Cálculo de Programas
Trabalho Prático
MiEI+LCC — 2017/18

Departamento de Informática
Universidade do Minho

Junho de 2018

Grupo nr.	62
a81451	Alexandre Rodrigues
a82145	Filipa Parente
a81403	Pedro Ferreira

1 Preâmbulo

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

Na sequência pedagógica dos planos de estudo dos dois cursos que têm esta disciplina, restringe-se a aplicação deste método à programação funcional em Haskell. Assim, o presente trabalho prático coloca os alunos perante problemas concretos que deverão ser implementados em Haskell. Há ainda um outro objectivo: o de ensinar a documentar programas e a produzir textos técnico-científicos de qualidade.

2 Documentação

Para cumprir de forma integrada os objectivos enunciados acima vamos recorrer a uma técnica de programação dita "literária" [1], cujo princípio base é o seguinte:

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

Por outras palavras, o código fonte e a documentação de um programa deverão estar no mesmo ficheiro.

O ficheiro `cp1718t.pdf` que está a ler é já um exemplo de programação literária: foi gerado a partir do texto fonte `cp1718t.lhs`[1] que encontrará no material pedagógico desta disciplina descompactando o ficheiro `cp1718t.zip` e executando

```
$ lhs2TeX cp1718t.lhs > cp1718t.tex
$ pdflatex cp1718t
```

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

```
$ cabal install lhs2tex
```

Por outro lado, o mesmo ficheiro `cp1718t.lhs` é executável e contém o "kit" básico, escrito em Haskell, para realizar o trabalho. Basta executar

```
$ ghci cp1718t.lhs
```

Abra o ficheiro `cp1718t.lhs` no seu editor de texto preferido e verifique que assim é: todo o texto que se encontra dentro do ambiente

[1] O 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 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, no local respectivo da folha de rosto.

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

```
$ bibtex cp1718t.aux
$ makeindex cp1718t.idx
```

e recompilar o texto como acima se indicou. Dever-se-á ainda instalar o utilitário QuickCheck, que ajuda a validar programas em Haskell, a biblioteca JuicyPixels para processamento de imagens e a biblioteca gloss para geração de gráficos 2D:

```
$ cabal install QuickCheck JuicyPixels gloss
```

Para testar uma propriedade QuickCheck *prop*, basta invocá-la com o comando:

```
> quickCheck prop
+++ OK, passed 100 tests.
```

Problema 1

Segundo uma notícia do Jornal de Notícias, referente ao dia 12 de abril, *"apenas numa hora, foram transacionadas 1.2 mil milhões de dólares em bitcoins. Nas últimas 24 horas, foram transacionados 8,5 mil milhões de dólares, num total de 24 mil milhões de dólares referentes às principais criptomoedas"*.

De facto, é inquestionável que as criptomoedas, e em particular as bitcoin, vieram para ficar. Várias moedas digitais, e em particular as bitcoin, usam a tecnologia de block chain para guardar e assegurar todas as transações relacionadas com a moeda. Uma block chain é uma coleção de blocos que registam os movimentos da moeda; a sua definição em Haskell é apresentada de seguida.

$$\mathbf{data}\ \mathit{Blockchain} = \mathit{Bc}\ \{\mathit{bc} :: \mathit{Block}\} \mid \mathit{Bcs}\ \{\mathit{bcs} :: (\mathit{Block}, \mathit{Blockchain})\}\ \mathbf{deriving}\ \mathit{Show}$$

Cada bloco numa block chain regista um número (mágico) único, o momento da execução, e uma lista de transações, tal como no código seguinte:

$$\mathbf{type}\ \mathit{Block} = (\mathit{MagicNo}, (\mathit{Time}, \mathit{Transactions}))$$

Cada transação define a entidade de origem da transferência, o valor a ser transacionado, e a entidade destino (por esta ordem), tal como se define de seguida.

$$\mathbf{type}\ \mathit{Transaction} = (\mathit{Entity}, (\mathit{Value}, \mathit{Entity}))$$
$$\mathbf{type}\ \mathit{Transactions} = [\mathit{Transaction}]$$

A partir de uma block chain, é possível calcular o valor que cada entidade detém, tipicamente designado de ledger:

$$\mathbf{type}\ \mathit{Ledger} = [(\mathit{Entity}, \mathit{Value})]$$

Seguem as restantes definições Haskell para completar o código anterior. Note que $Time$ representa o momento da transação, como o número de milisegundos que passaram desde 1970.

$$
\begin{array}{l}
\mathbf{type}\ MagicNo = String \\
\mathbf{type}\ Time = Int \quad \text{-- em milisegundos} \\
\mathbf{type}\ Entity = String \\
\mathbf{type}\ Value = Int
\end{array}
$$

Neste contexto, implemente as seguintes funções:

1. Defina a função $allTransactions :: Blockchain \rightarrow Transactions$, como um catamorfismo, que calcula a lista com todas as transações numa dada block chain.

   Propriedade QuickCheck 1 *As transações de uma block chain são as mesmas da block chain revertida:*

   $$prop1a = sort \cdot allTransactions \equiv sort \cdot allTransactions \cdot reverseChain$$

   Note que a função sort é usada apenas para facilitar a comparação das listas.

2. Defina a função $ledger :: Blockchain \rightarrow Ledger$, utilizando catamorfismos e/ou anamorfismos, que calcula o ledger (i.e., o valor disponível) de cada entidade numa uma dada block chain. Note que as entidades podem ter valores negativos; de facto isso acontecerá para a primeira transação que executarem.

   Propriedade QuickCheck 2 *O tamanho do ledger é inferior ou igual a duas vezes o tamanho de todas as transações:*

   $$prop1b = \mathsf{length} \cdot ledger \leqslant (2*) \cdot \mathsf{length} \cdot allTransactions$$

   Propriedade QuickCheck 3 *O ledger de uma block chain é igual ao ledger da sua inversa:*

   $$prop1c = sort \cdot ledger \equiv sort \cdot ledger \cdot reverseChain$$

3. Defina a função $isValidMagicNr :: Blockchain \rightarrow Bool$, utilizando catamorfismos e/ou anamorfismos, que verifica se todos os números mágicos numa dada block chain são únicos.

   Propriedade QuickCheck 4 *A concatenação de uma block chain com ela mesma nunca é válida em termos de números mágicos:*

   $$prop1d = \neg \cdot isValidMagicNr \cdot concChain \cdot \langle id, id \rangle$$

   Propriedade QuickCheck 5 *Se uma block chain é válida em termos de números mágicos, então a sua inversa também o é:*

   $$prop1e = isValidMagicNr \Rightarrow isValidMagicNr \cdot reverseChain$$

Problema 2

Uma estrutura de dados frequentemente utilizada para representação e processamento de imagens de forma eficiente são as denominadas quadtrees. Uma *quadtree* é uma árvore quaternária em que cada nodo tem quatro sub-árvores e cada folha representa um valor bi-dimensional.

$$
\begin{array}{l}
\mathbf{data}\ QTree\ a = Cell\ a\ Int\ Int \mid Block\ (QTree\ a)\ (QTree\ a)\ (QTree\ a)\ (QTree\ a) \\
\quad \mathbf{deriving}\ (Eq, Show)
\end{array}
$$

```
( 0 0 0 0 0 0 0 0 )
( 0 0 0 0 0 0 0 0 )
( 0 0 0 0 1 1 1 0 )
( 0 0 0 0 1 1 0 0 )
( 1 1 1 1 1 1 0 0 )
( 1 1 1 1 1 1 0 0 )
( 1 1 1 1 0 0 0 0 )
( 1 1 1 1 0 0 0 1 )
```

(a) Matriz de exemplo bm.

```
Block
(Cell 0 4 4) (Block
(Cell 0 2 2) (Cell 0 2 2) (Cell 1 2 2) (Block
(Cell 1 1 1) (Cell 0 1 1) (Cell 0 1 1) (Cell 0 1 1)))
(Cell 1 4 4)
(Block
(Cell 1 2 2) (Cell 0 2 2) (Cell 0 2 2) (Block
(Cell 0 1 1) (Cell 0 1 1) (Cell 0 1 1) (Cell 1 1 1)))
```

(b) Quadtree de exemplo qt.

