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

Departamento de Informática
Universidade do Minho

Junho de 2021

Grupo nr.	999 (preencher)
a11111	Nome1 (preencher)
a22222	Nome2 (preencher)
a33333	Nome3 (preencher)
a44444	Nome4 (preencher, se aplicável, ou apagar)

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

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

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.*

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

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 [?], 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:

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

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.

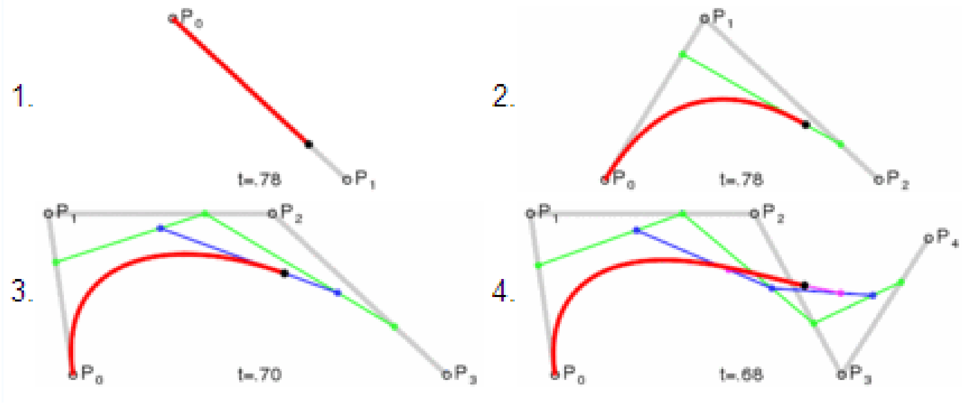

Figura 1: Exemplos de curvas de Bézier retirados da Wikipedia.

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:

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

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).

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

O anexo C tem definida a função

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

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

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

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.*

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

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

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

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

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:*

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

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:

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

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 [**?**].
[9] Cf. [**?**], página 102.

C Código fornecido

Problema 1

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

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:

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

Função auxiliar:

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

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:

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

[10]Fonte: Wikipedia.

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

Gloss:

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

Animação:

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

Propriedades e *main*:

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

Compilação e execução dentro do interpretador:[11]

$$
\begin{array}{l}
\mathit{main} = \mathit{runBezier} \\
\mathit{run} = \textbf{do}\ \{ \mathit{system}\ \texttt{"ghc cp2021t"}; \mathit{system}\ \texttt{"./cp2021t"} \}
\end{array}
$$

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

QuickCheck

Código para geração de testes:

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

Outras funções auxiliares

Lógicas:

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

D Soluções dos alunos

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

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

Problema 1

São dadas:

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

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

1. $outExpAr$ **e** $recExpAr$

As funções $outExpAr$ e $recExpAr$ são deduzidas através do tipo de dados do problema e com auxílio de alguns diagramas.
Sabendo que :

$$ExpAr\ A \xleftarrow[inExpAr]{} 1 + (A + (BinOp \times (ExpAr\ A \times ExpAr\ A) + (UnOp \times ExpAr\ A)))$$

$$inExpAr = [\underline{X}, [N, [bin, \widehat{Un}]]]$$
$$inExpAr \cdot outExpAr = id$$
$$outExpAr \cdot inExpAr = id$$

Pelas definições acima:

