Universidade do Minho

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

Departamento de Informática - Universidade do Minho

Maio de 2015

Grupo

(a) nome : Carlos Gonçalves
número : xxxxxxx
mail : xxxxxxxx.xxx

(b) nome : João Rua
número : 41841
mail : joaorua at mail.com

(c) nome : Miguel Guimarães
número : 66822
mail : migueguimaraess at hotmail.com

Contents

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 )
```

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

```
[ 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:

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

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

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

2. Seja dada:

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

   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:*

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

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.

   <u>Testes unitários</u> 4 *Faça os testes seguintes:*

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

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

   $$merge' :: Ord\ a \Rightarrow ([a], [a]) \rightarrow [a]$$
   $$merge' = hyloSList\ [id, cons]\ mgen$$

   para um dado gene $mgen$ que deverá definir.

   <u>Testes unitários</u> 5 *Faça os seguintes testes:*

   $$test05a = mgen\ ([0, 2, 5], [0, 6]) \equiv i_2\ (0, ([2, 5], [0, 6]))$$
   $$test05b = mgen\ ([0, 2, 5], [\,]) \equiv i_1\ [0, 2, 5]$$
   $$test05c = merge'\ ([\,], [0, 6]) \equiv [0, 6]$$

5 Parte B

O triângulo de Sierpinski é uma figura fractal que tem o aspecto da figura 2 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. 2.

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 2 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:

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

onde

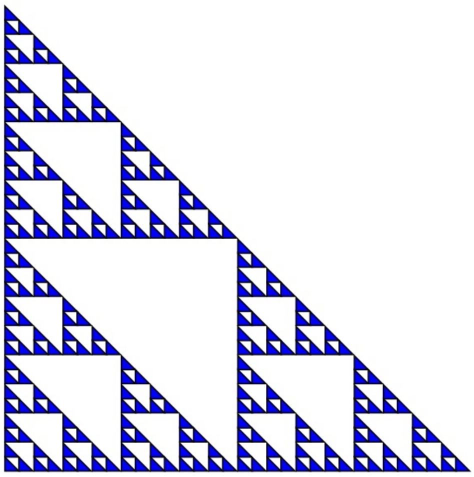


Figure 2: Um triângulo de Sierpinski

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

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:

$$\textbf{data } \mathsf{TLTree} = Tri\ Tri \mid Nodo\ \mathsf{TLTree}\ \mathsf{TLTree}\ \mathsf{TLTree}$$

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

$$sierpinski :: Tri \to Int \to [\,Tri\,]$$
$$sierpinski\ t = apresentaSierp \cdot (geraSierp\ t)$$

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,

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

e outra que as consome:

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

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

$$
\begin{array}{l}
tipsTLTree :: \mathsf{TLTree}\ b \to [b] \\
countTLTree :: \mathsf{TLTree}\ b \to Int \\
depthTLTree :: \mathsf{TLTree}\ b \to Int \\
invTLTree :: \mathsf{TLTree}\ b \to \mathsf{TLTree}\ b
\end{array}
$$

   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

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

   e faça os testes seguintes:

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

$$
\begin{array}{l}
test06a = depthTLTree\ ts \equiv 6 \\
test06b = countTLTree\ ts \equiv 243 \\
test06c = fromIntegral\ (countTLTree\ ts) \equiv length\ (tipsTLTree\ ts) \\
test06d = countTLTree\ ts \equiv countTLTree\ (invTLTree\ ts)
\end{array}
$$

Visualização: Para visualizarmos triângulos de Sierpinski vamos usar X3DOM, uma biblioteca "open-source" para construção e visualização de gráficos 3D no Web.[2] 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:

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

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

$$finalize :: String \to String$$

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

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

[2] 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`.

