

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

Departamento de Informática
Universidade do Minho

Fevereiro de 2018

Contents

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	4
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	9
6.4	Trabalho a realizar	10
A	Programa principal	11
B	Bibliotecas e código auxiliar	11
B.1	“Easy X3DOM access”	11
C	Soluções propostas	12

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 `[fontsize=] lhs2TeX cp1415t.lhs & cp1415t.tex pdflatex cp1415t` em que `lhs2tex` é um pre-processor 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 `[fontsize=] ghci cp1415t.lhs` para ver que assim é:

```
[fontsize=] 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 ... link-
ing ... 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, in-
terpreted ) [ 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, inter-
preted ) [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 (· → ·, ·)
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

¹O suffixo ‘lhs’ quer dizer *literate Haskell*.

```
\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 **BibT_EX**) e o índice remissivo (com **makeindex**) [fontsize=] 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

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

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

2. Seja dada:

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

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

$$\text{test01} = \text{balanced } t \equiv \text{False}$$

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

$$\text{balance} :: \text{LTree } a \rightarrow \text{LTree } a$$

que equilibra uma qualquer árvore binária.

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

$$\text{test02} = \text{balanced } (\text{balance } t) \equiv \text{True}$$

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) :

$$abpe\ (n, m) = anaBTree\ qspl\it\ (n, m)$$

Comece por definir o gene *qspl*it e depois construa a árvore

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

que será precisa na secção 6.4.

Testes unitários 3 *Faça os testes seguintes:*

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

4.3 Biblioteca para listas com sentinelas

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

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

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:*

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

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

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

para um dado gene *mgen* que deverá definir.

Testes unitários 5 *Faça os seguintes testes:*

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

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).

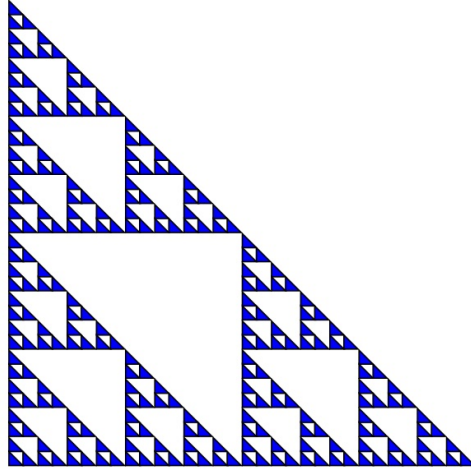


Figure 1: Um triângulo de Sierpinski

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:

```
type Tri = (Point, Side)
```

onde

```
type Side = Int
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:

```
data TLTree = Tri Tri | Nodo TLTree TLTree 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 → Int → [Tri]
sierpinski t = apresentaSierp · (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,

```
geraSierp :: Tri → Int → TLTree
geraSierp t 0 = Tri t
geraSierp ((x, y), s) n =
  let s' = s ÷ 2
  in Nodo
    (geraSierp ((x, y), s') (n - 1))
    (geraSierp ((x + s', y), s') (n - 1))
    (geraSierp ((x, y + s'), s') (n - 1))
```

e outra que as consome:

```
apresentaSierp :: TLTree → [Tri]
apresentaSierp (Tri t) = [t]
apresentaSierp (Nodo a b c) = (apresentaSierp a) ++ (apresentaSierp b) ++ (apresentaSierp c)
```

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

```
tipsTLTree :: TLTree b → [b]
countTLTree :: TLTree b → Int
depthTLTree :: TLTree b → Int
invTLTree :: TLTree b → TLTree b
```

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 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:

```
test06a = depthTLTree ts ≡ 6
test06b = countTLTree ts ≡ 243
test06c = countTLTree ts ≡ length (tipsTLTree ts)
test06d = countTLTree ts ≡ countTLTree (invTLTree ts)
```

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:

- Desenharam-se os triângulos, utilizando:

```
drawTriangle :: ((Int, Int), Int) → String
```

