Cálculo de Programas
Trabalho Prático
LCC+LEI — Ano Lectivo de 2014/15

Departamento de Informática
Universidade do Minho

Junho de 2015

Conteúdo

1 Preâmbulo **2**

2 Documentação **2**

3 Como realizar o trabalho **3**

4 Parte A **3**
4.1 Biblioteca LTree 3
4.2 Biblioteca BTree 4
4.3 Biblioteca para listas com sentinelas 4

5 Parte B **5**
5.1 Criação de Triângulos de Sierpinski 5
5.2 Trabalho a realizar 6

6 Parte C **8**
6.1 Mónades 8
6.2 Trabalho a realizar 9
6.3 Programação funcional paralela 10
6.4 Trabalho a realizar 11

A Programa principal **12**

B Bibliotecas e código auxiliar **12**
B.1 “Easy X3DOM access” 12

C Soluções propostas **13**

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 usa-se esses combinadores para construir programas *composicionalmente*, isto é, compondo 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 e simples os objectivos enunciados acima vamos recorrer a uma técnica de programação dita literária [**?**], cujo princípio base é o seguinte:

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

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

O ficheiro `cp1415t.pdf` que está a ler é já um exemplo de programação literária: foi gerado a partir do texto fonte `cp1415t.lhs`[1] que encontrará no material pedagógico desta disciplina descompactando o ficheiro `cp1415t.zip` e executando

```
lhs2TeX cp1415t.lhs > cp1415t.tex
pdflatex cp1415t
```

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 `cp1415t.lhs` é executável e contém o "kit" básico, escrito em Haskell, para realizar o trabalho. Basta executar

```
ghci cp1415t.lhs
```

para ver que assim é:

```
GHCi, version 7.8.3: http://www.haskell.org/ghc/  :? for help
Loading package ghc-prim ... linking ... done.
Loading package integer-gmp ... linking ... done.
Loading package base ... linking ... done.
[ 1 of 11] Compiling ListUtils        ( ListUtils.hs, interpreted )
[ 2 of 11] Compiling Cp               ( Cp.hs, interpreted )
[ 3 of 11] Compiling BTree            ( BTree.hs, interpreted )
[ 4 of 11] Compiling LTree            ( LTree.hs, interpreted )
[ 5 of 11] Compiling Exp              ( Exp.hs, interpreted )
[ 6 of 11] Compiling Nat              ( Nat.hs, interpreted )
[ 7 of 11] Compiling Show             ( Show.hs, interpreted )
[ 8 of 11] Compiling Probability      ( Probability.hs, interpreted )
[ 9 of 11] Compiling List             ( List.hs, interpreted )
[10 of 11] Compiling X3d              ( X3d.hs, interpreted )
[11 of 11] Compiling Main             ( cp1415t.lhs, interpreted )
Ok, modules loaded: List, Show, Nat, Exp, Cp, BTree, LTree, X3d,
Probability, Main, ListUtils.
```

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:

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

$\mathbf{import}\ \mathit{Data.List}$
$\mathbf{import}\ \mathit{System.Process}$
$\mathbf{import}\ \mathit{Cp}$
$\mathbf{import}\ \mathit{List}$
$\mathbf{import}\ \mathit{Nat}$
$\mathbf{import}\ \mathit{Exp}$
$\mathbf{import}\ \mathit{BTree}$
$\mathbf{import}\ \mathit{LTree}$
$\mathbf{import}\ \mathit{X3d}$
$\mathbf{import}\ \mathit{Control.Parallel.Strategies}$
$\mathbf{import}\ \mathit{Probability}\ \mathit{hiding}\ (\cdot \rightarrow \cdot, \cdot)$
$\mathbf{import}\ \mathit{System.Environment}\ (\mathit{getArgs})$

