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

Departamento de Informática
Universidade do Minho

Junho de 2017

Grupo nr.	37
a67637	Célia Figueiredo
a67662	Adriana Pereira
a60993	Luís Pedro Fonseca

Conteúdo

1 Preâmbulo

A disciplina de Cálculo de Programas tem como objectivo principal ensinar a programação de computadores como uma disciplina científica. Para isso parte-se de um repertório de *combinadores* que formam uma álgebra da programação (conjunto de leis universais e seus corolários) e usam-se esses combinadores para construir programas *composicionalmente*, isto é, agregando programas já existentes.

Na sequência pedagógica dos planos de estudo dos dois cursos que têm esta disciplina, restringe-se a aplicação deste método ao desenvolvimento de programas funcionais na linguagem Haskell.

O presente trabalho tem por objectivo concretizar na prática os objectivos da disciplina, colocando os alunos perante problemas de programação que deverão ser abordados composicionalmente e implementados em Haskell. Há ainda um outro objectivo: o de ensinar a documentar programas e a produzir textos técnico-científicos de qualidade.

2 Documentação

Para cumprir de forma integrada os objectivos enunciados acima vamos recorrer a uma técnica de programação dita "literária" [3], cujo princípio base é o seguinte:

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

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

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

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

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

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

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

```
ghci cp1617t.lhs
```

para ver que assim é:

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

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

import *Cp*
import *List*

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

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

Abra o ficheiro `cp1617t.lhs` no seu editor de texto preferido e verifique que assim é: todo o texto que se encontra dentro do ambiente

```
\begin{code}
...
\end{code}
```

vai ser seleccionado pelo GHCi para ser executado.

3 Como realizar o trabalho

Este trabalho teórico-prático deve ser realizado por grupos de três alunos. Os detalhes da avaliação (datas para submissão do relatório e sua defesa oral) são os que forem publicados na página da disciplina na *internet*. Recomenda-se uma abordagem equilibrada e participativa dos membros do grupo de trabalho por forma a poderem responder às questões que serão colocadas na defesa oral do relatório.

Em que consiste, então, o *relatório* a que se refere o parágrafo anterior? É a edição do texto que está a ser lido, preenchendo o anexo C com as suas respostas. O relatório deverá conter ainda a identificação dos membros do grupo de trabalho, no local respectivo da folha de rosto.

Para gerar o PDF integral do relatório deve-se ainda correr os comando seguintes, que actualizam a bibliografia (com BibTEX) e o índice remissivo (com `makeindex`),

```
bibtex cp1617t.aux
makeindex cp1617t.idx
```

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

Problema 1

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

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

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

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

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

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

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

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

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

Problema 2

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

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

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

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

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

$$
\begin{array}{l}
wc\_w :: [\mathit{Char}] \rightarrow \mathit{Int} \\
wc\_w\ [\,] = 0 \\
wc\_w\ (c : l) = \\
\quad \textbf{if}\ \neg\ (sep\ c) \wedge lookahead\_sep\ l \\
\quad \textbf{then}\ wc\_w\ l + 1 \\
\quad \textbf{else}\ wc\_w\ l \\
\qquad \textbf{where} \\
\qquad\quad sep\ c = c \in \texttt{" \textbackslash r\textbackslash n\textbackslash t\textbackslash v\textbackslash f\textbackslash 160"} \\
\qquad\quad lookahead\_sep\ [\,] = \mathit{True} \\
\qquad\quad lookahead\_sep\ (c : l) = sep\ c
\end{array}
$$

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 $lookahead\_sep$.)

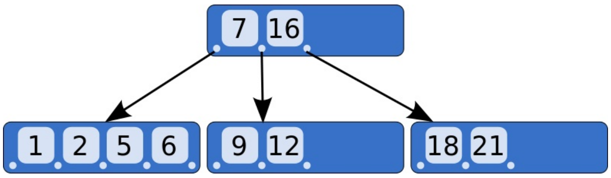
Problema 3

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

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

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 $inordB\_tree$ :: B-tree $t \rightarrow [t]$ que faça travessias "inorder"de árvores deste tipo.

