Cálculo de Programas
Trabalho Prático
MiEI+LCC — Ano Lectivo de 2016/17

Departamento de Informática
Universidade do Minho

Junho de 2017

Grupo nr.	04
A72053	Ângela Andrade Carvalho Fernandes
A65432	João Manuel Martins Cerqueira
A71506	Sónia Catarina Guerra Costa

Conteúdo

1 Preâmbulo

A disciplina de 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, restringe-se a aplicação deste método ao desenvolvimento de programas funcionais na linguagem Haskell.

O presente trabalho tem por objectivo concretizar na prática os objectivos da disciplina, colocando os alunos perante problemas de programação que deverão ser abordados composicionalmente e implementados em Haskell. Há ainda um outro objectivo: o de ensinar a documentar programas 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" [3], cujo princípio base é o seguinte:

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

Por outras palavras, o código fonte e a sua documentação deverão constar do mesmo documento (ficheiro).

O ficheiro `cp1617t.pdf` que está a ler é já um exemplo de programação literária: foi gerado a partir do texto fonte `cp1617t.lhs`[1] que encontrará no material pedagógico desta disciplina descompactando o ficheiro `cp1617t.zip` e executando

```
lhs2TeX cp1617t.lhs > cp1617t.tex
pdflatex cp1617t
```

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

> `https://hackage.haskell.org/package/lhs2tex`.

Por outro lado, o mesmo ficheiro `cp1617t.lhs` é executável e contém o "kit" básico, escrito em Haskell, para realizar o trabalho. Basta executar

```
ghci cp1617t.lhs
```

para ver que assim é:

```
GHCi, version 8.0.2: http://www.haskell.org/ghc/  :? for help
[ 1 of 11] Compiling Show             ( Show.hs, interpreted )
[ 2 of 11] Compiling ListUtils        ( ListUtils.hs, interpreted )
[ 3 of 11] Compiling Probability      ( Probability.hs, interpreted )
[ 4 of 11] Compiling Cp               ( Cp.hs, interpreted )
[ 5 of 11] Compiling Nat              ( Nat.hs, interpreted )
[ 6 of 11] Compiling List             ( List.hs, interpreted )
[ 7 of 11] Compiling LTree            ( LTree.hs, interpreted )
[ 8 of 11] Compiling St               ( St.hs, interpreted )
[ 9 of 11] Compiling BTree            ( BTree.hs, interpreted )
[10 of 11] Compiling Exp              ( Exp.hs, interpreted )
[11 of 11] Compiling Main             ( cp1617t.lhs, interpreted )
Ok, modules loaded: BTree, Cp, Exp, LTree, List, ListUtils, Main, Nat,
Probability, Show, St.
```

O facto de o interpretador carregar as bibliotecas do material pedagógico da disciplina, entre outras, deve-se ao facto de, neste mesmo sítio do texto fonte, se ter inserido o seguinte código Haskell:

import *Cp*
import *List*

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

```
import Nat
import Exp
import BTree
import LTree
import St
import Probability hiding (cond)
import Data.List
import Test.QuickCheck hiding ((×))
import System.Random hiding ⟨·, ·⟩
import GHC.IO.Exception
import System.IO.Unsafe
```

Abra o ficheiro `cp1617t.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}
```

vai ser seleccionado pelo GHCi para ser executado.

3 Como realizar o trabalho

Este trabalho teórico-prático deve ser realizado por grupos de três 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 equilibrada e 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 C com as suas 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 cp1617t.aux
makeindex cp1617t.idx
```

e recompilar o texto como acima se indicou. Dever-se-á ainda instalar o utilitário QuickCheck [2] que ajuda a validar programas em Haskell.

Problema 1

O controlador de um processo físico baseia-se em dezenas de sensores que enviam as suas leituras para um sistema central, onde é feito o respectivo processamento.

Verificando-se que o sistema central está muito sobrecarregado, surgiu a ideia de equipar cada sensor com um microcontrolador que faça algum pré-processamento das leituras antes de as enviar ao sistema central. Esse tratamento envolve as operações (em vírgula flutuante) de soma, subtracção, multiplicação e divisão.

Há, contudo, uma dificuldade: o código da divisão não cabe na memória do microcontrolador, e não se pretende investir em novos microcontroladores devido à sua elevada quantidade e preço.

Olhando para o código a replicar pelos microcontroladores, alguém verificou que a divisão só é usada para calcular inversos, $\frac{1}{x}$. Calibrando os sensores foi possível garantir que os valores a inverter estão entre $1 < x < 2$, podendo-se então recorrer à série de Maclaurin

$$\frac{1}{x} = \sum_{i=0}^{\infty} (1-x)^i$$

para calcular $\frac{1}{x}$ sem fazer divisões. Seja então

$$inv\ x\ n = \sum_{i=0}^{n} (1-x)^i$$

[2] Para uma breve introdução ver e.g. https://en.wikipedia.org/wiki/QuickCheck.

a função que aproxima $\frac{1}{x}$ com n iterações da série de MacLaurin. Mostre que $inv\ x$ é um ciclo-for, implementando-o em Haskell (e opcionalmente em C). Deverá ainda apresentar testes em QuickCheck que verifiquem o funcionamento da sua solução. (**Sugestão:** inspire-se no problema semelhante relativo à função ns da secção 3.16 dos apontamentos [4].)