Abra o ficheiro `cp1415t.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 respostas. O relatório deverá conter ainda a identificação dos membros do grupo de trabalho, na 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 cp1415t.aux
makeindex cp1415t.idx
```

e recompilar o texto como acima se indicou.

4 Parte A

Nesta primeira parte do trabalho pretende-se averiguar a capacidade de utilização por parte dos alunos das bibliotecas fornecidas no material pedagógico da disciplina. Algumas respostas são validadas por testes unitários. Sempre que o resultado de um teste unitário for False, a solução proposta falha a validação e deve ser revista.

4.1 Biblioteca LTree

1. A seguinte função

$$\mathit{balanced}\ (\mathit{Leaf}\ \_) = \mathit{True}$$
$$\mathit{balanced}\ (\mathit{Fork}\ (t, t')) = \mathit{balanced}\ t \wedge \mathit{balanced}\ t' \wedge \mathit{abs}\ (\mathit{depth}\ t - \mathit{depth}\ t') \leqslant 1$$

testa se uma árvore binária está equilibrada ou não. Defina como catamorfismo em LTree a função auxiliar depth.

2. Seja dada:

$$t = \mathit{Fork}\ (\mathit{Fork}\ (\mathit{Leaf}\ 10, \mathit{Fork}\ (\mathit{Leaf}\ 2, \mathit{Fork}\ (\mathit{Leaf}\ 5, \mathit{Leaf}\ 3))), \mathit{Leaf}\ 23)$$

Testes unitários 1 *Verifique que árvore t está desequilibrada:*

$$test01 = balanced\ t \equiv False$$

3. Recorrendo a funções da biblioteca LTree, escreva numa única linha de Haskell a função

$$balance :: LTree\ a \rightarrow LTree\ a$$

que equilibra uma qualquer árvore binária.

Testes unitários 2 *Verifique que $balance\ t$ é uma árvore equilibrada:*

$$test02 = balanced\ (balance\ t) \equiv True$$

4.2 Biblioteca BTree

Pretende-se construir um anamorfismo que produza uma árvore binária de procura *equilibrada* que contenha o intervalo definido por dois inteiros (n, m):

$$abpe\ (n, m) = anaBTree\ qsplit\ (n, m)$$

Comece por definir o gene $qsplit$ e depois construa a árvore

$$t1 = abpe\ (20, 30)$$

que será precisa na secção 6.4.

Testes unitários 3 *Faça os testes seguintes:*[2]

$$\begin{aligned}
&test03a = qsplit\ (4, 30) \equiv i_2\ (17, ((4, 16), (18, 30)))\\
&test03b = qsplit\ (4, 3) \equiv i_1\ ()\\
&test03c = qsplit\ (0, 0) \equiv i_1\ ()\\
&test03d = qsplit\ (1, 1) \equiv i_2\ (1, ((1, 0), (2, 1)))\\
&test03e = balBTree\ t1 \equiv True\\
&test03f = inordt\ t1 \equiv [20\mathinner{..}30]\\
&test03 = test03a \wedge test03b \wedge test03c \wedge\\
&\qquad\qquad test03d \wedge test03e \wedge test03f
\end{aligned}$$

4.3 Biblioteca para listas com sentinelas

Considere o tipo de dados que representa listas finitas com uma sentinela no fim:

$$\mathbf{data}\ SList\ a\ b = Sent\ b \mid Cons\ (a, SList\ a\ b)\ \mathbf{deriving}\ (Show, Eq)$$

1. Derive os isomorfismos $inSList$ e $outSList$, adicione-os a este ficheiro e passe aos testes que se seguem.

   Testes unitários 4 *Faça os testes seguintes:*

$$\begin{aligned}
&test04a = \mathbf{let}\ x = Cons\ (1, Sent\ \texttt{"end"})\ \mathbf{in}\ inSList\ (outSList\ x) \equiv x\\
&test04b = \mathbf{let}\ x = i_2\ (\texttt{"ola"}, Sent\ \texttt{"2"})\ \mathbf{in}\ outSList\ (inSList\ x) \equiv x\\
&test04 = test04a \wedge test04b
\end{aligned}$$

2. Derive os combinadores $cataSList$, $anaSList$ e $hyloSList$, e mostre que a função $merge$ da biblioteca LTree se pode escrever da forma seguinte,

$$\begin{aligned}
&merge' :: Ord\ a \Rightarrow ([a], [a]) \rightarrow [a]\\
&merge' = hyloSList\ [id, cons]\ mgen
\end{aligned}$$

   para um dado gene $mgen$ que deverá definir.

[2]Foi adicionado uma função test03 que é a conjunção de todos os testes 3.

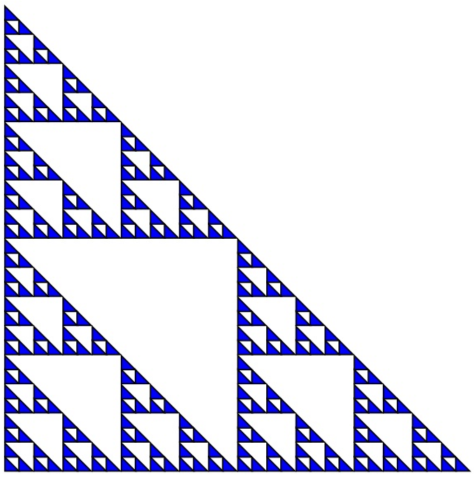


Figura 1: Um triângulo de Sierpinski

Testes unitários 5 *Faça os seguintes testes:*

$$\begin{aligned}
&test05a = mgen\ ([0,2,5],[0,6]) \equiv i_2\ (0,([2,5],[0,6]))\\
&test05b = mgen\ ([0,2,5],[\,]) \equiv i_1\ [0,2,5]\\
&test05c = merge'\ ([\,],[0,6]) \equiv [0,6]\\
&test05 = test05a \wedge test05b \wedge test05c
\end{aligned}$$

5 Parte B

