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

Departamento de Informática
Universidade do Minho

Junho de 2018

Grupo nr.	36
a82474	Ana Ribeiro
a82061	Jéssica Lemos
a82535	Pedro Pinto

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" [**?**], 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.

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

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

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

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.

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

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

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

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.

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

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.

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

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

(a) Matriz de exemplo bm.

(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:

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

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

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.

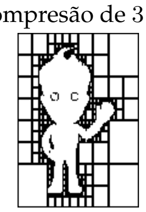
(d) Inversão de cores.

(e) Compresão de 1 nível.

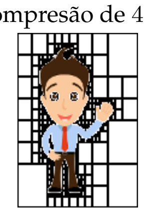
(f) Compresão de 2 níveis.

(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 $compressQTree :: Int \to QTree\ a \to QTree\ a$, 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 $outlineQTree :: (a \to Bool) \to QTree\ a \to Matrix\ Bool$, 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,

$$\binom{n}{k} = \frac{n!}{k! * (n-k)!} \tag{1}$$

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,

$$\binom{n}{k} = h\ k\ (n-k)$$

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

onde

$$h\ k\ d = \frac{f\ k\ d}{g\ d}$$
$$f\ k\ d = \frac{(d+k)!}{k!}$$
$$g\ d = d!$$

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

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

e

$$g\ 0 = 1$$
$$g\ (d+1) = \underbrace{(d+1)}_{s\ d} * g\ d$$
$$s\ 0 = 1$$
$$s\ (d+1) = s\ n+1$$

A partir daqui alguém derivou a seguinte implementação:

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

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):*

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

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:

$$\textbf{data}\ FTree\ a\ b = Unit\ b \mid Comp\ a\ (FTree\ a\ b)\ (FTree\ a\ b)\ \textbf{deriving}\ (Eq, Show)$$
$$\textbf{type}\ PTree = FTree\ Square\ Square$$
$$\textbf{type}\ Square = Float$$

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

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

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":

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

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.

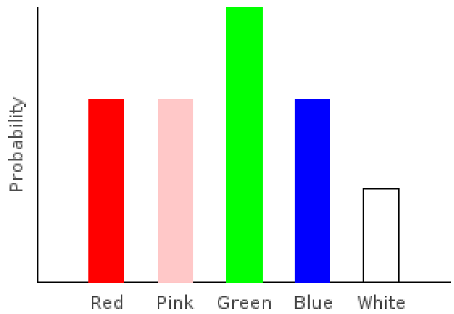


Figura 4: Distribuição de berlindes num saco.

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,

$$\begin{array}{l} \textbf{instance}\ Functor\ Bag\ \textbf{where} \\ \quad \mathsf{fmap}\ f = B \cdot \mathsf{map}\ (f \times id) \cdot unB \end{array}$$

$$\begin{array}{l} \textbf{instance}\ Monad\ Bag\ \textbf{where} \\ \quad x \gg\!= f = (\mu \cdot \mathsf{fmap}\ f)\ x\ \textbf{where} \\ \quad return = singletonbag \end{array}$$

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.*

Anexos

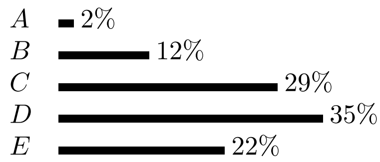
A Mónade para probabilidades e estatística

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

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

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*:

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

Igualdade de *bags*:

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

Ainda sobre o mónade Bag:

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

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):

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

Outras funções auxiliares:

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

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

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

Começamos por escrever em Haskell a função que devolve a lista com todas as transações numa dada block chain e de seguida aplicamos as Leis do Cálculo funcional de modo a obtermos o catamorfismo.

$$
\begin{array}{ll}
& \left\{\begin{array}{l} \mathit{allTransactions}\ (\mathit{Bc}\ (a, (b, c))) = c \\ \mathit{allTransactions}\ (\mathit{Bcs}\ ((a, (b, c)), d)) = c +\!\!+ \mathit{allTransactions}\ d \end{array}\right. \\
\equiv & \quad \{\ \text{Igualdade Extensional, Def-comp}\ \} \\
& \left\{\begin{array}{l} \mathit{allTransactions} \cdot \mathit{Bc} = \pi_2 \cdot \pi_2 \\ \mathit{allTransactions} \cdot \mathit{Bcs} = \pi_2 \cdot \pi_2 \cdot \pi_1 +\!\!+ \mathit{allTransactions} \cdot \pi_2 \end{array}\right. \\
\equiv & \quad \{\ \text{Definição conc, Def-split}\ \} \\
& \left\{\begin{array}{l} \mathit{allTransactions} \cdot \mathit{Bc} = \pi_2 \cdot \pi_2 \\ \mathit{allTransactions} \cdot \mathit{Bcs} = \mathsf{conc} \cdot \langle \pi_2 \cdot \pi_2 \cdot \pi_1, \mathit{allTransactions} \cdot \pi_2 \rangle \end{array}\right. \\
\equiv & \quad \{\ \text{Def-x}\ \}
\end{array}
$$

$$\begin{cases} allTransactions \cdot Bc = \pi_2 \cdot \pi_2 \\ allTransactions \cdot Bcs = \mathsf{conc} \cdot (\pi_2 \cdot \pi_2 \times allTransactions) \end{cases}$$