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

4. Definir como um anamorfismo a função $mirrorB\_tree$ :: B-tree $a \rightarrow$ B-tree a que roda a árvore argumento de 180º

5. Adaptar ao tipo B-tree o hilomorfismo "quick sort"do módulo BTree. O respectivo anamorfismo deverá basear-se no gene $lsplitB\_tree$ cujo funcionamento se sugere a seguir:

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

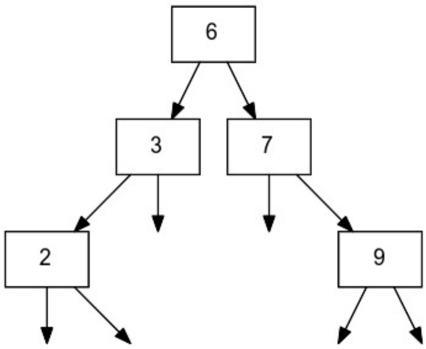
6. A biblioteca Exp permite representar árvores-expressão em formato DOT, que pode ser lido por aplicações como por exemplo Graphviz, produzindo as respectivas imagens. Por exemplo, para o caso de árvores BTree, se definirmos

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

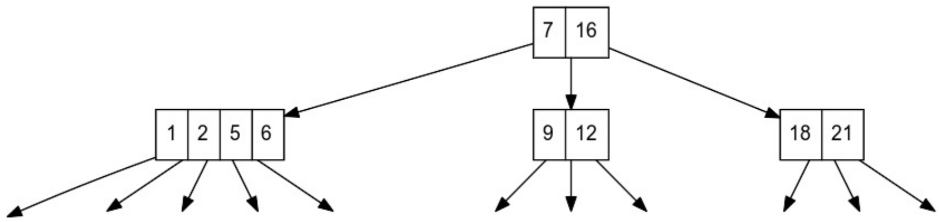
executando $dotBTree\ t$ para

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

obter-se-á a imagem



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



para a árvora dada acima.

Problema 4

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

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

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

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

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

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

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

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

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 -¿ Algae

   como anamorfismo de $Algae$ e da função

   showAlgae :: Algae -¿ 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:

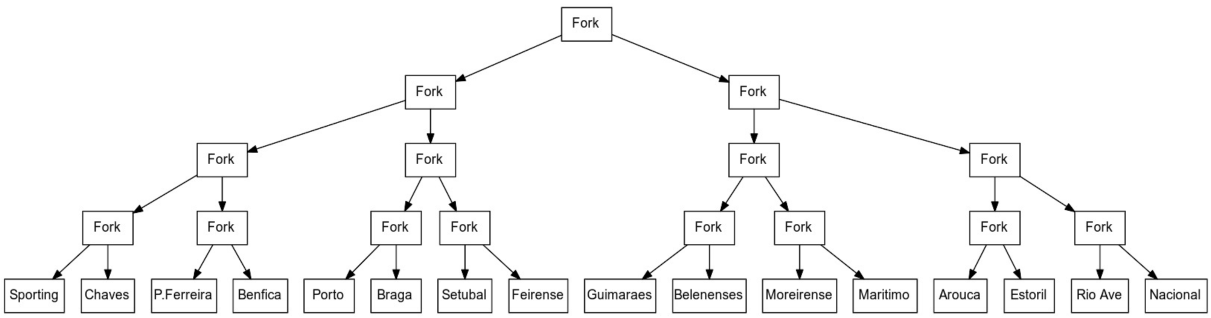
```
equipas :: [Equipa]
equipas = [
  "Arouca","Belenenses","Benfica","Braga","Chaves","Feirense",
  "Guimaraes","Maritimo","Moreirense","Nacional","P.Ferreira",
  "Porto","Rio Ave","Setubal","Sporting","Estoril"
  ]
```

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 $[Equipa] \rightarrow \mathsf{Dist}\ Equipa$,

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

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] -¿ 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 :: LTree Equipa -¿ Dist Equipa

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

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

Referências

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

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

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

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

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

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

Anexos

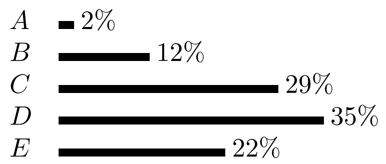
A Mónade para probabilidades e estatística

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

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

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

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



será representada pela distribuição

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

que o GHCi mostrará assim:

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

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

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

isto é

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

distribuição *normais*, eg.

