rightarrow (ExpAr\ a) \rightarrow a \\
eval\_exp\ a = cataExpAr\ (g\_eval\_exp\ a)
\end{array}
$$

$$
\begin{array}{l}
optmize\_eval :: (Floating\ a, Eq\ a) \Rightarrow a \rightarrow (ExpAr\ a) \rightarrow a \\
optmize\_eval\ a = hyloExpAr\ (gopt\ a)\ clean
\end{array}
$$

$$
\begin{array}{l}
sd :: Floating\ a \Rightarrow ExpAr\ a \rightarrow ExpAr\ a \\
sd = \pi_2 \cdot cataExpAr\ sd\_gen
\end{array}
$$

$$
\begin{array}{l}
ad :: Floating\ a \Rightarrow a \rightarrow ExpAr\ a \rightarrow a \\
ad\ v = \pi_2 \cdot cataExpAr\ (ad\_gen\ v)
\end{array}
$$

Exercício 1.1

Sabemos que $outExpAr$. $inExpAr$ = id e como tal, torna-se simples encontrar $outExpAr$:

$$
\begin{array}{ll}
 & outExpAr \cdot inExpAr = id \\
\equiv & \qquad \{ \text{ Definição inExpAr } \} \\
 & outExpAr \cdot [\underline{X}, num\_ops] \\
\equiv & \qquad \{ \text{ Fusão-+ } \} \\
 & [outExpAr \cdot \underline{X}, outExpAr \cdot num\_ops] = id \\
\equiv & \qquad \{ \text{ Universal-+ e Definição num\_ops } \} \\
 & \left\{ \begin{array}{l} outExpAr \cdot \underline{X} = i_1 \\ outExpAr \cdot [N, ops] = i_2 \end{array} \right. \\
\equiv & \qquad \{ \text{ Definição ops } \} \\
 & \left\{ \begin{array}{l} outExpAr \cdot \underline{X} = i_1 \\ outExpAr \cdot [N, [Bin\ op\ a\ b, \widehat{Un}]] = i_2 \end{array} \right. \\
\equiv & \qquad \{ \text{ Fusão-+ } \} \\
 & \left\{ \begin{array}{l} outExpAr \cdot \underline{X} = i_1 \\ [outExpAr \cdot N, outExpAr \cdot [Bin\ op\ a\ b, \widehat{Un}]] = i_2 \end{array} \right. \\
\equiv & \qquad \{ \text{ Universal-+ } \} \\
 & \left\{ \begin{array}{l} outExpAr \cdot \underline{X} = i_1 \\ \left\{ \begin{array}{l} outExpAr \cdot N = i_2 \cdot i_1 \\ outExpAr \cdot [Bin\ op\ a\ b, \widehat{Un}] = i_2 \cdot i_2 \end{array} \right. \end{array} \right. \\
\equiv & \qquad \{ \text{ Fusão-+ e Universal-+ } \} \\
 & \left\{ \begin{array}{l} outExpAr \cdot \underline{X} = i_1 \\ \left\{ \begin{array}{l} outExpAr \cdot N = i_2 \cdot i_1 \\ \left\{ \begin{array}{l} outExpAr \cdot (Bin\ op\ a\ b) = i_2 \cdot i_2 \cdot i_1 \\ outExpAr \cdot \widehat{Un} = i_2 \cdot i_2 \cdot i_2 \end{array} \right. \end{array} \right. \end{array} \right. \\
\equiv & \qquad \{ \text{ Introdução De Variáveis } \} \\
 & \left\{ \begin{array}{l} outExpAr\ X = i_1\ () \\ \left\{ \begin{array}{l} outExpAr\ (N\ x) = i_2\ (i_1\ x) \\ \left\{ \begin{array}{l} outExpAr\ (Bin\ a\ b\ c) = i_2\ (i_2\ (i_1\ (a, (b, c)))) \\ outExpAr\ (Un\ a\ b) = i_2\ (i_2\ (i_2\ (a, b))) \end{array} \right. \end{array} \right. \end{array} \right. \\
\Box &
\end{array}
$$

