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

Departamento de Informática
Universidade do Minho

Junho de 2017

Grupo nr.	58
a73909	Francisco Lira
a77249	Gil Cunha
a79742	Nuno Faria

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 $\mathbb{N}$
import *Exp*
import BTree
import LTree
import *St*
import *Probability hiding* $(\cdot \rightarrow \cdot, \cdot, choose)$
import *Data.List*
import *Test.QuickCheck hiding* $((\times))$
import *System.Random hiding* $\langle \cdot, \cdot \rangle$
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 $lookahead\_sep$.)

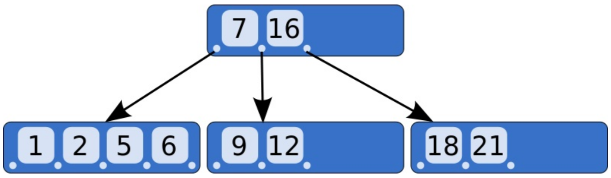
Problema 3

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

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

Por exemplo, a B-tree[3]

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



é representada no tipo acima por:

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

Pretende-se, neste problema:

1. Construir uma biblioteca para o tipo B-tree da forma habitual (in + out; ana + cata + hylo; instância na classe $\mathit{Functor}$).

2. Definir como um catamorfismo a função $\mathit{inordB\_tree} :: \text{B-tree}\ t \to [t]$ que faça travessias "inorder"de árvores deste tipo.

3. Definir como um catamorfismo a função $\mathit{largestBlock} :: \text{B-tree}\ a \to \mathit{Int}$ que detecta o tamanho do maior bloco da árvore argumento.

4. Definir como um anamorfismo a função $\mathit{mirrorB\_tree} :: \text{B-tree}\ a \to \text{B-tree}\ a$ 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 $\mathit{lsplitB\_tree}$ cujo funcionamento se sugere a seguir:

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

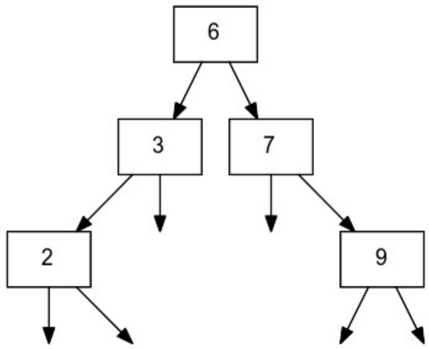
6. A biblioteca Exp permite representar árvores-expressão em formato DOT, que pode ser lido por aplicações como por exemplo Graphviz, produzindo as respectivas imagens. Por exemplo, para o caso de árvores BTree, se definirmos

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

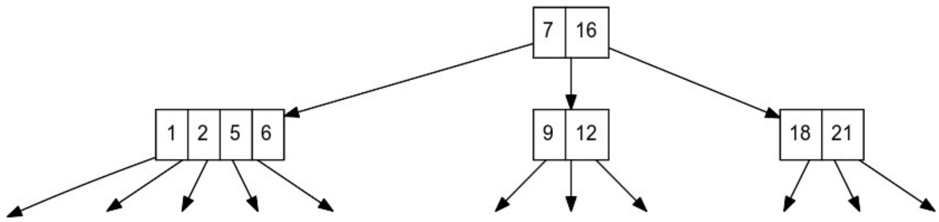
executando $\mathit{dotBTree}\ t$ para

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

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:

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

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

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

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,

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

e a mesma coisa para os Bs:

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

Pretende-se, neste problema:

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

   $$generateAlgae :: Int \rightarrow Algae$$

   como anamorfismo de $Algae$ e da função

   $$showAlgae :: Algae \rightarrow String$$

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

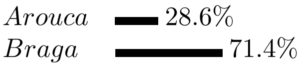
   $$\mathsf{length} \cdot showAlgae \cdot generateAlgae = fib \cdot \mathsf{succ}$$

Problema 5

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

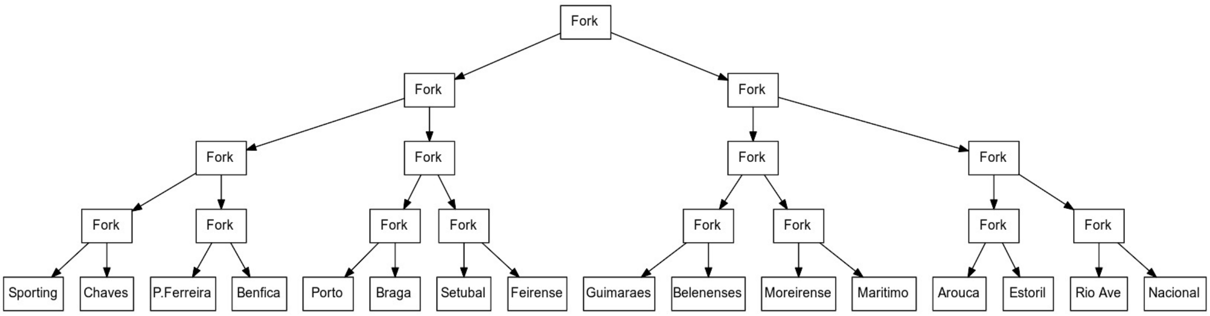
```
equipas :: [Equipa]
equipas = [
  "Arouca","Belenenses","Benfica","Braga","Chaves","Feirense",
  "Guimaraes","Maritimo","Moreirense","Nacional","P.Ferreira",
  "Porto","Rio Ave","Setubal","Sporting","Estoril"
  ]
```

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



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

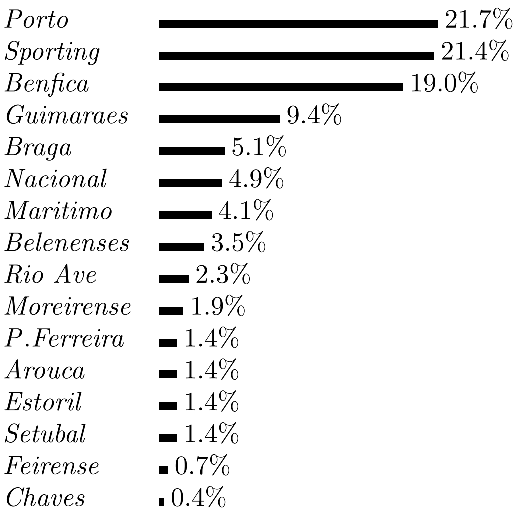
Para lidarmos com probabilidades vamos usar o mónade Dist a que vem descrito no apêndice A e que está implementado na biblioteca Probability [1] — ver definição (1) mais adiante. A primeira parte do problema consiste em sortear *aleatoriamente* os jogos das equipas. O resultado deverá ser uma LTree contendo, 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

