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

Departamento de Informática
Universidade do Minho

Junho de 2017

Grupo nr.	02
a77782	Mariana Miranda
a78377	Daniel Fernandes
a78633	Helena Poleri

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*.

$\mathbf{import}\ \mathit{Nat}$
$\mathbf{import}\ \mathit{Exp}$
$\mathbf{import}\ \mathsf{BTree}$
$\mathbf{import}\ \mathsf{LTree}$
$\mathbf{import}\ \mathit{St}$
$\mathbf{import}\ \mathit{Probability}\ \mathit{hiding}\ (\mathit{cond}, \mathit{choose})$
$\mathbf{import}\ \mathit{Data}.\mathit{List}$
$\mathbf{import}\ \mathit{Test}.\mathit{QuickCheck}\ \mathit{hiding}\ ((\times))$
$\mathbf{import}\ \mathit{System}.\mathit{Random}\ \mathit{hiding}\ \langle\cdot,\cdot\rangle$
$\mathbf{import}\ \mathit{GHC}.\mathsf{IO}.\mathit{Exception}$
$\mathbf{import}\ \mathit{System}.\mathsf{IO}.\mathit{Unsafe}$

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

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

[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:

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

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 $lookahead\_sep$.)

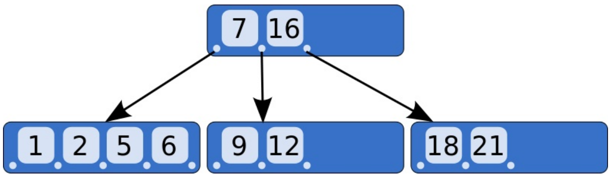
Problema 3

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

$$\mathbf{data}\ \mathsf{B\_tree}\ a = \mathit{Nil} \mid \mathit{Block}\ \{\mathit{leftmost} :: \mathsf{B\_tree}\ a, \mathit{block} :: [(a, \mathsf{B\_tree}\ a)]\}\ \mathbf{deriving}\ (\mathit{Show}, \mathit{Eq})$$

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 $inordB\_tree :: \mathsf{B\_tree}\ t \rightarrow [t]$ que faça travessias "inorder"de árvores deste tipo.
3. Definir como um catamorfismo a função $largestBlock :: \mathsf{B\_tree}\ a \rightarrow Int$ que detecta o tamanho do maior bloco da árvore argumento.
4. Definir como um anamorfismo a função $mirrorB\_tree :: \mathsf{B\_tree}\ a \rightarrow \mathsf{B\_tree}\ a$ 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:

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

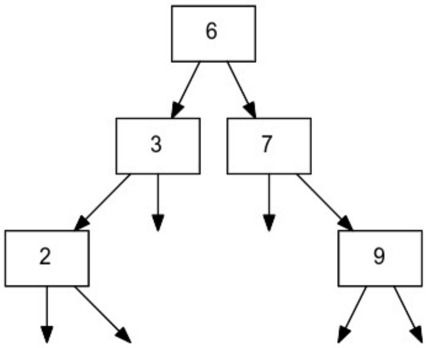
6. A biblioteca Exp permite representar árvores-expressão em formato DOT, que pode ser lido por aplicações como por exemplo Graphviz, produzindo as respectivas imagens. Por exemplo, para o caso de árvores BTree, se definirmos

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

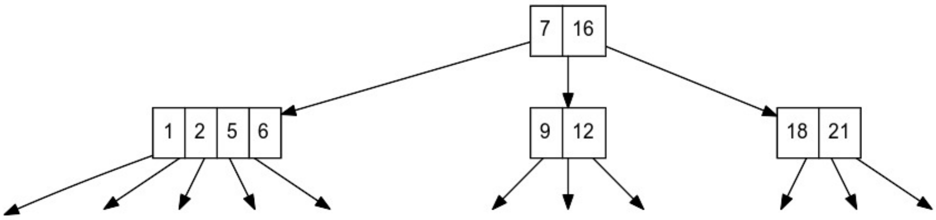
executando $dotBTree\ t$ para

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

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: $A \rightarrow A\ B, B \rightarrow A$.

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:

$$\mathbf{type}\ Algae = A$$
$$\mathbf{data}\ A = \mathrm{NA} \mid A\ A\ B\ \mathbf{deriving}\ Show$$
$$\mathbf{data}\ B = \mathrm{NB} \mid B\ A\ \mathbf{deriving}\ Show$$

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

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

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,

$$(\!|\cdot\ \cdot|\!)_A :: (1 + c \times d \to c) \to (1 + c \to d) \to A \to c$$
$$(\!|ga\ gb|\!)_A = ga \cdot (id + (\!|ga\ gb|\!)_A \times (\!|ga\ gb|\!)_B) \cdot outA$$

e a mesma coisa para os Bs:

$$(\!|\cdot\ \cdot|\!)_B :: (1 + c \times d \to c) \to (1 + c \to d) \to B \to d$$
$$(\!|ga\ gb|\!)_B = gb \cdot (id + (\!|ga\ gb|\!)_A) \cdot outB$$

Pretende-se, neste problema:

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

   $$generateAlgae :: Int \to Algae$$

   como anamorfismo de $Algae$ e da função

   $$showAlgae :: Algae \to String$$

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

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

Problema 5

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

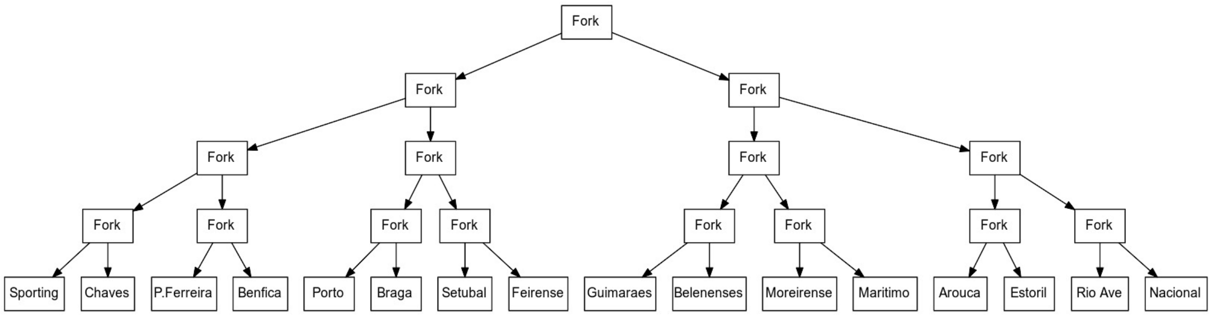
```
equipas :: [Equipa]
equipas = [
  "Arouca","Belenenses","Benfica","Braga","Chaves","Feirense",
  "Guimaraes","Maritimo","Moreirense","Nacional","P.Ferreira",
  "Porto","Rio Ave","Setubal","Sporting","Estoril"
  ]
```

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

Arouca	28.6%
Braga	71.4%

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:

Porto	21.7%
Sporting	21.4%
Benfica	19.0%
Guimaraes	9.4%
Braga	5.1%
Nacional	4.9%
Maritimo	4.1%
Belenenses	3.5%
Rio Ave	2.3%
Moreirense	1.9%
P.Ferreira	1.4%
Arouca	1.4%
Estoril	1.4%
Setubal	1.4%
Feirense	0.7%
Chaves	0.4%

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

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

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

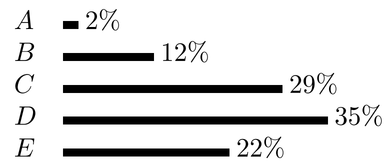
A Mónade para probabilidades e estatística

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.

$$d3 = normal\ [10\,..\,20]$$

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 $g : A \rightarrow \mathsf{Dist}\ B$ e $f : B \rightarrow \mathsf{Dist}\ C$ 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

O primeiro passo na resolução deste problema foi seguir a sugestão dada no enunciado, e inspirarmo-nos no problema semelhante relativo à função ns da secção 3.16 dos apontamentos [4].

Assim, este consistiu em escrever a função $\mathit{inv}\ x$ de forma *pointwise* em Haskell:

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

$$= \quad \{ \textit{ Propriedades matemáticas elementares } \}$$

$$\left\{ \begin{array}{l} (inv\ x)\ 0 = \sum_{i=0}^{0} (1-x)^i \\ (inv\ x)\ (n+1) = \sum_{i=0}^{n+1} (1-x)^i \end{array} \right.$$

$$= \quad \{ \textit{ Propriedades matemáticas elementares } \}$$

$$\left\{ \begin{array}{l} (inv\ x)\ 0 = (1-x)^0 \\ (inv\ x)\ (n+1) = (1-x)^{n+1} + \sum_{i=0}^{n} (1-x)^i \end{array} \right.$$

$$= \quad \{ \textit{ Propriedades matemáticas elementares}; (inv\ x)\ n = \sum_{i=0}^{n} (1-x)^i \ \}$$

$$\left\{ \begin{array}{l} (inv\ x)\ 0 = 1 \\ (inv\ x)\ (n+1) = (1-x)^{n+1} + inv\ x\ n \end{array} \right.$$

Depois disto e por causa da semelhança com o problema anteriormente mencionado, o próximo passo era introduzir uma outra função $elev\ x$, esperando que depois se pudesse aplicar a lei de Fokkinga a estas duas.

$$\left\{ \begin{array}{l} (elev\ x)\ 0 = (1-x) \\ (elev\ x)\ (n+1) = (1-x) * elev\ x\ n \end{array} \right.$$