$$
\begin{array}{ll}
& outExpAr \cdot inExpAr = id \\
\equiv & \quad \{ \text{ Definição de inExpAr } \} \\
& outExpAr \cdot [\underline{X}, [N, [bin, \widehat{Un}]]] = id \\
\equiv & \quad \{ \text{ Fusão + (20) } \} \\
& [outExpAr \cdot \underline{X}, outExpAr \cdot [N, [bin, \widehat{Un}]]] = id \\
\equiv & \quad \{ \text{ Universal + (17) e Natural-id (1) } \} \\
& \left\{ \begin{array}{l} outExpAr \cdot \underline{X} = i_1 \\ outExpAr \cdot [N, [bin, \widehat{Un}]] = i_2 \end{array} \right. \\
\equiv & \quad \{ \text{ Fusão + (20) , Universal + (17) e Natural-id (1) } \} \\
& \left\{ \begin{array}{l} outExpAr \cdot \underline{X} = i_1 \\ \left\{ \begin{array}{l} outExpAr \cdot N = i_2 \cdot i_1 \\ outExpAr \cdot [bin, \widehat{Un}] = i_2 \cdot i_2 \end{array} \right. \end{array} \right. \\
\equiv & \quad \{ \text{ Fusão + (20) , Universal + (17) e Natural-id (1) } \} \\
& \left\{ \begin{array}{l} outExpAr \cdot \underline{X} = i_1 \\ \left\{ \begin{array}{l} outExpAr \cdot N = i_2 \cdot i_1 \\ \left\{ \begin{array}{l} outExpAr \cdot bin = i_2 \cdot i_2 \cdot i_1 \\ outExpAr \cdot \widehat{Un} = i_2 \cdot i_2 \cdot i_2 \end{array} \right. \end{array} \right. \end{array} \right. \\
\equiv & \quad \{ \text{ Igualdade Extensional (71) e Def-Comp (72) } \} \\
& outExpAr\ \underline{X}\ x = i_1\ x \\
& outExpAr\ (N\ x) = i_2 \cdot i_1\ x \\
& outExpAr\ (bin\ (op, (a, b))) = i_2 \cdot i_2 \cdot i_1\ (op, (a, b)) \\
& outExpAr\ \widehat{Un}\ (op, a) = i_2 \cdot i_2 \cdot i_2\ (op, a) \\
\equiv & \quad \{ \text{ Definição de Bin e Un } \} \\
& outExpAr\ X = i_1\ ()
\end{array}
$$

$$
\begin{aligned}
&outExpAr\ (N\ x) = i_2 \cdot i_1\ x \\
&outExpAr\ (Bin\ op\ a\ b) = i_2 \cdot i_2 \cdot i_1\ (op, (a, b)) \\
&outExpAr\ (Un\ op\ a) = i_2 \cdot i_2 \cdot i_2\ (op, a)
\end{aligned}
$$

Concluiu-se que:

$$
\begin{aligned}
&outExpAr\ X = i_1\ () \\
&outExpAr\ (N\ a) = i_2 \cdot i_1\ \$\ a \\
&outExpAr\ (Bin\ op\ a\ b) = i_2 \cdot i_2 \cdot i_1\ \$\ (op, (a, b)) \\
&outExpAr\ (Un\ op\ a) = i_2 \cdot i_2 \cdot i_2\ \$\ (op, a)
\end{aligned}
$$

$$
ExpAr\ A \xrightarrow{outExpAr} 1 + (A + (BinOp \times (ExpAr\ A \times ExpAr\ A) + (UnOp \times ExpAr\ A)))
$$

$$
recExpAr\ x = baseExpAr\ id\ id\ id\ x\ x\ id\ x
$$

2. g_eval_exp

De seguida, apresenta-se a definição da função:

$$
\begin{aligned}
&g\_eval\_exp\ num = [\underline{num}, [id, [\widehat{f}, \widehat{g}]]] \\
&\quad \textbf{where} \\
&\qquad f\ Sum = \widehat{(+)} \\
&\qquad f\ Product = \widehat{(*)} \\
&\qquad g\ E = expd \\
&\qquad g\ Negate = negate
\end{aligned}
$$

E o seu diagrama:

$$
\begin{tikzcd}
ExpAr\ A \arrow[r, "outExpAr"] \arrow[d, "eval\_exp\ num"'] & 1 + (A + (BinOp \times (ExpAr\ A \times ExpAr\ A) + (UnOp \times ExpAr\ A))) \arrow[d, "id+(id+((id\times(g\_eval\_exp\ num)^2)+(id\times g\_eval\_exp\ num))"] \\
A & 1 + (A + (BinOp \times (A \times A) + (UnOp \times A))) \arrow[l, "g\_eval\_exp\ num"]
\end{tikzcd}
$$