Figura 1: Exemplos de representações de bitmaps.

Uma imagem monocromática em formato bitmap pode ser representada como uma matriz de bits[2], tal como se exemplifica na Figura 1a.

O anamorfismo $bm2qt$ converte um bitmap em forma matricial na sua codificação eficiente em quadtrees, e o catamorfismo $qt2bm$ executa a operação inversa:

$$\begin{array}{l} bm2qt :: (Eq\ a) \Rightarrow Matrix\ a \rightarrow QTree\ a \\ bm2qt = anaQTree\ f\ \mathbf{where} \\ \quad f\ m = \mathbf{if}\ one\ \mathbf{then}\ i_1\ u\ \mathbf{else}\ i_2\ (a, (b, (c, d))) \\ \quad\quad \mathbf{where}\ x = (nub \cdot toList)\ m \\ \quad\quad\quad\quad u = (head\ x, (ncols\ m, nrows\ m)) \\ \quad\quad\quad\quad one = (ncols\ m \equiv 1 \vee nrows\ m \equiv 1 \vee \mathsf{length}\ \ x \equiv 1) \\ \quad\quad\quad\quad (a, b, c, d) = splitBlocks\ (nrows\ m\ `div`\ 2)\ (ncols\ m\ `div`\ 2)\ m \end{array}$$

$$\begin{array}{l} qt2bm :: (Eq\ a) \Rightarrow QTree\ a \rightarrow Matrix\ a \\ qt2bm = cataQTree\ [f, g]\ \mathbf{where} \\ \quad f\ (k, (i, j)) = matrix\ j\ i\ \underline{k} \\ \quad g\ (a, (b, (c, d))) = (a \updownarrow b) \leftrightarrow (c \updownarrow d) \end{array}$$

O algoritmo $bm2qt$ particiona recursivamente a imagem em 4 blocos e termina produzindo folhas para matrizes unitárias ou quando todos os píxeis de um sub-bloco têm a mesma côr. Para a matriz bm de exemplo, a quadtree correspondente $qt = bm2qt\ bm$ é ilustrada na Figura 1b.

Imagens a cores podem ser representadas como matrizes de píxeis segundo o código de cores RGBA, codificado no tipo $PixelRGBA8$ em que cada pixel é um quádruplo de valores inteiros $(red, green, blue, alpha)$ contidos entre 0 e 255. Atente em alguns exemplos de cores:

$$\begin{array}{ll} whitePx & = PixelRGBA8\ 255\ 255\ 255\ 255 \\ blackPx & = PixelRGBA8\ 0\ 0\ 0\ 255 \\ redPx & = PixelRGBA8\ 255\ 0\ 0\ 255 \end{array}$$

O módulo BMP, disponibilizado juntamente com o enunciado, fornece funções para processar ficheiros de imagem bitmap como matrizes:

$$\begin{array}{l} readBMP :: FilePath \rightarrow \mathsf{IO}\ (Matrix\ PixelRGBA8) \\ writeBMP :: FilePath \rightarrow Matrix\ PixelRGBA8 \rightarrow \mathsf{IO}\ () \end{array}$$

Teste, por exemplo, no $GHCi$, carregar a Figura 2a:

```
> readBMP "cp1718t_media/person.bmp"
```

Esta questão aborda operações de processamento de imagens utilizando quadtrees:

1. Defina as funções $rotateQTree :: QTree\ a \rightarrow QTree\ a$, $scaleQTree :: Int \rightarrow QTree\ a \rightarrow QTree\ a$ e $invertQTree :: QTree\ a \rightarrow QTree\ a$, como catamorfismos e/ou anamorfismos, que rodam[3], redimensionam [4] e invertem as cores de uma quadtree[5], respectivamente. Tente produzir imagens similares às Figuras 2b, 2c e 2d:

```
> rotateBMP  "cp1718t_media/person.bmp" "person90.bmp"
> scaleBMP 2 "cp1718t_media/person.bmp" "personx2.bmp"
> invertBMP  "cp1718t_media/person.bmp" "personinv.bmp"
```

[2] Cf. módulo $Data.Matrix$.

[3] Segundo um ângulo de 90º no sentido dos ponteiros do relógio.

[4] Multiplicando o seu tamanho pelo valor recebido.

[5] Um pixel pode ser invertido calculando $255 - c$ para cada componente c de cor RGB, exceptuando o componente alpha.

(a) Bitmap de exemplo.

(b) Rotação.

(c) Redimensionamento.

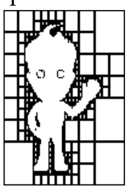
(d) Inversão de cores.

(e) Compresão de 1 nível.

(f) Compresão de 2 níveis.

(g) Compresão de 3 níveis.

(h) Compresão de 4 níveis.



(i) Bitmap de contorno.

(j) Bitmap com contorno.

Figura 2: Manipulação de uma figura bitmap utilizando quadtrees.

Propriedade QuickCheck 6 *Rodar uma quadtree é equivalente a rodar a matriz correspondente:*

$$prop2c = rotateMatrix \cdot qt2bm \equiv qt2bm \cdot rotateQTree$$

Propriedade QuickCheck 7 *Redimensionar uma imagem altera o seu tamanho na mesma proporção:*

$$prop2d\ (Nat\ s) = sizeQTree \cdot scaleQTree\ s \equiv ((s*) \times (s*)) \cdot sizeQTree$$

Propriedade QuickCheck 8 *Inverter as cores de uma quadtree preserva a sua estrutura:*

$$prop2e = shapeQTree \cdot invertQTree \equiv shapeQTree$$

2. Defina a função $compressQTree :: Int \rightarrow QTree\ a \rightarrow QTree\ a$, utilizando catamorfismos e/ou anamorfismos, que comprime uma quadtree cortando folhas da árvore para reduzir a sua profundidade num dado número de níveis. Tente produzir imagens similares (mas não necessariamente iguais) às Figuras 2e, 2f, 2g e 2h:

```
> compressBMP 1 "cp1718t_media/person.bmp" "person1.bmp"
> compressBMP 2 "cp1718t_media/person.bmp" "person2.bmp"
> compressBMP 3 "cp1718t_media/person.bmp" "person3.bmp"
> compressBMP 4 "cp1718t_media/person.bmp" "person4.bmp"
```

Propriedade QuickCheck 9 *A quadtree comprimida tem profundidade igual à da quadtree original menos a taxa de compressão:*

$$prop2f\ (Nat\ n) = depthQTree \cdot compressQTree\ n \equiv (-n) \cdot depthQTree$$

3. Defina a função $outlineQTree :: (a \rightarrow Bool) \rightarrow QTree\ a \rightarrow Matrix\ Bool$, utilizando catamorfismos e/ou anamorfismos, que recebe uma função que determina quais os píxeis de fundo e converte uma quadtree numa matriz monocromática, de forma a desenhar o contorno de uma malha poligonal contida na imagem. Tente produzir imagens similares (mas não necessariamente iguais) às Figuras 2i e 2j:

```
> outlineBMP    "cp1718t_media/person.bmp" "personOut1.bmp"
> addOutlineBMP "cp1718t_media/person.bmp" "personOut2.bmp"
```

Propriedade QuickCheck 10 *A matriz de contorno tem dimensões iguais às da quadtree:*

$$prop2g = sizeQTree \equiv sizeMatrix \cdot outlineQTree\ (<0)$$

Teste unitário 1 *Contorno da quadtree de exemplo qt:*

$$teste2a = outlineQTree\ (\equiv 0)\ qt \equiv qtOut$$

Problema 3

O cálculo das combinações de n k-a-k,

$$\left(\begin{array}{c} n \\ k \end{array}\right) = \frac{n!}{k! * (n-k)!} \tag{1}$$

envolve três factoriais. Recorrendo à lei de recursividade múltipla do cálculo de programas, é possível escrever o mesmo programa como um simples ciclo-for onde se fazem apenas multiplicações e somas. Para isso, começa-se por estruturar a definição dada da forma seguinte,

$$\left(\begin{array}{c} n \\ k \end{array}\right) = h\ k\ (n-k)$$

Figura 3: Passos de construção de uma árvore de Pitágoras de ordem 3.