Desta forma, temos que:

$$\left\{ \begin{array}{l} (inv\ x)\ 0 = 1 \\ (inv\ x)\ (n+1) = (1-x)^{n+1} + inv\ x\ n \end{array} \right.$$

$$= \quad \{ \ elev\ x\ n = (1-x)^{n+1} \ \}$$

$$\left\{ \begin{array}{l} (inv\ x)\ 0 = 1 \\ (inv\ x)\ (n+1) = elev\ x\ n + inv\ x\ n \end{array} \right.$$

$$= \quad \{ \ (74) \times 3; (73); (78); (76) \times 2; \mathsf{succ}\ n = n+1; \widehat{(+)}\ (x,y) = x+y \ \}$$

$$\left\{ \begin{array}{l} (inv\ x) \cdot \underline{0} = \underline{1} \\ (inv\ x) \cdot \mathsf{succ} = \widehat{(+)} \cdot \langle elev\ x, inv\ x \rangle \end{array} \right.$$

$$= \quad \{ \ (27) \ \}$$

$$[(inv\ x) \cdot \underline{0}, (inv\ x) \cdot \mathsf{succ}] = [\underline{1}, \widehat{(+)} \cdot \langle elev\ x, inv\ x \rangle]$$

$$= \quad \{ \ (20); (22); (1) \ \}$$

$$(inv\ x) \cdot [\underline{0}, \mathsf{succ}] = [\underline{1}, \widehat{(+)}] \cdot (id + \langle elev\ x, inv\ x \rangle)$$

E também que:

$$\left\{ \begin{array}{l} (elev\ x)\ 0 = (1-x) \\ (elev\ x)\ (n+1) = (1-x) * elev\ x\ n \end{array} \right.$$

$$= \quad \{ \ (74) \times 3; (73); (78); (76) \times 2; elev\ x = \pi_1 \cdot \langle elev\ x, inv\ x \rangle\ (7) \ \}$$

$$\left\{ \begin{array}{l} (elev\ x) \cdot \underline{0} = \underline{(1-x)} \\ (elev\ x) \cdot \mathsf{succ} = ((1-x)*) \cdot \pi_1 \cdot \langle elev\ x, inv\ x \rangle \end{array} \right.$$

$$= \quad \{ \ (27) \ \}$$

$$[(elev\ x) \cdot \underline{0}, (elev\ x) \cdot \mathsf{succ}] = [\underline{(1-x)}, ((1-x)*) \cdot \pi_1 \cdot \langle elev\ x, inv\ x \rangle]$$

$$= \quad \{ \ (20); (22); (1) \ \}$$

$$(elev\ x) \cdot [\underline{0}, \mathsf{succ}] = [\underline{(1-x)}, ((1-x)*) \cdot \pi_1] \cdot (id + \langle elev\ x, inv\ x \rangle)$$

Destas duas, pudemos então proceder a aplicar a lei de Fokkinga:

$$\begin{cases} (elev\ x) \cdot [\underline{0}, \mathsf{succ}] = [\underline{(1-x)}, ((1-x)*) \cdot \pi_1] \cdot (id + \langle elev\ x, inv\ x \rangle) \\ (inv\ x) \cdot [\underline{0}, \mathsf{succ}] = [\underline{1}, \widehat{(+)}] \cdot (id + \langle elev\ x, inv\ x \rangle) \end{cases}$$

$$= \quad \{ \ (50) \ \}$$

$$\langle elev\ x, inv\ x \rangle = (\!|\langle [\underline{(1-x)}, ((1-x)*) \cdot \pi_1], [\underline{1}, \widehat{(+)}] \rangle|\!)$$

$$= \quad \{ \ (28) \ \}$$

$$\langle elev\ x, inv\ x \rangle = (\!|[\langle \underline{(1-x)}, \underline{1} \rangle, \langle ((1-x)*) \cdot \pi_1, \widehat{(+)} \rangle]|\!)$$

$$= \quad \{ \ \underline{(a,b)} = \langle \underline{a}, \underline{b} \rangle \ \}$$

$$\langle elev\ x, inv\ x \rangle = (\!|[\underline{(1-x, 1)}, \langle ((1-x)*) \cdot \pi_1, \widehat{(+)} \rangle]|\!)$$

Donde podemos retirar $inv\ x$ aplicando π_2 a $\langle elev\ x, inv\ x \rangle$. Assim, ficamos com:

$$inv\ x = \pi_2 \cdot (\!|[\underline{(1-x, 1)}, \langle ((1-x)*) \cdot \pi_1, \widehat{(+)} \rangle]|\!)$$

Esta função pode ser testada usando os testes *quickCheck* abaixo. A propriedade baseia-se no facto de que o inverso do inverso de um qualquer número corresponde a esse número.

$$\begin{array}{l} prop\_inv\ e = forAll\ betweenOneAndTwo\ \$\ \lambda x \rightarrow \\ \quad forAll\ (largeN\ e)\ \$\ \lambda y \rightarrow inv\ (inv\ x\ y)\ y - x \leqslant e \\ betweenOneAndTwo :: Gen\ Float \\ betweenOneAndTwo = choose\ (1, 2) \\ largeN :: Float \rightarrow Gen\ Int \\ largeN\ e = \mathbf{if}\ (e \geqslant 1)\ \mathbf{then}\ choose\ (round\ (1000 * e), round\ (2000 * e)) \\ \quad \mathbf{else}\ choose\ (round\ (1\ /\ e) * 1000, round\ (1\ /\ e) * 2000) \end{array}$$

