

Universidade do Minho
Escola de Engenharia

Cálculo de Programas

Trabalho Prático (2023/24)

Lic. em Engenharia Informática

Grupo G99

xxxxxxx	Nome
xxxxxxx	Nome
xxxxxxx	Nome

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 abordarem os problemas propostos no trabalho, os grupos devem ler com atenção o anexo **A** onde encontrarão as instruções relativas ao software 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:

1	→	2	→	3
				↓
4	→	5		6
↑				↓
7	←	8	←	9

1	→	2	→	3	→	4
						↓
5	→	6	→	7		8
↑						↓
9	←	10	←	11	←	12

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

□

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 **D**.

1 Resolução

De maneira a resolver este problema decidimos fazer alguma pesquisa. Chegamos à conclusão que este problema era bastante conhecido como **matrot**.

A nossa resolução baseia-se na utilização das funções *transpose* e *reverse*. A ideia desta resolução baseia-se em recursivamente, tirar a primeira linha, inverter a ordem de cada uma das linhas e fazer *transpose* à matriz, até que obtemos uma matriz com só um elemento.

De maneira a testar esta resolução definimos então a função no formato *pointwise*. Alcançamos a seguinte resposta:

```
ex1pw :: [[a]] → [a]
ex1pw [] = []
ex1pw (h : t) = h ++ ex1 (reverse (transpose t))
```

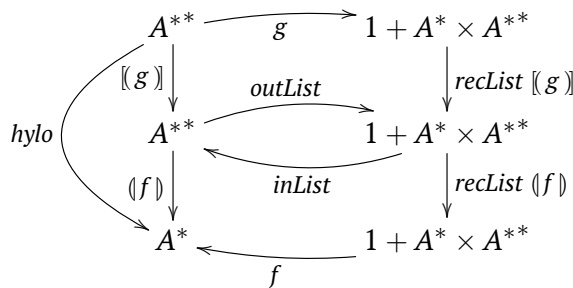
Testando este código, conseguimos entender que realmente utilizar a função *transpose* e *reverse* permitem nos obter a resolução certa.

Passamos então para definir esta solução *à la CP*, *pointfree*. Para isso começamos por definir o diagrama para resolver a mesma.

Tendo então o diagrama chegamos à seguinte solução

```
ex1 = [nil, conc] · recList (ex1 · reverse · transpose) · outList
```

Olhando para esta definição e analisando melhor o diagrama, conseguimos deduzir que podemos definir este problema como um hilomorfismo. Vamos então desenvolver o diagrama para provar isto.



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](#),

$$\tanh x = \frac{\sinh x}{\cosh x} \quad (1)$$

podendo estes ser definidos pelas seguintes [séries de Taylor](#):

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

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

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

que implementa $\sinh x$, uma das funções de $\tanh x$ (1), através da soma das i primeiras parcelas da sua série (2).

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 passageiros. 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

data Stop = S0 | S1 | S2 | S3 | S4 | S5 **deriving** (Show, Eq, Ord, Enum)

que formam a linha $[S0 \dots S5]$ assumindo a ordem determinada pela instância de *Stop* na classe *Enum*;

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

type *Segment* = (*Stop*, *Stop*)

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

dados :: [(*Segment*, *Delay*)]

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

A partir destes dados, há que:

- gerar a base de dados probabilística

db :: [(*Segment*, *Dist 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

mkdist :: *Eq a* \Rightarrow [*a*] \rightarrow *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

delay :: *Segment* \rightarrow *Dist Delay*

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

Finalmente, o objectivo principal é definir a função probabilística:

pdelay :: *Stop* \rightarrow *Stop* \rightarrow *Dist Delay*

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 $[a \dots b]$.

