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

Departamento de Informática
Universidade do Minho

Maio de 2015

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 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 [2], 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*.

```
import Data.List
import System.Process
import Cp
import List
import Nat
import Exp
import BTree
import LTree
  -- import TLTree
import X3d
import Control.Parallel.Strategies
import Probability hiding (· → ·, ·)
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

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

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

   $$balance :: LTree\ a \to LTree\ a$$

   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:*

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

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:*

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

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

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

   para um dado gene $mgen$ que deverá definir.

   Testes unitários 5 *Faça os seguintes testes:*

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

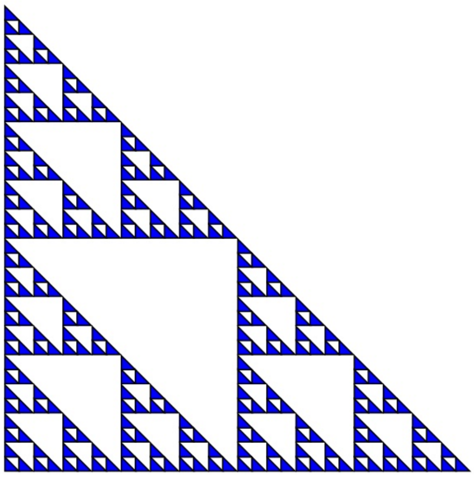


Figura 1: Um triângulo de Sierpinski

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:

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

onde

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

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:

$$\mathbf{data}\ \mathsf{TLTree} = \mathit{Tri}\ \mathit{Tri} \mid \mathit{Nodo}\ \mathsf{TLTree}\ \mathsf{TLTree}\ \mathsf{TLTree}$$

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

$$\mathit{sierpinski} :: \mathit{Tri} \rightarrow \mathit{Int} \rightarrow [\,\mathit{Tri}\,]$$
$$\mathit{sierpinski}\ t = \mathit{apresentaSierp} \cdot (\mathit{geraSierp}\ t)$$

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,

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

e outra que as consome:

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

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

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

   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:

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

[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`.

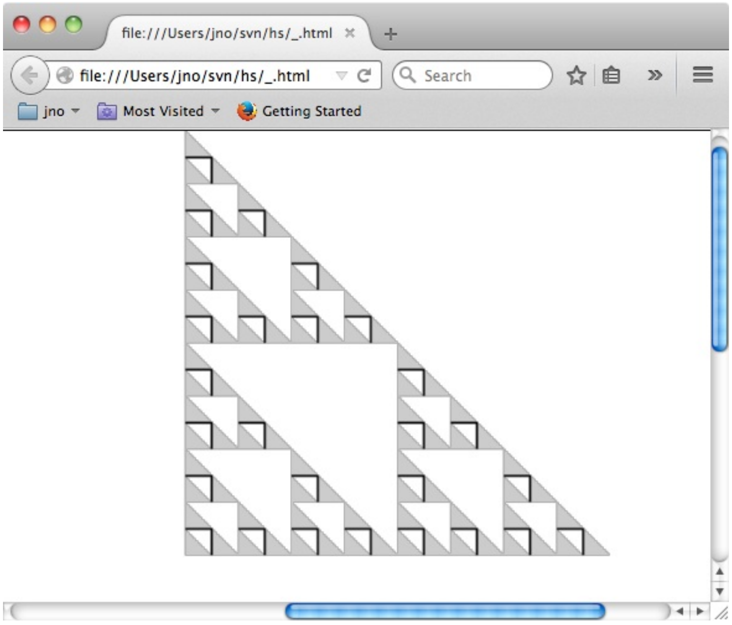


Figura 2: Um triângulo de Sierpinski em x3dom

- 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,

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

   (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.

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%.

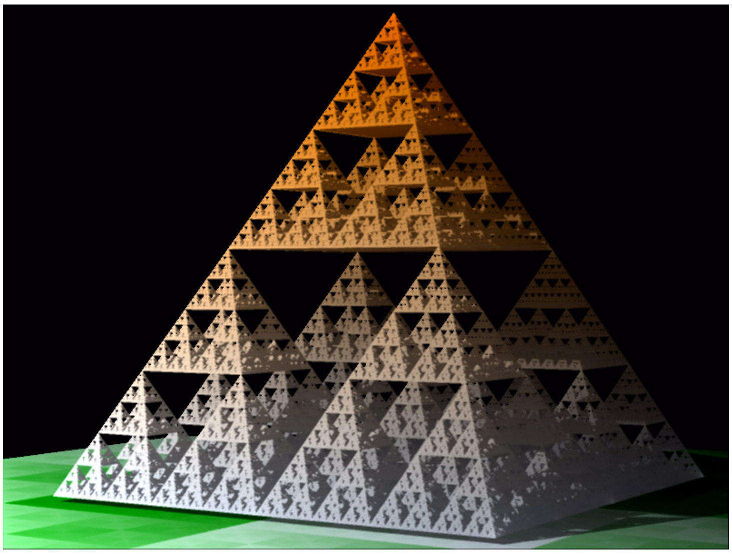
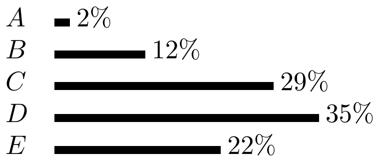


Figura 3: Uma pirâmide de Sierpinski

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 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 [1].

$Dist$ 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

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

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,

$$rpar :: a \to Eval\ a$$
$$rseq :: a \to Eval\ a$$

conforme se deseja que uma dada computação seja efectuada em paralelo ou sequencialmente.[4] Por exemplo,

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

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

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

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

[4] Esta explicação é bastante simplista, mas serve de momento. Para uma abordagem completa e elucidativa ver a referência [3].

[5] Intel Core 2 Duo a 2.53 GHz.

[6] Ver detalhes em [3].

