Universidade do Minho
Escola de Engenharia

Cálculo de Programas

Trabalho Prático (2023/24)

Lic. em Engenharia Informática

Grupo G05

aA100761 Carlos Ribeiro
aA100742 Júlio Pinto
aA100823 Pedro Sousa

janeiro 2024

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 corolários) e usam-se esses combinadores para construir programas *composicionalmente*, isto é, agregando programas já existentes.

Na sequência pedagógica dos planos de estudo dos 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.

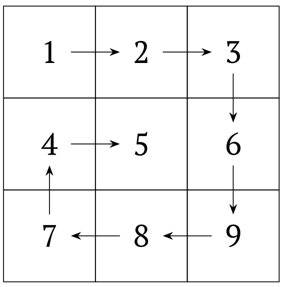
Antes de abodarem os problemas propostos no trabalho, os grupos devem ler com atenção o anexo **??** onde encontrarão as instruções relativas ao sofware a instalar, etc.

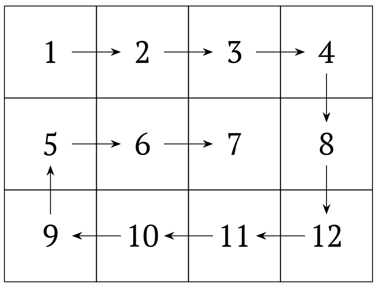
Valoriza-se a escrita de *pouco* código que corresponda a soluções simples e elegantes que utilizem os combinadores de ordem superior estudados na disciplina.

Problema 1

Este problema, retirado de um *site* de exercícios de preparação para entrevistas de emprego, tem uma formulação simples:

> *Dada uma matriz de uma qualquer dimensão, listar todos os seus elementos rodados em espiral.*
>
> *Por exemplo, dadas as seguintes matrizes:*





> *dever-se-á obter, respetivamente,* $[1,2,3,6,9,8,7,4,5]$ *e* $[1,2,3,4,8,12,11,10,9,5,6,7]$.
>
> $\square$

Valorizar-se-ão as soluções *pointfree* que empreguem os combinadores estudados na disciplina, e.g. $f \cdot g$, $\langle f, g \rangle$, $f \times g$, $[f, g]$, $f + g$, bem como catamorfismos e anamorfismos.

Recomenda-se a escrita de *pouco* código e de soluções simples e fáceis de entender. Recomenda-se que o código venha acompanhado de uma descrição de como funciona e foi concebido, apoiado em diagramas explicativos. Para instruções sobre como produzir esses diagramas e exprimir raciocínios de cálculo, ver o anexo **??**.

Problema 2

Este problema, que de novo foi retirado de um *site* de exercícios de preparação para entrevistas de emprego, tem uma formulação muito simples:

> *Inverter as vogais de um* string.

Esta formulação deverá ser generalizada a:

> *Inverter os elementos de uma dada lista que satisfazem um dado predicado.*

Valorizam-se as soluções tal como no problema anterior e fazem-se as mesmas recomendações.

Problema 3

Sistemas como chatGPT etc baseiam-se em algoritmos de aprendizagem automática que usam determinadas funções matemáticas, designadas *activation functions* (AF), para modelar aspectos não lineares do mundo real. Uma dessas AFs é a tangente hiperbólica, definida como o quociente do seno e coseno hiperbólicos,

$$\tanh x = \frac{\sinh x}{\cosh x} \tag{1}$$

podendo estes ser definidos pelas seguintes séries de Taylor:

$$\sum_{k=0}^{\infty} \frac{x^{2k+1}}{(2k+1)!} = \sinh x \tag{2}$$
$$\sum_{k=0}^{\infty} \frac{x^{2k}}{(2k)!} = \cosh x$$

Interessa que estas funções sejam implementadas de forma muito eficiente, desdobrando-as em operações aritméticas elementares. Isso pode ser conseguido através da chamada programação dinâmica que, em Cálculo de Programas, é feita de forma *correct-by-construction* derivando-se ciclos-**for** via lei de recursividade mútua generalizada a tantas funções quanto necessário — ver o anexo **??**.

O objectivo desta questão é codificar como um ciclo-for (em Haskell) a função

$$snh\ x\ i = \sum_{k=0}^{i} \frac{x^{2k+1}}{(2k+1)!} \tag{3}$$

que implementa *sinh x*, uma das funções de *tanh x* (**??**), através da soma das i primeiras parcelas da sua série (**??**).

Deverá ser seguida a regra prática do anexo **??** e documentada a solução proposta com todos os cálculos que se fizerem.

Problema 4

Uma empresa de transportes urbanos pretende fornecer um serviço de previsão de atrasos dos seus autocarros que esteja sempre actual, com base em *feedback* dos seus paassageiros. Para isso, desenvolveu uma *app* que instala num telemóvel um botão que indica coordenadas GPS a um serviço central, de forma anónima, sugerindo que os passageiros o usem preferencialmente sempre que o autocarro onde vão chega a uma paragem.

Com base nesses dados, outra funcionalidade da *app* informa os utentes do serviço sobre a probabilidade do atraso que possa haver entre duas paragens (partida e chegada) de uma qualquer linha.