Problema 2

$$look \cdot \mathbf{in} = h.F \langle look, wc\_w \rangle$$

$$= \quad \{ \ \mathbf{in} = [nil, cons]; Ff = id + id \times f; h = [h1, h2] \ \}$$

$$look \cdot [nil, cons] = [h1, h2] \cdot (id + id \times \langle look, wc\_w \rangle)$$

$$= \quad \{ \ (20); (22); (1) \ \}$$

$$[look \cdot nil, look \cdot cons] = [h1, h2 \cdot (id \times \langle look, wc\_w \rangle)]$$

$$= \quad \{ \ (27) \ \}$$

$$\begin{cases} look \cdot nil = h1 \\ look \cdot cons = h2 \cdot (id \times \langle look, wc\_w \rangle) \end{cases}$$

$$= \quad \{ \ (73) * 2; (74) * 3; nil\ \_ = [\,]; cons\ (h, t) = h : t \ \}$$

$$\begin{cases} look\ [\,] = h1\ x \\ look\ (h : t) = h2\ ((id \times \langle look, wc\_w \rangle)\ (h, t)) \end{cases}$$

$$= \quad \{ \ \text{definição look}\ ; (79); (78); (75) \ \}$$

$$\begin{cases} h1\ x = True \\ h2\ (h, (look\ t, wc\_w\ t)) = sep\ h \end{cases}$$

$$= \quad \{ \ \text{definição sep}\ ; (76); (73) \times 2; (81) \ \}$$

$$\begin{cases} h1 = \underline{True} \\ h2 = sep \cdot \pi_1 \end{cases}$$

$$
\begin{array}{ll}
 & wc\_w \cdot \mathbf{in} = k.F\langle look, wc\_w\rangle \\
= & \quad \{\ \mathbf{in} = [nil, cons]; Ff = id + id \times f; k = [k1, k2]\ \} \\
 & wc\_w \cdot [nil, cons] = [k1, k2] \cdot (id + id \times \langle look, wc\_w\rangle) \\
= & \quad \{\ (20), (22), (1), (27)\ \} \\
 & \left\{ \begin{array}{l} wc\_w \cdot nil = k1 \\ wc\_w \cdot cons = k2 \cdot (id \times \langle look, wc\_w\rangle) \end{array} \right. \\
= & \quad \{\ (73) * 2; (74) * 3; nil\_ = [\,]; cons\ (h, t) = h : t; (75); (78); (79)\ \} \\
 & \left\{ \begin{array}{l} wc\_w\ [\,] = k1\ x \\ wc\_w\ (h : t) = k2\ (h, (look\ t, wc\_w\ t)) \end{array} \right. \\
= & \quad \{\ \text{definição wc\_w ; (76); (73)}\ \} \\
 & \left\{ \begin{array}{l} k1 = \underline{0} \\ \mathbf{if}\ (\neg sep\ c \wedge look\ t)\ \mathbf{then}\ wc\_w + 1\ \mathbf{else}\ wc\_w = k2\ (h, (look\ t, wc\_w\ t)) \end{array} \right. \\
= & \quad \{\ (80); \widehat{f}\ (x, y) = f\ x\ y; (81) \times 4; (74) \times 5; (73)\ \} \\
 & \left\{ \begin{array}{l} k1 = \underline{0} \\ k2 = \widehat{(\&\&)} \cdot (not \cdot sep \times \pi_1) \rightarrow \mathsf{succ} \cdot \pi_2 \cdot \pi_2, \pi_2 \cdot \pi_2 \end{array} \right. \\
= & \quad \{\ (30)\ \} \\
 & \left\{ \begin{array}{l} k1 = \underline{0} \\ k2 = [\mathsf{succ} \cdot \pi_2 \cdot \pi_2, \pi_2 \cdot \pi_2] \cdot (\widehat{(\&\&)} \cdot (not \cdot sep \times \pi_1))? \end{array} \right.
\end{array}
$$

Donde retiramos que $h = [\underline{True}, sep \cdot \pi_1]$ e $k = [\underline{0}, [\mathsf{succ} \cdot \pi_2 \cdot \pi_2, \pi_2 \cdot \pi_2] \cdot (\widehat{(\&\&)} \cdot (\neg \cdot sep \times \pi_1))?]$
Pela lei de Fokkinga, temos então:

$$
\begin{array}{ll}
 & \left\{ \begin{array}{l} look \cdot \mathbf{in} = h \cdot F\langle look, wc\_w\rangle \\ wc\_w \cdot \mathbf{in} = k \cdot F\langle look, wc\_w\rangle \end{array} \right. \\
= & \quad \{\ \text{(50) ; definição h ; definição k}\ \} \\
 & \langle look, wc\_w\rangle = (\!|\langle[\underline{True}, sep \cdot \pi_1], [\underline{0}, [\mathsf{succ} \cdot \pi_2 \cdot \pi_2, \pi_2 \cdot \pi_2] \cdot (\widehat{(\&\&)} \cdot (\neg \cdot sep \times \pi_1))?]\rangle|\!) \\
= & \quad \{\ (28)\ \} \\
 & \langle look, wc\_w\rangle = (\!|[\langle\underline{True}, \underline{0}\rangle, \langle sep \cdot \pi_1, [\mathsf{succ} \cdot \pi_2 \cdot \pi_2, \pi_2 \cdot \pi_2] \cdot (\widehat{(\&\&)} \cdot (\neg \cdot sep \times \pi_1))?\rangle]|\!) \\
= & \quad \{\ \underline{(a, b)} = \langle\underline{a}, \underline{b}\rangle\ \} \\
 & \langle look, wc\_w\rangle = (\!|[\underline{(True, 0)}, \langle sep \cdot \pi_1, [\mathsf{succ} \cdot \pi_2 \cdot \pi_2, \pi_2 \cdot \pi_2] \cdot (\widehat{(\&\&)} \cdot (\neg \cdot sep \times \pi_1))?\rangle]|\!)
\end{array}
$$