```
./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

$$putStrLn \cdot show \cdot (map\ fib)\ \$\ [20\,..\,30]$$

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

$$putStrLn \cdot show \cdot runEval \cdot (parmap\ fib)\ \$\ [20\,..\,30]$$

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

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

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.

Referências

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

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

[3] S. Marlow. *Parallel and Concurrent Programming in Haskell*. O'Reilly, 2013.

Anexos

A Programa principal

$$
\begin{array}{l}
\mathit{main} :: \mathit{IO}\ ()\\
\mathit{main} = \mathit{getArgs} \ggg\!= (\neg \cdot \mathit{null}) \rightarrow \mathit{exemp\_or\_exer}, \mathit{errInvArgs}\\
\quad \mathbf{where}\\
\qquad \mathit{exemp\_or\_exer} = (((\equiv)\ \texttt{"exemplo"}) \cdot \mathit{head}) \rightarrow \mathit{exemp}, \mathit{exer}\\
\qquad \mathit{exemp} = (((\equiv)\ 2) \cdot \mathit{length}) \rightarrow \mathit{execExemp}, \mathit{errInvArgs}\\
\qquad \mathit{execExemp} = \mathit{isPar} \rightarrow \mathit{execExempPar}, \mathit{execExempSeq}\\
\qquad \mathit{exer} = (((\equiv)\ 3) \cdot \mathit{length}) \rightarrow \mathit{execExer}, \mathit{errInvArgs}\\
\qquad \mathit{execExer} = \mathit{isPar} \rightarrow \mathit{execExerPar}, \mathit{execExerSeq}\\
\qquad \mathit{execExempSeq} = \underline{(\mathit{putStrLn} \cdot \mathit{show} \cdot \mathit{runEval} \cdot (\mathit{parBTreeMap}\ \mathit{fib})\ \$\ \mathit{t1})}\\
\qquad \mathit{execExempPar} = \underline{(\mathit{putStrLn} \cdot \mathit{show} \cdot (\mathit{fmap}\ \mathit{fib})\ \$\ \mathit{t1})}
\end{array}
$$

B Bibliotecas e código auxiliar

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

B.1 "Easy X3DOM access"