Pretende-se implementar esta segunda funcionalidade assumindo disponíveis os dados da primeira. No que se segue, ir-se-á trabalhar sobre um modelo intencionalmente *muito simplificado* deste sistema, em que se usará o mónade das distribuições probabilísticas (ver o anexo **??**). Ter-se-á, então:

- paragens de autocarro

  $$\mathbf{data}\ \mathit{Stop} = \mathit{S0} \mid \mathit{S1} \mid \mathit{S2} \mid \mathit{S3} \mid \mathit{S4} \mid \mathit{S5}\ \mathbf{deriving}\ (\mathit{Show}, \mathit{Eq}, \mathit{Ord}, \mathit{Enum})$$

  que formam a linha $[\mathit{S0} \mathinner{..} \mathit{S5}]$ assumindo a ordem determinada pela instância de *Stop* na classe *Enum*;

- segmentos da linha, isto é, percursos entre duas paragens consecutivas:

  $$\mathbf{type}\ \mathit{Segment} = (\mathit{Stop}, \mathit{Stop})$$

- os dados obtidos a partir da *app* dos passageiros que, após algum processamento, ficam disponíveis sob a forma de pares *(segmento, atraso observado)*:

  $$\mathit{dados} :: [(\mathit{Segment}, \mathit{Delay})]$$

  (Ver no apêndice **??**, página **??**, uma pequena amostra destes dados.)

A partir destes dados, há que:

- gerar a base de dados probabilística

  $$\mathit{db} :: [(\mathit{Segment}, \mathsf{Dist}\ \mathit{Delay})]$$

  que regista, estatisticamente, a probabilidade dos atrasos (*Delay*) que podem afectar cada segmento da linha. Recomenda-se aqui a definição de uma função genérica

  $$\mathit{mkdist} :: \mathit{Eq}\ a \Rightarrow [a] \to \mathsf{Dist}\ a$$

  que faça o sumário estatístico de uma qualquer lista finita, gerando a distribuição de ocorrência dos seus elementos.

- com base em *db*, definir a função probabilística

  $$\mathit{delay} :: \mathit{Segment} \to \mathsf{Dist}\ \mathit{Delay}$$

  que dará, para cada segmento, a respectiva distribuição de atrasos.

Finalmente, o objectivo principal é definir a função probabilística:

$$\mathit{pdelay} :: \mathit{Stop} \to \mathit{Stop} \to \mathsf{Dist}\ \mathit{Delay}$$

$\mathit{pdelay}\ a\ b$ deverá informar qualquer utente que queira ir da paragem a até à paragem b de uma dada linha sobre a probabilidade de atraso acumulado no total do percurso $[a \mathinner{..} b]$.

Valorizar-se-ão as soluções que usem funcionalidades monádicas genéricas estudadas na disciplina e que sejam elegantes, isto é, poupem código desnecessário.

Anexos

A Natureza do trabalho a realizar

Este trabalho teórico-prático deve ser realizado por grupos de 3 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 em **todos** os exercícios do trabalho, para assim poderem responder a qualquer questão colocada na *defesa oral* do relatório.

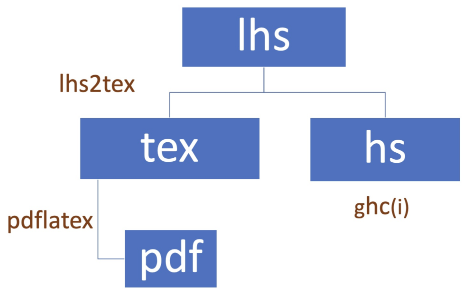
Para cumprir de forma integrada os objectivos do trabalho vamos recorrer a uma técnica de programação dita "literária" [**?**], cujo princípio base é o seguinte:

> *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 `cp2324t.pdf` que está a ler é já um exemplo de programação literária: foi gerado a partir do texto fonte `cp2324t.lhs`[1] que encontrará no material pedagógico desta disciplina descompactando o ficheiro `cp2324t.zip`.

Como se mostra no esquema abaixo, de um único ficheiro (*lhs*) gera-se um PDF ou faz-se a interpretação do código Haskell que ele inclui:



Vê-se assim que, para além do GHCi, serão necessários os executáveis pdflatex e lhs2TeX. Para facilitar a instalação e evitar problemas de versões e conflitos com sistemas operativos, é recomendado o uso do Docker tal como a seguir se descreve.

B Docker

Recomenda-se o uso do container cuja imagem é gerada pelo Docker a partir do ficheiro `Dockerfile` que se encontra na diretoria que resulta de descompactar `cp2324t.zip`. Este container deverá ser usado na execução do GHCi e dos comandos relativos ao LaTeX. (Ver também a `Makefile` que é disponibilizada.)

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

Após instalar o Docker e descarregar o referido zip com o código fonte do trabalho, basta executar os seguintes comandos:

```
$ docker build -t cp2324t .
$ docker run -v ${PWD}:/cp2324t -it cp2324t
```

NB: O objetivo é que o container seja usado *apenas* para executar o GHCi e os comandos relativos ao LaTeX. Deste modo, é criado um *volume* (cf. a opção `-v ${PWD}:/cp2324t`) que permite que a diretoria em que se encontra na sua máquina local e a diretoria `/cp2324t` no container sejam partilhadas.

Pretende-se então que visualize/edite os ficheiros na sua máquina local e que os compile no container, executando:

```
$ lhs2TeX cp2324t.lhs > cp2324t.tex
$ pdflatex cp2324t
```

lhs2TeX é o pre-processador que faz "pretty printing" de código Haskell em LaTeX e que faz parte já do container. Alternativamente, basta executar

```
$ make
```

para obter o mesmo efeito que acima.