$$
\begin{array}{l}
outExpAr\ X = i_1\ () \\
outExpAr\ (N\ x) = i_2\ (i_1\ x)
\end{array}
$$

$$outExpAr\ (Bin\ a\ b\ c) = i_2\ (i_2\ (i_1\ (a, (b, c))))$$
$$outExpAr\ (Un\ a\ b) = i_2\ (i_2\ (i_2\ (a, b)))$$

Temos então definido o seguinte isomorfismo:

$$\begin{tikzcd}
ExpAr \arrow[r, bend left, "outExpAr"] & 1 + (A + ((BinOp \times (ExpAr \times ExpAr)) + (UnOp \times ExpAr))) \arrow[l, bend left, "inExpAr"]
\end{tikzcd}$$

Tendo em conta a informação fornecida pelo professor no video '10a' que diz que, podemos obter o funtor recursivo, utilizando o funtor de base e trocando o primeiro parâmetro e passando-o a identidade (id). Na FAQ 'Q9' presente no *website* da unidade curricular, temos a definição de $baseExpAr'$ que faz com que seja bastante simples chegar à definição de *recExpAr*:

$$baseExpAr'\ g\ f = baseExpAr\ id\ g\ id\ f\ f\ id\ f$$
$$recExpAr\ f = baseExpAr'\ id\ f$$

Exercício 1.2

De forma algo intuitiva é possível perceber como será defenido este gene tendo em conta o seguinte. Se $ExpAra$:

- É igual a X, então o resultado pertendido é x;
- É igual a $N\ a$, o resultado pertendido é o próprio a;
- É igual a $BinOp\ Sum\ (a, b)$, o resultado pertendido é $a + b$;
- É igual a $BinOp\ Product\ (a, b)$, o resultado pertendido é $a \times b$;
- É igual a $UnOp\ Negate\ a$, o resultado pertendido é $(-1) \times a$;
- É igual a $UnOp\ E\ a$, o resultado pertendido é $expda$.

O gene do catamorfismo da função eval_exp será definido da seguinte maneira:

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

E o diagrama deste catamorfismo terá o seguinte aspecto:

$$\begin{tikzcd}
ExpAr \arrow[d, "(\!|g_eval_exp|\!)"'] \arrow[r, bend left, "outExpAr"] & 1 + (A + ((BinOp \times (ExpAr \times ExpAr)) + (UnOp \times ExpAr))) \arrow[l, bend left, "inExpAr"] \arrow[d, "id+(id+((id\times((\!|g_eval_exp|\!)\times(\!|g_eval_exp|\!)))+(id\times(\!|g_eval_exp|\!)))"] \\
A & 1 + (A + ((BinOp \times (A \times A)) + (UnOp \times A))) \arrow[l, bend left, "g_eval_exp"]
\end{tikzcd}$$

Exercício 1.3

Como dito nas aulas práticas, o que um tipico programador funcional faria seria generalizar o catamorfismo e um anamorfismo para se converter num hilomorfismo para otimização.

sabendo que hylo f g = cata f . ana g

cata f é representado como g_eval_exp a ana g é representado como outExpAr x onde ana consome os dados e cata tranforma dados

--

$clean\ (Bin\ Product\ \_\ (N\ 0)) = outExpAr\ \$\ N\ 0$
$clean\ (Bin\ Product\ (N\ 0)\ \_) = outExpAr\ \$\ N\ 0$
$clean\ x = outExpAr\ x$

$gopt\ a = g\_eval\_exp\ a$

Exercício 1.4

Como nos é dado no enunciado é necessário aplicar transformaçõoes a expressão original que respeitem as regras das derivadas. Para generalizar o que é dado no enunciado temos os seguintes casos modificados:

- Sum: sd_gen (Right (Right (Left (Sum, ((a,b),(c,d)))))) = (Bin Sum a c,Bin Sum b d)
- product: sd_gen (Right (Right (Left (Product, ((a,b),(c,d))))))= (Bin Product a c,Bin Sum (Bin Product a d) (Bin Product b c))
- negate: sd_gen (Right (Right (Right (Negate, (a,b)))))= (Un Negate a,Un Negate b)
- E: sd_gen (Right (Right (Right (E, (a,b)))))= (Un E a,Bin Product (Un E a) b)

Ainda para receber estes tipos e tranforma-los temos de os definir:

- para que o BinOp possa receber dois pares fica ((ExpAr a, ExpAr a), (ExpAr a, ExpAr a)))
- para que o UnOp possa receber também um par ficar (UnOp, (ExpAr a, ExpAr a)))

Aplicando assim:

Floating a $\Rightarrow Either()(Eithera(Either(BinOp,((ExpAra,ExpAra),(ExpAra,ExpAra)))(UnOp,(ExpAra,ExpA$
$(ExpAra,ExpAra)$

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

Exercício 1.5

O mesmo raciocinio do 1.4 se aplica a este exercício tendo em conta agora que queremos calcular o valor da derivada no ponto passado como argumento. Como queremos encontrar o gene de ad podemos assumir o próximo diagrama:

$$
\begin{tikzcd}
ExpAr \arrow[r, bend left, "outExpAr"] \arrow[d, "(\!|ad\_gen|\!)"] \arrow[dd, bend right, "ad"'] & 1 + (A + ((BinOp \times (ExpAr \times ExpAr)) + (UnOp \times ExpAr))) \arrow[l, bend left, "inExpAr"] \arrow[d, "id + (id + ((id \times ((\!|ad\_gen|\!) \times (\!|ad\_gen|\!))) + (id \times (\!|ad\_gen|\!))"] \\
(A \times A) \arrow[d, "\pi_2"] & 1 + (A + ((BinOp \times ((A \times A) \times (A \times A))) + (UnOp \times (A \times A)))) \arrow[l, "ad\_gen"] \\
A &
\end{tikzcd}
$$

A solução encontrada é a seguinte:

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

Problema 2

Resposta:

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

$$
\begin{aligned}
&loop\ (cat, h1, h2, i) = (cat * h1 \div h2, (f'\ 4\ 6\ 2\ i), (f'\ 1\ 3\ 2\ i), \mathsf{succ}\ \ i) \\
&inic = (1, 2, 2, 1) \\
&prj\ (cat, h1, h2, i) = cat
\end{aligned}
$$

O objetivo deste problema é, dada a fórmula C_n que calcula o n-ésimo número de Catalan, derivar um ciclo-for que calcule este número sem utilizar fatoriais nos cálculos.

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

Como não podemos utilizar fatoriais, temos de perceber como C_n pode ser representado como uma expressão recursiva. Para tal, calculamos C_{n+1} e simplificamos até obter então uma expressão recursiva. Os seguintes cálculos foram feitos para chegar ao pretendido:

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

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

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

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

$$= \frac{4n^2+6n+2}{n^2+3n+2} \times C_n$$

Mais simplificações poderiam ter sido feitas na parte da divisão dos dois polinómios de segundo grau, porém, como no enunciado é dada a definição de $fx = ax^2 + bx + c$, derivada em duas funções mutuamente recursivas, utilizá-la-emos na nossa resolução.
Para evitar erros de aproximação, optamos por fazer a divisão o mais tarde possível, e como tal, a nossa implementação de C_{n+1} terá a seguinte estrutura:

$$C_{n+1} = \frac{(4n^2+6n+2) \times C_n}{n^2+3n+2}$$

Posto isto, e tendo em conta as regras listadas no enunciado para derivar um ciclo-for, temos a informação necessária para resolver o problema. Temos de definir `prj`, `loop` e `inic`, tal que:

```
cat = prj.(for loop inic)
```