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

etc.[10]

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

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

em que $g{:}A \rightarrow \mathsf{Dist}\ B$ 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

$$
\begin{array}{l}
\mathit{inv} :: \mathit{Double} \rightarrow \mathit{Int} \rightarrow \mathit{Double} \\
\mathit{inv}\ x\ n = \pi_2\ \$\ \mathsf{for}\ (\mathit{invAccum}\ x)\ (0, 0)\ n
\end{array}
$$

$$
\begin{array}{l}
\textbf{where } \mathit{invAccum} :: \mathit{Double} \rightarrow (\mathit{Double}, \mathit{Double}) \rightarrow (\mathit{Double}, \mathit{Double}) \\
\quad \mathit{invAccum}\ x\ (\mathit{it}, \mathit{accum}) = (\mathit{it} + 1, \mathit{accum} + (1 - x) ** \mathit{it})
\end{array}
$$

$$
\begin{array}{l}
\mathit{prop\_inv\_correctness} :: \mathit{NonNegative\ Int} \rightarrow \mathit{Property} \\
\mathit{prop\_inv\_correctness}\ (\mathit{NonNegative}\ n) = \mathit{forAll}\ (\mathit{choose}\ (1, 1.999))\ \$\ \lambda x \rightarrow \mathit{diff}\ n\ x > \mathit{tolerance} \\
\quad \textbf{where } \mathit{diff} :: \mathit{Int} \rightarrow \mathit{Double} \rightarrow \mathit{Double} \\
\quad\quad \mathit{diff}\ n\ x = \mathit{abs}\ \$\ x - (\mathit{inv}\ x\ n) \\
\quad\quad \mathit{tolerance} = 0.0001
\end{array}
$$

A sua solução em C pode ser definida da seguinte forma:

```
float inv(float x, int n){
    float accum = 0;
    for(int i = 0; i < n; i++){
        accum += pow(1 - x, i);
    }
    return accum;
}
```

Problema 2

$$
\begin{array}{l}
\mathit{wc\_w\_final} :: [\mathit{Char}] \rightarrow \mathit{Int} \\
\mathit{wc\_w\_final} = \mathit{wrapper} \cdot \mathit{worker}
\end{array}
$$

$$
\begin{array}{l}
\mathit{wrapper} :: a \times b \rightarrow b \\
\mathit{wrapper} = \pi_2 \\
\mathit{worker} :: [\mathit{Char}] \rightarrow (\mathit{Bool}, \mathit{Int}) \\
\mathit{worker} = \mathit{cataList}\ [\underline{(\mathit{True}, 0)}, \mathit{wordAccum}] \\
\mathit{wordAccum} :: (\mathit{Char}, (\mathit{Bool}, \mathit{Int})) \rightarrow (\mathit{Bool}, \mathit{Int}) \\
\mathit{wordAccum}\ (a, (b, n)) \\
\quad \mid b \wedge (\neg \cdot \mathit{sep})\ a = (\mathit{sep}\ a, \mathsf{succ}\ \ n) \\
\quad \mid \mathit{otherwise} = (\mathit{sep}\ a, n) \\
\mathit{sep} :: \mathit{Char} \rightarrow \mathit{Bool} \\
\mathit{sep}\ c = c \in \texttt{" \textbackslash r\textbackslash n\textbackslash t\textbackslash v\textbackslash f\textbackslash 160"}
\end{array}
$$

$$
\begin{array}{l}
\mathit{prop\_wc\_w\_correctness1} :: \mathit{String} \rightarrow \mathit{Bool} \\
\mathit{prop\_wc\_w\_correctness1}\ \mathit{str} = \mathsf{length}\ \ (\mathit{words}\ \mathit{str}) \equiv \mathit{wc\_w\_final}\ \mathit{str} \\
\mathit{prop\_wc\_w\_correctness2} :: \mathit{String} \rightarrow \mathit{Bool} \\
\mathit{prop\_wc\_w\_correctness2}\ \mathit{str} = \mathit{wc\_w}\ \mathit{str} \equiv \mathit{wc\_w\_final}\ \mathit{str} \\
\mathit{prop\_worker\_first} :: \mathit{String} \rightarrow \mathit{Bool} \\
\mathit{prop\_worker\_first}\ [\,] = \mathit{True} \\
\mathit{prop\_worker\_first}\ \mathit{str}@(c : \_) = \mathit{sep}\ c \equiv (\pi_1\ (\mathit{worker}\ \mathit{str}))
\end{array}
$$