Problema 2

Se digitar *man wc* na shell do Unix (Linux) obterá:

```
NAME
     wc -- word, line, character, and byte count

SYNOPSIS
     wc [-clmw] [file ...]

DESCRIPTION
    The wc utility displays the number of lines, words, and bytes contained in
    each input file,  or standard input (if no file is specified) to the stan-
    dard  output.  A line is defined as  a string of characters delimited by a
    <newline> character.  Characters beyond the final <newline> character will
    not be included in the line count.
    (...)
    The following options are available:
    (...)
        -w   The number of words in each input file is written to the standard
             output.
    (...)
```

Se olharmos para o código da função que, em C, implementa esta funcionalidade [2] e nos focarmos apenas na parte que implementa a opção `-w`, verificamos que a poderíamos escrever, em Haskell, da forma seguinte:

$$
\begin{array}{l}
wc\_w :: [\,Char\,] \rightarrow Int \\
wc\_w\ [\,] = 0 \\
wc\_w\ (c : l) = \\
\quad \mathbf{if}\ \neg\ (sep\ c) \wedge lookahead\_sep\ l \\
\quad \mathbf{then}\ wc\_w\ l + 1 \\
\quad \mathbf{else}\ wc\_w\ l \\
\qquad \mathbf{where} \\
\qquad\quad sep\ c = (c \equiv \texttt{'\ '} \vee c \equiv \texttt{'\textbackslash n'} \vee c \equiv \texttt{'\textbackslash t'}) \\
\qquad\quad lookahead\_sep\ [\,] = True \\
\qquad\quad lookahead\_sep\ (c : l) = sep\ c
\end{array}
$$

Re-implemente esta função segundo o modelo *worker/wrapper* onde *wrapper* deverá ser um catamorfismos de listas. Apresente os cálculos que fez para chegar a essa sua versão de wc_w e inclua testes em QuickCheck que verifiquem o funcionamento da sua solução. (**Sugestão:** aplique a lei de recursividade múltipla às funções wc_w e $lookahead\_sep$.)

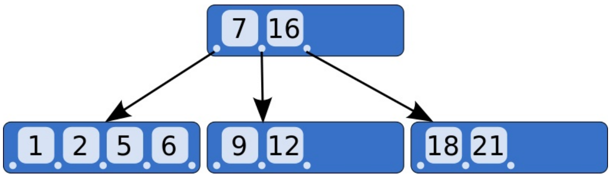
Problema 3

Uma "B-tree"é uma generalização das árvores binárias do módulo BTree a mais do que duas sub-árvores por nó:

$$\mathbf{data}\ \text{B-tree}\ a = Nil \mid Block\ \{leftmost :: \text{B-tree}\ a, block :: [(a, \text{B-tree}\ a)]\}\ \mathbf{deriving}\ (Show, Eq)$$

Por exemplo, a B-tree[3]

[3]Créditos: figura extraída de https://en.wikipedia.org/wiki/B-tree.



é representada no tipo acima por:

$$
\begin{array}{l}
t = Block\ \{\\
\quad leftmost = Block\ \{\\
\quad\quad leftmost = Nil,\\
\quad\quad block = [(1, Nil), (2, Nil), (5, Nil), (6, Nil)]\},\\
\quad block = [\\
\quad\quad (7, Block\ \{\\
\quad\quad\quad leftmost = Nil,\\
\quad\quad\quad block = [(9, Nil), (12, Nil)]\}),\\
\quad\quad (16, Block\ \{\\
\quad\quad\quad leftmost = Nil,\\
\quad\quad\quad block = [(18, Nil), (21, Nil)]\})\\
\quad\quad]\}
\end{array}
$$

Pretende-se, neste problema:

1. Construir uma biblioteca para o tipo B-tree da forma habitual (in + out; ana + cata + hylo; instância na classe $Functor$).
2. Definir como um catamorfismo a função $inordB\_tree$ :: B-tree $t \rightarrow [t]$ que faça travessias "inorder"de árvores deste tipo.
3. Definir como um catamorfismo a função $largestBlock$ :: B-tree $a \rightarrow Int$ que detecta o tamanho do maior bloco da árvore argumento.
4. Definir como um anamorfismo a função $mirrorB\_tree$ :: B-tree $a \rightarrow$ B-tree a que roda a árvore argumento de 180º
5. Adaptar ao tipo B-tree o hilomorfismo "quick sort"do módulo BTree. O respectivo anamorfismo deverá basear-se no gene $lsplitB\_tree$ cujo funcionamento se sugere a seguir:

$$
\begin{array}{l}
lsplitB\_tree\ [\,] = i_1\ ()\\
lsplitB\_tree\ [7] = i_2\ ([\,], [(7, [\,])])\\
lsplitB\_tree\ [5, 7, 1, 9] = i_2\ ([1], [(5, [\,]), (7, [9])])\\
lsplitB\_tree\ [7, 5, 1, 9] = i_2\ ([1], [(5, [\,]), (7, [9])])
\end{array}
$$

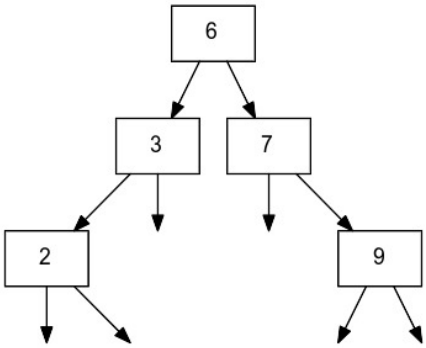
6. A biblioteca Exp permite representar árvores-expressão em formato DOT, que pode ser lido por aplicações como por exemplo Graphviz, produzindo as respectivas imagens. Por exemplo, para o caso de árvores BTree, se definirmos

$$
\begin{array}{l}
dotBTree :: Show\ a \Rightarrow \mathsf{BTree}\ a \rightarrow \mathsf{IO}\ ExitCode\\
dotBTree = dotpict \cdot bmap\ nothing\ (Just \cdot show) \cdot cBTree2Exp
\end{array}
$$

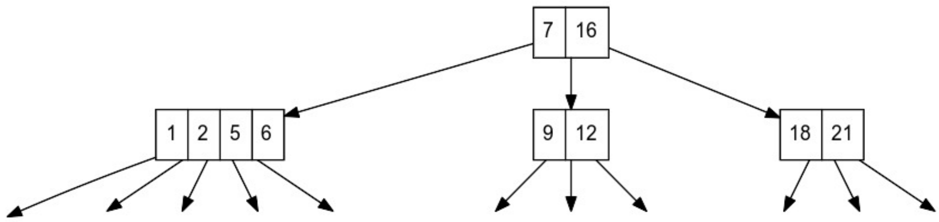
executando $dotBTree\ t$ para

$$
t = Node\ (6, (Node\ (3, (Node\ (2, (Empty, Empty)), Empty)), Node\ (7, (Empty, Node\ (9, (Empty, Empty))))))
$$

obter-se-á a imagem



Escreva de forma semelhante uma função $dotB\_tree$ que permita mostrar em Graphviz[4] árvores B-tree tal como se ilustra a seguir,



para a árvora dada acima.