Defina-se a seguinte composição de funções

$$\mathit{x3dom} = \mathit{html} \cdot \mathit{preamble} \cdot \mathit{body} \cdot \mathit{x3d} \cdot \mathit{scene} \cdot \mathit{items}$$

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

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

$$\quad (\texttt{"src"}, \mathit{quote}\ \texttt{"http://www.x3dom.org/x3dom/release/x3dom.js"})]$$
$$\mathit{ctag}\ t\ l = \mathit{tag}\ t\ l\ \texttt{""}$$

onde

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

De seguida dão-se mais algumas funções auxiliares facilitadoras:

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

C Soluções propostas

Grupo 61:
Tiago João Lopes Carvalhais (A70443)
José Carlos da Silva Brandão Gonçalves (A71223)
Gustavo da Costa Gomes (A72223)

Secção 4.1

A função depth recebe uma LTree e devolve um inteiro com o fator de balanceamento da árvore. Neste caso, usamos um catamorfismo de um $[\mathit{zero}, \mathit{succ} \cdot \widehat{\mathit{max}}]$, em que zero é o fator de balanceamento da árvore que contém só uma folha e $\mathit{succ}\ .\ \widehat{\mathit{max}}$ caso a árvore contenha mais que uma folha e, nessa situação, o balanceamento é calculado através do sucessor do maior fator de balanceamento entre as árvores da esquerda e da direita.

A função $\mathit{balance}$ contém uma função tips, que coloca todos os elementos de uma LTree numa lista com todos os elementos da mesma. Essa lista vai, depois, ser organizada pelo anamorfismo da função lsplit, que vai tratar de organizar a mesma, para depois devolver uma nova árvore, desta vez, balanceada, consoante a disposição dos elementos na lista.

$$\mathit{depth} :: \mathit{LTree}\ a \rightarrow \mathit{Integer}$$
$$\mathit{depth} = \mathit{cataLTree}\ [\mathit{zero}, \mathit{succ} \cdot \widehat{\mathit{max}}]$$
$$\mathit{balance} :: \mathit{LTree}\ a \rightarrow \mathit{LTree}\ a$$
$$\mathit{balance} = \mathit{anaLTree}\ \mathit{lsplit} \cdot \mathit{tips}$$

Secção 4.2

A função qsplit corresponde a um gene e funciona como função auxiliar da abpe. Recebe como argumento um par de inteiros, que corresponde ao intervalo em que a árvore vai ser definida, e pega no elemento médio do mesmo (através da função med) e coloca-o no topo a BTree, e de seguida, gera dois intervalos, que correspondem aos intervalos de valores da árvore, que se situam antes e depois do valor intermédio.

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

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.

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

Secção 5.2

Nesta secção e, tal como na anterior, estão presentes funções para uma biblioteca, neste caso a TLTree. A função $geraSierp$ gera um triângulo de Sierpinski através de um anamorfismo, cujo gene, a função $geneSierp$ divide a estrutura do tipo Tri em três partes, para assim formar o triângulo, diminuindo o nível de recursividade em uma unidade.

A função $apresentaSierp$ recebe uma estrutura do tipo TLTree e trasforma-a numa lista de estruturas do tipo Tri. Faz exatamente o mesmo que a função tipsTLTree, pelo que a sua presença é redundante.

A função rep transforma por fim a estrutura TLTree num triângulo de Sierpinski, que pode posteriormente ser visto numa página web, no ficheiro "sierpinski.html". A função começa por gerar a estrutura do triângulo, chamando a função $geraSierp$ que, posteriormente, deverá sofrer um $uncurryTLTree$ para ser adaptado e poder ser recebido como input da $apresentaSierp$, que vai transformar a TLTree numa lista de Tri's. Por fim, cada elemento da lista vai ser passado para uma String própria para ficheiros html pela função $drawTriangle$ e a função $finalize$ faz o acabamento final, para que seja possível o ficheiro html ser aberto numa página web e o triângulo possa ser visualizado.