Código:

$$
\begin{array}{l}
wc\_w\_final :: [Char] \rightarrow Int \\
wc\_w\_final = wrapper \cdot worker \\
wrapper = \pi_2 \\
worker = (\!|[worker\_i, worker\_b]|\!) \\
worker\_i = \underline{(True, 0)} \\
worker\_b = \langle sep \cdot \pi_1, cond\ wc\_cond\ (\mathsf{succ} \cdot \pi_2 \cdot \pi_2)\ (\pi_2 \cdot \pi_2)\rangle \\
wc\_cond = \widehat{(\wedge)} \cdot (\neg \cdot sep \times \pi_1) \\
sep\ c = (c \equiv \texttt{' '} \vee c \equiv \texttt{'\textbackslash n'} \vee c \equiv \texttt{'\textbackslash t'})
\end{array}
$$

Esta função pode ser testada usando os testes *quickCheck* abaixo. A propriedade baseia-se no facto de que a função criada deve dar o mesmo resultado que a função wc_w dada.

$$prop\_wc\ words = wc\_w\_final\ words \equiv wc\_w\ words$$

Problema 3

1. Biblioteca para o tipo B_tree:

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

Prova para o outB_tree:

$$\begin{aligned}
& outB_tree \cdot inB_tree = id \\
= \quad & \{ \; inB_tree = [\underline{Nil}, \widehat{Block}]; (19) \; \} \\
& outB_tree \cdot [\underline{Nil}, \widehat{Block}] = [i_1, i_2] \\
= \quad & \{ \; (20); (27) \; \} \\
& \begin{cases} outB_tree \cdot \underline{Nil} = i_1 \\ outB_tree \cdot \widehat{Block} = i_2 \end{cases} \\
= \quad & \{ \; (73); (76) \; \} \\
& \begin{cases} outB_tree \; Nil = i_1 \\ outB_tree \; (Block \; \{ leftmost = a, block = l \}) = i_2 \; (a, l) \end{cases}
\end{aligned}$$

$$\begin{aligned}
& outB_tree \; Nil = i_1 \; () \\
& outB_tree \; (Block \; \{ leftmost = a, block = l \}) = i_2 \; (a, l)
\end{aligned}$$

$$\begin{tikzcd}
\mathsf{B_tree} \; A \arrow[rr, bend left, "outB_tree"] \arrow[r, phantom, "\cong"] & {} & 1 + \mathsf{B_tree} \; A \times (A \times \mathsf{B_tree} \; A)^* \arrow[ll, bend left, "inB_tree"]
\end{tikzcd}$$

Figura 1: Diagrama do isomorfismo *inB_tree* / *outB_tree*

$$\begin{aligned}
& recB_tree \; f = baseB_tree \; id \; f \\
& baseB_tree \; g \; f = id + (f \times (\mathsf{map} \;\; (g \times f))) \\
& (\!| g |\!) = g \cdot recB_tree \; (\!| g |\!) \cdot outB_tree \\
& [\!(g)\!] = inB_tree \cdot (recB_tree \; [\!(g)\!]) \cdot g \\
& [\![f, g]\!] = (\!| f |\!) \cdot [\!(g)\!] \\
& \textbf{instance } Functor \; \mathsf{B_tree} \\
& \quad \textbf{where } \mathsf{fmap} \; f = (\!| inB_tree \cdot baseB_tree \; f \;\; id |\!)
\end{aligned}$$

2. Travessia in-order:

$$\begin{aligned}
& inordB_tree = (\!| inordB_t |\!) \\
& inordB_t = [nil, \mathsf{conc}] \cdot (id + id \times (concat \cdot (\mathsf{map} \;\; cons)))
\end{aligned}$$

$$\begin{tikzcd}
\mathsf{B_tree} \; A \arrow[r, "outB_tree"'] \arrow[d, "inordB_tree = (\!| g |\!)"'] & 1 + \mathsf{B_tree} \; A \times (A \times \mathsf{B_tree} \; A)^* \arrow[d, "id + in_ord \times \mathsf{map} \;\; (id \times in_ord)"] \\
A^* & 1 + A^* \times (A \times A^*)^* \arrow[l, "g = inordB_t"'] \arrow[d, "id + id \times \mathsf{map} \;\; cons"] \\
 & 1 + A^* \times (A^*)^* \arrow[d, "id + id \times concat"] \\
 & 1 + A^* \times A^* \arrow[uul, "{[nil, \mathsf{conc}]}"]
\end{tikzcd}$$