Problema 4

Nesta disciplina estudaram-se funções mutuamente recursivas e como lidar com elas. Os tipos indutivos de dados podem, eles próprios, ser mutuamente recursivos. Um exemplo dessa situação são os chamados L-Systems.

Um L-System é um conjunto de regras de produção que podem ser usadas para gerar padrões por re-escrita sucessiva, de acordo com essas mesmas regras. Tal como numa gramática, há um axioma ou símbolo inicial, de onde se parte para aplicar as regras. Um exemplo célebre é o do crescimento de algas formalizado por Lindenmayer[5] no sistema:

Variáveis: A e B
Constantes: nenhuma
Axioma: A
Regras: $A \to A\ B, B \to A$.

Quer dizer, em cada iteração do "crescimento"da alga, cada A deriva num par $A\ B$ e cada B converte-se num A. Assim, ter-se-á, onde n é o número de iterações desse processo:

- $n = 0$: A
- $n = 1$: $A\ B$
- $n = 2$: $A\ B\ A$
- $n = 3$: $A\ B\ A\ A\ B$
- etc

[4] Como alternativa a instalar Graphviz, podem usar WebGraphviz num browser.
[5] Ver `https://en.wikipedia.org/wiki/Aristid_Lindenmayer`.

Este L-System pode codificar-se em Haskell considerando cada variável um tipo, a que se adiciona um caso de paragem para poder expressar as sucessivas iterações:

$$\begin{array}{l}\textbf{type } \mathit{Algae} = A \\ \textbf{data } A = \mathrm{NA} \mid A\ A\ B \ \textbf{deriving } \mathit{Show} \\ \textbf{data } B = \mathrm{NB} \mid B\ A \ \textbf{deriving } \mathit{Show}\end{array}$$

Observa-se aqui já que A e B são mutuamente recursivos. Os isomorfismos in/out são definidos da forma habitual:

$$\begin{array}{l} \mathit{inA} :: 1 + A \times B \to A \\ \mathit{inA} = [\underline{\mathrm{NA}}, \widehat{A}] \\ \mathit{outA} :: A \to 1 + A \times B \\ \mathit{outA}\ \mathrm{NA} = i_1\ () \\ \mathit{outA}\ (A\ a\ b) = i_2\ (a, b) \\ \mathit{inB} :: 1 + A \to B \\ \mathit{inB} = [\underline{\mathrm{NB}}, B] \\ \mathit{outB} :: B \to 1 + A \\ \mathit{outB}\ \mathrm{NB} = i_1\ () \\ \mathit{outB}\ (B\ a) = i_2\ a \end{array}$$

O functor é, em ambos os casos, $\mathsf{F}\ X = 1 + X$. Contudo, os catamorfismos de A têm de ser estendidos com mais um gene, de forma a processar também os B,

$$\begin{array}{l} (\!|\cdot\ \cdot|\!)_A :: (1 + c \times d \to c) \to (1 + c \to d) \to A \to c \\ (\!|ga\ gb|\!)_A = ga \cdot (id + (\!|ga\ gb|\!)_A \times (\!|ga\ gb|\!)_B) \cdot \mathit{outA} \end{array}$$

e a mesma coisa para os Bs:

$$\begin{array}{l} (\!|\cdot\ \cdot|\!)_B :: (1 + c \times d \to c) \to (1 + c \to d) \to B \to d \\ (\!|ga\ gb|\!)_B = gb \cdot (id + (\!|ga\ gb|\!)_A) \cdot \mathit{outB} \end{array}$$

Pretende-se, neste problema:

1. A definição dos anamorfimos dos tipos A e B.
2. A definição da função

   $$\mathit{generateAlgae} :: \mathit{Int} \to \mathit{Algae}$$

   como anamorfismo de *Algae* e da função

   $$\mathit{showAlgae} :: \mathit{Algae} \to \mathit{String}$$

   como catamorfismo de *Algae*.
3. Use QuickCheck para verificar a seguinte propriedade:

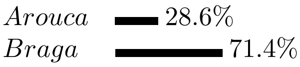
   $$\mathsf{length} \cdot \mathit{showAlgae} \cdot \mathit{generateAlgae} = \mathit{fib} \cdot \mathsf{succ}$$

Problema 5

O ponto de partida deste problema é um conjunto de equipas de futebol, por exemplo:

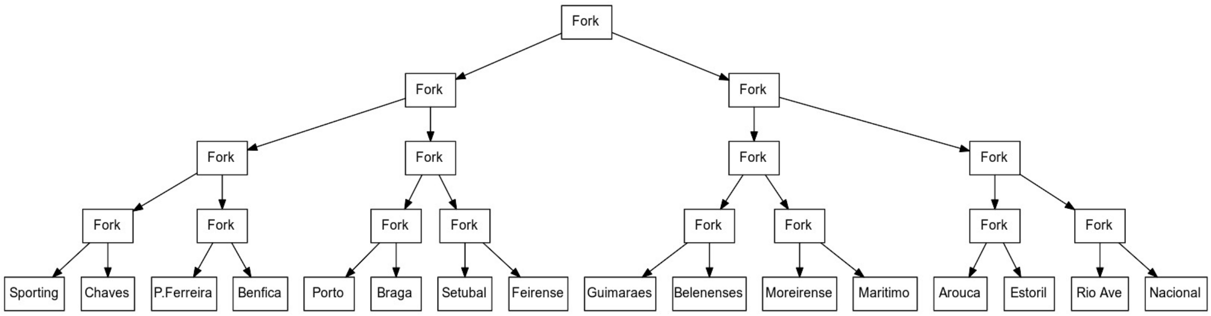
```
equipas :: [Equipa]
equipas = [
  "Arouca","Belenenses","Benfica","Braga","Chaves","Feirense",
  "Guimaraes","Maritimo","Moreirense","Nacional","P.Ferreira",
  "Porto","Rio Ave","Setubal","Sporting","Estoril"
  ]
```

Assume-se que há uma função $f\ (e_1, e_2)$ que dá — baseando-se em informação acumulada historicamente, e.g. estatística — qual a probabilidade de e_1 ou e_2 ganharem um jogo entre si.[6] Por exemplo, f (`"Arouca"`, `"Braga"`) poderá dar como resultado a distribuição



indicando que há 71.4% de probabilidades de `"Braga"` ganhar a `"Arouca"`.

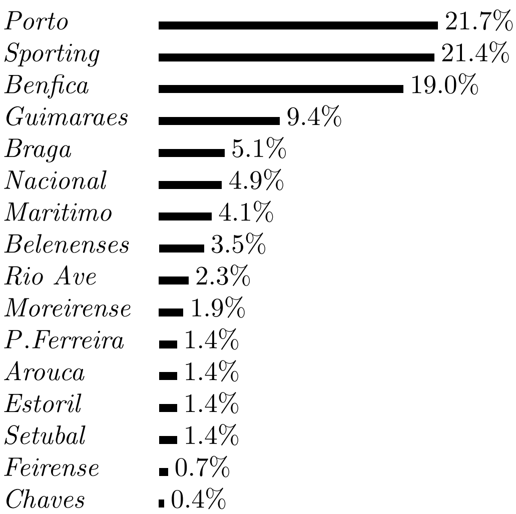
Para lidarmos com probabilidades vamos usar o mónade Dist a que vem descrito no apêndice A e que está implementado na biblioteca Probability [1] — ver definição (1) mais adiante. A primeira parte do problema consiste em sortear *aleatoriamente* os jogos das equipas. O resultado deverá ser uma LTree contendo, nas folhas, os jogos da primeira eliminatória e cujos nós indicam quem joga com quem (vencendo), à medida que a eliminatória prossegue:



A segunda parte do problema consiste em processar essa árvore usando a função

$$jogo :: (Equipa, Equipa) \rightarrow \mathsf{Dist}\ Equipa$$

que foi referida acima. Essa função simula um qualquer jogo, como foi acima dito, dando o resultado de forma probabilística. Por exemplo, para o sorteio acima e a função *jogo* que é dada neste enunciado[7], a probabilidade de cada equipa vir a ganhar a competição vem dada na distribuição seguinte:



Assumindo como dada e fixa a função *jogo* acima referida, juntando as duas partes obteremos um *hilomorfismo* de tipo $[Equipa] \rightarrow \mathsf{Dist}\ Equipa$,

$$\begin{array}{l} quem\_vence :: [Equipa] \rightarrow \mathsf{Dist}\ Equipa \\ quem\_vence = eliminatoria \cdot sorteio \end{array}$$

com características especiais: é aleatório no anamorfismo (sorteio) e probabilístico no catamorfismo (eliminatória).

[6] Tratando-se de jogos eliminatórios, não há lugar a empates.

[7] Pode, se desejar, criar a sua própria função *jogo*, mas para efeitos de avaliação terá que ser usada a que vem dada neste enunciado. Uma versão de *jogo* realista teria que ter em conta todas as estatísticas de jogos entre as equipas em jogo, etc etc.

O anamorfismo $sorteio :: [Equipa] \rightarrow \mathsf{LTree}\ Equipa$ tem a seguinte arquitectura, [8]

$$sorteio = anaLTree\ lsplit \cdot envia \cdot permuta$$

reutilizando o anamorfismo do algoritmo de "merge sort", da biblioteca LTree, para construir a árvore de jogos a partir de uma permutação aleatória das equipas gerada pela função genérica

$$permuta :: [a] \rightarrow \mathsf{IO}\ [a]$$

A presença do mónade de IO tem a ver com a geração de números aleatórios[9].

1. Defina a função monádica $permuta$ sabendo que tem já disponível

   $$getR :: [a] \rightarrow \mathsf{IO}\ (a, [a])$$

   $getR\ x$ dá como resultado um par (h, t) em que h é um elemento de x tirado à sorte e t é a lista sem esse elemento – mas esse par vem encapsulado dentro de IO.

2. A segunda parte do exercício consiste em definir a função monádica

   $$eliminatoria :: \mathsf{LTree}\ Equipa \rightarrow \mathsf{Dist}\ Equipa$$

   que, assumindo já disponível a função $jogo$ acima referida, dá como resultado a distribuição de equipas vencedoras do campeonato.

Sugestão: inspire-se na secção 4.10 (*'Monadification' of Haskell code made easy*) dos apontamentos [4].

Referências

[1] M. Erwig and S. Kollmansberger. Functional pearls: Probabilistic functional programming in Haskell. *J. Funct. Program.*, 16:21–34, January 2006.

[2] B.W. Kernighan and D.M. Richtie. *The C Programming Language*. Prentice Hall, Englewood Cliffs, N.J., 1978.