A nossa solução é a seguinte:

```
prj(cat, h1, h2, i) = cat
loop (cat, h1, h2, i) = (div (cat * h1) h2,(f' 4 6 2 i),(f' 1 3 2 i), succ i)
inic = (1,2,2,1)
```

Ou seja:

```
cat = div (cat * h1) h2
h1 = f' 4 6 3 i
h2 = f' 1 3 2 i
i = succ i
```

Para chegar a esta solução, para além das sugestões passadas no enunciado, tivemos em conta o seguinte:

- A $4n^2 + 6n + 2$ aplicamos a definição de `f'`, ficando com o seguinte código Haskell `f' 4 6 2 x` para calcular o valor deste polinómio no valor x.
- O mesmo é aplicado em $n^2 + 3n + 2$ resultando em `f' 1 3 2 x`.
- Adicionar uma variável, `i`, que irá incrementar a cada chamada recursiva. Sentimos a necessidade de a aplicar pois a omissão da mesma não nos permitia correr a função. Esta variável é igual a `n`. Por exemplo, calculando "manualmente" `cat 3` onde `n = 2` e `i = 2`:

```
cat (n+1) = div (cat n * h1 i) (h2 i)
cat 3 = cat (2+1) = div (cat 2 * h1 2) (h2 2) = 5
```

Reconhecemos que algo poderia ter sido feito para evitar a implementação deste `i`, porém não conseguimos utilizar a função `f'` omitindo o argumento.

$$C_n = \frac{(2n)!}{(n+1)!(n!)} \tag{4}$$

Problema 3

Como queremos implementar $calcLine$ como um catamorfismo de listas, assumimos $calcLine == (\!|gene|\!)$ e o seguinte diagrama representa o pretendido:

$$\begin{tikzcd}
NPoint^* \arrow[rr, bend left, "out"] \arrow[d, "(\!|gene|\!)"'] & & 1 + NPoint \times NPoint^* \arrow[ll, bend left, "in"] \arrow[d, "id+id\times(\!|gene|\!)"] \\
OverTimeNPoint & & 1 + NPoint \times OverTimeNPoint, \arrow[ll, bend left, "gene={[gCL1,gCL2]}"]
\end{tikzcd}$$

$gCL1\ () = \underline{nil}$
$gCL2\ (p, x) = (\overline{g}\ p\ x)$ **where**
$\quad g\ (d, f)\ l =$ **case** l **of**
$\quad\quad [\,] \rightarrow nil$
$\quad\quad (x : xs) \rightarrow \lambda z \rightarrow concat\ \$\ (sequenceA\ [singl \cdot linear1d\ \ d\ x, f\ \ xs])\ z$

$calcLine :: NPoint \rightarrow (NPoint \rightarrow OverTime\ NPoint)$
$calcLine = cataList\ h$ **where**
$\quad h = [gCL1, gCL2]$

Para definirmos o gene do anamorfismo e do catamorfismo temos de compreender o que deve acontecer no 'divide' e no 'conquer'. Para tal fizemos o seguinte esquema:

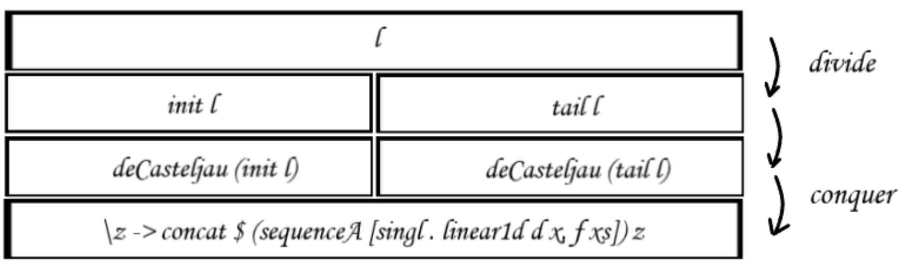


Figura 2: Esquema para percebermos o que se passa no divide e no conquer.