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

```
finalize :: String → 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 = do { writeFile "__.html" html; system "sensible-browser __.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](#).

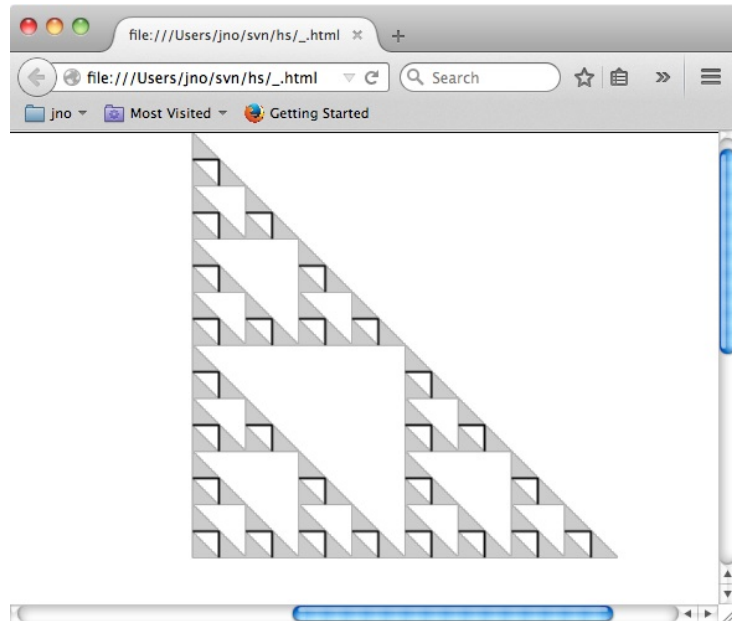


Figure 2: Um triângulo de Sierpinski em x3dom

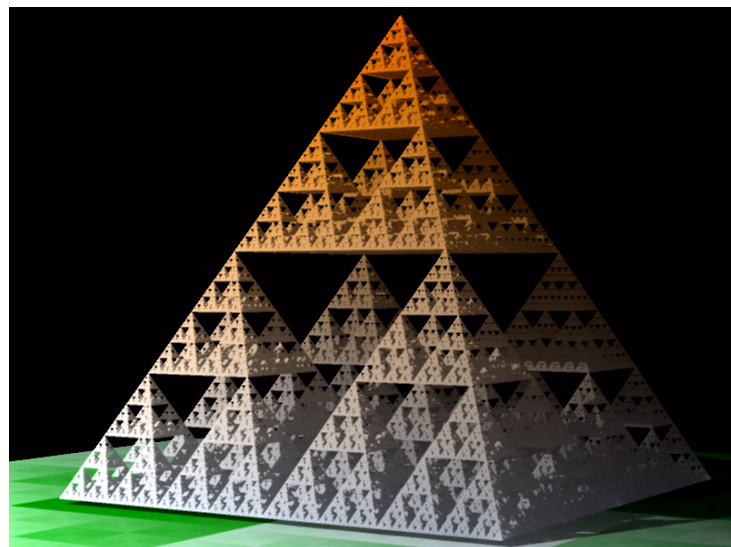


Figure 3: Uma pirâmide de Sierpinski

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

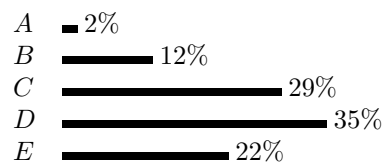
6.1 Mónades

Os mónades são funtores 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