[3] D.E. Knuth. *Literate Programming*. CSLI Lecture Notes Number 27. Stanford University Center for the Study of Language and Information, Stanford, CA, USA, 1992.

[4] J.N. Oliveira. *Program Design by Calculation*, 2008. Draft of textbook in preparation. viii+297 pages. Informatics Department, University of Minho.

[8] A função $envia$ não é importante para o processo; apenas se destina a simplificar a arquitectura monádica da solução.

[9] Quem estiver interessado em detalhes deverá consultar System.Random.

Anexos

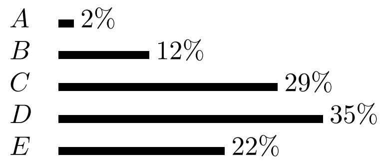
A Mónade para probabilidades e estatística

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

$$\textbf{newtype}\ \mathsf{Dist}\ a = D\ \{ unD :: [(a, ProbRep)] \} \tag{1}$$

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

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

que o GHCi mostrará assim:

```
'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.[10]

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 $g : A \to \mathsf{Dist}\ B$ e $f : B \to \mathsf{Dist}\ C$ 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 de programação monádica.

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

B Definições auxiliares

São dadas: a função que simula jogos entre equipas,

$$
\begin{array}{l}
\textbf{type}\ \mathit{Equipa} = \mathit{String}\\
\mathit{jogo} :: (\mathit{Equipa}, \mathit{Equipa}) \rightarrow \mathsf{Dist}\ \mathit{Equipa}\\
\mathit{jogo}\ (e_1, e_2) = D\ [(e_1, 1 - \mathit{r1}\ /\ (\mathit{r1} + \mathit{r2})), (e_2, 1 - \mathit{r2}\ /\ (\mathit{r1} + \mathit{r2}))]\ \textbf{where}\\
\quad \mathit{r1} = \mathit{rank}\ e_1\\
\quad \mathit{r2} = \mathit{rank}\ e_2\\
\quad \mathit{rank} = \mathit{pap}\ \mathit{ranks}\\
\quad \mathit{ranks} = [\\
\qquad (\texttt{"Arouca"}, 5),\\
\qquad (\texttt{"Belenenses"}, 3),\\
\qquad (\texttt{"Benfica"}, 1),\\
\qquad (\texttt{"Braga"}, 2),\\
\qquad (\texttt{"Chaves"}, 5),\\
\qquad (\texttt{"Feirense"}, 5),\\
\qquad (\texttt{"Guimaraes"}, 2),\\
\qquad (\texttt{"Maritimo"}, 3),\\
\qquad (\texttt{"Moreirense"}, 4),\\
\qquad (\texttt{"Nacional"}, 3),\\
\qquad (\texttt{"P.Ferreira"}, 3),\\
\qquad (\texttt{"Porto"}, 1),\\
\qquad (\texttt{"Rio Ave"}, 4),\\
\qquad (\texttt{"Setubal"}, 4),\\
\qquad (\texttt{"Sporting"}, 1),\\
\qquad (\texttt{"Estoril"}, 5)]
\end{array}
$$

a função (monádica) que parte uma lista numa cabeça e cauda *aleatórias*,

$$
\begin{array}{l}
\mathit{getR} :: [a] \rightarrow \mathsf{IO}\ (a, [a])\\
\mathit{getR}\ x = \textbf{do}\ \{\\
\quad i \leftarrow \mathit{getStdRandom}\ (\mathit{randomR}\ (0, \mathsf{length}\ \ x - 1));\\
\quad \mathit{return}\ (x\ !!\ i, \mathit{retira}\ i\ x)\\
\quad \}\ \textbf{where}\ \mathit{retira}\ i\ x = \mathit{take}\ i\ x + \!\! + \mathit{drop}\ (i + 1)\ x
\end{array}
$$

e algumas funções auxiliares de menor importância: uma que ordena listas com base num atributo (função que induz uma pré-ordem),

$$
\begin{array}{l}
\mathit{presort} :: (\mathit{Ord}\ a, \mathit{Ord}\ b) \Rightarrow (b \rightarrow a) \rightarrow [b] \rightarrow [b]\\
\mathit{presort}\ f = \mathsf{map}\ \ \pi_2 \cdot \mathit{sort} \cdot (\mathsf{map}\ \ (\mathit{fork}\ f\ \mathit{id}))
\end{array}
$$

e outra que converte “look-up tables"em funções (parciais):

$$
\begin{array}{l}
\mathit{pap} :: \mathit{Eq}\ a \Rightarrow [(a, t)] \rightarrow a \rightarrow t\\
\mathit{pap}\ m\ k = \mathit{unJust}\ (\mathit{lookup}\ k\ m)\ \textbf{where}\ \mathit{unJust}\ (\mathit{Just}\ a) = a
\end{array}
$$

C Soluções propostas

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

Problema 1

Código

$$
\mathit{inv}\ x = \pi_2 \cdot (\mathit{cataNat}\ \langle [\underline{(1 - x)}, \widehat{(*)} \cdot \langle \underline{(1 - x)}, \pi_1 \rangle], [\mathit{fromIntegral} \cdot \mathit{one}, \widehat{(+)}] \rangle)
$$

-- Testes com o QuickCheck

$$prop\_inv1 :: Float \rightarrow Integer \rightarrow Property$$
$$prop\_inv1\ x\ n = (x \geqslant 1) ==> (inv\ x\ 0 \equiv 1)$$
$$prop\_inv2 :: Float \rightarrow Integer \rightarrow Property$$
$$prop\_inv2\ x\ n = (x \geqslant 1) ==> (inv\ 1\ 10 \equiv inv\ 1\ 20)$$

Cálculos Auxiliares