onde

$$
\begin{array}{l}
h\ k\ d = \dfrac{f\ k\ d}{g\ d} \\
f\ k\ d = \dfrac{(d+k)!}{k!} \\
g\ d = d!
\end{array}
$$

assumindo-se $d = n - k \geqslant 0$. É fácil de ver que $f\ \ k$ e g se desdobram em 4 funções mutuamente recursivas, a saber

$$
\begin{array}{l}
f\ k\ 0 = 1 \\
f\ k\ (d+1) = \underbrace{(d+k+1)}_{l\ k\ d} * f\ k\ d \\
l\ k\ 0 = k+1 \\
l\ k\ (d+1) = l\ k\ d + 1
\end{array}
$$

e

$$
\begin{array}{l}
g\ 0 = 1 \\
g\ (d+1) = \underbrace{(d+1)}_{s\ d} * g\ d \\
s\ 0 = 1 \\
s\ (d+1) = s\ d + 1
\end{array}
$$

A partir daqui alguém derivou a seguinte implementação:

$$
\binom{n}{k} = h\ k\ (n-k)\ \textbf{where}\ h\ k\ n = \textbf{let}\ (a, \_, b, \_) = \mathsf{for}\ loop\ (base\ k)\ n\ \textbf{in}\ a \mathbin{/} b
$$

Aplicando a lei da recursividade múltipla para $\langle f\ k, l\ k \rangle$ e para $\langle g, s \rangle$ e combinando os resultados com a lei de banana-split, derive as funções $base\ k$ e $loop$ que são usadas como auxiliares acima.

Propriedade QuickCheck 11 *Verificação que* $\binom{n}{k}$ *coincide com a sua especificação (1):*

$$
prop3\ (NonNegative\ n)\ (NonNegative\ k) = k \leqslant n \Rightarrow \binom{n}{k} \equiv n! \mathbin{/} (k! * (n-k)!)
$$

Problema 4

Fractais são formas geométricas que podem ser construídas recursivamente de acordo com um conjunto de equações matemáticas. Um exemplo clássico de um fractal são as árvores de Pitágoras. A construção de uma árvore de Pitágoras começa com um quadrado, ao qual se unem dois quadrados redimensionados pela escala $\sqrt{2}/2$, de forma a que os cantos dos 3 quadrados coincidam e formem um triângulo rectângulo isósceles. Este procedimento é repetido recursivamente de acordo com uma dada ordem, definida como um número natural (Figura 3).

Uma árvore de Pitágoras pode ser codificada em Haskell como uma full tree contendo quadrados nos nodos e nas folhas, sendo um quadrado definido simplesmente pelo tamanho do seu lado:

$$\begin{array}{l}\mathbf{data}\ FTree\ a\ b = Unit\ b \mid Comp\ a\ (FTree\ a\ b)\ (FTree\ a\ b)\ \mathbf{deriving}\ (Eq, Show)\\ \mathbf{type}\ PTree = FTree\ Square\ Square\\ \mathbf{type}\ Square = Float\end{array}$$

1. Defina a função $generatePTree :: Int \to PTree$, como um anamorfismo, que gera uma árvore de Pitágoras para uma dada ordem.

   Propriedade QuickCheck 12 *Uma árvore de Pitágoras tem profundidade igual à sua ordem:*

   $$prop4a\ (SmallNat\ n) = (depthFTree \cdot generatePTree)\ n \equiv n$$

   Propriedade QuickCheck 13 *Uma árvore de Pitágoras está sempre balanceada:*

   $$prop4b\ (SmallNat\ n) = (isBalancedFTree \cdot generatePTree)\ n$$

2. Defina a função $drawPTree :: PTree \to [Picture]$, utilizando catamorfismos e/ou anamorfismos, que anima incrementalmente os passos de construção de uma árvore de Pitágoras recorrendo à biblioteca gloss. Anime a sua solução:

   ```
   > animatePTree 3
   ```

Problema 5

Uma das áreas em maior expansão no campo da informática é a análise de dados e machine learning. Esta questão aborda um *mónade* que ajuda a fazer, de forma simples, as operações básicas dessas técnicas. Esse mónade é conhecido por *bag*, *saco* ou *multi-conjunto*, permitindo que os elementos de um conjunto tenham multiplicidades associadas. Por exemplo, seja

$$\mathbf{data}\ Marble = Red \mid Pink \mid Green \mid Blue \mid White\ \mathbf{deriving}\ (Read, Show, Eq, Ord)$$

um tipo dado.[6] A lista $[Pink, Green, Red, Blue, Green, Red, Green, Pink, Blue, White]$ tem elementos repetidos. Assumindo que a ordem não é importante, essa lista corresponde ao saco

```
{ Red |-> 2 , Pink |-> 2 , Green |-> 3 , Blue |-> 2 , White |-> 1 }
```

que habita o tipo genérico dos "bags":

$$\mathbf{data}\ Bag\ a = B\ [(a, Int)]\ \mathbf{deriving}\ (Ord)$$

O mónade que vamos construir sobre este tipo de dados faz a gestão automática das multiciplidades. Por exemplo, seja dada a função que dá o peso de cada berlinde em gramas:

$$\begin{array}{l}marbleWeight :: Marble \to Int\\ marbleWeight\ Red\ \ = 3\\ marbleWeight\ Pink = 2\\ marbleWeight\ Green = 3\\ marbleWeight\ Blue\ = 6\\ marbleWeight\ White = 2\end{array}$$

Então, se quisermos saber quantos *berlindes* temos, de cada *peso*, não teremos que fazer contas: basta calcular

$$marbleWeights = \mathsf{fmap}\ marbleWeight\ bagOfMarbles$$

onde $bagOfMarbles$ é o saco de berlindes referido acima, obtendo-se:

```
{ 2 |-> 3 , 3 |-> 5 , 6 |-> 2 }.
```

[6] "Marble" traduz para "berlinde" em português.

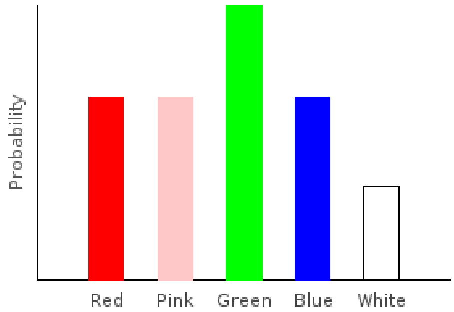


Figura 4: Distribuição de berlindes num saco.

Mais ainda, se quisermos saber o total de berlindes em $bagOfMarbles$ basta calcular fmap $(!)$ $bagOfMarbles$ obtendo-se `{ () |-> 10 }`; isto é, o saco tem 10 berlindes no total.

Finalmente, se quisermos saber a probabilidade da cor de um berlinde que tiremos do saco, basta converter o referido saco numa distribuição correndo:

$$marblesDist = dist\ bagOfMarbles$$

obtendo-se a distribuição (graças ao módulo Probability):

```
Green   30.0%
  Red   20.0%
 Pink   20.0%
 Blue   20.0%
White   10.0%
```

cf. Figura 4.

Partindo da seguinte declaração de Bag como um functor e como um mónade,

instance $Functor\ Bag$ **where**
  $\mathsf{fmap}\ f = B \cdot \mathsf{map}\ \ (f \times id) \cdot unB$
instance $Monad\ Bag$ **where**
  $x \gg\!\!= f = (\mu \cdot \mathsf{fmap}\ f)\ x$ **where**
  $return = singletonbag$

1. Defina a função μ (multiplicação do mónade Bag) e a função auxiliar $singletonbag$.
2. Verifique-as com os seguintes testes unitários:

Teste unitário 2 *Lei* $\mu \cdot return = id$*:*

$$test5a = bagOfMarbles \equiv \mu\ (return\ bagOfMarbles)$$

Teste unitário 3 *Lei* $\mu \cdot \mu = \mu \cdot \mathsf{fmap}\ \mu$*:*

$$test5b = (\mu \cdot \mu)\ b3 \equiv (\mu \cdot \mathsf{fmap}\ \mu)\ b3$$

onde $b3$ *é um saco dado em anexo.*

Referências

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

Anexos

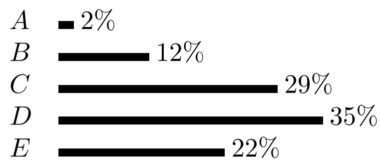
A Mónade para probabilidades e estatística

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

$$\mathbf{newtype}\ \mathsf{Dist}\ a = D\ \{ unD :: [(a, ProbRep)] \} \tag{2}$$

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

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



será representada pela distribuição

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

que o GHCi mostrará assim:

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

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.