$\equiv$ { Functor-x, Natural-id }

$$\begin{cases} allTransactions \cdot Bc = \pi_2 \cdot \pi_2 \\ allTransactions \cdot Bcs = \mathsf{conc} \cdot ((\pi_2 \cdot \pi_2 \times id) \cdot (id \times allTransactions)) \end{cases}$$

$\equiv$ { Eq-+ }

$$[allTransactions \cdot Bc, allTransactions \cdot Bcs] = [\pi_2 \cdot \pi_2, \mathsf{conc} \cdot ((\pi_2 \cdot \pi_2 \times id) \cdot (id \times allTransactions))]$$

$\equiv$ { Fusao-+, Natural-id }

$$allTransactions \cdot [Bc, Bcs] = [\pi_2 \cdot \pi_2 \cdot id, \mathsf{conc} \cdot ((\pi_2 \cdot \pi_2 \times id) \cdot (id \times allTransactions))]$$

$\equiv$ { Absorção-+ }

$$allTransactions \cdot [Bc, Bcs] = [\pi_2 \cdot \pi_2, \mathsf{conc} \cdot (\pi_2 \cdot \pi_2 \times id)] \cdot (id + (id \times allTransactions))$$

$\equiv$ { Universal-cata }

$$allTransactions = cataBlockchain\ [\pi_2 \cdot \pi_2, \mathsf{conc} \cdot (\pi_2 \cdot \pi_2 \times id)]$$

$$\begin{tikzcd} Blockchain \arrow[d, "allTransactions"'] & Block + Block \times Blockchain \arrow[l, "inBlockchain"'] \arrow[d, "id + id \times allTransactions"] \\ Transactions & Block + Block \times Transactions \arrow[l, "f"] \end{tikzcd}$$

Desta forma, o catamorfismo obtido é:

$$allTransactions = cataBlockchain\ [\pi_2 \cdot \pi_2, \mathsf{conc} \cdot (\pi_2 \cdot \pi_2 \times id)]$$

Para a resolução da função *ledger* foi necessário recorrer a duas funções auxiliares. A *insere* que atualiza o valor disponível de uma entidade e caso esta não se encontre na lista em que as entidades estão associadas ao seu valor disponível, insere-a. E a função *ledgerAux* responsável por gerir as transações, ou seja, indica a cada entidade o montante da transação.

$insere :: Ledger \to Entity \to Value \to Ledger$
$insere\ [\,]\ entity\ value = [(entity, value)]$
$insere\ ((h, y) : t)\ entity\ value = \mathbf{if}\ (h \equiv entity)\ \mathbf{then}\ (h, (y + value)) : t\ \mathbf{else}\ (h, y) : insere\ t\ entity\ value$

$ledgerAux :: Transactions \to Ledger$
$ledgerAux\ [\,] = [\,]$
$ledgerAux\ ((a, (h, c)) : t) = insere\ (insere\ (ledgerAux\ t)\ a\ (h))\ c\ h$

Deste modo, a função em Haskell que permite calcular o valor disponível numa dada block chain é a seguinte:

$$\begin{cases} ledger\ (Bc\ (a, (b, c))) = ledgerAux\ c \\ ledger\ (Bcs\ ((a, (b, c)), d)) = ledgerAux\ c +\!\!+\ ledger\ d \end{cases}$$

$\equiv$ { Igualdade Extensional, Def-comp }

$$\begin{cases} ledger \cdot Bc = lerdgerAux \cdot \pi_2 \cdot \pi_2 \\ ledger \cdot Bcs = ledgerAux \cdot \pi_2 \cdot \pi_2 \cdot \pi_1 +\!\!+\ ledger \cdot \pi_2 \end{cases}$$

$\equiv$ { Definição conc, Def-split }

$$\begin{cases} ledger \cdot Bc = \pi_2 \cdot \pi_2 \\ ledger \cdot Bcs = \mathsf{conc} \cdot \langle ledgerAux \cdot \pi_2 \cdot \pi_2 \cdot \pi_1, ledger \cdot \pi_2 \rangle \end{cases}$$