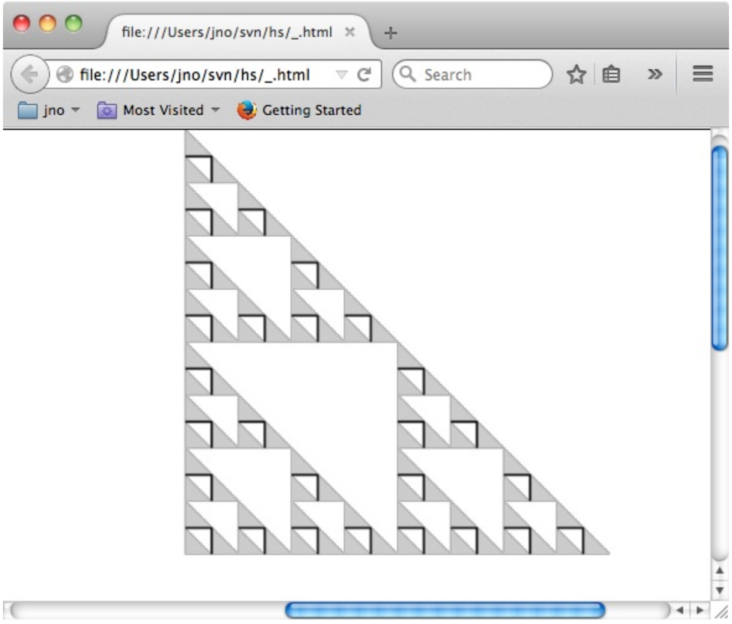


Figure 3: Um triângulo de Sierpinski em x3dom

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

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

(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 3.

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 4. Pode recorrer, se desejar, às funções disponibilizadas no anexo B.1.

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

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

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

Cada par (a, p) numa distribuição $d :: Dist\ a$ indica que a probabilidade de a é p, devendo ser garantida a propriedade de que todas as probabilidades de d somam

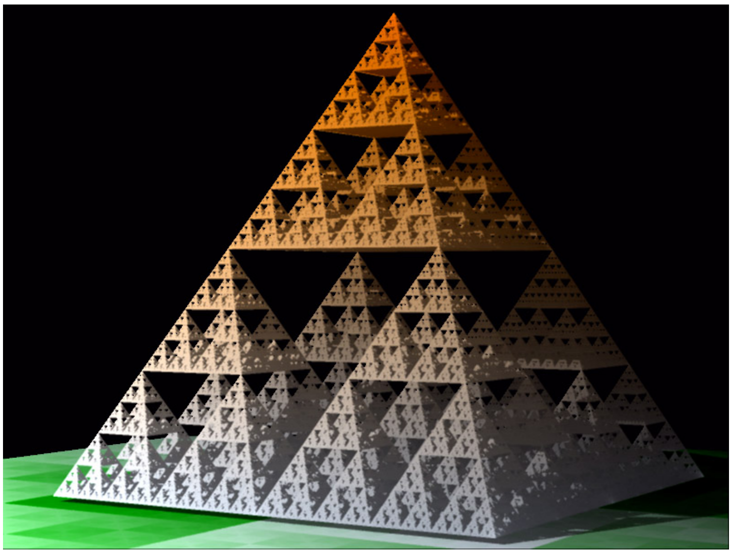
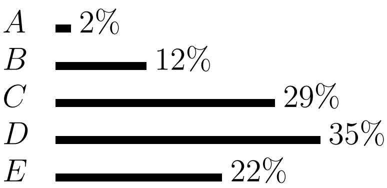


Figure 4: Uma pirâmide de Sierpinski

100%. Por exemplo, a seguinte distribuição de classificações por escalões de A a E,



será representada pela distribuição

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

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.[3]

[3] 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 [?].

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 $g : A \rightarrow Dist\ B$ e $f : B \rightarrow Dist\ C$ 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

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

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

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

que é muito parecido com

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

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:

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

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

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

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

$$transmitir = pcataList\ gene$$

descreve o comportamento do aparelho.

Testes unitários 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.[4] Por exemplo,

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

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

$$\begin{array}{l} lt' \leftarrow parmap\ f\ lt \\ return\ (a' : lt') \end{array}$$

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

$$map\ fib\ [20\mathinner{..}30] = [10946, 17711, 28657, 46368, 75025, 121393, 196418, 317811, 514229, 832040, 1$$

(cálculo dos números de Fibonacci do vigésimo ao trigésimo), o tempo que o cálculo vai demorar numa máquina com 2 cores[5] 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:[6]

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

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

   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

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

   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.

[5] Intel Core 2 Duo a 2.53 GHz.

[6] Ver detalhes em [?].

6.4 Trabalho a realizar

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

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

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.