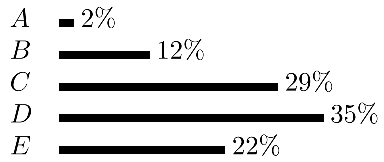
A Mónade para probabilidades e estatística

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

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

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

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

que o GHCi mostrará assim:

```
'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 $g : A \rightarrow \mathsf{Dist}\ B$ 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

Pointwise

Primeiro desenvolvemos a solução em *pointwise* para nos ajudar a chegar a uma *pointfree*.

$$
\begin{array}{l}
inv\_ \; x \; 0 = 1 \\
inv\_ \; x \; (n+1) = (f \; (1-x) \; (n+1)) + (inv\_ \; x \; n) \\
\quad \textbf{where} \\
\qquad f \; x \; 0 = 1 \\
\qquad f \; x \; (n+1) = x * (f \; x \; n)
\end{array}
$$

Onde **f** será a função potência.

Pointfree

Ao fazer a função **inv** em *pointwise* verificou-se que podia ser decomposta em duas, com uma função **f** auxiliar. A partir daqui notamos que seria possível obter um ciclo for (catamorfismo) pela lei de Fokkinga (ou lei de recursividade mútua), dada por:

$$
\left\{ \begin{array}{l} f \; y \cdot \mathbf{in} = h \cdot \mathsf{F} \; \langle f \; y, inv \; x \rangle \\ (inv \; x) \cdot \mathbf{in} = k \cdot \mathsf{F} \; \langle f \; y, inv \; x \rangle \end{array} \right. \equiv \langle f \; y, inv \; x \rangle = (\!|\langle h, k \rangle|\!)
$$

Para derivar h e k, precisamos converter primeiro **f** e **inv** em *pointfree*.
Comecemos por **f y**:

$$
\begin{array}{ll}
& \left\{ \begin{array}{l} f \; y \; 0 = 1 \\ f \; y \; (n+1) = y * f \; y \; n \end{array} \right. \\
\equiv & \quad \{ \text{ Mudança de variável y = 1-x, succ } n = n+1 \; \} \\
& \left\{ \begin{array}{l} f \; (1-x) \; 0 = 1 \\ f \; (1-x) \; (\mathsf{succ} \; n) = (1-x) * (f \; (1-x) \; n) \end{array} \right. \\
\equiv & \quad \{ \text{ Igualdade Extensional, } \underline{0} \; \_ = 0, \underline{1} \; \_ = 1, \text{Eq-+}, \mathbf{in} = [\underline{0}, \mathsf{succ}] \; \} \\
& f \; (1-x) \cdot \mathbf{in} = [\underline{1}, (1-x) * (f \; (1-x))] \\
\equiv & \quad \{ \text{ Absorção-+, Natural-}id \; \} \\
& f \; (1-x) \cdot \mathbf{in} = [\underline{1}, (1-x) *] \cdot (id + f \; (1-x)) \\
\equiv & \quad \{ \text{ Cancelamento-}\times \; \} \\
& f \; (1-x) \cdot \mathbf{in} = [\underline{1}, (1-x) *] \cdot (id + \pi_1 \cdot \langle f \; (1-x), inv \; x \rangle) \\
\equiv & \quad \{ \text{ Functor-+, Natural-}id \; \} \\
& f \; (1-x) \cdot \mathbf{in} = [\underline{1}, (1-x) *] \cdot (id + \pi_1) \cdot (id + \langle f \; (1-x), inv \; x \rangle) \\
\equiv & \quad \{ \; h = [\underline{1}, (1-x) *] \cdot (id + \pi_1), \mathsf{F} \; \langle f \; (1-x), inv \; x \rangle = (id + \langle f \; (1-x), inv \; x \rangle) \; \} \\
& f \; (1-x) \cdot \mathbf{in} = h \cdot \mathsf{F} \; \langle f \; (1-x), inv \; x \rangle
\end{array}
$$

Vejamos agora **inv x**:

$$
\begin{array}{ll}
& \left\{ \begin{array}{l} inv \; x \; 0 = 1 \\ inv \; x \; (n+1) = f \; (1-x) \; (n+1) + inv \; x \; n \end{array} \right. \\
\equiv & \quad \{ \; \mathsf{succ} \; n = n+1, \text{Def-}add \; \} \\
& \left\{ \begin{array}{l} inv \; x \; 0 = 1 \\ inv \; x \; (n+1) = add \cdot \langle f \; (1-x) \; (\mathsf{succ} \; n), inv \; x \; n \rangle \end{array} \right. \\
\equiv & \quad \{ \; f \; (1-x) \; (\mathsf{succ} \; n) = (1-x) * f \; (1-x), \text{Igualdade Extensional} \; \}
\end{array}
$$

$$\begin{cases} inv\ x\ 0 = 1 \\ inv\ x\ (n+1) = add \cdot \langle (1-x) * f\ (1-x), inv\ x \rangle \end{cases}$$

$$\equiv \quad \{\ \text{Eq-+}, \mathbf{in} = [\underline{0}, \mathsf{succ}], \mathsf{succ}\ n = n+1\ \}$$

$$inv\ x \cdot \mathbf{in} = [\underline{1}, add \cdot \langle (1-x) * f\ (1-x), inv\ x \rangle]$$

$$\equiv \quad \{\ \text{Absorção-}\times, \text{Natural-}id\ \}$$

$$inv\ x \cdot \mathbf{in} = [\underline{1}, add \cdot (((1-x)*) \times id) \cdot \langle f\ (1-x), inv\ x \rangle]$$

$$\equiv \quad \{\ \text{Absorção-+}\ \}$$

$$inv\ x \cdot \mathbf{in} = [\underline{1}, add \cdot (((1-x)*) \times id)] \cdot (id + \langle f\ (1-x), inv\ x \rangle)$$

$$\equiv \quad \{\ k = [\underline{1}, add\ (((1-x)*) \times id)], \mathsf{F}\ \langle f\ (1-x), inv\ x \rangle = id + \langle f\ (1-x), inv\ x \rangle\ \}$$

$$inv\ x \cdot \mathbf{in} = k \cdot \mathsf{F}\ \langle f\ (1-x), inv\ x \rangle$$

Agora podemos juntar as duas funções e aplicar a lei de Fokkinga:

$$\begin{cases} f\ (1-x) \cdot \mathbf{in} = h \cdot \mathsf{F}\ \langle f\ (1-x), inv\ x \rangle \\ inv\ x \cdot \mathbf{in} = k \cdot \mathsf{F}\ \langle f\ (1-x), inv\ x \rangle \end{cases}$$

$$\equiv \quad \{\ \text{Fokkinga}\ \}$$

$$\langle f\ (1-x), inv\ x \rangle = (\!|\langle h, k \rangle|\!)$$

$$\equiv \quad \{\ \text{Def-}h, \text{Def-}k\ \}$$

$$\langle f\ (1-x), inv\ x \rangle = (\!|\langle [\underline{1}, (1-x)*] \cdot (id + \pi_1), [\underline{1}, add\ (((1-x)*) \times id)] \rangle|\!)$$