O triângulo de Sierpinski é uma figura fractal que tem o aspecto da figura 1 e que se obtém da seguinte forma: considere-se um triângulo rectângulo e isósceles A cujos catetos têm comprimento s. A estrutura fractal é criada desenhando-se três triângulos no interior de A, todos eles rectângulos e isósceles e com catetos de comprimento $s/2$. Este passo é depois repetido para cada um dos triângulos desenhados, e assim sucessivamente. O resultado dos cinco primeiros passos é dado na Fig. 1.

Um triângulo de Sierpinski é gerado repetindo-se infinitamente o processo acima descrito. No entanto, para efeitos de visualização num monitor, cuja resolução é forçosamente finita, faz sentido escolher uma representação adequada do triângulo, parando o processo recursivo a um determinado nível. A figura a desenhar é constituída por um conjunto finito de triângulos todos da mesma dimensão (por exemplo, na figura 1 há 243 triângulos).

5.1 Criação de Triângulos de Sierpinski

Seja cada triângulo geometricamente descrito pelas coordenadas do seu vértice inferior esquerdo e o comprimento dos seus catetos:

$$\textbf{type}\ \mathit{Tri} = (\mathit{Point}, \mathit{Side})$$

onde

$$\begin{aligned}
&\textbf{type}\ \mathit{Side} = \mathit{Int}\\
&\textbf{type}\ \mathit{Point} = (\mathit{Int}, \mathit{Int})
\end{aligned}$$

A estrutura recursiva de (uma representação finita de) um triângulo de Sierpinski é captada por uma árvore ternária, em que cada nó é um triângulo com os respectivos três sub-triângulos: [3]

$$\textbf{data}\ \mathsf{TLTree} = \mathit{Tri}\ \ \mathit{Tri} \mid \mathit{Nodo}\ \mathsf{TLTree}\ \mathsf{TLTree}\ \mathsf{TLTree}\ \textbf{deriving}\ (\mathit{Eq}, \mathit{Show})$$

[3] Foi usada esta definição de TLTree em vez da que foi dada abaixo, porque o grupo começou a fazer com esta definição e só vimos a alternativa no final do trabalho.

Nas folhas dessa árvore encontram-se os triângulos mais pequenos, todos da mesma dimensão, que deverão ser desenhados. Apenas estes conterão informação de carácter geométrico, tendo os nós da árvore um papel exclusivamente estrutural. Portanto, a informação geométrica guardada em cada folha consiste nas coordenadas do vértice inferior esquerdo e no lado dos catetos do respectivo triângulo. A função

$$
\begin{aligned}
&\mathit{sierpinski} :: \mathit{Tri} \to \mathit{Int} \to [\,\mathit{Tri}\,]\\
&\mathit{sierpinski}\ t = \mathit{apresentaSierp} \cdot (\mathit{geraSierp}\ t)
\end{aligned}
$$

recebe a informação do triângulo exterior e o número de níveis pretendido, que funciona como critério de paragem do processo de construção do fractal. O seu resultado é a lista de triângulos a desenhar. Esta função é um hilomorfismo do tipo TLTree, i.e. a composição de duas funções: uma que gera TLTrees,

$$
\begin{aligned}
&\mathit{geraSierp} :: \mathit{Tri} \to \mathit{Int} \to \mathsf{TLTree}\\
&\mathit{geraSierp}\ t\ 0 = \mathit{Tri}\ t\\
&\mathit{geraSierp}\ ((x,y),s)\ n =\\
&\quad \mathbf{let}\ s' = s \div 2\\
&\quad \mathbf{in}\ \mathit{Nodo}\\
&\qquad\quad (\mathit{geraSierp}\ ((x,y),s')\ (n-1))\\
&\qquad\quad (\mathit{geraSierp}\ ((x+s',y),s')\ (n-1))\\
&\qquad\quad (\mathit{geraSierp}\ ((x,y+s'),s')\ (n-1))
\end{aligned}
$$

e outra que as consome:

$$
\begin{aligned}
&\mathit{apresentaSierp} :: \mathsf{TLTree} \to [\,\mathit{Tri}\,]\\
&\mathit{apresentaSierp}\ (\mathit{Tri}\ t) = [\,t\,]\\
&\mathit{apresentaSierp}\ (\mathit{Nodo}\ a\ b\ c) = (\mathit{apresentaSierp}\ a) +\!\!+ (\mathit{apresentaSierp}\ b) +\!\!+ (\mathit{apresentaSierp}\ c)
\end{aligned}
$$

5.2 Trabalho a realizar

Preparação:

1. Desenvolva a biblioteca "pointfree" `TLTree.hs` de forma análoga a outras bibliotecas que conhece (*eg.* BTree, LTree, etc) e que estão disponíveis no material pedagógico.

2. Defina como catamorfismos de TLTree as funções

$$
\begin{aligned}
&\mathit{tipsTLTree} :: \mathsf{TLTree}\ b \to [\,b\,]\\
&\mathit{countTLTree} :: \mathsf{TLTree}\ b \to \mathit{Int}\\
&\mathit{depthTLTree} :: \mathsf{TLTree}\ b \to \mathit{Int}\\
&\mathit{invTLTree} :: \mathsf{TLTree}\ b \to \mathsf{TLTree}\ b
\end{aligned}
$$

   respectivamente semelhantes a *tips*, *countLTree*, *depth* e *inv* ("mirror") de LTree.

3. Exprima as funções *geraSierp* e *apresentaSierp* recorrendo a anamorfismos e catamorfismos, respectivamente, do tipo TLTree.