Figura 2: Diagrama de *in_ord*

3. Tamanho do maior bloco da árvore argumento:

$$
\begin{aligned}
&largestBlock = (\!|[zero, maior \cdot (id \times maior)] \cdot (id + id \times (\langle length', maximum\rangle \cdot \mathsf{map}\ \ \pi_2))|\!)\\
&length' = toInteger \cdot \mathsf{length}\\
&maior = \widehat{max}
\end{aligned}
$$

$$
\begin{tikzcd}
\mathsf{B\_tree}\ A \arrow[r, "outB\_tree"'] \arrow[d, "largestBlock = (\!|g|\!)"'] & 1 + \mathsf{B\_tree}\ A \times (A \times \mathsf{B\_tree}\ A)^* \arrow[d, "id + largestBlock \times \mathsf{map}\ \ (id \times largestBlock)"] \\
\mathbb{N}_0 & 1 + \mathbb{N}_0 \times (A \times \mathbb{N}_0)^* \arrow[l, "g"] \arrow[d, "id + id \times \mathsf{map}\ \ \pi_2"] \\
 & 1 + \mathbb{N}_0 \times \mathbb{N}_0^* \arrow[d, "id + id \times \langle length', maximum\rangle"] \\
 & 1 + \mathbb{N}_0 \times (\mathbb{N}_0 \times \mathbb{N}_0) \arrow[uuul, "{[\underline{0}, maior \cdot (id \times maior)]}"]
\end{tikzcd}
$$

Figura 3: Diagrama de $largestBlock$

4. Rodar a árvore 180º:

$$
\begin{aligned}
&mirrorB\_tree = [\!(f1 \cdot f2 \cdot f3 \cdot outB\_tree)\!]\\
&f1 = id + id \times (reverse \cdot \widehat{zip})\\
&f2 = id + \langle last \cdot \pi_2 \cdot \pi_2, \langle \pi_1 \cdot \pi_2, cons \cdot (id \times \pi_2)\rangle\rangle\\
&f3 = id + id \times unzip
\end{aligned}
$$

$$
\begin{tikzcd}
\mathsf{B\_tree}\ A & 1 + \mathsf{B\_tree}\ A \times (A \times \mathsf{B\_tree}\ A)^* \arrow[l, "inB\_tree"'] \\
\mathsf{B\_tree}\ A \arrow[u, "mirror = [\!(g)\!]"'] \arrow[r, "g"] \arrow[d, "outB\_tree"'] & 1 + \mathsf{B\_tree}\ A \times (A \times \mathsf{B\_tree}\ A)^* \arrow[u, "id + mirror \times \mathsf{map}\ \ (id \times mirror)"] \\
1 + \mathsf{B\_tree}\ A \times (A \times \mathsf{B\_tree}\ A)^* \arrow[d, "f3"] & \\
1 + \mathsf{B\_tree}\ A \times (A^* \times (\mathsf{B\_tree}\ A)^*) \arrow[d, "f2"] & \\
1 + \mathsf{B\_tree}\ A \times (A^* \times (\mathsf{B\_tree}\ A)^*) \arrow[uuur, "f1"] &
\end{tikzcd}
$$

Figura 4: Diagrama de $mirrorB\_tree$

5. Quick-sort:

$$
\begin{aligned}
&qSortB\_tree :: Ord\ a \Rightarrow [\,a\,] \to [\,a\,]\\
&qSortB\_tree = [\![inordB\_t, lsplitB\_tree]\!]\\
&lsplitB\_tree\ [\,] \ \ = i_1\ ()\\
&lsplitB\_tree\ [\,a\,] = i_2\ ([\,], [\,(a, [\,])\,])\\
&lsplitB\_tree\ (p : s : t)\\
&\quad \mid (p < s) \qquad = i_2\ (a, [\,(p, b), (s, c)\,])\\
&\quad \mid otherwise \ \ = i_2\ (a, [\,(s, b), (p, c)\,])\\
&\qquad \mathbf{where}\ (a, b, c) = part3\ (<p)\ (<s)\ t
\end{aligned}
$$

$$
\begin{array}{l}
\mathit{part3} :: (a \rightarrow \mathit{Bool}) \rightarrow (a \rightarrow \mathit{Bool}) \rightarrow [\,a\,] \rightarrow ([\,a\,],[\,a\,],[\,a\,]) \\
\mathit{part3}\ \_\ \_\ [\,] \quad = ([\,],[\,],[\,]) \\
\mathit{part3}\ \mathit{v1}\ \mathit{v2}\ (h:t) \\
\quad \mid \mathit{v1}\ h \qquad\quad = \mathbf{let}\ (a,b,c) = \mathit{part3}\ \mathit{v1}\ \mathit{v2}\ t\ \mathbf{in}\ (h:a,b,c) \\
\quad \mid \mathit{v2}\ h \qquad\quad = \mathbf{let}\ (a,b,c) = \mathit{part3}\ \mathit{v1}\ \mathit{v2}\ t\ \mathbf{in}\ (a,h:b,c) \\
\quad \mid \mathit{otherwise} = \mathbf{let}\ (a,b,c) = \mathit{part3}\ \mathit{v1}\ \mathit{v2}\ t\ \mathbf{in}\ (a,b,h:c)
\end{array}
$$