Tendo-se que $inv\ x = \pi_2 \cdot \langle f\ (1-x), inv\ x \rangle = \pi_2 \cdot (\!|\langle h, k \rangle|\!)$. Com isto já podemos fazer a função:

$$\begin{array}{l} inv\ x = wrap \cdot cataNat\ \langle h, k \rangle \\ \quad \mathbf{where} \\ \qquad h = [\underline{1}, (1-x)*] \cdot (id + \pi_1) \\ \qquad k = [\underline{1}, \widehat{(+)} \cdot (((1-x)*) \times id)] \\ \qquad wrap = \pi_2 \end{array}$$

Podemos agora simplificar esta função usando um ciclo-*for*, sendo $\mathsf{for}\ b\ i = (\!|[\underline{i}, b]|\!)$

$$invFor\ x = \pi_2 \cdot (\mathsf{for}\ \langle ((1-x)*) \cdot \pi_1, \widehat{(+)} \cdot (((1-x)*) \times id) \rangle\ (1,1))$$

Alternativa com a regra de Horner

Uma alternativa à resolução deste problema seria através da aplicação da regra de Horner[11].

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

$$= 1 + (1-x) + (1-x)^2 + (1-x)^3 + ...$$

$$= 1 + (1-x)(1 + (1-x)(1 + (1-x)(...)...$$

O que é equivalente a:

$$inv\_alt\ x = \mathsf{for}\ (\mathsf{succ} \cdot ((1-x)*))\ 1$$

Testes (Com incerteza de 0.001% para 3000 iterações)

$$\begin{array}{l} inv\_test :: Double \to Bool \\ inv\_test\ x = abs\ (inv\ x\ 3000 - 1\ /\ x) < 0.001 \end{array}$$

$$\begin{array}{l} prop\_inv :: Property \\ prop\_inv = forAll\ (choose\ (1,2))\ \$\ \lambda x \to inv\_test\ x \end{array}$$

[11] https://pt.wikipedia.org/wiki/Esquema_de_Horner

Problema 2

Pela sugestão do enunciado, iremos aplicar a lei de recursividade (lei de Fokkinga) múltipla às funções **wc_w** (*wc*) e **lookahead_sep** (*look*).

$$
\begin{aligned}
& \langle wc, look \rangle \\
\equiv \quad & \{ \text{ Fokkinga } \} \\
& \begin{cases} wc \cdot \mathbf{in} = h \cdot (id + id \times \langle wc, look \rangle) \\ look \cdot \mathbf{in} = k \cdot (id + id \times \langle wc, look \rangle) \end{cases} \\
\equiv \quad & \{ \text{ Def-}+\text{(x2) } \} \\
& \begin{cases} wc \cdot \mathbf{in} = h \cdot [i_1 \cdot id, i_2 \cdot id \times \langle wc, look \rangle] \\ look \cdot \mathbf{in} = k \cdot [i_1 \cdot id, i_2 \cdot id \times \langle wc, look \rangle] \end{cases} \\
\equiv \quad & \{ \text{ Fusão-}+\text{(x2), Natural-}id\text{(x4) } \} \\
& \begin{cases} wc \cdot \mathbf{in} = [h \cdot i_1, h \cdot i_2 \cdot (id \times \langle wc, look \rangle)] \\ look \cdot \mathbf{in} = [k \cdot i_1, h \cdot i_2 \cdot (id \times \langle wc, look \rangle)] \end{cases} \\
\equiv \quad & \{ \mathbf{in} = [\mathbf{nil}, cons], \text{Eq-}+\text{(x2) } \} \\
& \begin{cases} \begin{cases} wc \cdot \mathbf{nil} = h \cdot i_1 \\ wc \cdot cons = h \cdot i_2 \cdot (id \times \langle wc, look \rangle) \end{cases} \\ \begin{cases} look \cdot \mathbf{nil} = k \cdot i_1 \\ look \cdot cons = k \cdot i_2 \cdot (id \times \langle wc, look \rangle) \end{cases} \end{cases} \\
\equiv \quad & \{ h = [h1, h2], k = [k1, k2], \text{Cancelamento-}+ \text{ (x2) } \} \\
& \begin{cases} \begin{cases} wc \cdot \mathbf{nil} = h1 \\ wc \cdot cons = h2 \cdot (id \times \langle wc, look \rangle) \end{cases} \\ \begin{cases} look \cdot \mathbf{nil} = k1 \\ look \cdot cons = k2 \cdot (id \times \langle wc, look \rangle) \end{cases} \end{cases}
\end{aligned}
$$

Derivando h:

$$
\begin{aligned}
& \begin{cases} wc \cdot \mathbf{nil} = h1 \\ wc \cdot cons = h2 \cdot (id \times \langle wc, look \rangle) \end{cases} \\
\equiv \quad & \{ wc \cdot \mathbf{nil} = \underline{0}, wc \cdot cons = (\widehat{(\wedge)} \cdot \langle \neg (sep \cdot \pi_1), look \cdot \pi_2 \rangle) \rightarrow (\mathsf{succ} \cdot wc \cdot \pi_2), (wc \cdot \pi_2) \} \\
& \begin{cases} \underline{0} = h1 \\ (\widehat{(\wedge)} \cdot \langle \neg (sep \cdot \pi_1), look \cdot \pi_2 \rangle) \rightarrow (\mathsf{succ} \cdot wc \cdot \pi_2), (wc \cdot \pi_2) = h2 \cdot (id \times \langle wc, look \rangle) \end{cases} \\
\equiv \quad & \{ \text{ Cancelamento-}\times\text{(x3) } \} \\
& \begin{cases} \underline{0} = h1 \\ \begin{array}{l} (\widehat{(\wedge)} \cdot \langle \neg (sep \cdot \pi_1), \pi_2 \cdot \langle wc, look \rangle \cdot \pi_2 \rangle) \rightarrow \\ \quad \mathsf{succ} \cdot \pi_1 \cdot \langle wc, look \rangle \cdot \pi_2, \\ \quad \pi_1 \cdot \langle wc, look \rangle \cdot \pi_2 \end{array} = h2 \cdot (id \times \langle wc, look \rangle) \end{cases} \\
\equiv \quad & \{ \text{ Def-}\times \text{ } \} \\
& \begin{cases} \underline{0} = h1 \\ \begin{array}{l} (\widehat{(\wedge)} \cdot ((\neg\ sep) \times (\pi_2 \cdot \langle wc, look \rangle))) \rightarrow \\ \quad \mathsf{succ} \cdot \pi_1 \cdot \langle wc, look \rangle \cdot \pi_2, \\ \quad \pi_1 \cdot \langle wc, look \rangle \cdot \pi_2 \end{array} = h2 \cdot (id \times \langle wc, look \rangle) \end{cases} \\
\equiv \quad & \{ \text{ Natural-}\pi_2\text{(x2) } \} \\