3. $clean$ **e** $gopt$

Optimizando a $ExpAr\ \ A$ dada, através da utilização das regras da absorção e do elemento neutro, obtém-se:

$$
\begin{aligned}
&clean\ (Bin\ Product\ (N\ 0)\ \_) = outExpAr\ (N\ 0) \\
&clean\ (Bin\ Product\ \_\ (N\ 0)) = outExpAr\ (N\ 0) \\
&clean\ (Bin\ Product\ (N\ 1)\ l) = outExpAr\ l \\
&clean\ (Bin\ Product\ l\ (N\ 1)) = outExpAr\ l \\
&clean\ (Bin\ Sum\ a\ (N\ 0)) = outExpAr\ a \\
&clean\ (Bin\ Sum\ (N\ 0)\ a) = outExpAr\ a \\
&clean\ (Bin\ op\ (N\ a)\ (N\ b)) = outExpAr\ (N\ ((f\ op)\ a\ b)) \\
&\quad \textbf{where} \\
&\qquad f\ Sum = (+) \\
&\qquad f\ Product = (*) \\
&clean\ (Un\ op\ (N\ a)) = outExpAr\ (N\ ((g\ op)\ a)) \\
&\quad \textbf{where} \\
&\qquad g\ E = expd \\
&\qquad g\ Negate = negate \\
&clean\ l = outExpAr\ l
\end{aligned}
$$

Feita essa otimização, fez-se a substituição do X pelo valor dado e calculou-se o respetivo valor da expressão, reutilizando a função g_eval_exp:

$$
\begin{array}{l}
gopt :: Floating\ c \Rightarrow c \rightarrow b + (c + ((BinOp, (c, c)) + (UnOp, c))) \rightarrow c \\
gopt = g\_eval\_exp
\end{array}
$$

De seguida, apresenta-se o diagrama do hilomorfismo:

$$
\begin{tikzcd}
ExpAr\ A \arrow[r, "outExpAr"] \arrow[d, "\lbrack\!\lparen clean \rparen\!\rbrack"'] & 1 + (A + (BinOp \times (ExpAr\ A \times ExpAr\ A) + (UnOp \times ExpAr\ A))) \arrow[d, "id+(id+((id\times(clean)^2)+(id\times clean))"] \\
ExpAr\ A \arrow[r, shift left, "inExpAr"] \arrow[d, "\llparenthesis gopt\ num \rrparenthesis"'] & 1 + (A + (BinOp \times (ExpAr\ A \times ExpAr\ A) + (UnOp \times ExpAr\ A))) \arrow[l, shift left, "outExpAr"] \arrow[d, "id+(id+((id\times(gopt\ num)^2)+(id\times gopt\ num))"] \\
A & 1 + (A + (BinOp \times (A \times A) + (UnOp \times A))) \arrow[l, "gopt\ num"]
\end{tikzcd}
$$

4. sd_gen

$$
\begin{array}{l}
sd\_gen :: Floating\ a \Rightarrow \\
\quad () + (a + ((BinOp, ((ExpAr\ a, ExpAr\ a), (ExpAr\ a, ExpAr\ a))) + (UnOp, (ExpAr\ a, ExpAr\ a)))) \\
\qquad \rightarrow (ExpAr\ a, ExpAr\ a) \\
sd\_gen = [\underline{(X, (N\ 1))}, [construi\_n, [construi\_bin, construi\_un]]] \\
\quad \mathbf{where} \\
\qquad construi\_n\ a = (N\ a, N\ 0) \\
\qquad construi\_bin\ (Sum, (a, b)) = (Bin\ Sum\ (\pi_1\ a)\ (\pi_1\ b), Bin\ Sum\ (\pi_2\ a)\ (\pi_2\ b)) \\
\qquad construi\_bin\ (Product, (a, b)) = (Bin\ Product\ (\pi_1\ a)\ (\pi_1\ b), segundo\_bin) \\
\qquad\quad \mathbf{where} \\
\qquad\qquad segundo\_bin = Bin\ Sum\ (Bin\ Product\ (\pi_1\ a)\ (\pi_2\ b))\ (Bin\ Product\ (\pi_2\ a)\ (\pi_1\ b)) \\
\qquad construi\_un\ (E, a) = (Un\ E\ (\pi_1\ a), Bin\ Product\ (Un\ E\ (\pi_1\ a))\ (\pi_2\ a)) \\
\qquad construi\_un\ (Negate, a) = (Un\ Negate\ (\pi_1\ a), Un\ Negate\ (\pi_2\ a))
\end{array}
$$