Por outro lado, o mesmo ficheiro `cp2324t.lhs` é executável e contém o "kit" básico, escrito em Haskell, para realizar o trabalho. Basta executar

```
$ ghci cp2324t.lhs
```

Abra o ficheiro `cp2324t.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.

C Em que consiste o TP

Em que consiste, então, o *relatório* a que se referiu acima? É a edição do texto que está a ser lido, preenchendo o anexo **??** 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 cp2324t.aux
$ makeindex cp2324t.idx
```

e recompilar o texto como acima se indicou. (Como já se disse, pode fazê-lo correndo simplesmente `make` no container.)

No anexo **??** disponibiliza-se algum código Haskell relativo aos problemas que são colocados. Esse anexo deverá ser consultado e analisado à medida que isso for necessário.

Deve ser feito uso da programação literária para documentar bem o código que se desenvolver, em particular fazendo diagramas explicativos do que foi feito e tal como se explica no anexo **??** que se segue.

D Como exprimir cálculos e diagramas em LaTeX/lhs2TeX

Como primeiro exemplo, estudar o texto fonte (lhs) do que está a ler[1] onde se obtém o efeito seguinte:[2]

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

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

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

E Regra prática para a recursividade mútua em $\mathbb{N}_0$

Nesta disciplina estudou-se como fazer programação dinâmica por cálculo, recorrendo à lei de recursividade mútua.[3]

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:

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

Obter-se-á de imediato

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

usando as regras seguintes:

[1] Procure e.g. por `"sec:diagramas"`.

[2] Exemplos tirados de [**?**].

[3] Lei (3.93) em [**?**], página 110.

- 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.[1]
- 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[2], 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:

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

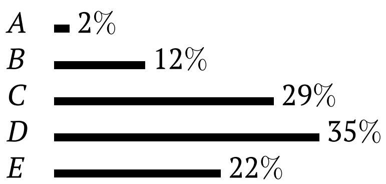
F O mónade das distribuições probabilísticas

Mónades são functores com propriedades adicionais que nos permitem obter efeitos especiais em programação. Por exemplo, a biblioteca Probability oferece um mónade para abordar problemas de probabilidades. Nesta biblioteca, o conceito de distribuição estatística é captado pelo tipo

$$\mathbf{newtype}\ \mathsf{Dist}\ a = D\ \{unD :: [(a, ProbRep)]\} \tag{4}$$

em que $ProbRep$ é um real de 0 a 1, equivalente a uma escala de 0 a 100%.

Cada par (a, p) numa distribuição $d :: \mathsf{Dist}\ a$ indica que a probabilidade de a é p, devendo ser garantida a propriedade de que todas as probabilidades de d somam 100%. Por exemplo, a seguinte distribuição de classificações por escalões de A a E,



será representada pela distribuição

$$
\begin{array}{l}
d1 :: \mathsf{Dist}\ Char \\
d1 = D\ [(\texttt{'A'}, 0.02), (\texttt{'B'}, 0.12), (\texttt{'C'}, 0.29), (\texttt{'D'}, 0.35), (\texttt{'E'}, 0.22)]
\end{array}
$$

que o GHCi mostrará assim:

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

[2] Secção 3.17 de [?] e tópico Recursividade mútua nos vídeos de apoio às aulas teóricas.

```
'D'   35.0%
'C'   29.0%
'E'   22.0%
'B'   12.0%
'A'    2.0%
```

É possível definir geradores de distribuições, por exemplo distribuições *uniformes*,

$$d2 = uniform\ (words\ \texttt{"Uma frase de cinco palavras"})$$

isto é