$$
\begin{tikzcd}
A^* \arrow[r, "g = \mathit{lsplitB\_tree}"'] \arrow[d, "[\![g]\!]"] \arrow[dd, bend right, "\mathit{qSortB\_tree}"'] & 1 + A^* \times (A \times A^*)^* \arrow[d, "\mathit{id} + [\![g]\!] \times \mathsf{map}\ (\mathit{id} \times [\![g]\!])"] \\
\mathsf{B\_tree}\ A \arrow[r, bend left, "\mathit{outB\_tree}"] \arrow[r, phantom, "\cong"] \arrow[d, "(\!| f |\!)"] & 1 + \mathsf{B\_tree}\ A \times (A \times \mathsf{B\_tree}\ A)^* \arrow[l, bend left, "\mathit{inB\_tree}"] \arrow[d, "\mathit{id} + (\!| f |\!) \times \mathsf{map}\ (\mathit{id} \times (\!| f |\!))"] \\
A^* & 1 + A^* \times (A \times A^*)^* \arrow[l, "f = \mathit{inordB\_t}"]
\end{tikzcd}
$$

Figura 5: Diagrama de *qSortB_tree*

6. Representação de uma B_tree:

$$
\begin{array}{l}
\mathit{dotB\_tree} :: \mathit{Show}\ a \Rightarrow \mathsf{B\_tree}\ a \rightarrow \mathsf{IO}\ \mathit{ExitCode} \\
\mathit{dotB\_tree} = \mathit{dotpict} \cdot \mathit{bmap}\ \mathit{nothing}\ (\mathit{Just} \cdot \mathit{intercalate}\ \texttt{"|"} \cdot \mathsf{map}\ \ \mathit{show}) \cdot \mathit{cB\_tree2Exp} \\
\mathit{cB\_tree2Exp} :: \mathsf{B\_tree}\ a \rightarrow \mathit{Exp}\ [\,\mathit{Char}\,]\ [\,a\,] \\
\mathit{cB\_tree2Exp} = (\!| [\underline{(\mathit{Var}\ \texttt{"nil"})}, h] |\!) \\
\quad \mathbf{where}\ h = \widehat{\mathit{Term}} \cdot \langle \pi_1 \cdot \pi_2, \mathit{cons} \cdot (\mathit{id} \times \pi_2) \rangle \cdot (\mathit{id} \times \mathit{unzip})
\end{array}
$$

$$
\begin{tikzcd}
\mathsf{B\_tree}\ A \arrow[r, "\mathit{out}"'] \arrow[d, "\mathit{cB\_tree2Exp} = (\!| g |\!)"'] & 1 + \mathsf{B\_tree}\ A \times (A \times \mathsf{B\_tree}\ A) \arrow[d, "\mathit{id} + \mathit{cB\_tree2Exp} \times \mathsf{map}\ (\mathit{id} \times \mathit{cB\_tree2Exp})"] \\
\mathit{Exp}\ A & 1 + \mathit{Exp}\ A \times (A \times \mathit{Exp}\ A)^* \arrow[l, "g"] \arrow[d, "\mathit{id} + \mathit{id} \times \mathit{unzip}"] \\
 & 1 + \mathit{Exp}\ A \times (A^* \times (\mathit{Exp}\ A)^*) \arrow[d, "\mathit{id} + \langle \pi_1 \cdot \pi_2, \mathit{cons} \cdot (\mathit{id} \times \pi_2) \rangle"] \\
 & 1 + (A^* \times (\mathit{Exp}\ A)^*) \arrow[uul, "{[\underline{(\mathit{Var}\ \texttt{"nil"})}, \widehat{\mathit{Term}}]}"]
\end{tikzcd}
$$

Figura 6: Diagrama de *cB_tree2Exp*

Problema 4

1. Definição dos anamorfismos dos tipos A e B:

$$
\begin{array}{l}
[\![\cdot\ \cdot]\!]_A :: (c \rightarrow 1 + c \times d) \rightarrow (d \rightarrow 1 + c) \rightarrow c \rightarrow A \\
[\![\mathit{ga}\ \mathit{gb}]\!]_A = \mathit{inA} \cdot (\mathit{id} + [\![\mathit{ga}\ \mathit{gb}]\!]_A \times [\![\mathit{ga}\ \mathit{gb}]\!]_B) \cdot \mathit{ga} \\
\\
[\![\cdot\ \cdot]\!]_B :: (c \rightarrow 1 + c \times d) \rightarrow (d \rightarrow 1 + c) \rightarrow d \rightarrow B \\
[\![\mathit{ga}\ \mathit{gb}]\!]_B = \mathit{inB} \cdot (\mathit{id} + [\![\mathit{ga}\ \mathit{gb}]\!]_A) \cdot \mathit{gb}
\end{array}
$$

$$
\begin{tikzcd}
A & 1 + A \times B \arrow[l, "\mathit{inA}"'] \\
c \arrow[u, "{[\![\mathit{ga}\ \mathit{gb}]\!]_A}"'] \arrow[r, "\mathit{ga}"] & 1 + c \times d \arrow[u, "{\mathit{id} + [\![\mathit{ga}\ \mathit{gb}]\!]_A \times [\![\mathit{ga}\ \mathit{gb}]\!]_B}"']
\end{tikzcd}
$$