É de notar que a função sd_gen retorna o par ($ExpAr\ A$,$ExpAr\ A$) , no qual, o segundo elemento irá ser a $ExpAr\ A$ derivada da $ExpAr\ A$ contida no primeiro elemento do par.

O diagrama correspondente à função é:

$$
\begin{tikzcd}
ExpAr\ A \arrow[r, "outExpAr"] \arrow[d, "sd"'] & () + (A + (BinOp \times (ExpAr\ A \times ExpAr\ A) + (UnOp \times ExpAr\ A))) \arrow[d, "id+(id+(id\times(sd\_gen\times sd\_gen)+(id\times sd\_gen)))"] \\
ExpAr\ A & (() + (A + (BinOp \times (ExpAr\ A \times ExpAr\ A) + (UnOp \times ExpAr\ A))))^2 \arrow[l, "\pi_2 \cdot sd\_gen"]
\end{tikzcd}
$$

5. ad_gen

$$
\begin{array}{l}
ad\_gen\ a = [\underline{(a, 1)}, [g, [\widehat{bin\_op}, \widehat{un\_op}]]] \\
\quad \mathbf{where} \\
\qquad g = \langle id, \underline{0} \rangle \\
\qquad bin\_op\ Sum\ (a, b) = ((+)\ (\pi_1\ a)\ (\pi_1\ b), (+)\ (\pi_2\ a)\ (\pi_2\ b)) \\
\qquad bin\_op\ Product\ (a, b) = ((*)\ (\pi_1\ a)\ (\pi_1\ b), (+)\ ((*)\ (\pi_1\ a)\ (\pi_2\ b))\ ((*)\ (\pi_2\ a)\ (\pi_1\ b))) \\
\qquad un\_op\ E\ a = (expd\ (\pi_1\ a), (*)\ (\pi_2\ a)\ (expd\ (\pi_1\ a))) \\
\qquad un\_op\ Negate\ a = (negate\ (\pi_1\ a), negate\ (\pi_2\ a))
\end{array}
$$

Note que a função ad_gen retorna o par (A,A), onde, o primeiro elemento irá ser o valor da $ExpAr\ A$ com a substituição do valor dado e o segundo, irá ser o valor da sua derivada.

O diagrama correspondente à função é:

$$\begin{tikzcd}
ExpAr\ A \arrow[r, "outExpAr"] \arrow[d, "ad"'] & () + (A + (BinOp \times (ExpAr\ A \times ExpAr\ A) + (UnOp \times ExpAr\ A))) \arrow[d, "id+(id+(id\times(ad_gen\ A\times ad_gen\ A)+(id\times ad_gen\ A)))"] \\
A & (() + (A + (BinOp \times ((A \times A) \times (A \times A)) + (UnOp \times (A \times A))))) \arrow[l, "\pi_2 \cdot ad_gen"]
\end{tikzcd}$$

Problema 2

Definir

$$loop\ (c,(s,h)) = (c * s \div h, (s+4, h+1))$$
$$inic = (1,(2,2))$$
$$prj = \pi_1$$

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.