Uma vez que temos três situações iniciais, nomeadamente:

- Lista vazia;
- Lista com apenas um elemento;

- Lista com dois ou mais elementos.

O gene do anamorfismo será

$$
\begin{array}{l}
\mathit{gCastel1}\ [\,] = i_1\ \mathit{nil} \\
\mathit{gCastel1}\ [p] = i_1\ \underline{p} \\
\mathit{gCastel1}\ l = i_2\ (\mathit{init}\ l, \mathit{tail}\ l)
\end{array}
$$

De notar que, a saída deste gene será um NPoint ou um par de listas de NPoint. Se formos aos apontamentos da disciplina podemos verificar que é igual ao das LTrees. Portanto serão utilizados o cataLTree e o anaLTree.

O gene do catamorfismo, uma vez que, recebe o resultado da chamada recursiva ficará:

$$
\begin{array}{l}
\mathit{gCastel2a}\ l = l \\
\mathit{gCastel2b}\ (x, y) = \lambda pt \rightarrow (\mathit{calcLine}\ (x\ pt)\ (y\ pt))\ pt \\
\mathit{gCastel2} = [\mathit{gCastel2a}, \mathit{gCastel2b}] \\
\mathit{deCasteljau} :: [\mathit{NPoint}] \rightarrow \mathit{OverTime}\ \mathit{NPoint} \\
\mathit{deCasteljau} = \mathit{hyloAlgForm}\ \mathit{alg}\ \mathit{coalg}\ \textbf{where} \\
\quad \mathit{coalg} = \llparenthesis\, \mathit{gCastel2}\, \rrparenthesis \\
\quad \mathit{alg} = [\!(\, \mathit{gCastel1}\,)\!]
\end{array}
$$

Como vimos nas aulas, um hilomorfismo é um catamorfismo apòs o anamorfismo. Ou seja:

$$
\mathit{hyloAlgForm}\ a\ b = b \cdot a
$$

Posto isto, podemos desenhar o diagrama deste hilomorfismo:

$$
\begin{tikzcd}
\mathit{NPoint}{*} \arrow[rr, "\mathit{gCastel1}"'] \arrow[d, "{[\!(\mathit{gCastel1})\!]}"'] & & \mathit{NPoint} + \mathit{NPoint}{*} \times \mathit{NPoint}{*} \arrow[d, "{id + [\!(\mathit{gCastel1})\!] \times [\!(\mathit{gCastel1})\!]}"] \\
\mathit{NPoint} + \mathit{NPoint}{*} \times \mathit{NPoint}{*} \arrow[d, "\llparenthesis \mathit{gCastel2} \rrparenthesis"'] & & \mathit{NPoint} + (\mathit{NPoint} + \mathit{NPoint}{*} \times \mathit{NPoint}{*}) \times (\mathit{NPoint} + \mathit{NPoint}{*} \times \mathit{NPoi} \arrow[d, "id + \llparenthesis \mathit{gCastel2} \rrparenthesis \times \llparenthesis \mathit{gCastel2} \rrparenthesis"] \\
\mathit{OverTime}\ \mathit{NPoint} & & \mathit{NPoint} + \mathit{OverTime}\ \mathit{NPoint} \times \mathit{OverTime}\ \mathit{NPoint} \arrow[ll, "\mathit{gCastel2}"]
\end{tikzcd}
$$

Temos aqui agora alguns resultados de correr `runBezier`:

Figura 3: *Print Screen* de um curva de Bezier com 3 pontos

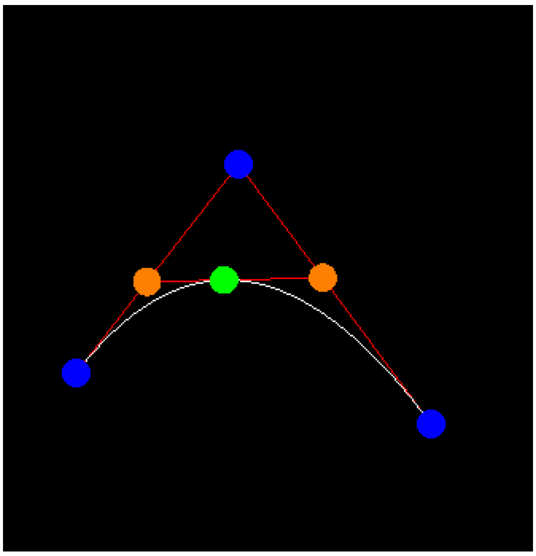
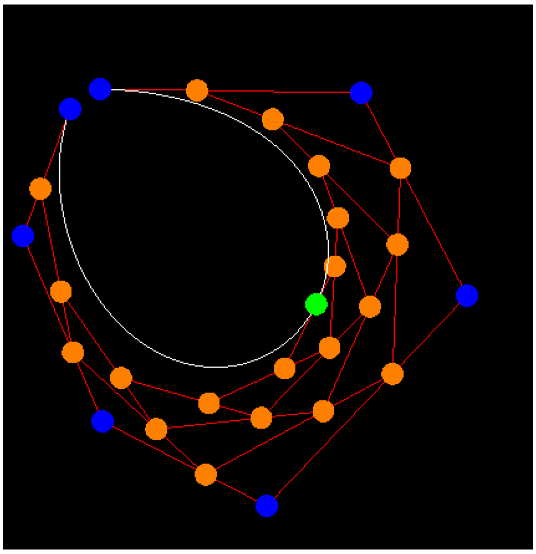


Figura 4: *Print Screen* de um curva de Bezier com vários pontos



Problema 4

Solução para listas não vazias:

De modo a resolvermos este problema precisamo de compreender primeiro qual o in de listas não vazias. Ora, este não difere muito do inNat definido na biblioteca List.hs. Temos apenas de ter em conta que existe um elemento na lista. Ou seja, o in será [singl, cons].

O *out* de listas não vazias será calculado da mesma maneira que calculamos o outExpAr no Problema 1, ou seja, tendo em conta que estamos perante um isomorfismo ($outListasNaoVazias \cdot inListasNaoVazias = id$) :

$$
\begin{array}{ll}
 & outListasNaoVazias \cdot inListasNaoVazias = id \\
\equiv & \quad \{ \text{ Definição inListasNaoVazias } \} \\
 & outListasNaoVazias \cdot [singl, cons] \\
\equiv & \quad \{ \text{ Fusão-+ } \} \\
 & [outListasNaoVazias \cdot singl, outListasNaoVazias \cdot cons] = id \\
\equiv & \quad \{ \text{ Universal-+ } \} \\
 & \left\{ \begin{array}{l} outListasNaoVazias \cdot singl = i_1 \\ outListasNaoVazias \cdot cons = i_2 \end{array} \right. \\
\equiv & \quad \{ \text{ Introdução de variáveis } \} \\
 & \left\{ \begin{array}{l} outListasNaoVazias\ [a] = i_1\ a \\ outListasNaoVazias\ (a, x) = i_2\ (a, x) \end{array} \right. \\
\square &
\end{array}
$$

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

$$avg\_aux = avgX$$

$$
\begin{array}{l}
inListasNaoVazias = [singl, cons] \\
outListasNaoVazias\ [a] = i_1\ (a) \\
outListasNaoVazias\ (a : x) = i_2\ (a, x)
\end{array}
$$

O nosso recListasNaoVazias será o mesmo que consta na biblioteca List.hs:

$$recListasNaoVazias = recList$$

Por fim o nosso cataListasNaoVazias pode ser definido:

$$cataListasNaoVazias\ g = g \cdot recListasNaoVazias\ (cataListasNaoVazias\ g) \cdot outListasNaoVazias$$