& \begin{cases} \underline{0} = h1 \\ \begin{array}{l} (\widehat{(\wedge)} \cdot ((\neg\ sep) \times (\pi_2 \cdot \langle wc, look \rangle))) \rightarrow \\ \quad \mathsf{succ} \cdot \pi_1 \cdot \pi_2 \cdot (id \times \langle wc, look \rangle), \\ \quad \pi_1 \cdot \pi_2 \cdot (id \times \langle wc, look \rangle) \end{array} = h2 \cdot (id \times \langle wc, look \rangle) \end{cases}
\end{aligned}
$$

$$\equiv \quad \{ \text{ Natural-}id\text{, Fuctor-}\times \}$$

$$\left\{\begin{array}{l} \underline{0} = h1 \\ \widehat{((\wedge)} \cdot ((\neg\ sep) \times \pi_2) \cdot (id \times \langle wc, look\rangle)) \rightarrow \\ \quad \mathsf{succ} \cdot \pi_1 \cdot \pi_2\ (id \times \langle wc, look\rangle), \qquad\qquad = h2 \cdot (id \times \langle wc, look\rangle) \\ \quad \pi_1 \cdot \pi_2 \cdot (id \times \langle wc, look\rangle) \end{array}\right.$$

$$\equiv \quad \{ \text{ 2ª Lei de fusão do condicional } \}$$

$$\left\{\begin{array}{l} \underline{0} = h1 \\ ((\widehat{(\wedge)} \cdot ((\neg\ sep) \times \pi_2)) \rightarrow (\mathsf{succ}\ \pi_1 \cdot \pi_2), (\pi_1 \cdot \pi_2)) \cdot (id \times \langle wc, look\rangle) = h2 \cdot (id \times \langle wc, look\rangle) \end{array}\right.$$

$$\equiv \quad \{ \text{ Leibniz } \}$$

$$\left\{\begin{array}{l} \underline{0} = h1 \\ (\widehat{(\wedge)} \cdot ((\neg\ sep) \times \pi_2)) \rightarrow (\mathsf{succ} \cdot \pi_1 \cdot \pi_2), (\pi_1 \cdot \pi_2) = h2 \end{array}\right.$$

Derivando k:

$$\left\{\begin{array}{l} look \cdot \mathbf{nil} = k1 \\ look \cdot cons = k2 \cdot (id \times \langle wc, look\rangle) \end{array}\right.$$

$$\equiv \quad \{ look \cdot \mathbf{nil} = \underline{True}, look \cdot cons = sep \cdot \pi_1 \}$$

$$\left\{\begin{array}{l} \underline{True} = k1 \\ sep \cdot \pi_1 = k2 \cdot (id \times \langle wc, look\rangle) \end{array}\right.$$

$$\equiv \quad \{ \text{ Natural-}id \}$$

$$\left\{\begin{array}{l} \underline{True} = k1 \\ sep \cdot id \cdot \pi_1 = k2 \cdot (id \times \langle wc, look\rangle) \end{array}\right.$$

$$\equiv \quad \{ \text{ Natural-}\pi_1 \}$$

$$\left\{\begin{array}{l} \underline{True} = k1 \\ sep \cdot \pi_1 \cdot (id \times \langle wc, look\rangle) = k2 \cdot (id \times \langle wc, look\rangle) \end{array}\right.$$

$$\equiv \quad \{ \text{ Leibniz } \}$$

$$\left\{\begin{array}{l} \underline{True} = k1 \\ sep \cdot \pi_1 = k2 \end{array}\right.$$

Com os valores de h e k podemos finalmente definir **wc_w** através de um modulo *worker/wrapper*.

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

$wrapper = \pi_1$

$worker = (\!|\langle [h1, h2], [k1, k2] \rangle|\!)$
  where
    $h1 = \underline{0}$
    $h2 = (\widehat{(\wedge)} \cdot ((\neg \cdot sep) \times \pi_2)) \rightarrow (\mathsf{succ} \cdot \pi_1 \cdot \pi_2), (\pi_1 \cdot \pi_2)$
    $k1 = \underline{True}$
    $k2 = sep \cdot \pi_1$

Testes

$char\_gen :: Gen\ Char$
$char\_gen = elements\ (\mathsf{conc}\ ([\texttt{'0'} \mathinner{..} \texttt{'z'}], [\texttt{' '}, \texttt{'\textbackslash n'}, \texttt{'\textbackslash t'}]))$

$text\_gen :: Gen\ String$
$text\_gen = vectorOf\ 10000\ char\_gen$

$prop\_wc :: Property$
$prop\_wc = forAll\ text\_gen\ \$\ \lambda x \rightarrow wc\_w\_final\ x \equiv wc\_w\ x$

Problema 3

Biblioteca para o tipo B-tree

$$
\begin{array}{l}
inB\_tree = [\underline{Nil}, \widehat{Block}] \\
outB\_tree\ Nil = i_1\ () \\
outB\_tree\ (Block\ l\ b) = i_2\ (l, b) \\
recB\_tree\ f = id + f \times (\mathsf{map}\ \ (id \times f)) \\
baseB\_tree\ g\ f = id + f \times (\mathsf{map}\ \ (g \times f)) \\
(\!|g|\!) = g \cdot (recB\_tree\ ((\!|g|\!))) \cdot outB\_tree \\
anaB\_tree\ g = inB\_tree \cdot (recB\_tree\ (anaB\_tree\ g)) \cdot g \\
hyloB\_tree\ f\ \ g = ((\!|f|\!)) \cdot (anaB\_tree\ g) \\
\textbf{instance}\ Functor\ \mathsf{B\text{-}tree} \\
\quad \textbf{where}\ \mathsf{fmap}\ f = (\!|(inB\_tree \cdot baseB\_tree\ f\ \ id)|\!)
\end{array}
$$

Inord[12]

$$
\begin{tikzcd}
\mathsf{B\text{-}tree}\ A \arrow[d, "inord"'] \arrow[r, bend right, "\mathbf{out}"'] & 1 + \mathsf{B\text{-}tree}\ A \times (A \times (Btree\ A))^{*} \arrow[l, bend right, "\mathbf{in}"'] \arrow[d, "id + inord \times \mathsf{map}\ \ (id \times inord)"] \\
A^{*} & 1 + A^{*} \times (A \times A^{*})^{*} \arrow[l, "inordB\_gene = [\mathbf{nil}, f]"] \\
 & A^{*} \times (A \times A^{*})^{*} \arrow[u, "i_2"] \arrow[d, "id \times (\mathsf{map}\ \ \cdot \mathsf{conc} \cdot (singl \times id))"] \arrow[ddd, bend left=60, "f"] \\
 & A^{*} \times (A^{*})^{*} \arrow[d, "id \times concat"] \\
 & A^{*} \times A^{*} \arrow[d, "\mathsf{conc}"] \\
 & A^{*} \arrow[uuuuul, dashed, bend left]
\end{tikzcd}
$$

$$
\begin{array}{l}
inordB\_tree = (\!|inordB\_gene|\!) \\
inordB\_gene = [\mathbf{nil}, f] \\
\quad \textbf{where}\ f = \mathsf{conc} \cdot (id \times (concat \cdot (\mathsf{map}\ \ (\mathsf{conc} \cdot (singl \times id)))))
\end{array}
$$

[12] $inordB\_gene$ será mais tarde usada em $qSortB\_tree$

Largest Block

$$
\begin{tikzcd}