$$
\begin{array}{l}
inTLTree = [L, N]\\
outTLTree :: \mathsf{TLTree}\ a \rightarrow Either\ a\ (\mathsf{TLTree}\ a, (\mathsf{TLTree}\ a, \mathsf{TLTree}\ a))\\
outTLTree\ (L\ a) = i_1\ a\\
outTLTree\ (N\ (t1, (t2, t3))) = i_2\ (t1, (t2, t3))\\
baseTLTree\ g\ f = g + (f \times (f \times f))\\
recTLTree\ f = id + (f \times (f \times f))\\
cataTLTree\ a = a \cdot (recTLTree\ (cataTLTree\ a)) \cdot outTLTree
\end{array}
$$

$$anaTLTree\ f = inTLTree \cdot (recTLTree\ (anaTLTree\ f)) \cdot f$$
$$hyloTLTree\ a\ c = cataTLTree\ a \cdot anaTLTree\ c$$
$$tipsTLTree = cataTLTree\ [singl, \widehat{(+\!\!+)} \cdot (id \times \widehat{(+\!\!+)})]$$
$$invTLTree = cataTLTree\ [L, N \cdot swap']$$
$$\quad \textbf{where}\ swap'\ (a, (b, c)) = (c, (b, a))$$
$$depthTLTree = cataTLTree\ [one, succ \cdot \widehat{max} \cdot (id \times \widehat{max})]$$
$$geraSierp :: Tri \to Int \to \mathsf{TLTree}\ Tri$$
$$geraSierp\ tri\ n = anaTLTree\ geneSierp\ (tri, n)$$
$$\quad \textbf{where}\ geneSierp\ (a, 0) = i_1\ a$$
$$\qquad geneSierp\ (((x, y), z), n) = \textbf{let}\ z' = z \div 2$$
$$\qquad\quad \textbf{in}\ i_2\ ((((x, y), z'), n - 1), ((((x + z', y), z'), n - 1), (((x, y + z'), z'), n - 1)))$$
$$apresentaSierp :: \mathsf{TLTree}\ Tri \to [\,Tri\,]$$
$$apresentaSierp = tipsTLTree$$
$$countTLTree :: \mathsf{TLTree}\ b \to Integer$$
$$countTLTree = cataTLTree\ [one, add \cdot (id \times add)]$$
$$draw = render\ html\ \textbf{where}$$
$$\quad html = rep\ dados$$
$$rep = finalize \cdot concat \cdot (map\ drawTriangle) \cdot apresentaSierp \cdot (uncurryTLTree\ geraSierp) \cdot split\ \pi_1\ \pi_2$$
$$\quad \textbf{where}\ uncurryTLTree\ f\ (((x, y), z), n) = f\ ((x, y), z)\ n$$

Secção 6.2

A função *gene* devolve um $[stop, perder]$, em que retorna a probabilidade de a última palavra, ou seja, a palavra "stop", se perder e também a probabilidade de qualquer outra palavra da lista ter o mesmo destino. Funciona como função auxiliar da *transmitir*.

A função *perder* calcula a probabilidade qualquer palavra da lista se perder, exceto a palavra "stop", cuja probabilidade de desaparecimento vai ser calculada pela função *stop*.

$$gene = [stop, perder]$$
$$perder\ (a, b) = D\ [((a : b), 0.95), (b, 0.05)]$$
$$stop = \underline{(D\ [([\,], 0.10), ([\texttt{"stop"}], 0.90)])}$$

A probabilidade de a palavra "atacar"se perder é de 4.1%.
A probabilidade de chegarem todas as palavras, mas faltar o "stop"no fim, é de 8.6%.
A probabilidade de a transmissão ser perfeita é de 77.2%.

Secção 6.4

A função *parBTreeMap* aplica a função *rpar* a todos os elementos de uma BTree.

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

Testes de performance:
Versão sequencial - Total time 0.94s (0.73s elapsed)
Versão paralela - Total time 0.63s (0.36s elapsed)
Analisando o "time elapsed"de ambas as versões, podemos verificar que a paralela é cerca de 2.03x mais rápida que a sequancial.

Índice