```
     "Uma"   20.0%
   "cinco"   20.0%
      "de"   20.0%
   "frase"   20.0%
"palavras"   20.0%
```

distribuição *normais*, eg.

$$d3 = normal\ [10\mathinner{..}20]$$

etc.[1] Dist forma um **mónade** cuja unidade é $return\ a = D\ [(a, 1)]$ e cuja composição de Kleisli é (simplificando a notação)

$$(f \bullet g)\ a = [(y, q * p) \mid (x, p) \leftarrow g\ a, (y, q) \leftarrow f\ x]$$

em que $g : A \to \text{Dist}\ B$ e $f : B \to \text{Dist}\ C$ são funções **monádicas** que representam *computações probabilísticas*.

Este mónade é adequado à resolução de problemas de *probabilidades e estatística* usando programação funcional, de forma elegante e como caso particular da programação monádica.

G Código fornecido

Problema 1

$$m1 = [[1, 2, 3], [4, 5, 6], [7, 8, 9]]$$
$$m2 = [[1, 2, 3, 4], [5, 6, 7, 8], [9, 10, 11, 12]]$$
$$m3 = words\ \texttt{"Cristina Monteiro Carvalho Sequeira"}$$

$$test1 = matrot\ m1 \equiv [1, 2, 3, 6, 9, 8, 7, 4, 5]$$
$$test2 = matrot\ m2 \equiv [1, 2, 3, 4, 8, 12, 11, 10, 9, 5, 6, 7]$$
$$test3 = matrot\ m3 \equiv \texttt{"CristinaooarieuqeSCMonteirhlavra"}$$

Problema 2

$$test4 = reverseVowels\ \texttt{""} \equiv \texttt{""}$$
$$test5 = reverseVowels\ \texttt{"ácidos"} \equiv \texttt{"ocidás"}$$
$$test6 = reverseByPredicate\ even\ [1\mathinner{..}20] \equiv [1, 20, 3, 18, 5, 16, 7, 14, 9, 12, 11, 10, 13, 8, 15, 6, 17, 4, 19, 2]$$

[1] Para mais detalhes ver o código fonte de Probability, que é uma adaptação da biblioteca PHP ("Probabilistic Functional Programming"). Para quem quiser saber mais recomenda-se a leitura do artigo [?].

Problema 3

Nenhum código é fornecido neste problema.

Problema 4

Os atrasos, medidos em minutos, são inteiros:

$$\textbf{type}\ \mathit{Delay} = \mathbb{Z}$$

Amostra de dados apurados por passageiros:

$$\begin{array}{l}
\mathit{dados} = [((\mathit{S0},\mathit{S1}),0),((\mathit{S0},\mathit{S1}),2),((\mathit{S0},\mathit{S1}),0),((\mathit{S0},\mathit{S1}),3),((\mathit{S0},\mathit{S1}),3),\\
\quad ((\mathit{S1},\mathit{S2}),0),((\mathit{S1},\mathit{S2}),2),((\mathit{S1},\mathit{S2}),1),((\mathit{S1},\mathit{S2}),1),((\mathit{S1},\mathit{S2}),4),\\
\quad ((\mathit{S2},\mathit{S3}),2),((\mathit{S2},\mathit{S3}),2),((\mathit{S2},\mathit{S3}),4),((\mathit{S2},\mathit{S3}),0),((\mathit{S2},\mathit{S3}),5),\\
\quad ((\mathit{S3},\mathit{S4}),2),((\mathit{S3},\mathit{S4}),3),((\mathit{S3},\mathit{S4}),5),((\mathit{S3},\mathit{S4}),2),((\mathit{S3},\mathit{S4}),0),\\
\quad ((\mathit{S4},\mathit{S5}),0),((\mathit{S4},\mathit{S5}),5),((\mathit{S4},\mathit{S5}),0),((\mathit{S4},\mathit{S5}),7),((\mathit{S4},\mathit{S5}),-1)]
\end{array}$$

"Funcionalização" de listas:

$$\begin{array}{l}
\mathit{mkf} :: \mathit{Eq}\ a \Rightarrow [(a,b)] \rightarrow a \rightarrow \mathit{Maybe}\ b\\
\mathit{mkf} = \mathit{flip}\ \mathit{Prelude.lookup}
\end{array}$$

Ausência de qualquer atraso:

$$\begin{array}{l}
\mathit{instantaneous} :: \mathsf{Dist}\ \mathit{Delay}\\
\mathit{instantaneous} = D\ [(0,1)]
\end{array}$$

H Soluções dos alunos

Problema 1

A nosse resolução consiste em tirar a primeira linha da matriz e rodar a matriz 90º no sentido positivo, até a matriz estar vazia.

A operação de rotação e a resolução do problema podem ser definidas da seguinte forma

$$
\begin{array}{l}
\mathit{rotate} :: [[a]] \to [[a]] \\
\mathit{rotate} = \mathit{reverse} \cdot \mathit{transpose} \\
\mathit{matrot} :: [[a]] \to [a] \\
\mathit{matrot}\ [\,] = [\,] \\
\mathit{matrot}\ (h : t) = h +\!\!+\ \mathit{matrot}\ (\mathit{rotate}\ t)
\end{array}
$$

Passamos então para definir esta solução *à la CP*, *pointfree*.

$$\mathit{matrot} = [\mathit{nil}, \mathsf{conc}] \cdot \mathit{recList}\ (\mathit{matrot} \cdot \mathit{rotate}) \cdot \mathit{outList}$$

Seja $F = \mathit{recList}, \mathit{in} = \mathit{inList}, \mathit{out} = \mathit{outList}$

$$
\begin{tikzcd}
A^{**} \arrow[r, "\mathit{out}"'] \arrow[d, "\mathit{matrot}"'] & 1 + A^{*} \times A^{**} \arrow[d, "F\ (\mathit{matrot} \cdot \mathit{rotate})"] \\
A^{*} & 1 + A^{*} \times A^{*} \arrow[l, "{[\mathit{nil}, \mathsf{conc}]}"]
\end{tikzcd}
$$

$$