\textsf{B-tree}\ A \arrow[d, "largestBlock"'] \arrow[r, bend right, "\textbf{out}"'] & 1 + \textsf{B-tree}\ A \times (A \times (Btree\ A))^* \arrow[l, bend right, "\textbf{in}"'] \arrow[d, "id + largestBlock \times \textsf{map}\ (id \times largestBlock)"] \\
\mathbb{N}_0 & 1 + \mathbb{N}_0 \times (A \times \mathbb{N}_0)^* \arrow[l, "{[\underline{0}, f]}"] \arrow[dl, "id + id \times h"] \\
1 + \mathbb{N}_0 \times \mathbb{N}_0 \arrow[u, "{[\underline{0}, \widehat{max}]}"] & \mathbb{N}_0 \times (A \times \mathbb{N}_0)^* \arrow[u, "i2"] \arrow[d, "\pi_2"] \arrow[dddd, bend left, "h"] \\
& (A \times \mathbb{N}_0)^* \arrow[d, "unzip"] \\
& A^* \times (\mathbb{N}_0)^* \arrow[d, "\textsf{length} \times maximum"] \\
& \mathbb{N}_0 \times \mathbb{N}_0 \arrow[d, "\widehat{max}"] \\
& \mathbb{N}_0
\end{tikzcd}
$$

$$
\begin{aligned}
& largestBlock = (\![[\underline{0}, f]]\!) \\
& \quad \textbf{where} \\
& \qquad f = \widehat{(max)} \cdot (id \times h) \\
& \qquad\quad \textbf{where}\ h = \widehat{(max)} \cdot (\textsf{length} \times maximum) \cdot unzip
\end{aligned}
$$

Mirror

$$
\begin{tikzcd}
\textsf{B-tree}\ A \arrow[rr, bend right, "\mathbf{out}"'] & & 1+\textsf{B-tree}\ A\times(A\times\textsf{B-tree}\ A)^{*} \arrow[ll, bend right, "\mathbf{in}"'] \\
\textsf{B-tree}\ A \arrow[u, "mirror"] \arrow[rr, dashed] \arrow[d, "\mathbf{out}"] & & 1+\textsf{B-tree}\ A\times(A\times\textsf{B-tree}\ A)^{*} \arrow[u, "id+mirror\times\mathsf{map}\ \ (id\times mirror)"'] \\
1+\textsf{B-tree}\ A\times(A\times\textsf{B-tree}\ A)^{*} \arrow[urr, bend right, "id+\langle head\cdot\pi_2\cdot f,g\rangle"] & &
\end{tikzcd}
$$

A função f irá partir do *Block* da B-tree e transformar num par, onde o primeiro elemento da lista é uma lista de valores e o segundo é uma lista com os respetivos *blocks*[13]. Seguidamente irá aplicar o $reverse$ nas respetivas listas obtidas[14].

$f :: \textsf{B-tree}\ A\times(A\times\textsf{B-tree}\ A)^{*} \xrightarrow{\pi_2} (A\times\textsf{B-tree}\ A)^{*} \xrightarrow{unzip} A^{*}\times(\textsf{B-tree}\ A)^{*} \xrightarrow{reverse\times reverse} A^{*}\times(\textsf{B-tree}\ A)^{*}$

A função g irá partir dum par, onde o primeiro elemento contem todos os elementos da árvore, e o segundo tem todos os $blocks$, juntando através de $\widehat{zip}$ (necessário para a definição de B-tree).

$g :: \textsf{B-tree}\ A\times(A\times\textsf{B-tree}\ A)^{*} \xrightarrow{\langle\pi_1\cdot f,\mathsf{conc}\cdot k\rangle} A^{*}\times(B\_Tree\ A)^{*} \xrightarrow{zip} (A\times(\textsf{B-tree}\ A))^{*}$

A função $k1$ irá partir da segunda lista do par produzido por f e aplicar o $tail$, correspondendo a todos os *blocks* da arvore criada pelo $mirror$ sem o *leftmost*[15] e a sua cabeça[16], sendo esta o *leftmost* da nova árvore.

A função $k2$ partirá do *leftmost* original da B-tree e transformar-la numa lista de B-tree, através de $singl$, para ser possível mais tarde a sua junção com o resultado de $k1$.

$$
\begin{tikzcd}
\textsf{B-tree}\ A \arrow[dr, "k1"'] & \textsf{B-tree}\ A\times(A\times\textsf{B-tree}\ A)^{*} \arrow[l, "\pi_1"'] \arrow[r, "\pi_2"] \arrow[d, "\langle k1,k2\rangle"] & (A\times\textsf{B-tree}\ A)^{*} \arrow[dl, "k2"] \\
 & (\textsf{B-tree}\ A)^{*}\times(\textsf{B-tree}\ A)^{*} &
\end{tikzcd}
$$

$k1 :: \textsf{B-tree}\ A\times(A\times\textsf{B-tree}\ A)^{*} \xrightarrow{f} (A^{*}\times(\textsf{B-tree}\ A)^{*}) \xrightarrow{\pi_2} (\textsf{B-tree}\ A)^{*} \xrightarrow{tail} (\textsf{B-tree}\ A)^{*}$

$k2 :: \textsf{B-tree}\ A\times(A\times\textsf{B-tree}\ A)^{*} \xrightarrow{\pi_1} (\textsf{B-tree}\ A) \xrightarrow{singl} (\textsf{B-tree}\ A)^{*}$

Fazendo $\langle head\cdot\pi_2\cdot f, g\rangle$ obtemos o par necessário para que a função inB_tree construa a nova árvore (espelhada) a partir da original.

$$
\begin{array}{l}
mirrorB\_tree = anaB\_tree\ ((id + \langle head\cdot\pi_2\cdot f, g\rangle)\cdot outB\_tree) \\
\quad \textbf{where} \\
\qquad g = \widehat{(zip)}\cdot\langle\pi_1\cdot f, \mathsf{conc}\cdot k\rangle \\
\qquad f = (reverse\times reverse)\cdot unzip\cdot\pi_2 \\
\qquad k = \langle tail\cdot\pi_2\cdot f, singl\cdot\pi_1\rangle
\end{array}
$$

[13]Notar que *Block* != *block*

[14]Será feito o $reverse$ devido à definição de $mirror$

[15]Será agora um *block*

[16]Que era o ultimo $block$, mas como se vez $reverse$, será a cabeça da lista.

Mirror-Testes

$$
\begin{aligned}
&mirror\_test :: [Int] \rightarrow Bool \\
&mirror\_test\ l = f\ l \equiv g\ l \\
&\quad \mathbf{where} \\
&\qquad f = reverse \cdot qSortB\_tree \\
&\qquad g = inordB\_tree \cdot mirrorB\_tree \cdot anaB\_tree\ (lsplitB\_tree) \\
&num\_gen :: Gen\ Int \\
&num\_gen = choose\ (0, 1000) \\
&intList\_gen :: Gen\ [Int] \\
&intList\_gen = vectorOf\ 50\ num\_gen \\
&prop\_mirror :: Property \\
&prop\_mirror = forAll\ intList\_gen\ \$\ \lambda x \rightarrow mirror\_test\ x
\end{aligned}
$$

Quicksort