Sabendo que:

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

Vamos calcular c (n+1) :

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

Considerando:

$$s\ n = 2*(2\ n+1) = 4*n+2$$
$$h\ n = n+2$$

Podendo, com isto concluir:

$$\begin{cases} c\ 0 = 1 \\ c\ (n+1) = c\ n * \frac{s\ n}{h\ n} \end{cases}$$
$$\begin{cases} s\ 0 = 2 \\ s\ (n+1) = s\ n + 4 \end{cases}$$
$$\begin{cases} h\ 0 = 2 \\ h\ (n+1) = h\ n + 1 \end{cases}$$

Analisando:

$$\begin{cases} c\ 0 = 1 \\ c\ (n+1) = c\ n * \frac{s\ n}{h\ n} \end{cases}$$
$$\begin{cases} s\ 0 = 2 \\ s\ (n+1) = s\ n + 4 \end{cases}$$
$$\begin{cases} h\ 0 = 2 \\ h\ (n+1) = h\ n + 1 \end{cases}$$

$\equiv$ { Igualdade extensional (71) (6 vezes) }

$$\begin{cases} c \cdot \underline{0} = \underline{1} \\ c \cdot \mathsf{succ} = \widehat{(*)} \cdot \langle c, \widehat{(/)} \cdot \langle s, h \rangle \rangle \end{cases}$$
$$\begin{cases} s \cdot \underline{0} = \underline{2} \\ s \cdot \mathsf{succ} = (+4) \cdot s \end{cases}$$
$$\begin{cases} h \cdot \underline{0} = \underline{2} \\ h \cdot \mathsf{succ} = \mathsf{succ} \cdot h \end{cases}$$

$\equiv$ { Eq- + (27) (3 vezes) }

$$[c \cdot \underline{0}, c \cdot \mathsf{succ}\,] = [\underline{1}, \widehat{(/)} \cdot \langle \widehat{(*)} \cdot \langle c, s\rangle, h\rangle]$$
$$[s \cdot \underline{0}, s \cdot \mathsf{succ}\,] = [\underline{2}, (+4) \cdot s]$$
$$[h \cdot \underline{0}, h \cdot \mathsf{succ}\,] = [\underline{2}, \mathsf{succ}\ \cdot h]$$

$\equiv$ { in = [const 0, succ] ,Fusão X (3 vezes) }

$$c \cdot \mathbf{in} = [\underline{1}, \widehat{(/)} \cdot \langle \widehat{(*)} \cdot \langle c, s\rangle, h\rangle]$$
$$s \cdot \mathbf{in} = [\underline{2}, (+4) \cdot s]$$
$$h \cdot \mathbf{in} = [\underline{2}, \mathsf{succ}\ \cdot h]$$

$\equiv$ { Absorção + }

$$c \cdot \mathbf{in} = [\underline{1}, \widehat{(/)} \cdot \langle \widehat{(*)} \cdot \langle \pi_1, \pi_1 \cdot \pi_2\rangle, \pi_2 \cdot \pi_2\rangle] \cdot (id + \langle c, \langle s, h\rangle\rangle)$$
$$s \cdot \mathbf{in} = [\underline{2}, (+4) \cdot (\pi_1 \cdot \pi_2)] \cdot (id + \langle c, \langle s, h\rangle\rangle)$$
$$h \cdot \mathbf{in} = [\underline{2}, \mathsf{succ}\ \cdot (\pi_2 \cdot \pi_2)] \cdot (id + \langle c, \langle s, h\rangle\rangle)$$

$\equiv$ { Def- x (lei 10) e Fokkinga com 3 elementos }

$$\langle c, \langle s, h\rangle\rangle = (\!|\langle [\underline{1}, \widehat{(/)} \cdot \langle \widehat{(*)} \cdot (id \times \pi_1), \pi_2 \cdot \pi_2\rangle], \langle [\underline{2}, (+4) \cdot \pi_1 \cdot \pi_2], [\underline{2}, \mathsf{succ}\ \cdot \pi_2 \cdot \pi_2]\rangle\rangle|\!)$$