```
aux_inv :: Float -> Integer -> Float
aux_inv x 0 = (1-x)
aux_inv x (n+1) = (1-x)^(n+1+1)

inv_p1 :: Float -> Integer -> Float
inv_p1 x 0 = 1
inv_p1 x (n+1) = aux_inv x n + inv_p x n
```

Aplicando as Leis

$$\left\{\begin{array}{l} inv\_p1\ x \cdot \mathbf{in} = [h1, h2] \cdot F < inv\_p1\ x, aux\_inv\ x> \\ aux\_inv\ x \cdot \mathbf{in} = [k1, k2] \cdot F < inv\_p1\ x, aux\_inv\ x> \end{array}\right.$$

$$= \quad \{ \text{ definição de F } \}$$

$$\left\{\begin{array}{l} inv\_p1\ x \cdot \mathbf{in} = [h1, h2] \cdot (id + inv\_p1\ x) \\ aux\_inv\ x \cdot \mathbf{in} = [k1, k2] \cdot (id + aux\_inv\ x) \end{array}\right.$$

$$= \quad \{ \text{ Absorção-+, def in } \}$$

$$\left\{\begin{array}{l} \left\{\begin{array}{l} inv\_p1\ x \cdot \underline{0} = h1 \\ inv\_p1\ x \cdot \mathsf{succ} = h2 \cdot (inv\_p1\ x) \end{array}\right. \\ \left\{\begin{array}{l} aux\_inv\ x \cdot \underline{0} = k1 \\ aux\_inv\ x \cdot \mathsf{succ} = k2 \cdot (aux\_inv\ x) \end{array}\right. \end{array}\right. .$$

$$= \quad \{ \text{ Igualdade Extensional } \}$$

$$\left\{\begin{array}{l} \left\{\begin{array}{l} inv\_p1\ x\ (\underline{0} \cdot x) = h1\ x \\ inv\_p1\ x\ (n+1) = h2\ (inv\_p1\ x)\ (n+1) \end{array}\right. \\ \left\{\begin{array}{l} aux\_inv\ x\ (\underline{0} \cdot x) = k1\ x \\ aux\_inv\ x\ (n+1) = k2\ (aux\ x)\ (n+1) \end{array}\right. \end{array}\right. .$$

$$= \quad \{ \text{ Natural-cons } \}$$

$$\left\{\begin{array}{l} \left\{\begin{array}{l} inv\_p1\ x\ \underline{0} = \underline{1} \\ inv\_p1\ x\ (n+1) = h2\ (inv\_p1\ x)\ (n+1) \end{array}\right. \\ \left\{\begin{array}{l} aux\_inv\ x\ \underline{0} = \underline{(1-x)} \\ aux\_inv\ x\ (n+1) = k2\ (aux\_inv\ x)\ (n+1) \end{array}\right. \end{array}\right. .$$

$$= \quad \{ \text{ Verifica-se que } \}$$

$$< inv\_p1x, aux\_invx >= (<[h1, h2], [k1, k2]>).$$

$$= \quad \{ \text{ ————— } \}$$

$$Onde : \left\{\begin{array}{l} \left\{\begin{array}{l} h1 = \underline{(1)} \\ h2 = \widehat{(*)}\ . < \underline{(1-x)}, \pi_1> \end{array}\right. \\ \left\{\begin{array}{l} k1 = \underline{(1-x)} \\ k2 = \widehat{(+)} \end{array}\right. \end{array}\right. .$$

Problema 2

Código

$$wc\_w\_final :: [Char] \rightarrow Integer$$
$$wc\_w\_final = wrapper \cdot worker$$

$sep\ c = (c \equiv \texttt{' '} \vee c \equiv \texttt{'\textbackslash n'} \vee c \equiv \texttt{'\textbackslash t'})$

-- gene é a condicional de McCarthy

$gene2 = cond\ (\widehat{(\wedge)} \cdot \langle \neg \cdot sep \cdot \pi_1, \pi_2 \cdot \pi_2 \rangle)\ ((1+) \cdot (\pi_1 \cdot \pi_2))\ (\pi_1 \cdot \pi_2)$

$wrapper = \pi_1$

$worker = (cataList\ [\langle zero, true \rangle, \langle gene2, sep \cdot \pi_1 \rangle])$

-- Testes com o QuickCheck

$prop\_wc1 :: [Char] \rightarrow Property$

$prop\_wc1\ l = (l \not\equiv [\,]) ==> ((wc\_w\_final\ \texttt{"as "}) \equiv 1)$

$prop\_wc2 :: [Char] \rightarrow Property$

$prop\_wc2\ l = (l \not\equiv [\,]) ==> ((wc\_w\_final\ [\,]) \equiv 0)$

$prop\_wc3 :: [Char] \rightarrow Property$

$prop\_wc3\ l = (l \not\equiv [\,]) ==> ((wc\_w\_final\ \texttt{" as b   "}) \equiv 2)$

Cálculos Auxiliares

Aplicando as Leis

$$
\begin{array}{ll}
 & \left\{ \begin{array}{l} f = wc \\ g = lookahead \end{array} \right. \\
= & \quad \{ \text{ Lei fokkinga } \} \\
 & \left\{ \begin{array}{l} wc \cdot \mathbf{in} = [h1, h2] \cdot F < f, g > \\ lookahead \cdot \mathbf{in} = [k1, k2] \cdot F < f, g > \end{array} \right. \\
= & \quad \{ \text{ definição de F } \} \\
 & \left\{ \begin{array}{l} wc \cdot \mathbf{in} = [h1, h2] \cdot (id + id\ x < wc, lookahead >) \\ lookahead \cdot \mathbf{in} = [k1, k2] \cdot (id + id\ x < wc, look >) \end{array} \right. \\
= & \quad \{ \text{ def in } \} \\