Anexos

A Programa principal

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

B Bibliotecas e código auxiliar

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

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:

$$
\begin{array}{l}
html = tag\ \texttt{"html"}\ [\,]\\
preamble = headx\ `with`\ [title\ \texttt{"CP/X3DOM generation"}, links, script]\\
body = tag\ \texttt{"body"}\ [\,]\\
x3d = tag\ \texttt{"x3d"}\ [(\texttt{"width"}, \texttt{"\textbackslash"500px\textbackslash""}), (\texttt{"height"}, \texttt{"\textbackslash"400px\textbackslash""})]\\
scene = tag\ \texttt{"scene"}\ [\,]
\end{array}
$$

$$
\begin{array}{l}
\mathit{items} = \mathit{concat} \\
\mathit{links} = \mathit{ctag}\ \texttt{"link"}\ [\\
\quad (\texttt{"rel"}, \mathit{quote}\ \texttt{"stylesheet"}), (\texttt{"type"}, \mathit{quote}\ \texttt{"text/css"}), \\
\quad (\texttt{"href"}, \mathit{quote}\ \texttt{"http://www.x3dom.org/x3dom/release/x3dom.css"})] \\
\mathit{script} = \mathit{ctag}\ \texttt{"script"}\ [\\
\quad (\texttt{"type"}, \mathit{quote}\ \texttt{"text/javascript"}), \\
\quad (\texttt{"src"}, \mathit{quote}\ \texttt{"http://www.x3dom.org/x3dom/release/x3dom.js"})] \\
\mathit{ctag}\ t\ l = \mathit{tag}\ t\ l\ \texttt{""}
\end{array}
$$

onde

$$
\begin{array}{l}
\mathit{tag}\ t\ l\ x = \texttt{"<"} +\!\!+\ t +\!\!+ \texttt{" "} +\!\!+\ \mathit{ps} +\!\!+ \texttt{">"} +\!\!+\ x +\!\!+ \texttt{"</"} +\!\!+\ t +\!\!+ \texttt{">"} \\
\quad \mathbf{where}\ \mathit{ps} = \mathit{unwords}\ [\mathit{concat}\ [t, \texttt{"="}, v] \mid (t, v) \leftarrow l] \\
\mathit{headx} = \mathit{tag}\ \texttt{"head"}\ [\,]
\end{array}
$$

De seguida dão-se mais algumas funções auxiliares facilitadoras:

$$
\begin{array}{l}
\mathit{transform}\ (x, y, z) = \mathit{tag}\ \texttt{"transform"}\ [(\texttt{"translation"}, \mathit{quote}\ (\mathit{show3D}\ (x, y, x)))] \\
\mathit{groupx}\ (x, y, z) = (\mathit{tag}\ \texttt{"group"}\ [(\texttt{"bboxSize"}, \mathit{quote}\ (\mathit{show3D}\ (x, y, z)))]) \cdot \mathit{items} \\
\mathit{shapex} = \mathit{tag}\ \texttt{"shape"}\ [\,] \\
\mathit{title} = \mathit{tag}\ \texttt{"title"}\ [\,] \\
\mathit{appearance} = \mathit{tag}\ \texttt{"appearance"}\ [\,] \\
\mathit{show3D}\ (x, y, z) = \mathit{show}\ x +\!\!+ \texttt{" "} +\!\!+\ \mathit{show}\ y +\!\!+ \texttt{" "} +\!\!+\ \mathit{show}\ z \\
t\ `\mathit{with}`\ l = ((t\ \$\ \mathit{items}\ l) +\!\!+) \\
\mathit{quote}\ s = \texttt{"\textbackslash""} +\!\!+\ s +\!\!+ \texttt{"\textbackslash""} \\
\mathit{prime}\ s = \texttt{"'"} +\!\!+\ s +\!\!+ \texttt{"'"} \\
\mathit{box}\ p\ \mathit{col} = (\mathit{transform}\ p \cdot \mathit{shapex} \cdot \mathit{items})\ [\mathit{color}\ \mathit{col}, \mathit{ctag}\ \texttt{"box"}\ [(\texttt{"size"}, \mathit{prime}\ \texttt{"2,2,2"})]] \\
\mathit{cone}\ p\ \mathit{col}\ b\ h = (\mathit{transform}\ p \cdot \mathit{shapex} \cdot \mathit{items}) \\
\quad [\mathit{color}\ \mathit{col}, \\
\qquad \mathit{ctag}\ \texttt{"cone"}\ [(\texttt{"bottomRadius"}, \mathit{prime}\ (\mathit{show}\ b)), (\texttt{"height"}, \mathit{prime}\ (\mathit{show}\ h))]] \\
\mathit{color}\ c = \mathit{appearance}\ (\mathit{ctag}\ \texttt{"material"}\ [(\texttt{"diffuseColor"}, \mathit{prime}\ c)])
\end{array}
$$

C Soluções propostas

Secção 4.1

A função [depth] recebe uma LTree e returna a altura da LTree. Usando um catamorfismo de um [either one (succ . uncurry max)], em que [one] é a altura da árvore que contém só uma folha e [succ] . [uncurry max] caso a árvore contenha mais que uma folha e, nessa situação, a altura é calculado através da maior arvore, esquerda ou direita.