B Definições auxiliares

Funções para mostrar *bags*:

$$\begin{array}{l}
\mathbf{instance}\ (Show\ a, Ord\ a, Eq\ a) \Rightarrow Show\ (Bag\ a)\ \mathbf{where} \\
\quad show = showbag \cdot consol \cdot unB\ \mathbf{where} \\
\quad\quad showbag = concat\ \cdot \\
\quad\quad\quad (+\!\!+ [\texttt{"\ \}"}]) \cdot (\texttt{"\{\ "}:) \cdot \\
\quad\quad\quad (intersperse\ \texttt{"\ ,\ "}) \cdot \\
\quad\quad\quad sort\ \cdot \\
\quad\quad\quad (\mathsf{map}\ \ f)\ \mathbf{where}\ f\ (a, b) = (show\ a) +\!\!+ \texttt{"\ |->\ "} +\!\!+ (show\ b) \\
unB\ (B\ x) = x
\end{array}$$

Igualdade de *bags*:

$$\begin{array}{l}
\mathbf{instance}\ (Eq\ a) \Rightarrow Eq\ (Bag\ a)\ \mathbf{where} \\
\quad b \equiv b' = (unB\ b)\ \text{`}lequal\text{`}\ (unB\ b') \\
\quad\quad \mathbf{where}\ lequal\ a\ b = isempty\ (a \ominus b) \\
\quad\quad\quad ominus\ a\ b = a +\!\!+ neg\ b \\
\quad\quad\quad neg\ x = [(k, -i) \mid (k, i) \leftarrow x]
\end{array}$$

Ainda sobre o mónade Bag:

$$
\begin{array}{l}
\textbf{instance } \mathit{Applicative\ Bag} \textbf{ where} \\
\quad \mathit{pure} = \mathit{return} \\
\quad (<*>) = \mathit{aap}
\end{array}
$$

O exemplo do texto:

$$\mathit{bagOfMarbles} = B\ [(\mathit{Pink}, 2), (\mathit{Green}, 3), (\mathit{Red}, 2), (\mathit{Blue}, 2), (\mathit{White}, 1)]$$

Um valor para teste (bags de bags de bags):

$$
\begin{array}{l}
\mathit{b3} :: \mathit{Bag}\ (\mathit{Bag}\ (\mathit{Bag\ Marble})) \\
\mathit{b3} = B\ [(B\ [(B\ [(\mathit{Pink}, 2), (\mathit{Green}, 3), (\mathit{Red}, 2), (\mathit{Blue}, 2), (\mathit{White}, 1)], 5) \\
\quad , (B\ [(\mathit{Pink}, 1), (\mathit{Green}, 2), (\mathit{Red}, 1), (\mathit{Blue}, 1)], 2)], 2)]
\end{array}
$$

Outras funções auxiliares:

$$
\begin{array}{l}
a \mapsto b = (a, b) \\
\mathit{consol} :: (\mathit{Eq}\ b) \Rightarrow [(b, \mathit{Int})] \rightarrow [(b, \mathit{Int})] \\
\mathit{consol} = \mathit{filter\ nzero} \cdot \mathsf{map}\ (\mathit{id} \times \mathit{sum}) \cdot \mathit{col}\ \textbf{where}\ \mathit{nzero}\ (\_, x) = x \not\equiv 0 \\
\mathit{isempty} :: \mathit{Eq}\ a \Rightarrow [(a, \mathit{Int})] \rightarrow \mathit{Bool} \\
\mathit{isempty} = \mathit{all}\ (\equiv 0) \cdot \mathsf{map}\ \pi_2 \cdot \mathit{consol} \\
\mathit{col}\ x = \mathit{nub}\ [k \mapsto [d' \mid (k', d') \leftarrow x, k' \equiv k] \mid (k, d) \leftarrow x] \\
\mathit{consolidate} :: \mathit{Eq}\ a \Rightarrow \mathit{Bag}\ a \rightarrow \mathit{Bag}\ a \\
\mathit{consolidate} = B \cdot \mathit{consol} \cdot \mathit{unB}
\end{array}
$$

C Soluções dos alunos

Os alunos devem colocar neste anexo as suas soluções aos exercícios propostos, de acordo com o "layout" que se fornece. Não podem ser alterados os nomes ou tipos das funções dadas, mas pode ser adicionado texto e / ou outras funções auxiliares que sejam necessárias.

Problema 1

Para a resolução deste problema tivemos como base o hilomorfismo de Blockchain

Anamorfismo de Blockchain:

$$
\begin{tikzcd}
A* \arrow[r, "f"] \arrow[d, "\mathit{anaBlockchain}\ f"'] & 1 + (A \times A*) \arrow[d, "\mathit{id} + (\mathit{id} \times \mathit{anaBlockchain}\ f)"] \\
\mathit{Blockchain} & \mathit{Block} + (\mathit{Block} \times \mathit{Blockchain}) \arrow[l, "\mathit{inBlockchain}"]
\end{tikzcd}
$$

Catamorfismo de Blockchain:

$$
\begin{tikzcd}
\mathit{Blockchain} \arrow[d, "\mathit{cataBlockchain}\ g"'] & \mathit{Block} + (\mathit{Block} \times \mathit{Blockchain}) \arrow[l, "\mathit{inBlockchain}"'] \arrow[d, "\mathit{id} + \mathit{id} \times \mathit{cataBlockchain}\ g"] \\
A* & 1 + (A \times A*) \arrow[l, "g"]
\end{tikzcd}
$$

definindo-se assim as seguintes funções

$$
\begin{array}{l}
inBlockchain = [Bc, Bcs] \\
outBlockchain\ (Bc\ b) = i_1\ (b) \\
outBlockchain\ (Bcs\ bs) = i_2\ (bs) \\
recBlockchain\ f = id + id \times (f) \\
cataBlockchain\ g = g \cdot (recBlockchain\ (cataBlockchain\ g)) \cdot outBlockchain \\
anaBlockchain\ g = inBlockchain \cdot (recBlockchain\ (anaBlockchain\ g)) \cdot g \\
hyloBlockchain\ c\ a = cataBlockchain\ c \cdot anaBlockchain\ a
\end{array}
$$

allTransactions

Utiliza um catamorfismo cujo caso base é a lista de transações e a recursividade concatena a lista do bloco com a lista resultante.

$$allTransactions = cataBlockchain\ [\pi_2 \cdot \pi_2, \widehat{(+\!\!+)} \cdot ((\pi_2 \cdot \pi_2) \times id)]$$

ledger

O catamorfismo transforma a Blockchain numa lista de Ledgers com elementos repetidos. De seguida o `ledgeNub` utiliza um `cataList` para unir todas os elementos repetidos.