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

2 Resolução

Para solucionar este problema dividimo-lo em 3 partes:

2.1 mkDB

Para formar a base de dados no formato pretendido [(*Segment*, *Dist Delay*)] temos que transformar os dados fornecidos (tomando como exemplo o código fornecido no anexo E). Reparamos que esta etapa pode ser resolvida dando uso a um hilomorfismo que começa por usar a função *groupBy* para agrupar os elementos com o segmento equivalente numa lista, e de seguida a cada sublista gerada formar só um par com o segmento e com a distribuição gerada de todos os *Delay*'s contidos na sublista.

$$mkDB :: Eq\ a \Rightarrow [(a, b)] \rightarrow [(a, Dist\ b)]$$

$$mkDB = map\ (\pi_1 \cdot head, uniform \cdot map\ \pi_2) \cdot groupBy\ (\lambda x\ y \rightarrow \pi_1\ x \equiv \pi_1\ y)$$

Este hilomorfismo pode ser demonstrado no seguinte diagrama.

$$(A \succ B) * \{->\}[\mathbb{R}] \searrow \text{ar} -> [d]^{ana}$$

$$1 + (A \succ B) \succ (A \succ B) * \{->\}[d]^{\{Fa$$

2.2 delay

Após termos os dados no formato correto, pretendemos conseguir facilmente obter o delay associado a um determinado *Segment* e também de forma eficiente. Esta operação é bastante elementar, para a realizar simplesmente fazemos um *lookup* na nossa base de dados do segmento pretendido como apresentado no código seguinte.

$$delay :: Segment \rightarrow Dist\ Delay$$

$$delay = fromJust \cdot \widehat{List.lookup} \cdot \langle id, hashT \rangle$$

2.3 pdelay

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.

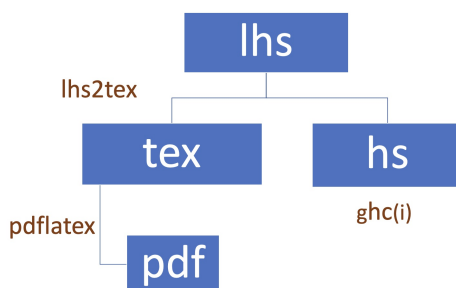
Para cumprir de forma integrada os objectivos do trabalho vamos recorrer a uma técnica de programação dita “[literária](#)” [?], cujo princípio base é o seguinte:

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 de um [container](#) Docker que deverá ser gerado 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 [L^AT_EX](#). (Ver também a `Makefile` que é disponibilizada.)

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

¹ O sufixo ‘lhs’ quer dizer *literate Haskell*.

```
$ 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 [L^AT_EX](#). 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-processor que faz “pretty printing” de código Haskell em [L^AT_EX](#) 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 [F](#) 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 [Bib_TE_X](#)) 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 [E](#) 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 [D](#) que se segue.

D Como exprimir cálculos e diagramas em LaTeX/lhs2TeX

Como primeiro exemplo, estudar o texto fonte ([lhs](#)) do que está a ler¹ onde se obtém o efeito seguinte:²

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

Os diagramas podem ser produzidos recorrendo à *package* [xymatrix](#), por exemplo:

$$\begin{array}{ccc} \mathbb{N}_0 & \xleftarrow{\text{in}} & 1 + \mathbb{N}_0 \\ \text{\scriptsize (g)} \downarrow & & \downarrow \text{\scriptsize $id+(g)$} \\ B & \xleftarrow{g} & 1 + B \end{array}$$

E 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 "Cristina Monteiro Carvalho Sequeira"
test1 = matrot m1 ≡ [1, 2, 3, 6, 9, 8, 7, 4, 5]
test2 = matrot m2 ≡ [1, 2, 3, 4, 8, 12, 11, 10, 9, 5, 6, 7]
test3 = matrot m3 ≡ "CristinaooarieuqeSCMonteirlavra"
```

Problema 2

```
test4 = reverseVowels "" ≡ ""
test5 = reverseVowels "ácidos" ≡ "ocidás"
test6 = reverseByPredicate even [1..20] ≡ [1, 20, 3, 18, 5, 16, 7, 14, 9, 12, 11, 10, 13, 8, 15, 6, 17, 4, 19, 2]
```

F 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 ao anexo, bem como diagramas e/ou outras funções auxiliares que sejam necessárias.

¹ Procure e.g. por "sec:diagramas".

² Exemplos tirados de [?].

Importante: Não pode ser alterado o texto deste ficheiro fora deste anexo.

Problema 1

$matrot :: Eq\ a \Rightarrow [[a]] \rightarrow [a]$
 $matrot = \perp$

Problema 2

$reverseVowels :: String \rightarrow String$
 $reverseVowels = \perp$
 $reverseByPredicate :: (a \rightarrow Bool) \rightarrow [a] \rightarrow [a]$
 $reverseByPredicate\ p = \perp$