Segundo o enunciado pode-se deduzir que:

$$
\left\{
\begin{array}{l}
\mathit{wc\_w} \cdot \mathit{nil} = \underline{0} \\
\mathit{wc\_w} \cdot \mathit{cons} = x \rightarrow (\mathsf{succ}\ \cdot \mathit{wc\_w} \cdot \pi_2), (\mathit{wc\_w} \cdot \pi_2)
\end{array}
\right.
$$

, em que:

$$
\begin{array}{ll}
& x = \widehat{(\wedge)} \cdot \langle \neg \cdot \mathit{sep} \cdot \pi_1, \mathit{look} \cdot \pi_2 \rangle \\
= & \quad \{ \text{ Absorção-X (11) } \} \\
& x = \widehat{(\wedge)} \cdot (\neg \cdot \mathit{sep} \times \mathit{look}) \cdot \langle \pi_1, \pi_2 \rangle \\
= & \quad \{ \text{ Reflexão-X (8) } \} \\
& x = \widehat{(\wedge)} \cdot (\neg \cdot \mathit{sep} \times \mathit{look})
\end{array}
$$

Então temos que:

$=$ $\{$ Eq- + (27) $\}$

$wc\_w \cdot \mathbf{in} = [\underline{0}, (\widehat{(\wedge)} \cdot (\neg \cdot sep \times look)) \rightarrow (\mathsf{succ} \cdot wc\_w \cdot \pi_2), (wc\_w \cdot \pi_2)]$

$=$ $\{$ $look = \pi_1 \cdot \langle look, wc\_w \rangle$ e $wc\_w = \pi_2 \cdot \langle look, wc\_w \rangle$ $\}$

$$wc\_w \cdot \mathbf{in} = [\underline{0}, \begin{array}{l} \widehat{(\wedge)} \cdot (\neg \cdot sep \times \pi_1 \cdot \langle look, wc\_w \rangle) \rightarrow \\ \mathsf{succ} \cdot \pi_2 \cdot (\langle look, wc\_w \rangle \cdot \pi_2), \\ \pi_2 \cdot (\langle look, wc\_w \rangle \cdot \pi_2) \end{array}]$$

$=$ $\{$ Functor- X (14), Natural π_2 (13) $\}$

$$wc\_w \cdot \mathbf{in} = [\underline{0}, \begin{array}{l} \widehat{(\wedge)} \cdot (\neg \cdot sep \times \pi_1) \cdot (id \times \langle look, wc\_w \rangle) \rightarrow \\ \mathsf{succ} \cdot \pi_2 \cdot \pi_2 \cdot (id \times \langle look, wc\_w \rangle), \\ \pi_2 \cdot \pi_2 \cdot (id \times \langle look, wc\_w \rangle) \end{array}]$$

$=$ $\{$ 2ª Lei de Fusão do Condicional (32) $\}$

$wc\_w \cdot \mathbf{in} = [\underline{0}, ((\widehat{(\wedge)} \cdot (\neg \cdot sep \times \pi_1)) \rightarrow (\mathsf{succ} \cdot \pi_2 \cdot \pi_2), (\pi_2 \cdot \pi_2)) \cdot (id \times \langle look, wc\_w \rangle)]$

$=$ $\{$ Absorção- + (22) $\}$

$wc\_w \cdot \mathbf{in} = [\underline{0}, (\widehat{(\wedge)} \cdot (\neg \cdot sep \times \pi_1)) \rightarrow (\mathsf{succ} \cdot \pi_2 \cdot \pi_2), (\pi_2 \cdot \pi_2)] \cdot (id + (id \times \langle look, wc\_w \rangle))$

Vamos agora trabalhar a expressão de modo a que seja possivel utilizá-la na lei Fokkinga

$$\begin{cases} look \cdot nil = True \\ look \cdot cons = sep\ c \end{cases}$$

$=$ $\{$ Eq-+ (27), fusão-+(20), Natural-π_1(12) $\}$

$look \cdot \mathbf{in} = [True, sep \cdot \pi_1]$

$=$ $\{$ Eq- + (27) $\}$

$look \cdot \mathbf{in} = [True, sep \cdot \pi_1]$

$=$ $\{$ Natural-π_1 (12) $\}$

$look \cdot \mathbf{in} = [True, \pi_1 \cdot (sep \times look)]$

$=$ $\{$ Natural_id (1) Absorção- + (22) $\}$

$look \cdot \mathbf{in} = [True, \pi_1] \cdot (id + (sep \times look))$

$=$ $\{$ $look = \pi_1 \cdot \langle look, wc\_w \rangle$ $\}$

$look \cdot \mathbf{in} = [True, \pi_1] \cdot (id + (sep \times \pi_1 \cdot \langle look, wc\_w \rangle))$