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

$$\langle c, \langle s, h\rangle\rangle = (\!|\langle [\underline{1}, \widehat{(/)} \cdot \langle \widehat{(*)} \cdot (id \times \pi_1), \pi_2 \cdot \pi_2\rangle], [\langle \underline{2}, \underline{2}\rangle, \langle (+4) \cdot \pi_1 \cdot \pi_2, \mathsf{succ}\ \cdot \pi_2 \cdot \pi_2\rangle]\rangle|\!)$$

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

$$\langle c, \langle s, h\rangle\rangle = (\!|[\langle \underline{1}, \langle \underline{2}, \underline{2}\rangle\rangle, \langle \widehat{(/)} \cdot \langle \widehat{(*)} \cdot (id \times \pi_1), \pi_2 \cdot \pi_2\rangle, \langle (+4) \cdot \pi_1 \cdot \pi_2, \mathsf{succ}\ \cdot \pi_2 \cdot \pi_2\rangle\rangle]|\!)$$

$\equiv$ { for b i = $(\underline{i}, b)$ }

$$\langle c, \langle s, h\rangle\rangle = \mathsf{for}\ (1, (2, 2))\ \langle \widehat{(/)} \cdot \langle \widehat{(*)} \cdot (id \times \pi_1), \pi_2 \cdot \pi_2\rangle, \langle (+4) \cdot \pi_1 \cdot \pi_2, \mathsf{succ}\ \cdot \pi_2 \cdot \pi_2\rangle\rangle$$

Problema 3

```
calcLine :: NPoint → (NPoint → OverTime NPoint)
calcLine = cataList h where
  h = [_· nil, g]
  g (d, f) l = case l of
    [] → nil
    (x : xs) → λz → concat $ (sequenceA [singl · linear1d d x, f xs]) z
deCasteljau :: [NPoint] → OverTime NPoint
deCasteljau = hyloAlgForm alg coalg where
    -- coalg [] = i1 ()
    -- coalg l = i2 split(init)(tail)l
    -- coalg l = i2 split(> deCasteljau.init)(> deCasteljau.tail)l
  coalg = ⊥
    -- coalg l = i2 split(deCasteljau.g)(deCasteljau.tail)l
    -- g (a:t) = a
  alg = ⊥
    -- coalg = split (deCasteljau.cons.(split p1 (init.p2))) (deCasteljau.p2)
    -- coalg .. (p,q)
    -- alg = either (const nil) (f)
    -- f (a,b) = calcLine a b
    -- f (a,b) = -¿(calcLine (a z) (b z) z)
    -- coalg = split id calcLine
    -- alg = either (const (nil,([[0.0]],id))) funcao
    -- funcao (lista,((a,b),(c,d))) = ((fmap d a) :lista, (a,b) )
    -- alg ( (a,f1),(b, l)) = either (const ((0,id),(0,[]))) ( (a,f1) , (b, (f1 b):l) )
    -- alg (a, ((a,f1), (b,f2))) = either (const ([],((0,id), (0,id))) ( (f1 b):a , )
```

$$hyloAlgForm\ h\ g = cataList\ h \cdot anaList\ g$$

Problema 4

1. Solução para listas não vazias:

$$
\begin{array}{l}
avg = \pi_1 \cdot avg\_aux \\
avg\_aux = cataList\ gene \\
\quad \textbf{where} \\
\qquad gene = [\langle \underline{0}, \underline{0} \rangle, \langle \widehat{(/)} \cdot \langle \widehat{(+)} \cdot \langle \pi_1, \widehat{(*)} \cdot \pi_2 \rangle, \mathsf{succ}\ \cdot \pi_2 \cdot \pi_2 \rangle, \mathsf{succ}\ \cdot \pi_2 \cdot \pi_2 \rangle]
\end{array}
$$

Descrita pelo diagrama seguinte:

$$
\begin{tikzcd}
A^* \arrow[r, "outList"] \arrow[d, "avg"'] & 1 + (A \times A^*) \arrow[d, "id + (id \times avg\_aux)"] \\
A & 1 + (A \times (A \times A)) \arrow[l, "\pi_1 \cdot avg\_aux"]
\end{tikzcd}
$$