A função [balance] usa a função [tips], que coloca todos os elementos de uma LTree numa lista com todos os elementos da mesma. Essa lista é organizada pelo anamorfismo da função [lsplit], que organiza a mesma, para depois devolver uma nova árvore, desta vez, balanceada, com os elementos da lista já organizados.

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

$$
\begin{aligned}
&\mathit{balance} :: \mathit{LTree}\ a \rightarrow \mathit{LTree}\ a\\
&\mathit{balance} = \mathit{anaLTree}\ \mathit{lsplit} \cdot \mathit{tips}
\end{aligned}
$$

Secção 4.2

A função [qsplit] corresponde a um gene e funciona como função auxiliar da [abpe]. Esta função recebe como parametro um par de dois numeros inteiros que vai ser usados para gerar uma BTree, colocando o valor medio na cabeça e criando a subarvore a esquerda com valores inferiores a cabeça e a subarvore a direita com os valores maiores.

$$
\begin{aligned}
&\mathit{qsplit} :: \mathit{Integral}\ a \Rightarrow (a, a) \rightarrow \mathit{Either}\ ()\ (a, ((a, a), (a, a)))\\
&\mathit{qsplit}\ (x, y) = \mathbf{if}\ (x > y) \vee (x \equiv 0 \wedge y \equiv 0)\ \mathbf{then}\ i_1\ ()\ \mathbf{else}\ i_2\ (\mathit{media}, ((x, \mathit{media} - 1), (\mathit{media} +\\
&\quad \mathbf{where}\ \mathit{media} = x + y \div 2
\end{aligned}
$$

Secção 4.3

Nesta secção estão presentes várias funções para a biblioteca SList, tal como estão definidas para as restantes bibliotecas. Contém também uma função [mgen], que é o gene de [merge]. Recebe um par de listas e parte-a, separando a cabeça da primeira lista do par com a sua cauda e com a segunda lista recebida como parâmetro.

$$
\begin{aligned}
&\mathit{inSList} :: \mathit{Either}\ a\ (a1, \mathit{SList}\ a1\ a) \rightarrow \mathit{SList}\ a1\ a\\
&\mathit{inSList} = [\mathit{Sent}, \mathit{Cons}]\\
&\mathit{outSList} :: \mathit{SList}\ b\ a \rightarrow \mathit{Either}\ a\ (b, \mathit{SList}\ b\ a)\\
&\mathit{outSList}\ (\mathit{Sent}\ a) = i_1\ a\\
&\mathit{outSList}\ (\mathit{Cons}\ (a, \mathit{sl})) = i_2\ (a, \mathit{sl})\\
&\mathit{recSList}\ c = \mathit{id} + \mathit{id} \times c\\
&\mathit{anaSList} :: (c \rightarrow \mathit{Either}\ a\ (b, c)) \rightarrow c \rightarrow \mathit{SList}\ b\ a\\
&\mathit{anaSList}\ g = \mathit{inSList} \cdot \mathit{recSList}\ (\mathit{anaSList}\ g) \cdot g\\
&\mathit{cataSList} :: (\mathit{Either}\ b\ (a, d) \rightarrow d) \rightarrow \mathit{SList}\ a\ b \rightarrow d\\
&\mathit{cataSList}\ g = g \cdot \mathit{recSList}\ (\mathit{cataSList}\ g) \cdot \mathit{outSList}\\
&\mathit{hyloSList} :: (\mathit{Either}\ b\ (d, c) \rightarrow c) \rightarrow (a \rightarrow \mathit{Either}\ b\ (d, a)) \rightarrow a \rightarrow c\\
&\mathit{hyloSList}\ h\ g = \mathit{cataSList}\ h \cdot \mathit{anaSList}\ g\\
&\mathit{mgen} :: \mathit{Ord}\ a \Rightarrow ([a], [a]) \rightarrow \mathit{Either}\ [a]\ (a, ([a], [a]))\\
&\mathit{mgen}\ (a, [\,]) = i_1\ (a)\\
&\mathit{mgen}\ ([\,], b) = i_1\ (b)\\
&\mathit{mgen}\ (a, b) = i_2\ (\mathit{head}\ a, (\mathit{tail}\ a, b))
\end{aligned}
$$