$=$ $\{$ Natural_id (1), Functor-x (14) $\}$

$look \cdot \mathbf{in} = [True, \pi_1] \cdot (id + (sep \times \pi_1) \cdot (id \times \langle look, wc\_w \rangle))$

$=$ $\{$ Functor-+ (25) $\}$

$look \cdot \mathbf{in} = [True, \pi_1] \cdot (id + (sep \times \pi_1)) \cdot (id + (id \times \langle look, wc\_w \rangle))$

$=$ $\{$ Absorção-+ (22) $\}$

$look \cdot \mathbf{in} = [True, \pi_1 \cdot (sep \times \pi_1)] \cdot (id + (id \times \langle look, wc\_w \rangle))$

$=$ $\{$ Natural-π_1 (12) $\}$

$look \cdot \mathbf{in} = [True, sep \cdot \pi_1] \cdot (id + (id \times \pi_1 \cdot \langle look, wc\_w \rangle))$

Podemos concluir que wordAccum poderá ser representada da seguinte maneira:

$wordAccum = \langle sep \cdot \pi_1, (\widehat{(\wedge)} \cdot ((\neg \cdot sep) \times \pi_1)) \rightarrow (\mathsf{succ} \cdot \pi_2 \cdot \pi_2), (\pi_2 \cdot \pi_2) \rangle$

Portanto:

$$\begin{cases} look \cdot \mathbf{in} = [True, sep \cdot \pi_1] \cdot (id + (id \times \langle look, wc\_w \rangle)) \\ wc\_w \cdot \mathbf{in} = ([\underline{0}, (\widehat{(\wedge)} \cdot (\neg \cdot sep \times \pi_1))] \rightarrow \mathsf{succ} \cdot \pi_2 \cdot \pi_2, \pi_2 \cdot \pi_2) \cdot (id + (id \times \langle look, wc\_w \rangle)) \end{cases}$$

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

$$\langle look, wc\_w \rangle = cataList \ \langle [True, sep \cdot \pi_1], [\underline{0}, \widehat{(\wedge)} \cdot (\neg \cdot sep \times \pi_1) \rightarrow \mathsf{succ} \cdot \pi_2 \cdot \pi_2, \pi_2 \cdot \pi_2] \rangle$$

$= \quad \{$ Lei da Troca (28) $\}$

$$\langle look, wc\_w \rangle = cataList \ [\langle True, \underline{0} \rangle, \langle sep \cdot \pi_1, \widehat{(\wedge)} \cdot (\neg \cdot sep \times \pi_1) \rightarrow \mathsf{succ} \cdot \pi_2 \cdot \pi_2, \pi_2 \cdot \pi_2 \rangle]$$

$= \quad \{$ Def-split(78), def de wordAccum $\}$

$$\langle look, wc\_w \rangle = cataList \ [\underline{(True, 0)}, wordAccum]$$

$= \quad \{$ trivial $\}$

Problema 3

Para gerar o gráfico basta correr este comando: dot -Tpng digraphG.dot -o diagrama.png

$inB\_tree :: () + (\text{B-tree } a, [(a, \text{B-tree } a)]) \rightarrow (\text{B-tree } a)$
$inB\_tree = [\underline{Nil}, joinB\_trees]$
$joinB\_trees :: (\text{B-tree } a, [(a, \text{B-tree } a)]) \rightarrow \text{B-tree } a$
$joinB\_trees \ (t1, ts) = Block \ \{ leftmost = t1, block = ts \}$

Uma versão alternativa seria:

```
inB_tree :: Either () (B_tree a, [(a, B_tree a)]) -> B_tree a
inB_tree = either (const Nil) (uncurry Block)
```

$outB\_tree :: \text{B-tree } a \rightarrow () + (\text{B-tree } a, [(a, \text{B-tree } a)])$
$outB\_tree \ Nil = i_1 \ ()$
$outB\_tree \ (Block \ left \ block) = i_2 \ (left, block)$

$recB\_tree :: (b \rightarrow d) \rightarrow z + (b, [(a, b)]) \rightarrow z + (d, [(a, d)])$
$recB\_tree \ f = id + (f \times (\mathsf{map} \ (id \times f)))$