O `b2l` retira a Ledger de um Block com o `t2l` que retira a Ledger de uma lista de transações.

$$\mathbf{type}\ Ledge = (Entity, Value)$$

$$ledger = ledgeNub \cdot cataBlockchain\ [b2l, \widehat{(+\!\!+)} \cdot (b2l \times id)]$$

$$
\begin{array}{l}
ledgeNub = cataList\ [\underline{([\,])}, \widehat{(aux)}] \\
\quad \mathbf{where} \\
\qquad aux :: Ledge \to Ledger \to Ledger \\
\qquad aux\ s\ [\,] = [s] \\
\qquad aux\ s\ (x : xs) = cond\ p\ f\ g\ (x, xs) \\
\qquad\quad \mathbf{where} \\
\qquad\qquad p :: (Ledge, Ledger) \to Bool \\
\qquad\qquad p = (\pi_1\ s\ \equiv) \cdot \pi_1 \cdot \pi_1 \\
\qquad\qquad f :: (Ledge, Ledger) \to Ledger \\
\qquad\qquad f = \widehat{(:)} \cdot ((id \times ((\pi_2\ s) +)) \times id) \\
\qquad\qquad g :: (Ledge, Ledger) \to Ledger \\
\qquad\qquad g = \widehat{(:)} \cdot (id \times (aux\ s)) \\
b2l :: Block \to Ledger \\
b2l\ b = (concat \cdot (\mathsf{map}\ \ t2l) \cdot \pi_2 \cdot \pi_2)\ b \\
t2l :: Transaction \to Ledger \\
t2l\ (origin, (value, destination)) = [(origin, -value), (destination, value)]
\end{array}
$$

isValidMagicNr

Compara a length da Blockchain com a length do resultado de um `nub`.

$$isValidMagicNr = \widehat{(\equiv)} \cdot \langle lenChain, \mathsf{length}\ \cdot nub \cdot cataBlockchain\ [return \cdot \pi_1, \widehat{(:)} \cdot (\pi_1 \times id)]\rangle$$

Problema 2

Para a resolução deste problema tivemos como base o hilomorfismo de QTree

Anamorfismo de QTree:

$$\begin{tikzcd}
Matrix\ a \arrow[r, "f"] \arrow[d, "anaQTree\ f"'] & QTree\ a + (Matrix\ a \times (Matrix\ a \times (Matrix\ a \times Matrix\ a))) \arrow[d, "id + (anaQTree\ f \times (anaQTree\ f \times (anaQTree\ f \times anaQTree\ f)))"] \\
QTree\ a & QTree\ a + (QTree\ a \times (QTree\ a \times (QTree\ a \times QTree\ a))) \arrow[l, "inQTree"]
\end{tikzcd}$$

Catamorfismo de QTree:

$$\begin{tikzcd}
QTree\ a \arrow[r, "outQTree"] \arrow[d, "cataQTree\ g"'] & QTree\ a + (QTree\ a \times (QTree\ a \times (QTree\ a \times QTree\ a))) \arrow[d, "id + (cataQTree\ g \times (cataQTree\ g \times (cataQTree\ g \times cataQTree\ g)))"] \\
Matrix\ a & QTree\ a + (Matrix\ a \times (Matrix\ a \times (Matrix\ a \times Matrix\ a))) \arrow[l, "g"]
\end{tikzcd}$$

definindo-se assim as seguintes funções

data $QTree\ a = Cell\ a\ Int\ Int \mid Block\ (QTree\ a)\ (QTree\ a)\ (QTree\ a)\ (QTree\ a)$
  deriving $(Eq, Show)$

$inQTreeCell :: (a, (Int, Int)) \rightarrow QTree\ a$
$inQTreeCell\ x = Cell\ (\pi_1\ x)\ ((\pi_1 \cdot \pi_2)\ x)\ ((\pi_2 \cdot \pi_2)\ x)$

$inQTreeBlock :: (QTree\ a, (QTree\ a, (QTree\ a, QTree\ a))) \rightarrow QTree\ a$
$inQTreeBlock\ x = Block\ (\pi_1\ x)\ ((\pi_1 \cdot \pi_2)\ x)\ ((\pi_1 \cdot \pi_2 \cdot \pi_2)\ x)\ ((\pi_2 \cdot \pi_2 \cdot \pi_2)\ x)$

$inQTree :: (a, (Int, Int)) + (QTree\ a, (QTree\ a, (QTree\ a, QTree\ a))) \rightarrow QTree\ a$
$inQTree = [inQTreeCell, inQTreeBlock]$

$outQTree :: QTree\ a \rightarrow (a, (Int, Int)) + (QTree\ a, (QTree\ a, (QTree\ a, QTree\ a)))$
$outQTree\ (Cell\ a\ x\ y) = i_1\ (a, (x, y))$
$outQTree\ (Block\ a\ b\ c\ d) = i_2\ (a, (b, (c, d)))$

$baseQTree :: (a1 \rightarrow b) \rightarrow (a2 \rightarrow d1) \rightarrow (a1, d2) + (a2, (a2, (a2, a2))) \rightarrow (b, d2) + (d1, (d1, (d1, d1)))$
$baseQTree\ f\ g = (f \times id) + (g \times (g \times (g \times g)))$

$baseQTreeScale :: (d2 \rightarrow b) \rightarrow (a2 \rightarrow d1) \rightarrow (a1, d2) + (a2, (a2, (a2, a2))) \rightarrow (a1, b) + (d1, (d1, (d1, d1)))$
$baseQTreeScale\ f\ g = (id \times f) + (g \times (g \times (g \times g)))$

$recQTree :: (a \rightarrow d1) \rightarrow (b, d2) + (a, (a, (a, a))) \rightarrow (b, d2) + (d1, (d1, (d1, d1)))$
$recQTree\ f = baseQTree\ id\ f$

$cataQTree :: ((b, (Int, Int)) + (d, (d, (d, d))) \rightarrow d) \rightarrow QTree\ b \rightarrow d$
$cataQTree\ g = g \cdot (recQTree\ (cataQTree\ g)) \cdot outQTree$

$anaQTree :: (a1 \rightarrow (a2, (Int, Int)) + (a1, (a1, (a1, a1)))) \rightarrow a1 \rightarrow QTree\ a2$
$anaQTree\ f = inQTree \cdot (recQTree\ (anaQTree\ f)) \cdot f$

$hyloQTree :: ((b, (Int, Int)) + (c, (c, (c, c))) \rightarrow c) \rightarrow (a \rightarrow (b, (Int, Int)) + (a, (a, (a, a)))) \rightarrow a \rightarrow c$
$hyloQTree\ g\ f = cataQTree\ g \cdot anaQTree\ f$

instance $Functor\ QTree$ **where**
  fmap $f = cataQTree\ (inQTree \cdot baseQTree\ f\ id)$

A resolução das alíneas foi feita com base nos diagramas anteriormente apresentados.

rotateQTree

Para a resolução desta questão definiu-se uma função que fazia a rotação dos blocos `myfunction. Também`se definiu uma função (`rotateAux`) que dado o functor de QTree fazia apenas a rotação dos blocos mantendo intactas as células.
Fica a faltar a troca dos tamanhos das células que é feito na função principal.

-- funções auxiliares–

$myfunction :: (a, (a, (a, a))) \rightarrow (a, (a, (a, a)))$
$myfunction\ (x, (y, (z, w))) = (z, (x, (w, y)))$

$rotateAux :: (b, d2) + (a, (a, (a, a))) \rightarrow (b, d2) + (a, (a, (a, a)))$
$rotateAux = id + myfunction$

-- função principal–

$rotateQTree = inQTree \cdot (baseQTreeScale\ swap\ rotateQTree) \cdot rotateAux \cdot outQTree$

scaleQTree

Para a resolução desta questão apenas se fez o catamorfismo da função `scaleAux`a todas as células da àrvore

-- função auxiliar–

$scaleAux :: Int \rightarrow (Int, Int) \rightarrow (Int, Int)$
$scaleAux\ n\ (a, b) = (n * a, n * b)$

-- função principal–

$scaleQTree\ n = cataQTree\ (inQTree \cdot baseQTreeScale\ (scaleAux\ n)\ id)$

invertQTree

Para a resolução desta questão apenas se fez o catamorfismo da função `changeColor`a todas as células da àrvore

-- função auxiliar–

$changeColor :: PixelRGBA8 \rightarrow PixelRGBA8$
$changeColor\ (PixelRGBA8\ x\ y\ z\ w) = PixelRGBA8\ (255 - x)\ (255 - y)\ (255 - z)\ (255 - w)$

-- função principal–

$invertQTree = cataQTree\ (inQTree \cdot baseQTree\ (changeColor)\ id)$

compressQTree

Para resolver esta questão recorreu-se a uma função auxiliar que cortava a QTree a uma determinada profundidade `compressQTreeAux`. Enquanto não chegasse à profundidade de corte fazia a recursividade da função nos blocos. Quando se chegasse lá usava-se a função `cutQTree` que substituia os blocos por células com o tamanho (`sizeQTree)` do bloco respetivo.