\begin{array}{ll}
& \mathit{matrot} = [\mathit{nil}, \mathsf{conc}] \cdot F\ (\mathit{matrot} \cdot \mathit{rotate}) \cdot \mathit{out} \\
\equiv & \qquad \{ \ \} \\
& \mathit{matrot} = [\mathit{nil}, \mathsf{conc}] \cdot F\ (\mathit{matrot}) \cdot F\ (\mathit{rotate}) \cdot \mathit{out} \\
\equiv & \qquad \{ \text{ matrot = hylo f g } \} \\
& [\![f, g]\!] = [\mathit{nil}, \mathsf{conc}] \cdot F\ [\![f, g]\!] \cdot F\ (\mathit{rotate}) \cdot \mathit{out} \\
\equiv & \qquad \{ \text{ def-hylo } \} \\
& (\!| f |\!) \cdot [\!(g)\!] = [\mathit{nil}, \mathsf{conc}] \cdot F\ ((\!| f |\!) \cdot [\!(g)\!]) \cdot F\ (\mathit{rotate}) \cdot \mathit{out} \\
\equiv & \qquad \{ \text{ Functor-F , Natural-id } \} \\
& (\!| f |\!) \cdot [\!(g)\!] = [\mathit{nil}, \mathsf{conc}] \cdot F\ (\!| f |\!) \cdot \mathit{id} \cdot F\ [\!(g)\!] \cdot F\ (\mathit{rotate}) \cdot \mathit{out} \\
\equiv & \qquad \{ \text{ in . out = out . in = id } \} \\
& (\!| f |\!) \cdot [\!(g)\!] = [\mathit{nil}, \mathsf{conc}] \cdot F\ (\!| f |\!) \cdot \mathit{out} \cdot \mathbf{in} \cdot F\ [\!(g)\!] \cdot F\ (\mathit{rotate}) \cdot \mathit{out} \\
\equiv & \qquad \{ \text{ Leibnitz } \} \\
& \left\{ \begin{array}{l} (\!| f |\!) = [\mathit{nil}, \mathsf{conc}] \cdot F\ (\!| f |\!) \cdot \mathit{out} \\ [\!(g)\!] = \mathbf{in} \cdot F\ [\!(g)\!] \cdot F\ (\mathit{rotate}) \cdot \mathit{out} \end{array} \right. \\
\equiv & \qquad \{ \text{ Shunt-left, Shunt-right, Cancelamento-cata, Cancelamento-ana } \} \\
& \left\{ \begin{array}{l} f = [\mathit{nil}, \mathsf{conc}] \\ g = F\ (\mathit{rotate}) \cdot \mathit{out} \end{array} \right.
\end{array}
$$

$\square$

ficamos então com

$$
\begin{aligned}
&\mathit{matrot} = \mathit{hyloList}\ f\ g \\
&\quad \textbf{where} \\
&\qquad f = [\mathit{nil}, \mathsf{conc}] \\
&\qquad g = \mathit{recList}\ (\mathit{rotate}) \cdot \mathit{outList}
\end{aligned}
$$

$$
\begin{tikzcd}
A^{**} \arrow[r, bend left, "g"] \arrow[d, "{[\!(\, g \,)\!]}"'] \arrow[dd, bend right=60, "{\mathit{matrot}=[\![f,g]\!]}"'] & 1 + A^{*} \times A^{**} \arrow[d, "{F\ [\!(\, g \,)\!]}"] \\
A^{**} \arrow[r, bend left, "\mathit{outList}"] \arrow[d, "{(\!|\, f \,|\!)}"'] & 1 + A^{*} \times A^{**} \arrow[l, bend left, "\mathit{inList}"] \arrow[d, "{F\ (\!|\, f \,|\!)}"] \\
A^{*} & 1 + A^{*} \times A^{*} \arrow[l, bend left, "f"]
\end{tikzcd}
\tag{5}
$$

Curiosamente, a própria função rotate é composta por duas funções sobre listas, a intuição diz-nos que talvez estas se tratem de anamorfismos ou catamorfismos.

É efetivamente possível definir as funções que servem inverter uma lista e transpor uma matriz como um catamorfismo ou anamorfismo (ambos). No entanto, como chamamos a reverse após a transpose decidimos defini-las de modo a que a função rotate passasse a ser um hilomorfismo.

$$
\begin{aligned}
&\mathit{reverse\_gen} :: () + a \times [a] \rightarrow [a] \\
&\mathit{reverse\_gen} = [\mathit{nil}, \mathsf{conc} \cdot \mathit{swap} \cdot (\mathit{singl} \times \mathit{id})]
\end{aligned}
$$

$$
\begin{aligned}
&\mathit{transpose\_gen} :: [[a]] \rightarrow () + ([a] \times [[a]]) \\
&\mathit{transpose\_gen}\ ([]: \_) = i_1\ () \\
&\mathit{transpose\_gen}\ [] = i_1\ () \\
&\mathit{transpose\_gen}\ l = i_2\ ((\mathsf{map}\ \ \mathit{head}\ l), (\mathsf{map}\ \ \mathit{tail}\ l))
\end{aligned}
$$

$$
\mathit{rotate} = \mathit{hyloList}\ \mathit{reverse\_gen}\ \mathit{transpose\_gen}
$$

Problema 2

Numa primiera tentativa decicidmos duplicar a lista, mantendo a cópia original e uma cópia filtrada pelo predicado e invertida, para, de seguida, substituir os elementos da lista original, que cumprem o predicado, pelos elementos da lista invertida.

Podemos definir a função para o exercício e a função auxiliar que faz esta fusão das listas, este *replaceWhen*, da seguinte maneira:

$$
\begin{array}{l}
\mathit{replaceWhen} :: (a \to \mathit{Bool}) \to ([a],[a]) \to [a] \\
\mathit{replaceWhen}\ f\ ((\mathit{h1} : \mathit{t1}), l_2@(\mathit{h2} : \mathit{t2})) = \\
\quad \textbf{if}\ f\ \mathit{h1}\ \textbf{then} \\
\qquad \mathit{h2} : (\mathit{replaceWhen}\ f\ (\mathit{t1}, \mathit{t2})) \\