2. Solução para árvores de tipo LTree:

$$
\begin{array}{l}
avgLTree = \pi_1 \cdot (\!|\, gene \,|\!)\ \textbf{where} \\
\quad gene = [\langle id, \underline{1} \rangle, \mathit{final}] \\
\quad \mathit{final} = \langle \widehat{(/)}, \pi_2 \rangle \cdot (\widehat{(+)} \times \widehat{(+)}) \cdot \langle \widehat{(*)} \times \widehat{(*)}, \pi_2 \times \pi_2 \rangle
\end{array}
$$

Na definição do gene acima, apresenta-se a versão pointfree da função "final", correspondendo à versão pointwise seguinte:

$$
\begin{array}{ll}
 & \mathit{final} = \langle \widehat{(/)}, \pi_2 \rangle \cdot (\widehat{(+)} \times \widehat{(+)}) \cdot \langle \widehat{(*)} \times \widehat{(*)}, \pi_2 \times \pi_2 \rangle \\
\equiv & \quad \{\ \text{Igualdade Extensional (71) e Def-comp(72)}\ \} \\
 & \mathit{final}\ ((a, b), (c, d)) = \langle \widehat{(/)}, \pi_2 \rangle \cdot (\widehat{(+)} \times \widehat{(+)}) \cdot \langle \widehat{(*)} \times \widehat{(*)}, \pi_2 \times \pi_2 \rangle\ ((a, b), (c, d)) \\
\equiv & \quad \{\ \text{Def-split (76) e Natural-}\pi_2\text{ (13) (2 vezes)}\ \} \\
 & \mathit{final}\ ((a, b), (c, d)) = \langle \widehat{(/)}, \pi_2 \rangle\ ((\widehat{(+)} \times \widehat{(+)})\ (((\widehat{(*)} \times \widehat{(*)})\ ((a, b), (c, d)), (\pi_2 \times \pi_2)\ ((a, b), (c, d)))) \\
\equiv & \quad \{\ \text{Def-X (77)}\ \} \\
 & \mathit{final}\ ((a, b), (c, d)) = \langle \widehat{(/)}, \pi_2 \rangle\ ((\widehat{(+)} \times \widehat{(+)})\ ((\widehat{(*)}\ (a, b), \widehat{(*)}\ (c, d)), (\pi_2\ (a, b), \pi_2\ (c, d)))) \\
\equiv & \quad \{\ \text{Uncurry (84) , Definição de * (multiplicação) e Natural-}\pi_2\text{ (13)}\ \} \\
 & \mathit{final}\ ((a, b), (c, d)) = \langle \widehat{(/)}, \pi_2 \rangle\ ((\widehat{(+)} \times \widehat{(+)})\ ((a * b, c * d), (b, d))) \\
\equiv & \quad \{\ \text{Def-split (76) , Uncurry (84) e Definição da soma}\ \} \\
 & \mathit{final}\ ((a, b), (c, d)) = \langle \widehat{(/)}, \pi_2 \rangle\ (a * b + c * d, b + d) \\
\equiv & \quad \{\ \text{Def-split (76) , Uncurry (84) , Definição da divisão e Natural-p2 (13)}\ \} \\
 & \mathit{final}\ ((a, b), (c, d)) = ((a * b + c * d)\ /\ (b + d), b + d)
\end{array}
$$

Chegando assim à versão pointwise:

$$\mathit{final}\ ((a, b), (c, d)) = ((a * b + c * d)\ /\ (b + d), b + d)$$

O diagrama correspondente é:

$$
\begin{tikzcd}
\mathsf{LTree}\ A \arrow[r, "outLtree"] \arrow[d, "avgLTree"'] & A + ((\mathsf{LTree}\ A) \times (\mathsf{LTree}\ A)) \arrow[d, "id + (gene \times gene)"] \\
A & A + ((A \times A) \times (A \times A)) \arrow[l, "\pi_1 \cdot gene"]
\end{tikzcd}
$$

Problema 5

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