-- funções auxiliares–

$celuralize :: QTree\ a \rightarrow a$
$celuralize\ (Cell\ x\ y\ z) = x$
$celuralize\ (Block\ o\ b\ c\ d) = celuralize\ o$

$cutQTree :: QTree\ a \rightarrow QTree\ a$
$cutQTree\ q = Cell\ (celuralize\ q)\ ((\pi_1 \cdot sizeQTree)\ q)\ ((\pi_2 \cdot sizeQTree)\ q)$

$compressQTreeAux :: Int \rightarrow QTree\ a \rightarrow QTree\ a$
$compressQTreeAux\ n = \mathbf{if}\ (n > 0)\ \mathbf{then}\ cataQTree\ (inQTree \cdot recQTree\ (compressQTreeAux\ (n - 1)))$
$\mathbf{else}\ cutQTree$

-- função principal–

$compressQTree\ n\ q = compressQTreeAux\ ((depthQTree\ q) - n)\ q$

outlineQTree

Esta questão necessitou de dois passos para ser resolvida:

- aplicação da negação da função f a todas as células da àrvore
- definição da função `outlineAux` que faz o catamorfismo de uma função f que dada uma célula do tipo Bool, caso ela fosse False, preenchia-se as bordas com True, caso contrário preenchia-se matriz a False, e uma função g que junta as submatrizes formadas.

-- funções auxiliares–

$pintaCell :: a \rightarrow Int \rightarrow Int \rightarrow Matrix\ a \rightarrow Matrix\ a$

$pintaCell\ n\ r\ c = (mapRow\ (\backslash\_x \rightarrow n)\ 1) \cdot (mapRow\ (\backslash\_x \rightarrow n)\ r) \cdot (mapCol\ (\backslash\_x \rightarrow n)\ 1) \cdot (mapCol\ (\backslash\_x \rightarrow n)$

$inQTreeAdapted :: (Bool, (Int, Int)) \rightarrow Matrix\ Bool$

$inQTreeAdapted\ (a, (b, c)) = \mathbf{if}\ (a)\ \mathbf{then}\ (qt2bm \cdot \mathsf{fmap}\ (\neg) \cdot inQTreeCell)\ (a, (b, c))$

$\quad \mathbf{else}\ ((pintaCell\ True\ b\ c) \cdot qt2bm \cdot inQTreeCell)\ (a, (b, c))$

$outlineAux :: QTree\ Bool \rightarrow Matrix\ Bool$

$outlineAux = (cataQTree\ [f, g])$

$\quad \mathbf{where}$

$\quad\quad f\ (x, (y, z)) = inQTreeAdapted\ (x, (y, z))$

$\quad\quad g\ (a, (b, (c, d))) = joinBlocks\ (a, b, c, d)$

-- função principal–

$outlineQTree\ f = (outlineAux) \cdot (\mathsf{fmap}\ (\neg \cdot f))$

Problema 3

Para a resolução desta questão foi necessário simplificar as expressões

$$\left\{ \begin{array}{l} f\ k\ 0 = 1 \\ f\ k\ (d+1) = (l\ k\ d) * f\ k\ d \end{array} \right.$$

$$\left\{ \begin{array}{l} l\ k\ 0 = k+1 \\ l\ k\ (d+1) = (l\ k\ d) + 1 \end{array} \right.$$

$$\left\{ \begin{array}{l} s\ 0 = 1 \\ s\ (d+1) = (s\ d) + 1 \end{array} \right.$$

$$\left\{ \begin{array}{l} g\ 0 = 1 \\ g\ (d+1) = (s\ d) * g\ d \end{array} \right.$$

Então:

$$\begin{array}{ll}
& \left\{ \begin{array}{l} f\ k\ 0 = 1 \\ f\ k\ (d+1) = (l\ k\ d) * f\ k\ d \end{array} \right. \\
\equiv & \quad \{\ \text{def succ, def mul, 73}\ \} \\
& \left\{ \begin{array}{l} f\ k \cdot 0 = 1 \\ f\ k \cdot \mathsf{succ}\ = mul \cdot \langle l\ k, f\ k \rangle \end{array} \right. \\
= & \quad \{\ 27, 20\ \} \\
& f\ k \cdot [0, \mathsf{succ}\,] = [1, mul\ (l\ k, f\ k)] \\
\equiv & \quad \{\ \text{def in}\ \} \\
& f\ k \cdot \mathbf{in} = [1, mul \cdot \langle l\ k, f\ k \rangle]
\end{array}$$

$$\left\{ \begin{array}{l} l\ k\ 0 = k+1 \\ l\ k\ (d+1) = (l\ k\ d) + 1 \end{array} \right.$$

$$\equiv \quad \{ \text{ def succ, 73 } \}$$

$$\left\{ \begin{array}{l} l\ k \cdot 0 = \mathsf{succ}\ \cdot k \\ l\ k \cdot \mathsf{succ}\ = \mathsf{succ}\ \cdot l\ k \end{array} \right.$$

$$\equiv \quad \{ \text{ 27, 20 } \}$$

$$l\ k \cdot [0, \mathsf{succ}\ ] = [\mathsf{succ}\ \cdot k, \mathsf{succ}\ \cdot l\ k]$$

$$\equiv \quad \{ \text{ def in } \}$$

$$l\ k \cdot \mathbf{in} = [\mathsf{succ}\ \cdot k, \mathsf{succ}\ \cdot l\ k]$$

e

$$\left\{ \begin{array}{l} g\ 0 = 1 \\ g\ (d+1) = (s\ d) * g\ d \end{array} \right.$$

$$\equiv \quad \{ \text{ def succ, def mul, 73 } \}$$

$$\left\{ \begin{array}{l} g \cdot 0 = 1 \\ g \cdot \mathsf{succ}\ = mul\ (s, g) \end{array} \right.$$

$$\equiv \quad \{ \text{ 27, 20 } \}$$

$$g \cdot [0, \mathsf{succ}\ ] = [1, mul \cdot \langle s, g \rangle]$$

$$\equiv \quad \{ \text{ def in } \}$$

$$g \cdot \mathbf{in} = [1, mul \cdot \langle s, g \rangle]$$

$$\left\{ \begin{array}{l} s\ 0 = 1 \\ s\ (d+1) = (s\ d) + 1 \end{array} \right.$$

$$\equiv \quad \{ \text{ def succ, 73 } \}$$

$$\left\{ \begin{array}{l} s \cdot 0 = 1 \\ s \cdot \mathsf{succ}\ = \mathsf{succ}\ \cdot s \end{array} \right.$$

$$\equiv \quad \{ \text{ 27, 20 } \}$$

$$s \cdot [0, \mathsf{succ}\ ] = [1, \mathsf{succ}\ \cdot s]$$

$$\equiv \quad \{ \text{ def in } \}$$

$$s \cdot \mathbf{in} = [1, \mathsf{succ}\ \cdot s]$$

Após isso foi usada a lei de `Fokkinga(50)` com o functor dos naturais

$$f\ k \cdot \mathbf{in} = h \cdot id + \langle f\ k, l\ k \rangle$$
$$l\ k \cdot \mathbf{in} = k \cdot id + \langle f\ k, l\ k \rangle$$

e

$$g \cdot \mathbf{in} = h \cdot id + \langle g, s \rangle$$
$$s \cdot \mathbf{in} = k \cdot id + \langle g, s \rangle$$