```
lsplitB_tree (h:h2:t) =
        ([menores que h], [(h, [maiores que h e menores que h2]),
          (h2, [maiores que h2])])
```

Onde [elementos menores que h] = *leftmost*, h e h2 = são os valores, [elementos maiores que h e menores que h2] e [elementos maiores que h2] = *blocks* dos valores h e h2, respetivamente.

$$lsplitB\_tree :: Ord\ a \Rightarrow [a] \rightarrow \text{Either}\ ()\ ([a], [(a, [a])])$$

$$
\begin{aligned}
&lsplitB\_tree\ [\,] = i_1\ () \\
&lsplitB\_tree\ [e] = i_2\ ([\,], [(e, [\,])]) \\
&lsplitB\_tree\ (h : h2 : t) = i_2\ (((filter\ (<minpar)\ t)), \\
&\qquad\qquad [((minpar), ((filter\ (condition)\ t))), ((maxpar), ((filter\ (>maxpar)\ t)))]) \\
&\quad \mathbf{where} \\
&\qquad minpar = (min\ h\ h2) \\
&\qquad maxpar = (max\ h\ h2) \\
&\qquad condition = \lambda a \rightarrow (a > minpar) \wedge (a < maxpar)
\end{aligned}
$$

$$
\begin{aligned}
&qSortB\_tree :: Ord\ a \Rightarrow [a] \rightarrow [a] \\
&qSortB\_tree = hyloB\_tree\ inordB\_gene\ lsplitB\_tree
\end{aligned}
$$

$$
\begin{tikzcd}
A^* \arrow[r, "lsplitB\_tree"] \arrow[d, "{[\![lsplitB\_tree]\!]}"'] & 1 + A^* \times (A \times A^*)^* \arrow[d, "{id + [\![lsplitB\_tree]\!] \times \mathsf{map}\ (id \times [\![lsplitB\_tree]\!])}"] \\
\text{B-tree}\ A \arrow[r, bend left, "\mathbf{out}"] \arrow[d, "{(\!| inordB\_gene |\!)}"'] & 1 + B\_treeA \times (A \times \text{B-tree}\ A)^* \arrow[l, bend left, "\mathbf{in}"] \arrow[d, "{id + (\!| inordB\_gene |\!) \times \mathsf{map}\ (id \times (\!| inordB\_gene |\!))}"] \\
A^* & 1 + A^* \times (A \times A^*)^* \arrow[l, "inord"]
\end{tikzcd}
$$

Quicksort-Testes

$$
\begin{array}{l}
isSorted :: Ord\ a \Rightarrow [\,a\,] \rightarrow Bool \\
isSorted\ [\,] = True \\
isSorted\ (a : [\,]) = True \\
isSorted\ (h : hs) = \mathbf{if}\ (h \leqslant head\ hs)\ \mathbf{then}\ isSorted\ hs \\
\quad \mathbf{else}\ False
\end{array}
$$

$$qsort\_test\ x = isSorted\ (qSortB\_tree\ x)$$

$$prop\_qsort = forAll\ intList\_gen\ \$\ \lambda x \rightarrow qsort\_test\ x$$

dotB_tree

$$
\begin{tikzcd}
\mathsf{B\text{-}tree}\ A \arrow[d, "cB\_tree2Exp"'] \arrow[r, bend right, "\mathbf{out}"'] & 1 + \mathsf{B\text{-}tree}\ A \times (A \times (Btree\ A))^{*} \arrow[l, bend right, "\mathbf{in}"'] \arrow[d, "id + cB\_tree2Exp \times \mathsf{map}\ (id \times cB\_tree2Exp)"] \\
Exp\ A & 1 + Exp\ A \times (A \times (Exp\ A))^{*} \arrow[l, "{[\underline{(Var\ \texttt{"nil"})}, cB2E\_aux]}"] \\
 & Exp\ A \times (A \times (Exp\ A))^{*} \arrow[u, "i_2"] \arrow[d, "id \times unzip"] \arrow[dddd, bend left, "cB2E\_aux"] \\
 & Exp\ A \times (A^{*} \times (Exp\ A)^{*}) \arrow[d, "\langle \pi_1 \cdot \pi_2, \langle \pi_1, \pi_2 \cdot \pi_2 \rangle \rangle"] \\
 & A^{*} \times (Exp\ A \times (Exp\ A)^{*}) \arrow[d, "id \times cons"] \\
 & A^{*} \times (Exp\ A)^{*} \arrow[d, "\widehat{Term}"] \\
 & Exp\ A \arrow[uuuuul, dashed]
\end{tikzcd}
$$

$$
\begin{array}{l}
dotB\_Tree :: Show\ a \Rightarrow \mathsf{B\text{-}tree}\ a \rightarrow \mathsf{IO}\ ExitCode \\
dotB\_Tree = dotpict \cdot bmap\ nothing\ (Just \cdot init \cdot concat \cdot (\mathsf{map}\ \ (+\!\!+ \texttt{"|"})) \cdot (\mathsf{map}\ \ show)) \cdot cB\_tree2Exp \\
cB\_tree2Exp = (\!|\,[\underline{(Var\ \texttt{"nil"})}, cB2E\_aux]\,|\!) \\
\quad \mathbf{where}\ cB2E\_aux = \widehat{(Term)} \cdot (id \times cons) \cdot \langle \pi_1 \cdot \pi_2, \langle \pi_1, \pi_2 \cdot \pi_2 \rangle \rangle \cdot (id \times unzip)
\end{array}
$$

Exemplos de mirror, dotB_tree e qsort

Lista para árvore para *dot*

$$listToBtreeToDot\ t = (dotB\_Tree \cdot anaB\_tree\ (lsplitB\_tree))\ t$$

Mirror de lista para árvore para *dot*

$$mirrorListToBtreeToDot\ t = (dotB\_Tree \cdot mirrorB\_tree \cdot (anaB\_tree\ (lsplitB\_tree)))\ t$$

Geramos a árvore a partir da seguinte lista:

`[870,241,548,743,974,399,620,68,266,12,436,540,51,902,350,926,53,567,4,676]`

sendo a sua representação da seguinte forma:

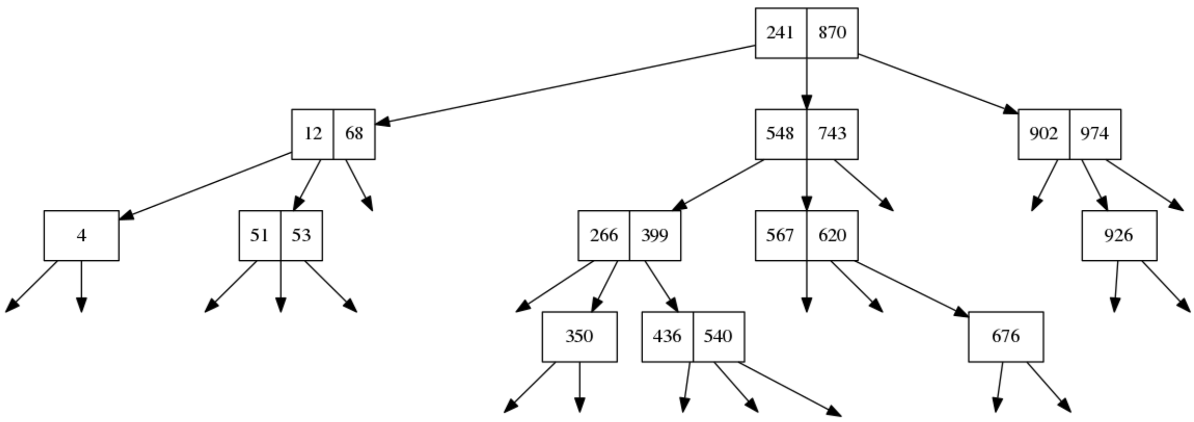


Figura 1: Árvore exemplo

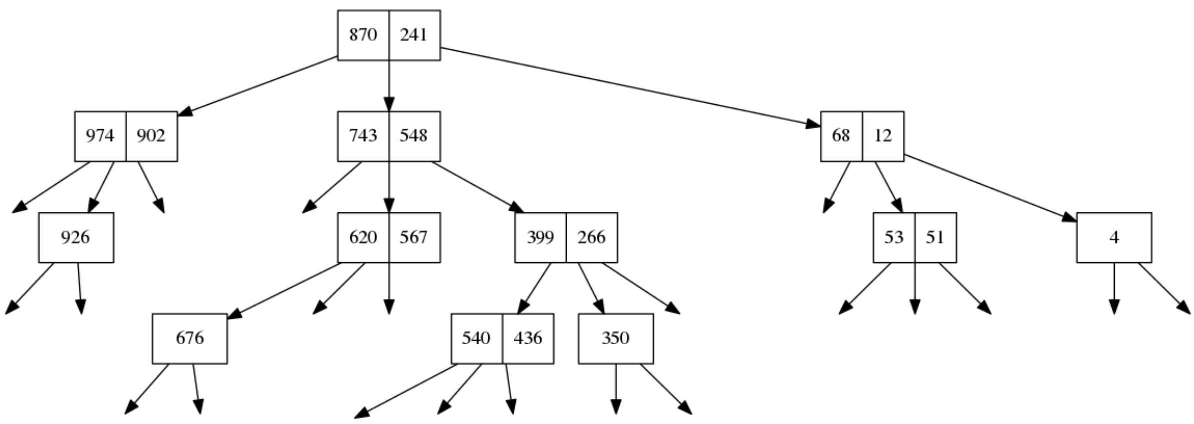


Figura 2: Árvore exemplo após mirror

Após a aplicação de $qsortB\_tree$:

`[4,12,51,53,68,241,266,350,399,436,540,548,567,620,676,743,870,902,926,974]`

Problema 4

1

$$\begin{tikzcd}
Algae\ A \arrow[rr, bend right, "\mathbf{out}"'] \arrow[d, "(\!|gagb|\!)_A"'] & & 1 + A \times B \arrow[ll, bend right, "\mathbf{in}"'] \arrow[d, "id + (\!|gagb|\!)_A \times (\!|gagb|\!)_B"] \\
X & & 1 + X \times X \arrow[ll, "ga"]
\end{tikzcd}
\qquad
\begin{tikzcd}
B \arrow[rr, bend right, "\mathbf{out}"'] \arrow[d, "(\!|gagb|\!)_B"'] & & 1 + A \arrow[ll, bend right, "\mathbf{in}"'] \arrow[d, "id + (\!|gagb|\!)_A"] \\
X & & 1 + X \arrow[ll, "gb"]
\end{tikzcd}$$

$$\begin{tikzcd}
A \arrow[rr, bend right, "\mathbf{out}"'] & & 1 + A \times B \arrow[ll, bend right, "\mathbf{in}"'] \\
X \arrow[u, "\llbracket gagb \rrbracket_A"] \arrow[rr, "ga"'] & & 1 + X \times X \arrow[u, "id + \llbracket gagb \rrbracket_A \times \llbracket gagb \rrbracket_B"']
\end{tikzcd}
\qquad
\begin{tikzcd}
B \arrow[rr, bend right, "\mathbf{out}"'] & & 1 + A \arrow[ll, bend right, "\mathbf{in}"'] \\
X \arrow[u, "\llbracket gagb \rrbracket_B"] \arrow[rr, "gb"'] & & 1 + X \arrow[u, "id + \llbracket gagb \rrbracket_A"']
\end{tikzcd}$$

$$(\!|gagb|\!)_A = ga \cdot (id + ((\!|gagb|\!)_A) \times ((\!|gagb|\!)_B)) \cdot outA$$
$$(\!|gagb|\!)_B = gb \cdot (id + ((\!|gagb|\!)_A)) \cdot outB$$
$$\llbracket gagb \rrbracket_A = inA \cdot (id + ((\llbracket gagb \rrbracket_A) \times (\llbracket gagb \rrbracket_B))) \cdot ga$$
$$\llbracket gagb \rrbracket_B = inB \cdot (id + (\llbracket gagb \rrbracket_A)) \cdot gb$$

2 - Pointwise

$$generateAlgaeA\ 0 = \mathrm{NA}$$
$$generateAlgaeA\ n = A\ (generateAlgaeA\ (n-1))\ (generateAlgaeB\ (n-1))$$

$$generateAlgaeB\ 0 = \mathrm{NB}$$
$$generateAlgaeB\ n = B\ (generateAlgaeA\ (n-1))$$

$$showAlgaeA\ \mathrm{NA} = \texttt{"A"}$$
$$showAlgaeA\ (A\ a\ b) = (showAlgaeA\ a) +\!\!+\ showAlgaeB\ b$$

$$showAlgaeB\ \mathrm{NB} = \texttt{"B"}$$
$$showAlgaeB\ (B\ a) = showAlgaeA\ a$$

2 - Pointfree

$$
\begin{tikzcd}
A \arrow[rr, bend right, "\mathbf{out}"'] & & 1 + A \times B \arrow[ll, bend right, "\mathbf{in}"'] \\
\mathbb{N}_0 \arrow[u, "generateA"] \arrow[r, "\mathbf{out}"'] \arrow[rr, bend right, "ga1"'] & 1 + \mathbb{N}_0 \arrow[r, "id + \langle id, id \rangle"'] & 1 + \mathbb{N}_0 \times \mathbb{N}_0 \arrow[u, "id + generateA \times generateB"']
\end{tikzcd}
\qquad
\begin{tikzcd}
B \arrow[r, bend right, "\mathbf{out}"'] & 1 + A \arrow[l, bend right, "\mathbf{in}"'] \\
\mathbb{N}_0 \arrow[u, "generateB"] \arrow[r, "gb1 = \mathbf{out}"'] & 1 + \mathbb{N}_0 \arrow[u, "id + generateA"']
\end{tikzcd}
$$

$$
\begin{aligned}
&generateAlgae = [\![ga1\ gb1]\!]_A \\
&\quad \textbf{where} \\
&\qquad ga1 = (id + \langle id, id \rangle) \cdot outNat \\
&\qquad gb1 = outNat
\end{aligned}
$$

$$
\begin{tikzcd}
A \arrow[rr, bend right, "\mathbf{out}"'] \arrow[d, "showA"'] & & 1 + A \times B \arrow[ll, bend right, "\mathbf{in}"'] \arrow[d, "id + showA \times showB"] \\
String & & 1 + String \times String \arrow[ll, "ga2 = {[\underline{\texttt{"A"}}, \mathsf{conc}]}"]
\end{tikzcd}
\qquad
\begin{tikzcd}
B \arrow[r, bend right, "\mathbf{out}"'] \arrow[d, "showB"'] & 1 + A \arrow[l, bend right, "\mathbf{in}"'] \arrow[d, "id + showA"] \\
String & 1 + String \arrow[l, "gb2 = {[\underline{\texttt{"B"}}, id]}"]
\end{tikzcd}
$$

$$
\begin{aligned}
&showAlgae = (\!| ga2\ gb2 |\!)_A \\
&\quad \textbf{where} \\
&\qquad ga2 = [\underline{\texttt{"A"}}, \mathsf{conc}] \\
&\qquad gb2 = [\underline{\texttt{"B"}}, id]
\end{aligned}
$$

Exemplos de *generate* e *show*