 & \left\{ \begin{array}{l} \left\{ \begin{array}{l} wc \cdot nil = h1 \\ wc \cdot cons = h2\ (id\ x < wc, lookahead >) \end{array} \right. \\ \left\{ \begin{array}{l} lookahead \cdot nil = k1 \\ lookahead \cdot cons = k2\ (id\ x < wc, lookahead >) \end{array} \right. \end{array} \right. . \\
= & \quad \{ \text{ def nil e cons } \} \\
 & \left\{ \begin{array}{l} \left\{ \begin{array}{l} wc\ [\,] = \underline{0} \\ wc\ (c : l) = h2\ (c, (wc\ t, lookahead\ t)) \end{array} \right. \\ \left\{ \begin{array}{l} lookahead\ [\,] = True \\ lookahead\ (c : t) = k2\ (c, (wc\ t, lookahead\ t)) \end{array} \right. \end{array} \right. . \\
= & \quad \{ \text{ Lei fokkinga } \} \\
 & < wc, lookahead >= (<[\underline{0}, gene2], [true, sep \cdot \pi_1]>) \\
= & \quad \{ \text{ Lei da Troca } \} \\
 & < wc, lookahead >= ([<\underline{0}, true>, <gene2, sep \cdot \pi_1>]) onde : \left\{ \begin{array}{l} \left\{ \begin{array}{l} h1 = \underline{0} \\ h2 = \widehat{(\wedge)}\ . < \neg \cdot sep \cdot \pi_1, \pi_2 \cdot \pi_2 > \end{array} \right. \\ \left\{ \begin{array}{l} k1 = True \\ k2 = sep \cdot \pi_1 \end{array} \right. \end{array} \right. .
\end{array}
$$

Problema 3

$inB\_tree :: () + (\textsf{B-tree}\ a, [(a, \textsf{B-tree}\ a)]) \rightarrow \textsf{B-tree}\ a$

$inB\_tree = [\underline{Nil}, \widehat{(Block)}]$

$$
\begin{array}{l}
\mathit{outB\_tree} :: \textsf{B-tree}\ a \to () + (\textsf{B-tree}\ a, [(a, \textsf{B-tree}\ a)])\\
\mathit{outB\_tree}\ \mathit{Nil} = i_1\ ()\\
\mathit{outB\_tree}\ (\mathit{Block}\ a\ ((b,c):t)) = i_2\ (a, ((b,c):t))
\end{array}
$$

$$
\begin{array}{l}
\mathit{recB\_tree}\ f = \mathit{id} + f \times (\textsf{map}\ \ (\mathit{id} \times f))\\
\mathit{baseB\_tree}\ f\ g = \mathit{id} + g \times (\textsf{map}\ \ (f \times g))\\
\mathit{cataB\_tree}\ g = g \cdot (\mathit{recB\_tree}\ (\mathit{cataB\_tree}\ g)) \cdot \mathit{outB\_tree}\\
\mathit{anaB\_tree}\ g = \mathit{inB\_tree} \cdot (\mathit{recB\_tree}\ (\mathit{anaB\_tree}\ g)) \cdot g\\
\mathit{hyloB\_tree}\ h\ g = \mathit{cataB\_tree}\ h \cdot \mathit{anaB\_tree}\ g
\end{array}
$$

$$
\begin{array}{l}
\textbf{instance}\ \mathit{Functor}\ \textsf{B-tree}\\
\quad \textbf{where}\ \textsf{fmap}\ f = \mathit{cataB\_tree}\ (\mathit{inB\_tree} \cdot \mathit{baseB\_tree}\ f\ \mathit{id})
\end{array}
$$

$$
\begin{array}{l}
\mathit{inordB\_tree} = \mathit{cataB\_tree}\ [\mathit{nil}, (\widehat{(+\!\!+)}) \cdot (\mathit{id} \times \mathit{concat} \cdot (\textsf{map}\ \ (\\
\quad \mathit{cons}))))]
\end{array}
$$

$$
\mathit{cataBTree}\ g = \mathit{cataB\_tree}\ g
$$

$$
\begin{tikzcd}
\textsf{B-tree}\ A \arrow[d, "cata\ g"'] & & 1 + \textsf{B-tree}\ A \times (A \times \textsf{B-tree}\ A)* \arrow[ll, "inB\_tree"'] \arrow[d, "id+(cata\ g)"] \\
D & & 1 + C \arrow[ll, "g"]
\end{tikzcd}
$$

-- largestBlock =
-- let b = (uncurry max) . (split (maximum.(map p2)) (length))
-- in cataBTree (either (const 0) (uncurry max.(id >< b)))

$$
\begin{tikzcd}
\textsf{B-tree}\ A \arrow[d, "cata\ g"'] & & 1 + \textsf{B-tree}\ A \times (A \times \textsf{B-tree}\ A)* \arrow[ll, "inB\_tree"'] \arrow[d, "id+(cata\ g)"] \\
\mathit{Int} & & 1 + \mathit{Int} \times (\textsf{B-tree}\ A \times \mathit{Int})* \arrow[ll, "g"]
\end{tikzcd}
$$

$$
\mathit{mirrorB\_tree} = \bot
$$

$$
\begin{array}{l}
\mathit{lsplitB\_tree} :: [\mathit{Int}] \to () + ([\mathit{Int}], [(\mathit{Int}, [\mathit{Int}])])\\
\mathit{lsplitB\_tree}\ [\,] = i_1\ ()\\
\mathit{lsplitB\_tree}\ x = i_2\ (\mathit{separaM}\ x)
\end{array}
$$

$$
\begin{array}{l}
\mathit{separaM} :: \mathit{Ord}\ t \Rightarrow [t] \to ([t], [(t, [t])])\\
\mathit{separaM}\ [x] = ([\,], [(x, [\,])])
\end{array}
$$

$$
\begin{array}{l}
\mathit{separaM}\ (x : y : t) = \textbf{let}\\
\quad \mathit{maior} = \mathit{max}\ x\ y\\
\quad \mathit{menor} = \mathit{min}\ x\ y\\