```
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 $d :: \text{Dist } a$ 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 [( 'A' , 0.02), ( 'B' , 0.12), ( 'C' , 0.29), ( 'D' , 0.35), ( 'E' , 0.22) ]
```

que o **GHCI** mostrará assim: [fontsize=] '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 "Uma frase de cinco palavras")
```

isto é [fontsize=] "Uma" 20.0"cinco" 20.0"de" 20.0"frase" 20.0"palavras" 20.0 distribuição *normais*, eg.

```
d3 = normal [10..20]
```

etc.³

Dist forma um **mónade** cuja unidade é $\text{return } a = D [(a, 1)]$ 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 $g : A \rightarrow \text{Dist } B$ e $f : B \rightarrow \text{Dist } C$ 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

```
do { x <- uniform [1..6]; y <- uniform [1..6]; return (x + y) }
```

e obter-se-á: [fontsize=] *Main> do { x <- uniform [1..6]; y <- uniform [1..6]; return(x+y) } 7 16.76 13.98 13.95 11.19 11.14 8.310 8.33 5.611 5.62 2.812 2.8 A soma mais provável é 7, com 16.7%.

²Ver <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>.

³Para 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 [?].

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:

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

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

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

obter-se-á: [fontsize=] 35 81.025 9.05 9.015 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

$$transmitir = pcataList \text{ gene}$$

descreve o comportamento do aparelho.

Testes unitários 7 Faça o seguinte teste unitário da sua versão para *gene*:

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

Responda então às perguntas do problema acima correndo a expressão:

$$transmitir (words "Vamos atacar hoje")$$

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,

$$\begin{aligned} rpar &:: a \rightarrow Eval a \\ rseq &:: a \rightarrow Eval a \end{aligned}$$

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

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

```

parmap :: (a -> b) -> [a] -> Eval [b]
parmap f [] = return []
parmap f (a : lt) = do
  a' <- rpar (f a)
  lt' <- parmap f lt
  return (a' : lt')

```

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

Se correremos 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 cores⁵ 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 · show · (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 · show · runEval · (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 -> b) -> (BTree a) -> 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.

⁵Intel Core 2 Duo a 2.53 GHz.

⁶Ver detalhes em [?].

References

- [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

```
main :: IO ()
main = getArgs >>= (\_ → null) → exemp_or_exer, errInvArgs
  where
    exemp_or_exer = (((≡) "exemplo") · head) → exemp, exer
    exemp = (((≡) 2) · length) → execExemp, errInvArgs
    execExemp = isPar → execExempPar, execExempSeq
    exer = (((≡) 2) · length) → execExer, errInvArgs
    execExer = isPar → execExerPar, execExerSeq
    execExempSeq = (putStrLn · show · (map fib) $ [20..30])
    execExempPar = (putStrLn · show · runEval · (parmap fib) $ [20..30])
```

B Bibliotecas e código auxiliar

```
errInvArgs :: a → IO ()
errInvArgs = _ $ putStrLn msgInvArgs
  where
    msgInvArgs = "Invalid arguments"
t2 = abpe (1,33)
execExerPar :: [String] → IO ()
execExerPar = (putStrLn · show · runEval · (parBTreeMap fib) $ t2)
execExerSeq :: [String] → IO ()
execExerSeq = (putStrLn · show · (fmap fib) $ t2)
isPar :: [String] → Bool
isPar = (((≡) "par") · head · tail) → True, False
pcataList g = mfoldr (curry (g · i2)) ((g · i1) ()) where
  mfoldr f d [] = d
  mfoldr f d (a : x) = do { y ← mfoldr f d x; f a y }
```

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:

```
html = tag "html" []
preamble = headx 'with' [title "CP/X3DOM generation", links, script]
body = tag "body" []
x3d = tag "x3d" [("width", "\"500px\""), ("height", "\"400px\"")]
scene = tag "scene" []
items = concat
links = ctag "link" [
  ("rel", quote "stylesheet"), ("type", quote "text/css"),
  ("href", quote "http://www.x3dom.org/x3dom/release/x3dom.css")]
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, z)))]
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

```

depth :: LTree a → Integer
depth = cataLTree [zero, succ · max]
balance :: LTree a → LTree a
balance = (anaLTree (lsplit)) · tips

```

Secção 4.2