4. Defina a árvore

$$
\mathit{ts} = \mathit{geraSierp}\ \mathit{tri}\ 5\ \mathbf{where}\ \mathit{tri} = ((0,0),256)
$$

   e faça os testes seguintes:[4]

   Testes unitários 6 *Verifique a profundidade da árvore gerada e o respectivo número de triângulos:*

$$
\begin{aligned}
&\mathit{test06a} = \mathit{depthTLTree}\ \mathit{ts} \equiv 6\\
&\mathit{test06b} = \mathit{countTLTree}\ \mathit{ts} \equiv 243\\
&\mathit{test06c} = \mathit{countTLTree}\ \mathit{ts} \equiv \mathit{genericLength}\ (\mathit{tipsTLTree}\ \mathit{ts})\\
&\mathit{test06d} = \mathit{countTLTree}\ \mathit{ts} \equiv \mathit{countTLTree}\ (\mathit{invTLTree}\ \mathit{ts})\\
&\mathit{test06} = \mathit{test06a} \wedge\\
&\quad \mathit{test06b} \wedge\\
&\quad \mathit{test06c} \wedge\\
&\quad \mathit{test06d}
\end{aligned}
$$

[4]Foi adicionado uma função test03 que é a conjunção de todos os testes 3.

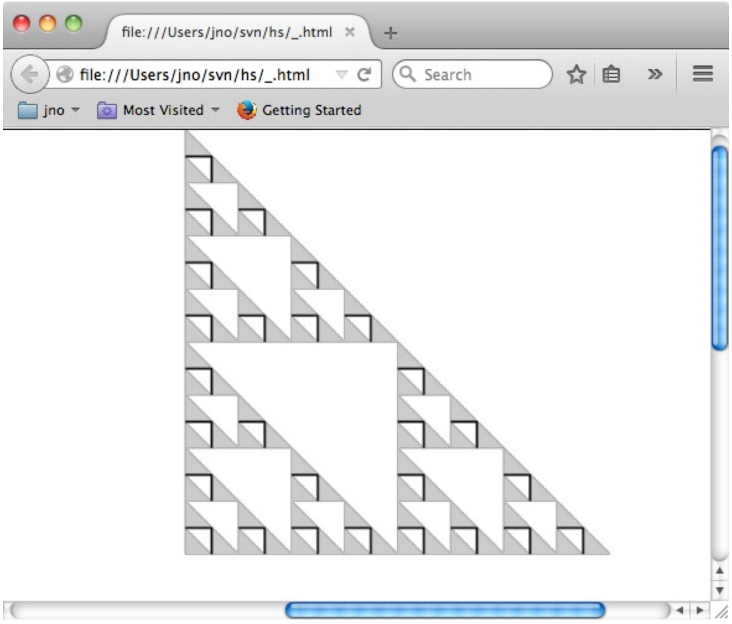


Figura 2: Um triângulo de Sierpinski em x3dom

Visualização: Para visualizarmos triângulos de Sierpinski vamos usar X3DOM, uma biblioteca "open-source" para construção e visualização de gráficos 3D no Web.[5] No pacote disponibilizado para a realização deste trabalho encontra a biblioteca $X3d$, que inclui a função $drawTriangle$ para geração de triângulos em 3D, usando X3DOM. Nesta abordagem, um ficheiro $x3dom$ é construído em dois passos:

- Desenham-se os triângulos, utilizando:

$$drawTriangle :: ((Int, Int), Int) \rightarrow String$$

- Finaliza-se o ficheiro com as tags de início e final:

$$finalize :: String \rightarrow String$$

1. Usando estas funções e as que definiu anteriormente, faça a geração do HTML que representa graficamente o triângulo de Sierpinski definido por

$$dados = (((0,0), 32), 4)$$

isto é, centrado na origem, com lado 32 e 4 níveis de recursividade. No anexo C sugere-se o recurso à função,

$$render\ html = \mathbf{do}\ \{ writeFile\ \texttt{"\_.html"}\ html; system\ \texttt{"chromium \_.html"} \}$$

(adapte-a, se necessário) para visualizar o triângulo gerado num "browser". Espera-se que o resultado final seja como o que se mostra na Figura 2.

Valorização

Se tiver tempo, investigue como é que a sua resolução desta parte do trabalho evolui para o desenho, não de *triângulos* de Sierpinski, mas sim de *pirâmides* de Sierpinski — ver a imagem da figura 3. Pode recorrer, se desejar, às funções disponibilizadas no anexo B.1.

[5] Ver `http://examples.x3dom.org` para mais informação. Em `http://examples.x3dom.org/IG/buddha-anim/x3dom_imageGeometry.html`, por exemplo, pode ser visualizado um objecto gráfico com mais de um milhão de triângulos. Mais documentação em: `http://doc.x3dom.org/tutorials/index.html`.

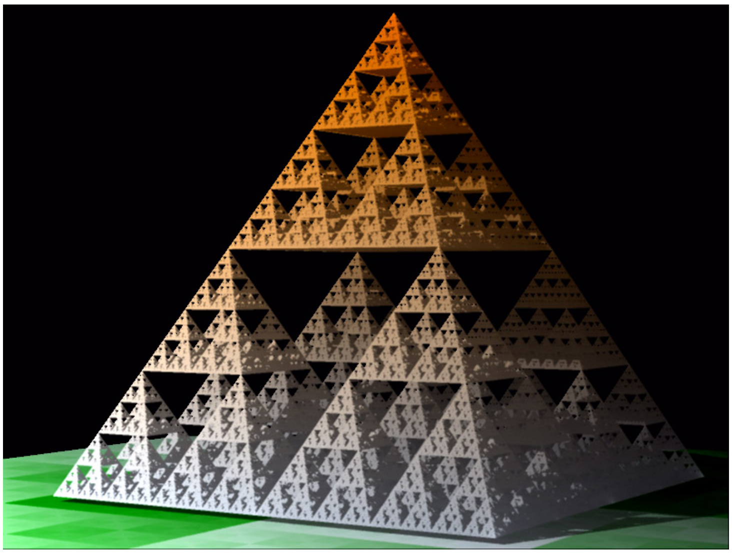


Figura 3: Uma pirâmide de Sierpinski

6 Parte C

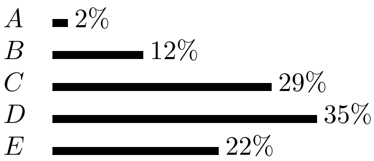
6.1 Mónades

Os 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

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

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

Cada par (a, p) numa distribuição $d :: \mathit{Dist}\ \ a$ 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

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

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*,

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

isto é

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

distribuição *normais*, eg.

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

etc.[6]

$Dist$ forma um **mónade** cuja unidade é $return\ a = D\ [(a, 1)]$ e cuja multiplicação é dada por (simplificando a notação)

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

em que $g : A \to Dist\ B$ e $f : B \to Dist\ C$ 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. Vejamos um exemplo:

Problema: qual é a soma de faces mais provável quando lançamos dois dados num tabuleiro?

Assumindo que os dados não estão viciados, cada um oferece uma distribuição uniforme das suas faces (1 a 6). Basta correr a expressão monádica

$$\mathbf{do}\ \{x \leftarrow uniform\ [1\,..\,6]; y \leftarrow uniform\ [1\,..\,6]; return\ (x + y)\}$$

e obter-se-á:

```
*Main> do { x <- uniform [1..6] ;  y <- uniform [1..6] ; return(x+y) }
 7  16.7%
 6  13.9%
 8  13.9%
 5  11.1%
 9  11.1%
 4   8.3%
10   8.3%
 3   5.6%
11   5.6%
 2   2.8%
12   2.8%
```

A soma mais provável é 7, com 16.7%.

6.2 Trabalho a realizar

É possível pensarmos em catamorfismos, anamorfismos etc probabilísticos, quer dizer, programas recursivos que dão distribuições como resultados. Por exemplo, neste enunciado é dado o combinador

$$pcataList :: (Either\ ()\ (a, b) \to Dist\ b) \to [a] \to Dist\ b$$

que é muito parecido com

$$cataList :: (Either\ ()\ (a, b) \to b) \to [a] \to b$$

da biblioteca List. A única diferença é que o gene de $pcataList$ é uma função probabilística.

Exemplo de utilização: recorde-se que $cataList\ [zero, add]$ soma todos os elementos da lista argumento, por exemplo:

$$cataList\ [zero, add]\ [20, 10, 5] = 35.$$

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

Considere agora a função $padd$ (adição probabilística) que, com probabilidade 90% soma dois números e com probabilidade 10% os subtrai:

$$padd\ (a, b) = D\ [(a + b, 0.9), (a - b, 0.1)]$$

Se se correr

$$d4 = pcataList\ [pzero, padd]\ [20, 10, 5]\ \textbf{where}\ pzero = return \cdot zero$$

obter-se-á:

```
35  81.0%
25   9.0%
 5   9.0%
15   1.0%
```

Com base nestes exemplos, resolva o seguinte

***Problema**: Uma unidade militar pretende enviar uma mensagem urgente a outra, mas tem o aparelho de telegrafia meio avariado. Por experiência, o telegrafista sabe que a probabilidade de uma palavra se perder (não ser transmitida) é 5%; no final de cada mensagem, o aparelho envia o código* `"stop"`*, mas (por estar meio avariado), falha 10% das vezes.*

Qual a probabilidade de a palavra `"atacar"` *da mensagem words* `"Vamos atacar hoje"` *se perder, isto é, o resultado da transmissão ser* `["Vamos","hoje","stop"]`*? e a de seguirem todas as palavras, mas faltar o* `"stop"` *no fim? E a da transmissão ser perfeita?*

Responda a todas estas perguntas encontrando g tal que

$$transmitir = (pcataList\ gene) \cdot (+\!\!+[\texttt{"stop"}])$$

descreve o comportamento do aparelho.[7]

<u>Testes unitários</u> 7 *Faça o seguinte teste unitário da sua versão para gene:*

$$test07 = gene\ (i_2\ (\texttt{"a"}, [\texttt{"b"}])) \equiv D\ [([\texttt{"a"}, \texttt{"b"}], 0.95), ([\texttt{"b"}], 0.05)]$$

Responda então às perguntas do problema acima correndo a expressão:

$$transmitir\ (words\ \texttt{"Vamos atacar hoje"})$$

6.3 Programação funcional paralela

Uma outra aplicação do conceito de mónade é a programação funcional paralela. A biblioteca Control.Parallel.Strategies, já carregada no início deste texto, implementa esse tipo de programação, que hoje está na ordem do dia. O mónade respectivo chama-se $Eval$ e disponibiliza duas funções,

$$rpar :: a \rightarrow Eval\ a$$
$$rseq :: a \rightarrow Eval\ a$$

conforme se deseja que uma dada computação seja efectuada em paralelo ou sequencialmente.[8] Por exemplo,

$$\begin{array}{l} parmap :: (a \rightarrow b) \rightarrow [a] \rightarrow Eval\ [b] \\ parmap\ f\ [\,] = return\ [\,] \\ parmap\ f\ (a : lt) = \textbf{do} \\ \quad a' \leftarrow rpar\ (f\ a) \\ \quad lt' \leftarrow parmap\ f\ lt \\ \quad return\ (a' : lt') \end{array}$$

é um map monádico que usa $rpar$ para aplicar f a todos os elementos de uma lista *em paralelo*.

Se corrermos o map habitual em

[7] Esta função foi modificado de modo a meter sempre um stop no final na mensagem automaticamente.

[8] Esta explicação é bastante simplista, mas serve de momento. Para uma abordagem completa e elucidativa ver a referência [**?**].

$$map\ fib\ [20\mathinner{\ldotp\ldotp}30] = [10946, 17711, 28657, 46368, 75025, 121393, 196418, 317811, 514229, 832040, 1346269]$$

(cálculo dos números de Fibonacci do vigésimo ao trigésimo), o tempo que o cálculo vai demorar numa máquina com 2 cores[9] será da ordem de 1.1s. Já no caso de usar $parmap$ em vez de map, fará o mesmo cálculo em cerca de 60% desse tempo.

Para verificar esta diferença siga as instruções seguintes:[10]

1. Compile o presente enunciado correndo:

   ```
   ghc -O2 cp1415t -rtsopts -threaded
   ```

2. De seguida execute numa "shell" o seguinte comando,

   ```
   ./cp1415t exemplo seq +RTS -s -N2
   ```

   onde o 2 em $N2$ indica 2 *cores* (se a máquina em questão tiver mais *cores*, este número deverá ser actualizado). Como pode ver inspecionando o código da função $main$ na secção A, o que vai ser executado é

   $$putStrLn \cdot show \cdot (map\ fib)\ \$\ [20\mathinner{\ldotp\ldotp}30]$$

   Das estatísticas que lhe aparecem no écran retenha esta:

   ```
   Total   time    1.41s  (  1.11s elapsed)
   ```

   Em particular, o campo $elapsed$ apresenta o tempo decorrido desde o início da execução do programa até ao respectivo fim.

3. De seguida execute

   ```
   ./cp1415t exemplo par +RTS -s -N2
   ```

   que irá chamar, desta vez

   $$putStrLn \cdot show \cdot runEval \cdot (parmap\ fib)\ \$\ [20\mathinner{\ldotp\ldotp}30]$$

   A estatística correspondente à de cima será, desta vez, da ordem seguinte:

   ```
   Total   time    1.13s  (  0.69s elapsed)
   ```

   Em suma, a versão paralela é cerca de 1.61x mais rápida ($\frac{1.11}{0.69}$) que a sequencial.

6.4 Trabalho a realizar

Com base na definição de $parmap$ acima, defina a função

$$parBTreeMap :: (a \rightarrow b) \rightarrow (BTree\ a) \rightarrow Eval\ (BTree\ b)$$

que implemente o "map paralelo" sobre BTree's.

De seguida, corra testes semelhantes aos apresentados acima para apurar o ganho em *performance* da aplicação da função fib a todos os números da árvore $t1$ da secção 4.2, em duas versões:

1. $fmap\ fib$ (sem paralelismo, usando a função definida em BTree), ou
2. usando $parBTreeMap\ fib$.

Em máquinas mais rápidas e/ou com mais "cores" deve usar números maiores para obter uma melhor distinção entre as duas versões.

[9] Intel Core 2 Duo a 2.53 GHz.
[10] Ver detalhes em [?].

Anexos

A Programa principal

$$
\begin{array}{l}
main :: IO\ ()\\
main = getArgs \gg\!\!= (\neg \cdot null) \rightarrow exemp\_or\_exer, errInvArgs\\
\quad \textbf{where}\\
\quad\quad exemp\_or\_exer = (((\equiv)\ \texttt{"exemplo"}) \cdot head) \rightarrow exemp, exer\\
\quad\quad exemp = (((\equiv)\ 2) \cdot length) \rightarrow execExemp, errInvArgs\\
\quad\quad execExemp = isPar \rightarrow execExempPar, execExempSeq\\
\quad\quad execExempSeq = \underline{(putStrLn \cdot show \cdot (map\ fib)\ \$\ [20\,..\,30])}\\
\quad\quad execExempPar = \underline{(putStrLn \cdot show \cdot runEval \cdot (parmap\ fib)\ \$\ [20\,..\,30])}\\
\quad\quad exer = (((\equiv)\ 3) \cdot length) \rightarrow execExer, errInvArgs\\
\quad\quad execExer = isPar \rightarrow execExerPar, execExerSeq
\end{array}
$$

B Bibliotecas e código auxiliar

$$
\begin{array}{l}
errInvArgs :: a \rightarrow IO\ ()\\
errInvArgs = \underline{\cdot}\ \$\ putStrLn\ msgInvArgs\\
\quad \textbf{where}\\
\quad msgInvArgs = \texttt{"Invalid arguments"}\\
execExerPar :: [String] \rightarrow IO\ ()\\
execExerPar = \underline{(putStrLn \cdot show \cdot runEval \cdot (parBTreeMap\ fib)\ \$\ t1)}\\
execExerSeq :: [String] \rightarrow IO\ ()\\
execExerSeq = \underline{(putStrLn \cdot show \cdot (fmap\ fib)\ \$\ t1)}\\
isPar :: [String] \rightarrow Bool\\
isPar = (((\equiv)\ \texttt{"par"}) \cdot head \cdot tail) \rightarrow \underline{True}, \underline{False}\\
pcataList\ g = mfoldr\ (curry\ (g \cdot i_2))\ ((g \cdot i_1)\ ())\ \textbf{where}\\
\quad mfoldr\ f\ d\ [\,] = d\\
\quad mfoldr\ f\ d\ (a : x) = \textbf{do}\ \{y \leftarrow mfoldr\ f\ d\ x; f\ a\ y\}
\end{array}
$$

B.1 "Easy X3DOM access"

Defina-se a seguinte composição de funções

$$x3dom = html \cdot preamble \cdot body \cdot x3d \cdot scene \cdot items$$

para gerar um texto HTML que represente um objecto gráfico em X3DOM. Esta função usa as seguintes funções auxiliares:[11]

$$
\begin{array}{l}
html = tag\ \texttt{"html"}\ [\,]\\
preamble\ rest = (headx\ \$\ concat\ [title\ \texttt{"CP/X3DOM generation"}, links, script]) +\!\!+\ rest\\
body = tag\ \texttt{"body"}\ [\,]\\
x3d = tag\ \texttt{"x3d"}\ [(\texttt{"width"}, \texttt{"\textbackslash"500px\textbackslash""}), (\texttt{"height"}, \texttt{"\textbackslash"400px\textbackslash""})]\\
scene = tag\ \texttt{"scene"}\ [\,]\\
items = concat\\
links = ctag\ \texttt{"link"}\ [\\
\quad (\texttt{"rel"}, quote\ \texttt{"stylesheet"}), (\texttt{"type"}, quote\ \texttt{"text/css"}),\\
\quad (\texttt{"href"}, quote\ \texttt{"http://www.x3dom.org/x3dom/release/x3dom.css"})]
\end{array}
$$

[11] Preamble foi modificado porque não conseguimos o que era pedido com a função preamble dada pelo professor.