Neste momento ainda não se sabia qual era o `h` e o `k`. Para isso foram usadas as igualdades anteriormente obtidas

$$h \cdot id + (f\ k, l\ k) = [1, mul \cdot \langle l\ k, f\ k \rangle]$$

$$\equiv \quad \{ \text{ 18, 22 } \}$$

$$[h \cdot i_1 \cdot id, h \cdot i_2 \cdot \langle f\ k, l\ k \rangle] = [1, mul \cdot \langle l\ k, f\ k \rangle]$$

$$\equiv \quad \{\ 27\ \}$$

$$\begin{cases} h \cdot i_1 = 1 \\ h \cdot i_2 \cdot \langle l\ k, f\ k \rangle = mul \cdot \langle l\ k, f\ k \rangle \end{cases}$$

$$\equiv \quad \{\ 73\ \}$$

$$\begin{cases} h \cdot i_1 = 1 \\ h \cdot i_2 = mul \end{cases}$$

$$\equiv \quad \{\ 17\ \}$$

$$h = [1, mul, \cdot]$$

$$k \cdot id + (f\ k, l\ k) = [\mathsf{succ}\ \cdot k, \mathsf{succ}\ \cdot l\ k]$$

$$\equiv \quad \{\ 18, 22\ \}$$

$$[k \cdot i_1 \cdot id, k \cdot i_2 \cdot \langle f\ k, l\ k \rangle] = [\mathsf{succ}\ \cdot k, \mathsf{succ}\ \cdot l\ k]$$

$$\equiv \quad \{\ 27\ \}$$

$$\begin{cases} k \cdot i_1 = \mathsf{succ}\ \cdot k \\ k \cdot i_2 \cdot \langle f\ k, l\ k \rangle = \mathsf{succ}\ \cdot l\ k \end{cases}$$

$$\equiv \quad \{\ 7, 73\ \}$$

$$\begin{cases} k \cdot i_1 = \mathsf{succ}\ \cdot k \\ k \cdot i_2 = \mathsf{succ}\ \cdot \pi_2 \end{cases}$$

$$\equiv \quad \{\ 17\ \}$$

$$k = [\mathsf{succ}\ \cdot k, \mathsf{succ}\ \cdot \pi_2]$$

e

$$h \cdot id + (g, s) = [1, mul\ (s)\ (g)]$$

$$\equiv \quad \{\ 18, 22\ \}$$

$$[h \cdot i_1 \cdot id, h \cdot i_2 \cdot \langle g, s \rangle] = [1, mul \cdot \langle s, g \rangle]$$

$$\equiv \quad \{\ 27\ \}$$

$$\begin{cases} h \cdot i_1 = 1 \\ h \cdot i_2 \cdot \langle g, s \rangle = mul \cdot \langle s, g \rangle \end{cases}$$

$$\equiv \quad \{\ 73\ \}$$

$$\begin{cases} h \cdot i_1 = 1 \\ h \cdot i_2 = mul \end{cases}$$

$$\equiv \quad \{\ 17\ \}$$

$$h = [1, mul]$$

$$k \cdot id + (g, s) = [1, \mathsf{succ}\ \cdot s]$$

$$\equiv \quad \{ 18, 22 \}$$

$$[k \cdot i_1 \cdot id, k \cdot i_2 \cdot \langle g, s \rangle] = [1, \mathsf{succ}\ \cdot s]$$

$$\equiv \quad \{\ 27\ \}$$

$$\begin{cases} k \cdot i_1 = 1 \\ k \cdot i_2 \cdot \langle g, s \rangle = \mathsf{succ}\ \cdot s \end{cases}$$

$$\equiv \quad \{\ 7, 73\ \}$$

$$\begin{cases} k \cdot i_1 = 1 \\ k \cdot i_2 = \mathsf{succ}\ \cdot \pi_2 \end{cases}$$

$$\equiv \quad \{\ 17\ \}$$

$$k = [1, \mathsf{succ} \cdot \pi_2]$$

Fica, assim, com a lei de `Fokkinga(50)`.

$$
\begin{aligned}
& f\ k \cdot \mathbf{in} = [1, mul] \cdot id + \langle f\ k, l\ k \rangle \\
& l\ k \cdot \mathbf{in} = [\mathsf{succ} \cdot k, \mathsf{succ} \cdot \pi_2] \cdot id + \langle f\ k, l\ k \rangle \\
\equiv \quad & \quad \{\ 50\ \} \\
& \langle f\ k, l\ k \rangle = (\!| \langle [1, mul], [\mathsf{succ} \cdot k, \mathsf{succ} \cdot \pi_2] \rangle |\!)
\end{aligned}
$$

e

$$
\begin{aligned}
& g \cdot \mathbf{in} = [1, mul] \cdot id + \langle g, s \rangle \\
& s \cdot \mathbf{in} = [1, \mathsf{succ} \cdot \pi_2] \cdot id + \langle g, s \rangle \\
\equiv \quad & \quad \{\ 50\ \} \\
& \langle g, s \rangle = (\!| \langle [1, mul], [1, \mathsf{succ} \cdot \pi_2] \rangle |\!)
\end{aligned}
$$

Juntou-se depois os dois pares e simplificou-se através do uso da lei de Banana-split (51) e de outras leis

$$
\begin{aligned}
& ((f\ k, l\ k), (g, s)) = \langle (\!| \langle [1, mul], [\mathsf{succ} \cdot k, \mathsf{succ} \cdot \pi_2] \rangle |\!), (\!| \langle [1, mul], [1, \mathsf{succ} \cdot \pi_2] \rangle |\!) \rangle \\
\equiv \quad & \quad \{\ 51\ \} \\
& (\!| (([1, mul], [\mathsf{succ} \cdot k, \mathsf{succ} \cdot \pi_2]) \times ([1, mul], [1, \mathsf{succ} \cdot \pi_2])) \cdot \langle id + \pi_1, id + \pi_2 \rangle |\!) \\
\equiv \quad & \quad \{\ 7, 22\ \} \\
& (\!| \langle \langle [1, mul \cdot \pi_1], [\mathsf{succ} \cdot k, \mathsf{succ} \cdot \pi_2 \cdot \pi_1] \rangle, \langle [1, mul \cdot \pi_2], [1, \mathsf{succ} \cdot \pi_2 \cdot \pi_2] \rangle \rangle |\!) \\
\equiv \quad & \quad \{\ 28\ \} \\
& (\!| \langle [\langle 1, \mathsf{succ} \cdot k \rangle, \langle mul \cdot \pi_1, \mathsf{succ} \cdot \pi_2 \cdot \pi_1 \rangle], [\langle 1, 1 \rangle, \langle mul \cdot \pi_2, \mathsf{succ} \cdot \pi_2 \cdot \pi_2 \rangle] \rangle |\!) \\
\equiv \quad & \quad \{\ 28\ \} \\
& (\!| [\langle \langle 1, \mathsf{succ} \cdot k \rangle, \langle 1, 1 \rangle \rangle, \langle \langle mul \cdot \pi_1, \mathsf{succ} \cdot \pi_2 \cdot \pi_1 \rangle, \langle mul \cdot \pi_2, \mathsf{succ} \cdot \pi_2 \cdot \pi_2 \rangle \rangle] |\!)
\end{aligned}
$$

Como

$$\mathsf{for}\ loop\ (base\ k) = (\!| [base\ k, loop] |\!)$$

, então obteve-se

$$
\begin{aligned}
& base\ k = ((1, \mathsf{succ} \cdot k), (1, 1)) \\
& loop = \langle mul, \mathsf{succ} \cdot \pi_2 \rangle \times \langle mul, \mathsf{succ} \cdot \pi_2 \rangle
\end{aligned}
$$

$\square$

Para colocar o tuplo de tuplos num 4-tuplo e vice-versa foram criadas as funções auxiliares `flatTotal`e `flatTotalRev`.
Ficaram, assim, as funções construidas

$$
\begin{aligned}
& base :: (Enum\ x, Num\ a, Num\ b, Num\ c) \Rightarrow x \rightarrow (a, x, b, c) \\
& base\ k = (1, \mathsf{succ}\ \ k, 1, 1) \\
& loop :: (Integer, Integer, Integer, Integer) \rightarrow (Integer, Integer, Integer, Integer) \\
& loop = flatTotal \cdot (\langle mul, \mathsf{succ} \cdot \pi_2 \rangle \times \langle mul, \mathsf{succ} \cdot \pi_2 \rangle) \cdot flatTotalRev \\
& flatTotal :: ((a, b), (c, d)) \rightarrow (a, b, c, d) \\
& flatTotal\ ((a, b), (c, d)) = (a, b, c, d) \\
& flatTotalRev :: (a, b, c, d) \rightarrow ((a, b), (c, d)) \\
& flatTotalRev\ (a, b, c, d) = ((a, b), (c, d))
\end{aligned}
$$