\quad \textbf{else} \\
\qquad \mathit{h1} : (\mathit{replaceWhen}\ f\ (\mathit{t1}, l_2)) \\
\mathit{replaceWhen}\ \_\ (\mathit{l1}, \_) = \mathit{l1}
\end{array}
$$

$$
\begin{array}{l}
\mathit{reverseByPredicate} :: (a \to \mathit{Bool}) \to [a] \to [a] \\
\mathit{reverseByPredicate}\ \_\ [\,] = [\,] \\
\mathit{reverseByPredicate}\ f\ l = \mathit{replaceWhen}\ f\ l\ ((\mathit{reverse} \cdot (\mathit{filter}\ f))\ l)
\end{array}
$$

Deste modo temos então as funções em *pointwise*. Aplicando equivalências de cálculo de programas chegamos à seguinte definição *pointfree*:

$$
\begin{array}{l}
\mathit{replaceWhen} :: (a \to \mathit{Bool}) \to ([a],[a]) \to [a] \\
\mathit{replaceWhen}\ f = [g, h] \cdot \mathit{alpha} \\
\quad \textbf{where}\ \mathit{alpha} = \mathit{coassocr} \cdot (\mathit{distr} + \mathit{distr}) \cdot \mathit{distl} \cdot (\mathit{coswap} \times \mathit{coswap}) \cdot (\mathit{outList} \times \mathit{outList}) \\
\qquad g = \mathit{cons} \cdot (\mathit{cond}\ (f \cdot \pi_1 \cdot \pi_1)\ \mathit{true}'\ \mathit{false}') \\
\qquad \mathit{true}' = \langle \pi_1 \cdot \pi_2, (\mathit{replaceWhen}\ f) \cdot \langle \pi_2 \cdot \pi_1, \pi_2 \cdot \pi_2 \rangle \rangle \\
\qquad \mathit{false}' = \langle \pi_1 \cdot \pi_1, (\mathit{replaceWhen}\ f) \cdot \langle \pi_2 \cdot \pi_1, \mathit{cons} \cdot \langle \pi_1 \cdot \pi_2, \pi_2 \cdot \pi_2 \rangle \rangle \rangle \\
\qquad h = \mathit{inList} \cdot [i_2 \cdot \pi_1, [i_1 \cdot \pi_1, i_1 \cdot \pi_1]]
\end{array}
$$

$$
\begin{array}{c}
A^* \times B^* \\
\Big\downarrow \alpha \\
(A \times A^*) \times (B \times B^*) + ((A \times A^*) \times 1 + (1 \times (B \times B^*) + 1 \times 1)) \\
\Big\downarrow [g,h] \\
A^*
\end{array} \tag{6}
$$

Percebemos que esta definição trás imensa complexidade, e então definimos um functor dos pares de listas para ajudar a resolver o problema.

H.1 ListPair

$$ListPar(A) \xrightarrow{out} A^* + A \times ListPar(A) \tag{7}$$

$\textbf{type}\ ListPair\ a = ([a],[a])$

$outListPair :: ListPair\ a \rightarrow [a] + (a, ListPair\ a)$
$outListPair\ ([\,], l) = i_1\ l$
$outListPair\ ((h:t), l) = i_2\ (h, (t, l))$

$inListPair :: [a] + (a, ListPair\ a) \rightarrow ListPair\ a$
$inListPair = [\langle \underline{[\,]}, id\rangle, (cons \times id) \cdot assocl]$

$recListPair\ f = id + id \times f$

No *out* deste functor desdobramos a lista da esquerda num par cabeça cauda e mantemos a lista da direita, caso a lista da esquerda for vazia ficamos apenas com a lista da direita. O **in** faz o converso deste *out*.

Com este functor podemos então definir a replaceWhen usando um anamorfismo sobre este tipo.

$replaceWhen\_ana\_aux :: (a \rightarrow Bool) \rightarrow ([a],[a]) \rightarrow ([a],[a])$
$replaceWhen\_ana\_aux\ f = anaListPair\ ((id + (aux\_)) \cdot outListPair)$
$\quad \textbf{where}\ aux\_\ (a, (as, (b:bs))) = \textbf{if}\ f\ a\ \textbf{then}\ (b, (as, bs))\ \textbf{else}\ (a, (as, b:bs))$
$\quad\quad aux\_\ \_ = error$ `"lista B mais pequena que lista A"`

$replaceWhen :: (a \rightarrow Bool) \rightarrow ([a],[a]) \rightarrow [a]$
$replaceWhen\ f = (\pi_1 \cdot replaceWhen\_ana\_aux\ f)$

O functor simplifica bastante o processo uma vez que o seu out encpasula os dois casos relevantes: o caso em que lista da esuqerda tem ou não elementos.

No entanto como se faz *reverse* e *filter* efetivamente itera-se a lista 3 vezes. Resolvemos então definir uma solução alternativa que faça tudo numa iteração sobre a lista.

$splitOn :: (a \rightarrow Bool) \rightarrow [a] \rightarrow (a, [a])$
$splitOn\ \_\ [x] = (x, [\,])$
$splitOn\ f\ (h:t) = \textbf{if}\ f\ h\ \textbf{then}\ (h, t)\ \textbf{else}\ (left, right)\ \textbf{where}\ (left, right) = splitOn\ f\ t$
$splitOn\ \_\ \_ = error$ `"Lista vazia"`

$replaceWhenReversed :: (a \rightarrow Bool) \rightarrow ([a],[a]) \rightarrow ([a],[a])$
$replaceWhenReversed\ f = cataListPair\ gene$
$\quad \textbf{where}$
$\quad\quad gene = inListPair \cdot (id + (cond\ (f \cdot \pi_1)\ aux\ id))$
$\quad\quad aux\ (\_, (y, z)) = (h, (y, t))\ \textbf{where}\ (h, t) = splitOn\ f\ z$