Figura 7: Anamorfismo do tipo A

$$
\begin{tikzcd}
B & 1 + A \arrow[l, "\mathit{inB}"'] \\
d \arrow[u, "{[\![\mathit{ga}\ \mathit{gb}]\!]_B}"'] \arrow[r, "\mathit{gb}"] & 1 + c \arrow[u, "{\mathit{id} + [\![\mathit{ga}\ \mathit{gb}]\!]_A}"']
\end{tikzcd}
$$

Figura 8: Anamorfismo do tipo B

2. Funções generateAlgae e showAlgae:

$$
\begin{aligned}
&generateAlgae = [\![(genA\ genB)]\!]_A \\
&\quad \textbf{where} \\
&\qquad genA = (id + \langle id, id \rangle) \cdot outNat \\
&\qquad genB = outNat
\end{aligned}
$$

$$\mathbb{N}_0 \xrightarrow{outNat} 1 + \mathbb{N}_0 \xrightarrow{id + \langle id, id \rangle} 1 + \mathbb{N}_0 \times \mathbb{N}_0$$

$$\mathbb{N}_0 \xrightarrow{outNat} 1 + \mathbb{N}_0$$

Figura 9: Diagramas dos genes de *generateAlgae*

$$
\begin{aligned}
&showAlgae = (\![genA\ genB]\!)_A \\
&\quad \textbf{where} \\
&\qquad genA = [\underline{\texttt{"A"}}, \mathsf{conc}] \\
&\qquad genB = [\underline{\texttt{"B"}}, id]
\end{aligned}
$$

$$String \xleftarrow{[\underline{\texttt{"A"}}, \mathsf{conc}]} 1 + String \times String$$

$$String \xleftarrow{[\underline{\texttt{"B"}}, id]} 1 + String$$

Figura 10: Diagramas dos genes de *showAlgae*

3. Testes quickCheck usando a propriedade dada:

$$
\begin{aligned}
&generatePos :: Gen\ Int \\
&generatePos = choose\ (0, 20) \\
&prop\_alg = forAll\ generatePos\ \$\ \lambda x \rightarrow \\
&\quad (\mathsf{length}\ \cdot showAlgae \cdot generateAlgae)\ x \equiv (fib \cdot \mathsf{succ})\ x \\
&\quad \textbf{where} \\
&\qquad fib :: Int \rightarrow Int \\
&\qquad fib = [\![[\underline{1}, \widehat{(+)}], fibd]\!]
\end{aligned}
$$

Para representação das árvores de Algae:

$$
\begin{aligned}
&dotAlgae :: Algae \rightarrow \mathsf{IO}\ ExitCode \\
&dotAlgae = dotpict \cdot bmap\ Just\ Just \cdot cAlgae2Exp \\
&cAlgae2Exp = (\![[\underline{(Var\ \texttt{"A"})}, h1]\ [\underline{(Var\ \texttt{"B"})}, h2]]\!)_A \\
&\quad \textbf{where}\ h1\ (a, b) = Term\ \texttt{"A"}\ [a, b] \\
&\qquad\qquad h2\ (a) = Term\ \texttt{"B"}\ [a]
\end{aligned}
$$

Problema 5

1. Definição da função permuta: esta função escolhe aletoriamente um elemento de uma lista, através da função getR, juntando esse elemento ao resultado de fazer o mesmo para o resto da lista sem esse elemento.

$$
\begin{aligned}
&permuta\ [\,] = return\ [\,] \\
&permuta\ l = \textbf{do}\ \{ \\
&\quad (h, t) \leftarrow getR\ l; \\
&\quad x \leftarrow permuta\ t; \\
&\quad return\ (h : x) \\
&\quad \}
\end{aligned}
$$

2. Definição da função eliminatória:

$$eliminatoria = (\!|[return, (\gg\!\!= jogo) \cdot \widehat{prod}]|\!)_L$$

$$\begin{tikzcd}
\mathsf{LTree}\ Equipa \arrow[r, "outLTree"'] \arrow[d, "eliminatoria=(\!|g|\!)"'] & Equipa + (\mathsf{LTree}\ Equipa)^2 \arrow[d, "id+eliminatoria^2"] \\
\mathsf{Dist}\ Equipa & Equipa + (\mathsf{Dist}\ Equipa)^2 \arrow[l, "g"] \arrow[d, "id+\widehat{prod}"] \\
 & Equipa + \mathsf{Dist}\ (Equipa, Equipa) \arrow[ul, "{[return, \gg\!\!= jogo]}"]
\end{tikzcd}$$

Figura 11: Diagrama de *eliminatoria*

Índice