Problema 4

Para a resolução deste problema tivemos como base o anamorfismo e o catamorfismo de FTree

Anamorfismo de FTree:

$$
\begin{tikzcd}
A \arrow[rr, "f"] \arrow[d, "anaFTree\ f"'] & & C + (B \times (A \times A)) \arrow[d, "id+(id\times(anaFTree\ f \times anaFTree\ f))"] \\
FTree\ B\ C & & C + (B \times (FTree\ B\ C \times FTree\ B\ C)) \arrow[ll, "inFTree"]
\end{tikzcd}
$$

Catamorfismo de FTree:

$$
\begin{tikzcd}
FTree\ B\ C \arrow[rr, "outFTree"] \arrow[d, "cataFTree\ g"'] & & C + (B \times (FTree\ B\ C \times FTree\ B\ C)) \arrow[d, "id+(id\times(cataFTree\ g \times cataFTree\ g))"] \\
A & & C + (B \times (A \times A)) \arrow[ll, "g"]
\end{tikzcd}
$$

definindo-se assim as seguintes funções

$inFTreeUnit :: b \to FTree\ a\ b$
$inFTreeUnit\ b = Unit\ b$
$inFTreeComp :: (a, (FTree\ a\ b, FTree\ a\ b)) \to FTree\ a\ b$
$inFTreeComp\ x = Comp\ (\pi_1\ x)\ ((\pi_1 \cdot \pi_2)\ x)\ ((\pi_2 \cdot \pi_2)\ x)$
$inFTree :: b + (a, (FTree\ a\ b, FTree\ a\ b)) \to FTree\ a\ b$
$inFTree = [inFTreeUnit, inFTreeComp]$
$outFTree :: FTree\ a1\ a2 \to a2 + (a1, (FTree\ a1\ a2, FTree\ a1\ a2))$
$outFTree\ (Unit\ b) = i_1\ (b)$
$outFTree\ (Comp\ a\ t1\ t2) = i_2\ (a, (t1, t2))$
$baseFTree :: (a1 \to b1) \to (a2 \to b2) \to (a3 \to d) \to a2 + (a1, (a3, a3)) \to b2 + (b1, (d, d))$
$baseFTree\ f\ g\ h = g + (f \times (h \times h))$
$recFTree :: (a \to d) \to b1 + (b2, (a, a)) \to b1 + (b2, (d, d))$
$recFTree\ f\ g = baseFTree\ id\ id\ f\ g$
$cataFTree :: (b1 + (b2, (d, d)) \to d) \to FTree\ b2\ b1 \to d$
$cataFTree\ f = f \cdot (recFTree\ (cataFTree\ f)) \cdot outFTree$
$anaFTree :: (a1 \to b + (a2, (a1, a1))) \to a1 \to FTree\ a2\ b$
$anaFTree\ f = inFTree \cdot (recFTree\ (anaFTree\ f)) \cdot f$
$hyloFTree :: (b1 + (b2, (c, c)) \to c) \to (a \to b1 + (b2, (a, a))) \to a \to c$
$hyloFTree\ f\ g = cataFTree\ f \cdot anaFTree\ g$

instance $Bifunctor\ FTree$ **where**
$\quad bimap\ f\ g = cataFTree\ (inFTree \cdot baseFTree\ f\ g\ id)$

generatePTree

Para a resolução desta questão definiu-se uma função auxiliar (`generateFTree`) que, combinada com o anamorfismo da FTree, gera uma PTree que contém as iterações de uma árvore de pitágoras cujo valor de lado decresce com uma escala de $\sqrt{2}/2$ por iteração.
A função auxiliar `generateFTree`utiliza dois inteiros como argumentos, um não é alterado e guarda o número de iterações iniciais pretendidas e outro é sempre decrementado e guarda a iteração atual até chegar a zero. Esses dois argumentos são usados em conjunto na função para obter as vezes que é necessário aplicar a escala ao valor inicial de lado dependendo da iteração.
A função `generatePTree`é obtida através do anamorfismo da FTree com argumentos `generateFTreen` (n representa o inteiro que não é alterado) e `n`.

$$generatePTree\ n = anaFTree\ (generateFTree\ n)\ n$$

$$generateFTree :: Int \rightarrow Int \rightarrow Square + (Square, (Int, Int))$$
$$generateFTree\ nInicial\ n = \mathbf{if}\ (n \equiv 0)\ \mathbf{then}\ i_1\ (100 * (sqrt\ (2)\ /\ 2) \uparrow (nInicial))$$
$$\quad \mathbf{else}\ i_2\ (100 * (sqrt\ (2)\ /\ 2) \uparrow (nInicial - n), (n - 1, n - 1))$$

drawPtree

Efetua `fmap rt` rodando todos os elementos subconsequentes de [Picture]. A função `rt` roda e translada uma Picture, juntamente com `rt'` que faz o mesmo salvo dois sinais negativos.

$$wind :: Int \rightarrow \mathsf{IO}\ ()$$
$$wind = display\ window\ white \cdot pictures \cdot drawPTree \cdot generatePTree$$

$$drawPTree = cataFTree\ [return \cdot square, drawAux]$$

$$drawAux :: (Square, ([Picture], [Picture])) \rightarrow [Picture]$$
$$drawAux\ (c, (a : [\,], b : [\,])) = [(square\ c), a, b]$$
$$drawAux\ (c, (a, b)) = square\ c : (\mathsf{fmap}\ (rt\ c)\ a) +\!\!+ (\mathsf{fmap}\ (rt'\ c)\ b)$$

$$rt = (rotate\ 45\cdot) \cdot aap\ (translate \cdot negate \cdot ((1\ /\ 2)*) \cdot sqrt \cdot (/2) \cdot (\uparrow 2))\ (((3\ /\ 2)*) \cdot sqrt \cdot (/2) \cdot (\uparrow 2))$$
$$rt' = (rotate\ (-45)\cdot) \cdot aap\ (translate \cdot ((1\ /\ 2)*) \cdot sqrt \cdot (/2) \cdot (\uparrow 2))\ (((3\ /\ 2)*) \cdot sqrt \cdot (/2) \cdot (\uparrow 2))$$

Problema 5

Para a resolução deste problema foi necessário analisar o tipo de dados do mónade que vamos instanciar (neste caso é do tipo `Bag a`).
Então, a definição de μ foi feita em 3 passos:

- obtenção da lista de bags que estão dentro da bag principal;
- obtenção do conteúdo de cada bag pertencente à lista para posterior concatenação;
- transformação em bag.

$$\mu\ q = B\ ((concat \cdot \mathsf{fmap}\ (unB \cdot \pi_1) \cdot unB)\ q)$$

A definição de singletonBag foi apenas a transformação de um tipo `a` para `Bag`:

$$singletonbag\ b = (B\ [(b, 1)])$$

Para definir a função dist definiram-se 2 funções auxiliares:

- `totalBag` que devolve o número de berlindes do saco;
- `prob` que dado o número de berlindes do saco e o saco devolve a distribuição finalizada recorrendo ao fmap de listas.

$$totalBag :: Bag\ Marble \to Int$$
$$totalBag = \pi_2 \cdot head \cdot unB \cdot consolidate \cdot (\mathsf{fmap}\ (!))$$
$$f :: Int \to (a, Int) \to (a, ProbRep)$$
$$f\ n\ (x, y) = (x, (fromIntegral\ y\ /\ fromIntegral\ n))$$
$$prob :: Int \to Bag\ a \to \mathsf{Dist}\ a$$
$$prob\ n\ l = D\ (\mathsf{fmap}\ (f\ n)\ (unB\ l))$$

Assim a função dist resume-se à invocação da função `prob`:

$$dist\ b = (prob\ (totalBag\ b)\ b)$$

Índice