$reverseByPredicate :: (a \rightarrow Bool) \rightarrow [a] \rightarrow [a]$
$reverseByPredicate\ g = \pi_1 \cdot (replaceWhenReversed\ g) \cdot dup$

A função splitOn retorna, num par,o primeiro elemento de uma lista que satisfaz um predicato e os restantes elementos. Esta função falha se se passar uma lista vazia ou uma lista sem elementos que satisfaçam o predicado, no entanto estes casos não acontecem devido aos argumentos que lhe passamos.

Para cada elemento da primeira lista, este catamorfismo verifica se este satisfaz o predicado, se tal for verdade o gene troca o elemento que será inserido na lista pelo inListPair pelo primeiro elemento que cumpre o predicado na segunda lista, e remove da mesma todos os elementos desde o início da lista até esse elemento.

Pela natureza da recursão a inserção dos elementos da lista da direita é invertida.

Assim, essencialmente, a função splitOn cumpre a responsabilidade da filter da primeira solução, e a estrutura recursiva da função faz com que os elementos sejam inseridos em ordem inversa, surtindo o efeito da reverse.

A função *reverseVowels* define-se então à custa da *reverseByPredicate* e de um predicado *isVowel*

$$isVowel :: Char \rightarrow Bool$$
$$isVowel = flip\ elem\ \texttt{"aeiouAEIOU"}$$

$$reverseVowels = reverseByPredicate\ isVowel$$

Problema 3

Para resolver o problema 3, é necessário implementar o somatório seguinte:

$$\sum_{k=0}^{\infty} \frac{x^{2k+1}}{(2k+1)!} \tag{8}$$

Para o fazer de forma eficiente é necessário definir a sucessão no corpo do somatório de forma recursiva. Seja então

$s(k) = \frac{x^{2k+1}}{(2k+1)!}$

Temos que

$s(0) = \frac{x^{2*0+1}}{(2*0+1)!} = \frac{x^1}{1!} = x$

E, por

$\frac{\frac{x^{2(k+1)+1}}{(2(k+1)+1)!}}{\frac{x^{2k+1}}{(2k+1)!}} = \frac{\frac{x^{2k+3}}{x^{2k+1}}}{\frac{(2k+3)!}{(2k+1)!}} = \frac{\frac{x^{2k+1}x^2}{x^{2k+1}}}{\frac{(2k+3)(2k+2)(2k+1)!}{(2k+1)!}} = \frac{x^2}{(2k+3)(2k+2)}$

Temos também que

$s(k+1) = s(k) * \frac{x^2}{(2k+3)(2k+2)}$

Como $(2k+3)(2k+2) = (4k^2 + 10k + 6)$

E para todos os polinómios p de grau 2 ou inferior

$p(n) = 3p(n-1) - 3p(n-2) + p(n-3)$

É possivel definir totalmente o polinómio no denominador da fracção também de forma recursiva, sabendo os três primeiros termos

Utilizando também uma estratégia de memoização para não termos de calcular o x^2 em cada iteração chega-se à seguinte deifinição

$$
\begin{array}{l}
\mathit{ex3} :: (\mathit{Floating}\ p) \Rightarrow p \rightarrow \mathit{Int} \rightarrow p \\
\mathit{ex3}\ x = \mathit{wrapper} \cdot \mathit{worker} \\
\quad \textbf{where}\ \mathit{wrapper} = \pi_1 \\
\quad\quad \mathit{worker} = \mathsf{for}\ \mathit{loop}\ \mathit{start}\ x
\end{array}
$$

$$
\begin{array}{l}
\mathit{loop}\ (\mathit{acc}, (\mathit{prev}, \mathit{prev\_q}, \mathit{x\_squared})) = (\mathit{acc} + \mathit{next}, (\mathit{next}, (\mathit{next\_q}'), \mathit{x\_squared})) \\
\quad \textbf{where} \\
\quad\quad \mathit{next} = \mathit{prev} * \mathit{x\_squared}\ /\ \mathit{fromInteger}\ (\pi_2\ \mathit{next\_q}') \\
\quad\quad \mathit{next\_q}' = (((3 * \mathit{m1}) - (3 * \mathit{m2}) + \mathit{m3}, \mathit{m1}), \mathit{m2})\ \textbf{where}\ ((\mathit{m1}, \mathit{m2}), \mathit{m3}) = \mathit{prev\_q}
\end{array}
$$

$$\mathit{start}\ x = (x, (x, ((20, 6), 0), x ** 2))$$

Problema 4

De maneira a solucionar este problema, podemos dividi-lo em 4 partes:

H.2 mkDist

Para se conseguir obter estatisitcas dos dados, é necessário definir uma função que gere uma distribuição. A partir da funções abaixo definidas, é possível obter uma distribuição que dependa do número de ocorrências de um determinado valor numa lista.

$$
\begin{array}{l}
\mathit{msetplus} :: \mathit{Eq}\ a \Rightarrow [a] \rightarrow ([(a, \mathit{Int})], \mathit{Int}) \\
\mathit{msetplus}\ [\,] = ([\,], 0) \\
\mathit{msetplus}\ (h : t) = (((h, c) : (x)), y + c) \\
\quad \textbf{where}\ (x, y) = \mathit{msetplus}\ \mathit{rest} \\
\qquad (c, \mathit{rest}) = (1 + \mathit{length}\ (\mathit{filter}\ (h \equiv)\ t), (\mathit{filter}\ (h \not\equiv))\ t)
\end{array}
$$

$$
\begin{array}{l}