```

qsplrit :: Integral a ⇒ (a, a) → Either () (a, ((a, a), (a, a)))
qsplrit (n, m) | (m < n) ∨ (n ≡ 0 ∧ m ≡ 0) = i1 ()
  | n < m = let p = n + (m - n) + 1 ÷ 2 in i2 ((p, ((n, p - 1), (p + 1, m))))
  | otherwise = i2 ((n, ((n, m - 1), (n + 1, m))))

```

Secção 4.3

```

inSList :: Either a (a1, SList a1 a) → SList a1 a
inSList = [Sent, Cons]

```

```

outSList :: SList b a → Either a (b, SList b a)
outSList (Sent b) = i1 b
outSList (Cons (a, s)) = i2 (a, s)
recSList f = id + id × f
anaSList :: (c → Either a (b, c)) → c → SList b a
anaSList g = inSList · recSList (anaSList g) · g
cataSList :: (Either b (a, d) → d) → SList a b → d
cataSList g = g · recSList (cataSList g) · outSList
hyloSList :: (Either b (d, c) → c) → (a → Either b (d, a)) → a → c
hyloSList h g = cataSList h · anaSList g
mgen :: Ord a ⇒ ([a], [a]) → Either [a] (a, ([a], [a]))
mgen (l, []) = i1 l
mgen ([], l) = i1 l
mgen ((x : xs), (y : ys)) | x ≤ y = i2 (x, (xs, (y : ys)))
  | otherwise = i2 (y, ((x : xs), ys))
{- mgen :: Ord a ⇒ ([a], [a]) → Either [a] (a, ([a], [a]))
mgen (l, []) = i1 l
mgen ([], l) = i1 l
mgen = cond (uncurry

```

Secção 5.2

```

inTLTree = [L, N]
outTLTree (L a) = i1 a
outTLTree (N (t1, (t2, t3))) = i2 (t1, (t2, t3))
baseTLTree g f = g + (f × (f × f))
recTLTree f = id + (f × (f × f))
anaTLTree g = inTLTree · recTLTree (anaTLTree g) · g
cataTLTree g = g · recTLTree (cataTLTree g) · outTLTree
hyloTLTree h g = cataTLTree h · anaTLTree g
tipsTLTree = cataTLTree [singl, conc · (id × conc)]
invTLTree = cataTLTree [L, N · assoc · swap · (id × swap)]
depthTLTree = fromIntegral · (cataTLTree [one, succ · (max) · (id × (max))])
transFirst = (id × ('div'2)) × (-1+)
transSnd (((x, y), c), n) = (((x + c ÷ 2, y), c ÷ 2), n - 1)
transTh (((x, y), c), n) = (((x, y + c ÷ 2), c ÷ 2), n - 1)
geneSierp :: (Tri, Int) → Either Tri ((Tri, Int), ((Tri, Int), (Tri, Int)))
geneSierp (t, n) | n ≡ 0 = i1 t
  | otherwise = i2 (transFirst (t, n), (transSnd (t, n), transTh (t, n)))
geraSierp = curry (anaTLTree geneSierp)
apresentaSierp :: TLTree Tri → [Tri]
apresentaSierp = tipsTLTree
sierpinski :: Tri → Int → [Tri]
sierpinski t = apresentaSierp · (geraSierp t)
countTLTree :: TLTree b → Int
countTLTree = fromIntegral · (cataTLTree [one, add · (id × add)])
instance Functor TLTree
  where fmap f = cataTLTree (inTLTree · baseTLTree f id)
draw = render html where
  html = rep dados
rep = finalize · concat · tipsTLTree · (fmap drawTriangle) · (geraSierp)
type Pir = (Point3D, Side)
type Point3D = (Int, Int, Int)

```