$baseB\_tree :: (a1 \rightarrow b1) \rightarrow (a \rightarrow d) \rightarrow b + (a, [(a1, a)]) \rightarrow b + (d, [(b1, d)])$
$baseB\_tree \ g \ f = id + (f \times (\mathsf{map} \ (g \times f)))$

$cataB\_tree :: (() + (d, [(a, d)]) \rightarrow d) \rightarrow \text{B-tree } a \rightarrow d$
$cataB\_tree \ g = g \cdot (recB\_tree \ (cataB\_tree \ g)) \cdot outB\_tree$
$anaB\_tree :: (b \rightarrow () + (b, [(a, b)])) \rightarrow b \rightarrow \text{B-tree } a$
$anaB\_tree \ g = inB\_tree \cdot (recB\_tree \ (anaB\_tree \ g)) \cdot g$
$hyloB\_tree :: (() + (c, [(a, c)]) \rightarrow c) \rightarrow (b \rightarrow () + (b, [(a, b)])) \rightarrow b \rightarrow c$
$hyloB\_tree \ f \ g = cataB\_tree \ f \cdot anaB\_tree \ g$

instance $Functor$ B-tree
  where fmap $f = cataB\_tree \ (inB\_tree \cdot baseB\_tree \ f \ id)$

  -- esq meio dir
$inordB\_tree :: \text{B-tree } t \rightarrow [t]$
$inordB\_tree = cataB\_tree \ inorder\_g$
$inorder\_g :: () + ([a], [(a, [a])]) \rightarrow [a]$

$$inorder\_g = [\underline{[\,]}, join]$$
$$\quad \textbf{where } join :: ([\,a\,], [(a, [\,a\,])]) \rightarrow [\,a\,]$$
$$\qquad join\ (xs, ys) = xs +\!\!+\ concat\ (\mathsf{map}\ \ (\lambda(n, zs) \rightarrow (n : zs))\ ys)$$

$$largestBlock :: \mathsf{B\text{-}tree}\ t \rightarrow Int$$
$$largestBlock = cataB\_tree\ [\underline{0}, size]$$
$$\quad \textbf{where } size :: (Int, [(a, Int)]) \rightarrow Int$$
$$\qquad size\ (x, ys) = max\ x\ (\mathsf{length}\ \ ys)$$

$$mirrorB\_tree :: \mathsf{B\text{-}tree}\ a \rightarrow \mathsf{B\text{-}tree}\ a$$
$$mirrorB\_tree = cataB\_tree\ (inB\_tree \cdot (id + (id \times reverse)))$$

$$dotB\_tree :: Show\ a \Rightarrow \mathsf{B\text{-}tree}\ a \rightarrow \mathsf{IO}\ ExitCode$$
$$dotB\_tree = dotpict \cdot bmap\ nothing\ (Just \cdot show) \cdot cB\_tree2Exp$$
$$cB\_tree2Exp :: \mathsf{B\text{-}tree}\ a \rightarrow Exp\ [\,Char\,]\ [\,a\,]$$
$$cB\_tree2Exp = cataB\_tree\ [\underline{(Var\ \texttt{"nil"})}, connect]$$
$$connect :: (Exp\ v\ [\,a\,], [(a, Exp\ v\ [\,a\,])]) \rightarrow Exp\ v\ [\,a\,]$$
$$connect\ (x, xs) = Term\ fsts\ (x : snds)$$
$$\quad \textbf{where } (fsts, snds) = unzip\ xs$$

$$qSortB\_tree :: Ord\ a \Rightarrow [\,a\,] \rightarrow [\,a\,]$$
$$qSortB\_tree = hyloB\_tree\ inorder\_g\ lsplitB\_tree$$
$$lsplitB\_tree :: Ord\ a \Rightarrow [\,a\,] \rightarrow () + ([\,a\,], [(a, [\,a\,])])$$
$$lsplitB\_tree\ [\,] = i_1\ ()$$
$$lsplitB\_tree\ (x : xs) = i_2\ (l1, [(x, l2)])$$
$$\quad \textbf{where } (l1, l2) = partB\_tree\ (<x)\ xs$$
$$partB\_tree :: (a \rightarrow Bool) \rightarrow [\,a\,] \rightarrow ([\,a\,], [\,a\,])$$
$$partB\_tree = partition$$