\mathit{relativeFrequence} :: (\mathit{Eq}\ b) \Rightarrow [b] \rightarrow [(b, \mathit{Float})] \\
\mathit{relativeFrequence}\ l = \mathsf{map}\ \ (\mathit{id} \times (((/\mathit{fromIntegral}\ s) \cdot \mathit{fromIntegral})))\ \mathit{mset} \\
\quad \textbf{where}\ (\mathit{mset}, s) = \mathit{msetplus}\ l
\end{array}
$$

$$
\begin{array}{l}
\mathit{mkdist} :: \mathit{Eq}\ a \Rightarrow [a] \rightarrow \mathsf{Dist}\ a \\
\mathit{mkdist} = \mathit{mkD} \cdot \mathit{relativeFrequence}
\end{array}
$$

A função *msetplus* define a partir de uma lista um *MultiSet*, um *Set* que guarda um elemento e a quantidade de vezes que este aparece numa lista, (retorna também o tamanho da lista incial)

A função *relativeFrequence* calcula a frequência relativa de cada um dos elementos deste *MultiSet*.

Apartir dai a *mkdist*, utiliza a *mkD*, para geraar a distribuição de ocorrências dos valores.

H.3 DB

De maneira a conseguirmos gerar *DB* definimos a seguinte função:

$$
\begin{array}{l}
\mathit{db} :: [(\mathit{Segment}, \mathsf{Dist}\ \mathit{Delay})] \\
\mathit{db} = \mathsf{map}\ \ \langle \pi_1 \cdot \mathit{head}, (\mathit{mkdist} \cdot (\mathsf{map}\ \ \pi_2)) \rangle \cdot \mathit{groupBy}\ (\lambda x\ y \rightarrow \pi_1\ x \equiv \pi_1\ y)\ \$\ \mathit{dados}
\end{array}
$$

A partir da *groupBy* e da utilização da funlão anónima $\lambda x\ \ y \ \rightarrow\ \pi_1\ \ x\ \equiv\ \pi_1\ \ y$, agrupam-se os dados pelo seu segmento. De seguida, utiliza-se o $\langle \pi_1 \cdot \mathit{head}, \mathit{mkdist} \cdot (\mathsf{map}\ \ \pi_2) \rangle$ para criar os pares $(\mathit{Segment}, \mathsf{Dist}\ \mathit{Delay})$.

H.4 delay

Utilizando a função DB e mkf define-se a função $delay$.

$$
\begin{array}{l}
mkf :: Eq\ a \Rightarrow [(a,b)] \rightarrow a \rightarrow Maybe\ b \\
mkf = flip\ Prelude.lookup \\
\\
instantaneous :: \mathsf{Dist}\ Delay \\
instantaneous = D\ [(0,1)] \\
\\
delay :: Segment \rightarrow \mathsf{Dist}\ Delay \\
delay = [\underline{instantaneous}, id] \cdot outMaybe \cdot mkf\ \ db
\end{array}
$$

Através da utilização da função mkf funcionliza-se a DB. No entanto, caso o segmento dado como argumento não exsita na base de dados um simples $lookup$ retorna $Nothing$, para evitar poetenciais erros, nesse retorna-se a distribuição "instanânea" que, no contexto deste problema, é o elemento neutro das operações que é necessário fazer sobre estas distribuições

H.5 pdelay

Definir a $pdelay$, passa por criar uma lista com todos os segmentos num dado caminho, e efetuar uma operação de convolução discreta sobre as distribuições associadas a cada segmento da lista. Implementamos esta função como um hilomorfismo sobre listas.

$$
\begin{array}{l}
devide :: (Eq\ b, Enum\ b, Ord\ b) \Rightarrow (b,b) \rightarrow () + ((b,b),(b,b)) \\
devide\ (s, final) \\
\quad \mid s \geqslant final = i_1\ () \\
\quad \mid otherwise = i_2\ ((s, \mathsf{succ}\ \ s), (\mathsf{succ}\ \ s, final)) \\
\\
conquer :: a + (Segment, \mathsf{Dist}\ Delay) \rightarrow \mathsf{Dist}\ Delay \\
conquer = [\underline{instantaneous}, aux] \\
\quad \mathbf{where}\ aux = (joinWith\ \widehat{(+)} \cdot delay)
\end{array}
$$

A convolução passa essencialemente por agrupar os atrasos iguais num novo evento probabilístico.

Como referido acima o elemento neutro da opereção de convolução no contexto deste problema é a distribuição dada por $instantaneous$.

$$
\begin{tikzcd}
Stop \times Stop \arrow[r, "devide"] \arrow[d, "{[\![devide]\!]}"'] \arrow[dd, bend right=60, "{pdelay = [\![conquer, devide]\!]}"'] & 1 + Segment \times (Stop \times Stop) \arrow[d, "{F\ [\![devide]\!]}"] \\
Segment^* \arrow[r, bend left, "outList"] \arrow[d, "{(\![conquer]\!)}"'] & 1 + Segment \times Segment* \arrow[l, bend left, "inList"] \arrow[d, "{F\ (\![conquer]\!)}"] \\
\mathsf{Dist}\ Delay & 1 + (Segment \times \mathsf{Dist}\ Delay) \arrow[l, "conquer"]
\end{tikzcd}
$$

$$
\begin{array}{l}
pdelay :: Stop \rightarrow Stop \rightarrow \mathsf{Dist}\ Delay \\
pdelay = \overline{hyloList\ conquer\ devide}
\end{array}
$$

Finalmente, com a função $conquer$ e $divide$ obtemos então o hilomorfismo da $pdelay$.