Secção 5.2

$$
\begin{aligned}
&\mathit{inTLTree} = [L, N]\\
&\mathit{outTLTree} :: \mathsf{TLTree}\ a \rightarrow \mathit{Either}\ a\ (\mathsf{TLTree}\ a, (\mathsf{TLTree}\ a, \mathsf{TLTree}\ a))\\
&\mathit{outTLTree}\ (L\ a) = i_1\ a\\
&\mathit{outTLTree}\ (N\ (a1, (a2, a3))) = i_2\ (a1, (a2, a3))\\
&\mathit{baseTLTree}\ g\ f = g + (f \times (f \times f))\\
&\mathit{recTLTree}\ f = \mathit{id} + (f \times (f \times f))\\
&\mathit{cataTLTree}\ a = a \cdot (\mathit{recTLTree}\ (\mathit{cataTLTree}\ a)) \cdot \mathit{outTLTree}
\end{aligned}
$$

$$
\begin{aligned}
&anaTLTree\ f = inTLTree \cdot (recTLTree\ (anaTLTree\ f)) \cdot f\\
&hyloTLTree\ a\ c = cataTLTree\ a \cdot anaTLTree\ c\\
&tipsTLTree = cataTLTree\ [singl, \widehat{(+\!\!+)} \cdot (id \times \widehat{(+\!\!+)})]\\
&invTLTree = cataTLTree\ [L, N \cdot swap']\\
&\quad \textbf{where}\ swap'\ (a,(b,c)) = (c,(b,a))\\
&depthTLTree = cataTLTree\ [one, succ \cdot \widehat{max} \cdot (id \times \widehat{max})]\\
&geraSierp :: Tri \to Int \to \mathsf{TLTree}\ Tri\\
&geraSierp\ tri\ n = anaTLTree\ geneSierp\ (tri, n)\\
&\quad \textbf{where}\ geneSierp\ (a, 0) = i_1\ a\\
&\qquad geneSierp\ (((x,y),z),n) = \textbf{let}\ z' = z \div 2\\
&\qquad\quad \textbf{in}\ i_2\ ((((x,y),z'),n-1),((((x+z',y),z'),n-1),(((x,y+z'),z'),n-1)))\\
&\quad \text{-- Aux Mostra Sierp}\\
&mostraSierp :: \mathsf{TLTree}\ Tri \to [Tri]\\
&mostraSierp = tipsTLTree\\
&countTLTree :: \mathsf{TLTree}\ b \to Integer\\
&countTLTree = cataTLTree\ [one, add \cdot (id \times add)]\\
&draw = render\ html\ \textbf{where}\\
&\quad html = rep\ dados\\
&rep = finalize \cdot concat \cdot (map\ drawTriangle) \cdot mostraSierp \cdot (uncurryTLTree\ geraSierp) \cdot split\ \pi_1\\
&\quad \textbf{where}\ uncurryTLTree\ f\ (((x,y),z),n) = f\ ((x,y),z)\ n
\end{aligned}
$$

Secção 6.2

Sendo gene um [either sendStop wordLost], que nos dá a probabilidade de stop ou outra palavra da lista se perder, esta função será usada como uma função auxiliar da função [transmitir], a função [wordLost] calcula a probabilidade de uma palavra da lista se perder, a probabilidade de a palavra stop se perder é calculada pela função [sendStop].

$$
\begin{aligned}
&sendStop = \underline{(D\ [([\,], 0.10), ([\texttt{"stop"}], 0.90)])}\\
&wordLost\ (a,b) = D\ [((a:b), 0.95), (b, 0.05)]\\
&gene = [sendStop, wordLost]
\end{aligned}
$$

Transmissão Perfeita:
["Vamos","atacar","hoje","stop"] 77.2
Não chegar a palavra "stop" no fim:
["Vamos","atacar","hoje"] 8.6
Se perder a palavra "atacar":
["Vamos","hoje","stop"] 4.1

Secção 6.4

A função [parBTreeMap] aplica a funcão [rpar] a todos os elementos de uma BTree.

$$
\begin{aligned}
&parBTreeMap\ f\ Empty = return\ Empty\\
&parBTreeMap\ f\ (Node\ (x,(y,z))) = \textbf{do}\\
&\quad x' \leftarrow rpar\ (f\ x)\\
&\quad y' \leftarrow parBTreeMap\ f\ y
\end{aligned}
$$

$$z' \leftarrow parBTreeMap\ f\ z$$
$$return\ (Node\ (x', (y', z')))$$