```
script = ctag "script" [
  ("type", quote "text/javascript"),
  ("src", quote "http://www.x3dom.org/x3dom/release/x3dom.js")]
ctag t l = tag t l ""
```

onde

```
tag t l x = "<" ++ t ++ " " ++ ps ++ ">" ++ x ++ "</" ++ t ++ ">"
  where ps = unwords [concat [t, "=", v] | (t, v) <- l]
headx = tag "head" []
```

De seguida dão-se mais algumas funções auxiliares facilitadoras:

```
transform (x, y, z) = tag "transform" [("translation", quote (show3D (x, y, x)))]
groupx (x, y, z) = (tag "group" [("bboxSize", quote (show3D (x, y, z)))]) · items
shapex = tag "shape" []
title = tag "title" []
appearance = tag "appearance" []
show3D (x, y, z) = show x ++ " " ++ show y ++ " " ++ show z
t `with` l = ((t $ items l)++)
quote s = "\"" ++ s ++ "\""
prime s = "'" ++ s ++ "'"
box p col = (transform p · shapex · items) [color col, ctag "box" [("size", prime "2,2,2")]]
cone p col b h = (transform p · shapex · items)
  [color col,
    ctag "cone" [("bottomRadius", prime (show b)), ("height", prime (show h))]]
color c = appearance (ctag "material" [("diffuseColor", prime c)])
```

C Soluções propostas

Os alunos devem colocar neste anexo as suas soluções aos exercícios propostos, de acordo com o "layout" que se fornece. Podem ser adicionadas outras funções auxiliares que sejam necessárias.

Secção 4.1

$$
\begin{array}{l}
depth :: LTree\ a \rightarrow Integer \\
depth = cataLTree\ [one, succ \cdot \widehat{max}]
\end{array}
$$

$$
\begin{array}{l}
balance :: LTree\ a \rightarrow LTree\ a \\
balance = anaLTree\ balanceGene \cdot tips
\end{array}
$$

$$
\begin{array}{l}
balanceGene :: [a] \rightarrow Either\ a\ ([a],[a]) \\
balanceGene\ [x] = i_1\ x \\
balanceGene\ xs = i_2\ (splitAt\ (length\ xs\ `div`\ 2)\ xs)
\end{array}
$$

Secção 4.2

$$
\begin{array}{l}
qsplit :: Integral\ a \Rightarrow (a,a) \rightarrow Either\ ()\ (a,((a,a),(a,a))) \\
qsplit\ (n,m) \\
\quad \mid n > m \quad = i_1\ () \\
\quad \mid m \equiv 0 \quad = i_1\ () \\
\quad \mid otherwise = i_2\ (mid,((n, mid-1),(mid+1, m))) \\
\quad \mathbf{where}\ mid = avg2\ (n,m)
\end{array}
$$

$$
\begin{array}{l}
avg2 :: Integral\ a \Rightarrow (a,a) \rightarrow a \\
avg2 = (`div`2) \cdot \widehat{(+)}
\end{array}
$$

Secção 4.3

$inSList :: Either\ a\ (a1, SList\ a1\ a) \rightarrow SList\ a1\ a$
$inSList = [Sent, Cons]$

$outSList :: SList\ b\ a \rightarrow Either\ a\ (b, SList\ b\ a)$
$outSList\ (Sent\ b) = i_1\ b$
$outSList\ (Cons\ (a, sl)) = i_2\ (a, sl)$

$recSList :: (a \rightarrow b) \rightarrow Either\ c\ (d, a) \rightarrow Either\ c\ (d, b)$
$recSList\ f = id + (id \times f)$

$anaSList :: (c \rightarrow Either\ a\ (b, c)) \rightarrow c \rightarrow SList\ b\ a$
$anaSList\ g = inSList \cdot (recSList\ (anaSList\ g)) \cdot g$

$cataSList :: (Either\ b\ (a, d) \rightarrow d) \rightarrow SList\ a\ b \rightarrow d$
$cataSList\ g = g \cdot (recSList\ (cataSList\ g)) \cdot outSList$

$hyloSList :: (Either\ b\ (d, c) \rightarrow c) \rightarrow (a \rightarrow Either\ b\ (d, a)) \rightarrow a \rightarrow c$
$hyloSList\ f\ g = cataSList\ f \cdot anaSList\ g$

$mgen :: Ord\ a \Rightarrow ([a], [a]) \rightarrow Either\ [a]\ (a, ([a], [a]))$
$mgen\ ([\,], ys) = i_1\ ys$
$mgen\ (xs, [\,]) = i_1\ xs$
$mgen\ ((x : xs), ys) = i_2\ (x, (xs, ys))$

Secção 5.2

$inTLTree :: Either\ Tri\ (\mathsf{TLTree}, (\mathsf{TLTree}, \mathsf{TLTree})) \rightarrow \mathsf{TLTree}$
$inTLTree = [Tri, joinTrees]$

$joinTrees :: (\mathsf{TLTree}, (\mathsf{TLTree}, \mathsf{TLTree})) \rightarrow \mathsf{TLTree}$
$joinTrees\ (t1, (t2, t3)) = Nodo\ t1\ t2\ t3$

$outTLTree :: \mathsf{TLTree} \rightarrow Either\ Tri\ (\mathsf{TLTree}, (\mathsf{TLTree}, \mathsf{TLTree}))$
$outTLTree\ (Tri\ tri) = i_1\ tri$
$outTLTree\ (Nodo\ t1\ t2\ t3) = i_2\ (t1, (t2, t3))$

$baseTLTree\ g\ f = (g \times g) + (f \times (f \times f))$

$recTLTree :: (a \rightarrow b) \rightarrow Either\ (c, d)\ (a, (a, a)) \rightarrow Either\ (c, d)\ (b, (b, b))$
$recTLTree\ f = (id \times id) + (f \times (f \times f))$

$cataTLTree :: (Either\ Tri\ (a, (a, a)) \rightarrow a) \rightarrow \mathsf{TLTree} \rightarrow a$
$cataTLTree\ g = g \cdot (recTLTree\ (cataTLTree\ g)) \cdot outTLTree$

$anaTLTree :: (a \rightarrow Either\ Tri\ (a, (a, a))) \rightarrow a \rightarrow \mathsf{TLTree}$
$anaTLTree\ f = inTLTree \cdot (recTLTree\ (anaTLTree\ f)) \cdot f$

$hyloTLTree\ a\ c = cataTLTree\ a \cdot anaTLTree\ c$

$tipsTLTree :: \mathsf{TLTree} \rightarrow [Tri]$
$tipsTLTree = cataTLTree\ [singl, concatenate]$
  where $concatenate\ (a, (b, c)) = a +\!\!+ b +\!\!+ c$

$apresentaSierp = tipsTLTree$

$invTLTree :: \mathsf{TLTree} \rightarrow \mathsf{TLTree}$
$invTLTree = cataTLTree\ [Tri, invTuple]$
  where $invTuple\ (t1, (t2, t3)) = Nodo\ t3\ t2\ t1$

  -- invTLTree :: TLTree -¿ TLTree
  -- invTLTree (Tri tri) = Tri tri
  -- invTLTree (Nodo t1 t2 t3) = Nodo (invTLTree t3) (invTLTree t2) (invTLTree t1)

$depthTLTree :: \mathsf{TLTree} \rightarrow Integer$
$depthTLTree = cataTLTree\ [one, succ \cdot max3]$
  where $max3\ (a, (b, c)) = max\ a\ (max\ b\ c)$

$countTLTree :: \mathsf{TLTree} \rightarrow Integer$
$countTLTree = cataTLTree\ [one, add3]$

$$\mathbf{where}\ add3\ (a,(b,c)) = a + b + c$$

$$geraSierp :: Tri \rightarrow Int \rightarrow \mathsf{TLTree}$$
$$geraSierp\ t\ d = anaTLTree\ geneSierp\ (t,d)$$

$$geneSierp :: (Tri, Int) \rightarrow Either\ Tri\ ((Tri, Int),((Tri, Int),(Tri, Int)))$$
$$geneSierp\ (t,0) = i_1\ t$$
$$geneSierp\ (t@((x,y),s),n) = i_2\ ((((x,y),s'),(n-1)),$$
$$((((x+s',y),s'),(n-1)),$$
$$(((x,y+s'),s'),(n-1))))$$
$$\mathbf{where}\ s' = s\ `div`\ 2$$

$$draw = render\ html\ \mathbf{where}$$
$$html = rep\ dados$$

$$rep = x3dom \cdot map\ drawTriangle \cdot tipsTLTree \cdot \widehat{geraSierp}$$

Secção 6.2

Defina

$$gene :: Either\ ()\ (String,[String]) \rightarrow Dist\ [String]$$
$$gene = [empty, transmitter]$$
$$transmitter :: (String,[String]) \rightarrow Dist\ [String]$$
$$transmitter\ (\texttt{"stop"}, words) = D\ [(\texttt{"stop"} : words, 0.90),(words, 0.10)]$$
$$transmitter\ (word, words) = D\ [(word : words, 0.95),(words, 0.05)]$$
$$empty :: () \rightarrow Dist\ [a]$$
$$empty\ () = return\ [\,]$$
$$ataque = transmitir\ (words\ \texttt{"Vamos atacar hoje"})$$

e responda ao problema do enunciado aqui.