Mostramos de seguida como proceder à definição da árvore:

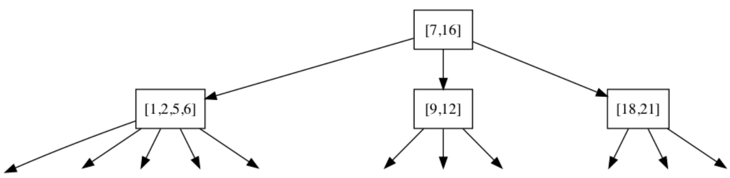
$$t = Block\ \{\,leftmost = Block\ \{\,leftmost = Nil, block = [(1, Nil), (2, Nil), (5, Nil), (6, Nil)]\,\}, block = [(7, Block\ \{\,lef$$

Seguido do comando para a geração do ficheiro .dot:

dotB_tree t

Para a obtenção da imagem é necessário correr o seguinte comando:

dot -Tpng digraphG.dot -o diagrama.png



Problema 4

$$[\!(\cdot\ \cdot)\!]_A :: (a \rightarrow 1 + (a, b)) \rightarrow (b \rightarrow 1 + a) \rightarrow a \rightarrow A$$
$$[\!(ga\ gb)\!]_A = inA \cdot (id + [\!(ga\ gb)\!]_A \times [\!(ga\ gb)\!]_B) \cdot ga$$
$$[\!(\cdot\ \cdot)\!]_B :: (a \rightarrow 1 + (a, b)) \rightarrow (b \rightarrow 1 + a) \rightarrow b \rightarrow B$$
$$[\!(ga\ gb)\!]_B = inB \cdot (id + [\!(ga\ gb)\!]_A) \cdot gb$$

$$
\begin{array}{l}
generateAlgae :: Int \rightarrow Algae \\
generateAlgae = [\!(ga\ gb)\!]_A \\
\quad \textbf{where}\ ga :: Int \rightarrow 1 + Int \times Int \\
\qquad ga\ 0 = i_1\ () \\
\qquad ga\ n = i_2\ (pred\ n, pred\ n) \\
\qquad gb :: Int \rightarrow 1 + Int \\
\qquad gb\ 0 = i_1\ () \\
\qquad gb\ n = i_2\ (pred\ n)
\end{array}
$$

$$
\begin{array}{l}
showAlgae :: Algae \rightarrow String \\
showAlgae = (\!| ga\ gb |\!)_A \\
\quad \textbf{where}\ ga :: a + (String, String) \rightarrow String \\
\qquad ga = [\underline{\texttt{"A"}}, \widehat{(+\!\!+)}] \\
\qquad gb :: a + String \rightarrow String \\
\qquad gb = [\underline{\texttt{"B"}}, id]
\end{array}
$$

$$
\begin{array}{l}
prop\_algae :: Property \\
prop\_algae = forAll\ (elements\ [0\,..\,25])\ \$\ \lambda n \rightarrow (\mathsf{length}\ \cdot showAlgae \cdot generateAlgae)\ n \equiv (fromIntegral\ ((fib \cdot \mathsf{suc}
\end{array}
$$

Problema 5

$$
\begin{array}{l}
permuta :: [\,a\,] \rightarrow \mathsf{IO}\ [\,a\,] \\
permuta\ [\,] = return\ [\,] \\
permuta\ input = \textbf{do} \\
\quad (x, xs) \leftarrow getR\ input \\
\quad permuted \leftarrow permuta\ xs \\
\quad return\ (x : permuted)
\end{array}
$$

$$
\begin{array}{l}
eliminatoria :: \mathsf{LTree}\ Equipa \rightarrow \mathsf{Dist}\ Equipa \\
eliminatoria = cataLTree\ g \\
\quad \textbf{where}\ g :: Equipa + (\mathsf{Dist}\ Equipa, \mathsf{Dist}\ Equipa) \rightarrow \mathsf{Dist}\ Equipa \\
\qquad g = [return, distJogo] \\
\qquad distJogo :: (\mathsf{Dist}\ Equipa, \mathsf{Dist}\ Equipa) \rightarrow \mathsf{Dist}\ Equipa \\
\qquad distJogo\ (de1, de2) = \textbf{do} \\
\qquad\quad e_1 \leftarrow de1 \\
\qquad\quad e_2 \leftarrow de2 \\
\qquad\quad jogo\ (e_1, e_2)
\end{array}
$$

Índice