```

data PLTree a = Pi a | No (PLTree a, (PLTree a, (PLTree a, (PLTree a, PLTree a)))) deriving (Eq, Show)
inPLTree = [Pi, No]
outPLTree (Pi a) = i1 a
outPLTree (No (t1, (t2, (t3, (t4, t5))))) = i2 (t1, (t2, (t3, (t4, t5))))
basePLTree g f = g + (f × (f × (f × (f × f))))
recPLTree f = id + (f × (f × (f × (f × f))))
anaPLTree g = inPLTree · recPLTree (anaPLTree g) · g
cataPLTree g = g · recPLTree (cataPLTree g) · outPLTree
hyloPLTree h g = cataPLTree h · anaPLTree g
instance Functor PLTree
  where fmap f = cataPLTree (inPLTree · basePLTree f id)
  tPir1 ((p, c), n) = let comp = (c 'div' 2) in ((p, comp), n - 1)
  tPir2 (((x, y, z), c), n) = let comp = (c 'div' 2) in (((x + comp, y, z), comp), n - 1)
  tPir3 (((x, y, z), c), n) = let comp = (c 'div' 2) in (((x, y + comp, z), comp), n - 1)
  tPir4 (((x, y, z), c), n) = let comp = (c 'div' 2) in (((x, y + comp, z + comp), comp), n - 1)
  tPir5 (((x, y, z), c), n) = let comp = (c 'div' 2) in (((x + comp, y + comp, z), comp), n - 1)
  genePiramide (p, n) | n ≡ 0 = i1 p
    | otherwise = i2 (tPir1 (p, n), (tPir2 (p, n), (tPir3 (p, n), (tPir4 (p, n), tPir5 (p, n)))))
  geraPiramide = curry (anaPLTree genePiramide)
  apresentaPiramide = cataPLTree [singl, conc · (id × conc) · (id × (id × conc)) · (id × (id × (id × conc)))]
  pierp t = apresentaPiramide · (geraPiramide t)
  drawPyramid :: Pir → String
  drawPyramid ((x, y, z), side) = ⊥
  draw' = render html' where
    html' = rep' dados'
  dados' = (((0, 0, 0), 10), 2)
  rep' = finalize · concat · apresentaPiramide · (fmap drawPyramid) · (geraPiramide)

```

Seção 6.2

Defina

```

gene (i2 (x, y)) = D [(x : y), 0.95], (y, 0.05)]
gene (i1 ()) = D [["stop"], 0.90], ([], 0.1)]

```

e responda ao problema do enunciado aqui.

Seção 6.4

Defina

```

parBTreeMap f Empty = return Empty
parBTreeMap f (Node (r, (d, e))) = do {
  root ← rpar (f r);
  esq ← parBTreeMap f d;
  dir ← parBTreeMap f e;
  return (Node (root, (esq, dir)))
}

```

```

cp1415 exercicio seq +RTS -s -N4
Total time 4.05s ( 2.69s elapsed)
cp1415 exercicio par +RTS -s -N4
Total time 3.34s ( 1.34s elapsed)

```

Index

Cálculo de Programas, [3](#)

Material Pedagógico, [2](#), [3](#), [6](#)

BTree.hs, [4](#), [6](#), [10](#)

List.hs, [9](#)

LTree.hs, [3](#), [4](#), [6](#)

Combinador “pointfree”

either, [4](#), [9](#), [12–14](#)

Fractal, [4](#)

Pirâmide de Sierpinski, [7](#)

Triângulo de Sierpinski, [4](#), [5](#), [7](#)

Função

uncurry, [12–14](#)

Haskell, [2](#)

“Literate Haskell”, [2](#)

lhs2TeX, [2](#)

Biblioteca

PFP, [8](#)

Probability, [8](#)

Control

Parallel.Strategies, [9](#)

interpretador

GHCi, [3](#), [8](#)

Programação literária, [2](#)

U.Minho

Departamento de Informática, [1](#)

Utilitário

LaTeX

bibtex, [3](#)

makeindex, [3](#)

X3DOM, [6](#), [11](#)