\quad \mathit{lista\_menores} = \mathit{filter}\ (\lambda n \to n < \mathit{menor})\ t\\
\quad e = \mathit{filter}\ (\lambda n \to n > \mathit{menor} \wedge n < \mathit{maior})\ t\\
\quad \mathit{lista\_maiores} = \mathit{filter}\ (\lambda n \to n > \mathit{maior})\ t\\
\quad \textbf{in}\ (\mathit{lista\_menores}, [(\mathit{menor}, e), (\mathit{maior}, \mathit{lista\_maiores})])
\end{array}
$$

$$
\mathit{gene} = [\mathit{nil}, (\widehat{(+\!\!+)}) \cdot (\mathit{id} \times \mathit{concat} \cdot (\textsf{map}\ \ (\mathit{cons}))))]
$$

$$
\mathit{qSortB\_tree} = (\mathit{hyloB\_tree}\ \mathit{gene}\ \mathit{lsplitB\_tree})
$$

$$
\mathit{dotB\_tree} = \bot
$$

$$
\mathit{cB\_tree2Exp} = \bot
$$

Problema 4

$$
\begin{array}{l}
[\![\cdot\ \cdot]\!]_A = \bot\\
[\![\cdot\ \cdot]\!]_B = \bot
\end{array}
$$

$$
\begin{array}{l}
\mathit{generateAlgae} = \bot\\
\mathit{showAlgae} = \bot
\end{array}
$$

Problema 5

Código

$$
\begin{array}{l}
permuta\ [\,] = return\ [\,] \\
permuta\ lista = \mathbf{do}\ \{(h,t) \leftarrow getR\ lista; \\
\quad resultado \leftarrow permuta\ t; \\
\quad return\ (h : resultado)\}
\end{array}
$$

Aplicando as Leis

$$
\begin{array}{ll}
& id = cata(in) \\
= & \quad \{\ \text{id = permuta}\ \} \\
& permuta = cata(in) \\
= & \quad \{\ \text{Universal-cata}\ \} \\
& permuta.in = in.(id + id >< f) \\
= & \quad \{\ \text{def in}\ \} \\
& permuta.[nil, cons] = [nil, cons](id + id >< f) \\
= & \quad \{\ \text{def Fusão-+}\ \} \\
& [permuta.nil, permuta.cons] = [nil.id, cons.(id + id >< f)]. \\
= & \quad \{\ \text{Universal-+}\ \} \\
& \left\{ \begin{array}{l} permuta \cdot nil = ([nil, cons] \cdot (id + id \times f)) \cdot i_1 \\ permuta \cdot cons = ([nil, cons] \cdot (id + id \times f)) \cdot i_2 \end{array} \right. \\
= & \quad \{\ \text{Absorção-+}\ \} \\
& \left\{ \begin{array}{l} permuta \cdot nil = ([nil, cons] \cdot (id + id \times f)) \cdot i_1 \\ permuta \cdot cons = ([nil, cons] \cdot (id + id \times f)) \cdot i_2 \end{array} \right. \\
= & \quad \{\ \text{igualdade extensional}\ \} \\
& \left\{ \begin{array}{l} permuta \cdot nil\ () = ([nil \cdot id, cons \cdot (id \times f)] \cdot i_1)\ () \\ permuta \cdot cons\ (h,t) = ([nil \cdot id, cons \cdot (id \times f)] \cdot i_2)\ (h,t) \end{array} \right. \\
= & \quad \{\ \text{Cancelamento-+}\ \} \\
& \left\{ \begin{array}{l} permuta\ (nil\ ()) = (nil \cdot id)\ () \\ permuta\ (cons\ (h,t)) = (cons \cdot (id \times f))\ (h,t) \end{array} \right. \\
= & \quad \{\ \text{Def comp}\ \} \\
& \{\ permuta\ (nil\ ()) = nil\ (id\ ())\ \ )\ (permuta\ (cons\ (h,t)) = (cons \cdot (id \times f))\ (h,t)) \\
= & \quad \{\ \text{Def id}\ \} \\
& \left\{ \begin{array}{l} permuta\ (nil\ ()) = nil\ (id\ ()) \\ permuta\ (cons\ (h,t)) = cons\ ((id \times f))\ (h,t) \end{array} \right. \\
= & \quad \{\ \text{Definição ao ponto}\ \} \\
& \left\{ \begin{array}{l} permuta\ (nil\ ()) = nil\ () \\ permuta\ (cons\ (h,t)) = cons\ (id\ h, (f\ \ t)) \end{array} \right. \\
= & \quad \{\ \text{Definição id}\ \} \\
& \left\{ \begin{array}{l} permuta\ (nil\ ()) = nil\ (id\ ()) \\ permuta\ (cons\ (h,t)) = (cons \cdot (id \times f))\ (h,t) \end{array} \right. \\
= & \quad \{\ \text{Definição ao ponto}\ \}
\end{array}
$$

$$\left\{\begin{array}{l} permuta\ (nil\ ()) = nil\ () \\ permuta\ (cons\ (h, t)) = cons\ (h, (f\ \ t)) \end{array}\right.$$

$$= \qquad \{\ \text{Def-cons, Def-nil}\ \}$$

$$\left\{\begin{array}{l} permuta\ [\,] = [\,] \\ permuta\ (h : t) = (h : (f\ \ t)) \end{array}\right.$$

Código

$$eliminatoria = \perp$$

$$\begin{tikzcd} \mathsf{LTree}\ Equipa \arrow[d, "el"'] & & Equipa + (\mathsf{LTree}\ Equipa \times \mathsf{LTree}\ Equipa) \arrow[ll, "inLTree"'] \arrow[d, "id+el \times el"] \\ Equipa & & Equipa + (Equipa \times Equipa) \arrow[ll, "g"] \end{tikzcd}$$

g = [id, jogo]

Índice