```
generateAlgae 0 = NA
showAlgae (generateAlgae 0) = "A"

generateAlgae 1 = A NA NB
showAlgae (generateAlgae 1) = "AB"

generateAlgae 5 = A (A (A (A (A NA NB) (B NA)) (B (A NA NB))) (B (A (A NA NB)
  (B NA)))) (B (A (A (A NA NB) (B NA)) (B (A NA NB))))
showAlgae (generateAlgae 5) = "ABAABABAABAAB"
```

Testes

Para os testes tivemos que redefinir a função *fib*, pois só assim é possível alterar o seu tipo (de **Integer** para **Int**). Esta alteração deve-se ao facto de tornar compatíveis os tipos de length e *fib*, para que o teste seja realizável.

$$
\begin{array}{l}
algae\_test :: Int \rightarrow Bool \\
algae\_test\ n = (\mathsf{length} \cdot showAlgae \cdot generateAlgae)\ n \equiv (fib \cdot \mathsf{succ})\ n \\
\qquad\qquad\qquad \textbf{where} \\
\qquad\qquad\qquad\quad fib :: Int \rightarrow Int \\
\qquad\qquad\qquad\quad fib = hyloLTree\ [\underline{1}, \widehat{(+)}]\ fibd \\
prop\_algae :: Property \\
prop\_algae = forAll\ (choose\ (0, 20))\ \$\ \lambda x \rightarrow algae\_test\ x
\end{array}
$$

Problema 5

Pointwise

Mais uma vez, iremos primeiro fazer as funções em *pointwise*, atingindo posteriormente os resultados finais. Para isso, começaremos por criar novas funções jogo e getR, onde o fator de aleatoriedade está ausente (sendo que isso envolve probabilidades, o que implicaria o uso de Monades). A partir destas faremos depois a sua monadificação[17], de modo a obter as funções finais pedidas pelo enunciado.

$jogo\_$ indica que ganha sempre a primeira equipa (da casa) e $getR\_$ tira sempre a cabeça da lista.

$$
\begin{array}{l}
jogo\_ :: (Equipa, Equipa) \rightarrow Equipa \\
jogo\_ = \pi_1
\end{array}
$$

$$
\begin{array}{l}
eliminatoria\_\ (Leaf\ \ a) = a \\
eliminatoria\_\ (Fork\ (l, r)) = jogo\_\ (e_1, e_2) \\
\quad \textbf{where} \\
\qquad e_1 = eliminatoria\_\ l \\
\qquad e_2 = eliminatoria\_\ r
\end{array}
$$

$$
\begin{array}{l}
getR\_ :: [a] \rightarrow (a, [a]) \\
getR\_ = \langle head, tail \rangle
\end{array}
$$

$$
\begin{array}{l}
permuta\_ :: [a] \rightarrow [a] \\
permuta\_\ [\,] = [\,] \\
permuta\_\ list = (a : permuta\_\ b) \\
\quad \textbf{where} \\
\qquad (a, b) = getR\_\ list
\end{array}
$$

"Monadificação"

$$
\begin{array}{l}
permuta\ [\,] = return\ [\,] \\
permuta\ list = \textbf{do}\ \{(a, b) \leftarrow getR\ list; c \leftarrow permuta\ b; return\ (a : c)\}
\end{array}
$$

$$
\begin{array}{l}
eliminatoria\ (Leaf\ \ a) = return\ a \\
eliminatoria\ (Fork\ (l, r)) = \textbf{do}\ \{e_1 \leftarrow eliminatoria\ l; e_2 \leftarrow eliminatoria\ r; jogo\ (e_1, e_2)\}
\end{array}
$$

[17] Secção 4.10 - ("Monadification"of Haskell code made easy) - apontamentos teóricos

Índice