```
*Main> ataque
["Vamos","atacar","hoje","stop"]  77.2%
       ["Vamos","atacar","hoje"]   8.6%
        ["atacar","hoje","stop"]   4.1%
       ["Vamos","atacar","stop"]   4.1%
         ["Vamos","hoje","stop"]   4.1%
              ["Vamos","atacar"]   0.5%
                ["Vamos","hoje"]   0.5%
               ["atacar","hoje"]   0.5%
                ["Vamos","stop"]   0.2%
               ["atacar","stop"]   0.2%
                 ["hoje","stop"]   0.2%
                      ["atacar"]   0.0%
                       ["Vamos"]   0.0%
                        ["hoje"]   0.0%
                        ["stop"]   0.0%
                              []   0.0%

*Main>
```

Qual a probabilidade de a palavra `"atacar"` da mensagem *words* `"Vamos atacar hoje"` se perder, isto é, o resultado da transmissão ser `["Vamos","hoje","stop"]`?

4.1%

E a de seguirem todas as palavras, mas faltar o `"stop"` no fim?

8.6%

E a da transmissão ser perfeita?

77.2%

Secção 6.4

Defina

$$
\begin{array}{l}
\mathit{parBTreeMap}\ f\ \mathit{Empty} = \mathit{return}\ \mathit{Empty} \\
\mathit{parBTreeMap}\ f\ (\mathit{Node}\ (a, (\mathit{bt1}, \mathit{bt2}))) = \mathbf{do}\ a' \leftarrow \mathit{rpar}\ (f\ a) \\
\quad \mathit{bt1}' \leftarrow \mathit{parBTreeMap}\ f\ \mathit{bt1} \\
\quad \mathit{bt2}' \leftarrow \mathit{parBTreeMap}\ f\ \mathit{bt2} \\
\quad \mathit{return}\ (\mathit{Node}\ (a', (\mathit{bt1}', \mathit{bt2}')))
\end{array}
$$

e apresente aqui os resultados das suas experiências com essa função.

```
$ ./cp1415t exercise seq -N2 +RTS -s 2>&1 | grep Total
    Total   time    0.915s  (  0.933s elapsed)
                                $ ./cp1415t exercise par -N2 +RTS -s 2>&1 | grep Tota
        Total   time    0.928s  (  0.933s elapsed)
```

Na versão feita para a BTree, a versão paralela é cerca de 1.01x mais rápida ($\frac{0.928}{0.915}$) que a sequencial.