Assim, após as contas e considerações feitas anteriormente, é possível construir o seguinte diagrama para mais facilmente analisar o problema:

$$\begin{tikzcd}
A^{+} \arrow[rr, bend left, "outListasNaoVazias"] \arrow[d, "<avg{,}length>"'] & & A + A \times A^{+} \arrow[ll, bend left, "inListasNaoVazias"] \arrow[d, "id+id\times<avg{,}length>"] \\
\mathbb{R} \times \mathbb{N}^{+} & & A + A \times (\mathbb{R} \times \mathbb{N}^{+}) \arrow[ll, bend left, "g={[g1,g2]}"]
\end{tikzcd}$$

Se tivermos uma lista com apenas um elemento, a entrada será o próprio elemento e a saída será um par onde a primeira componente, que é relativa à média, será o próprio elemento e a segunda componente que corresponde ao comprimento será 1. Se tivermos uma lista com mais que um elemento, a entrada será um par em que a primeira componente será um elemento e a segunda componente será outro par em que a primeira componente deste segundo par será a média e a segunda será o comprimento. Ou seja, a sua saída será um par em que a primeira componente, de acordo com a fórmula será a soma do elemento com a multiplicação da média pelo comprimento e tudo isto a dividir pela média incrementada por um, e a segunda componente será apenas o comprimento incrementado por um.
Dado que queremos encontrar *avg_aux*, e sabendo que $avg\_aux = (\![b, q]\!)$, queremos encontrar o b e o q. Tendo em conta o nosso diagrama, vemos que $[b, q] = [g1, g2]$.Posto isto a nossa solução é a seguinte:

$$\begin{aligned}
&g1\ \ a = (a, 1)\\
&g2\ (a, (b, c)) = (((b * c) + a)\ /\ (c + 1), c + 1)\\
&avgX\ = cataListasNaoVazias\ [g1, g2]
\end{aligned}$$

Solução para árvores de tipo LTree:

Se tivermos apenas o elemento da raiz, a entrada será o próprio elemento e a saída será um par onde a primeira componente, que é relativa à média será o próprio elemento, e a segunda componente que corresponde ao comprimento será 1. Se tivermos mais do que apenas a raiz, a entrada será um par de dois pares, em que o da esquerda será relativo à média e ao comprimento da árvore do lado esquerdo e o da direita será relativo à média e ao comprimento da árvore do lado direito. Ou seja, na saída, que também será um par, teremos na primeira componente a soma da multiplicação da média e do comprimento do lado esquerdo e do lado direito a dividir pela soma dos dois comprimentos e na segunda a soma dos comprimentos de ambas as árvores. Em termos de diagramas, estamos perante o seguinte:

$$\begin{tikzcd}
A^{+} \arrow[rr, bend left, "out"] \arrow[d, "(\!|gene|\!)"'] & & A + LTree\ A^{2} \arrow[ll, bend left, "in"] \arrow[d, "id+\langle gene\rangle"] \\
\mathbb{R} \times \mathbb{N} & & A + ((\mathbb{R}, \mathbb{N}^{+}) \times (\mathbb{R}, \mathbb{N}^{+})) \arrow[ll, bend left, "gene={[gLT1,gLT2]}"]
\end{tikzcd}$$

Assim sendo, a nossa solução é a seguinte:

$$
\begin{array}{l}
\mathit{avgLTree} = \pi_1 \cdot (\!|\, \mathit{gene} \,|\!) \ \mathbf{where} \\
\quad \mathit{gene} = [\mathit{gLT1}, \mathit{gLT2}] \\
\mathit{gLT1}\ \ a = (a, 1) \\
\mathit{gLT2}\ ((a, b), (c, d)) = (((a * b) + (c * d)) \,/\, (b + d), b + d)
\end{array}
$$

Índice