$\equiv$ { Def-x }

$$\begin{cases} ledger \cdot Bc = \pi_2 \cdot \pi_2 \\ ledger \cdot Bcs = \mathsf{conc} \cdot (ledgerAux \cdot \pi_2 \cdot \pi_2 \times ledger) \end{cases}$$

$$\equiv \quad \{ \text{ Functor-x, Natural-id } \}$$

$$\begin{cases} ledger \cdot Bc = \pi_2 \cdot \pi_2 \\ ledger \cdot Bcs = \mathsf{conc} \cdot ((ledgerAux \cdot \pi_2 \cdot \pi_2 \times id) \cdot (id \times ledger)) \end{cases}$$

$$\equiv \quad \{ \text{ Eq-+ } \}$$

$$[ledger \cdot Bc, ledger \cdot Bcs] = [ledgerAux \cdot \pi_2 \cdot \pi_2, \mathsf{conc} \cdot ((ledgerAux \cdot \pi_2 \cdot \pi_2 \times id) \cdot (id \times ledger))]$$

$$\equiv \quad \{ \text{ Fusão-+, Natural-id } \}$$

$$ledger \cdot [Bc, Bcs] = [ledgerAux \cdot \pi_2 \cdot \pi_2 \cdot id, \mathsf{conc} \cdot ((ledgerAux \cdot \pi_2 \cdot \pi_2 \times id) \cdot (id \times ledger))]$$

$$\equiv \quad \{ \text{ Absorção-+ } \}$$

$$ledger \cdot [Bc, Bcs] = [ledgerAux \cdot \pi_2 \cdot \pi_2, \mathsf{conc} \cdot (ledgerAux \cdot \pi_2 \cdot \pi_2 \times id)] \cdot (id + (id \times ledger))$$

$$\equiv \quad \{ \text{ Universal-cata } \}$$

$$ledger = cataBlockchain\ [ledgerAux \cdot \pi_2 \cdot \pi_2, \mathsf{conc} \cdot (ledgerAux \cdot \pi_2 \cdot \pi_2 \times id)]$$

Através dos Leis do Cálculo Funcional, obtemos o catamorfismo que calcula o ledger, valor disponível de cada entidade numa dada block chain:

$$ledger = cataBlockchain\ [ledgerAux \cdot \pi_2 \cdot \pi_2, \mathsf{conc} \cdot ((ledgerAux \cdot \pi_2 \cdot \pi_2) \times id)]$$

Para desenvolver a função *isValidMagicNr* tornou-se imperativo implementar a função auxiliar *nMag* que guarda numa lista todos os números mágicos existentes numa block chain.

$$\begin{cases} nMag\ Bc\ (a, (x, y)) = [\,a\,] \\ nMag\ Bcs\ ((a, (x, y)), b) = [\,a\,] + nMag\ b \end{cases}$$

$$\equiv \quad \{ \text{ Def-comp, Igualdade extensional, definição conc e singl, Def-x } \}$$

$$\begin{cases} nMag \cdot Bc = singl \cdot \pi_1 \\ nMag \cdot Bcs = \mathsf{conc} \cdot (singl \cdot \pi_1 \times nMag) \end{cases}$$

$$\equiv \quad \{ \text{ Eq-+, Fusão-+ } \}$$

$$nMag \cdot [Bc, Bcs] = [singl \cdot \pi_1, \mathsf{conc} \cdot (singl \cdot \pi_1 \times nMag)]$$

$$\equiv \quad \{ \text{ Functor-x, Natural-id } \}$$

$$nMag \cdot [Bc, Bcs] = [singl \cdot \pi_1, \mathsf{conc} \cdot (singl \cdot \pi_1 \times id \cdot id \times nMag)]$$

$$\equiv \quad \{ \text{ Absorção-+ } \}$$

$$nMag \cdot [Bc, Bcs] = [singl \cdot \pi_1, \mathsf{conc} \cdot (singl \cdot \pi_1 \times id)] \cdot (id + id \times nMag)$$

$$\equiv \quad \{ \text{ Universal-cata } \}$$

$$nMag = cataBlockchain\ [singl \cdot \pi_1, \mathsf{conc} \cdot (singl \cdot \pi_1 \times id)]$$

$$\begin{tikzcd}
Blockchain \arrow[d, "nMags"'] & Block + Block \times Blockchain \arrow[l, "inBlockchain"'] \arrow[d, "id + id \times nMags"] \\
{[\,MagicNo\,]} & Block + Block \times [\,MagicNo\,] \arrow[l, "f"]
\end{tikzcd}$$

Pelos Leis do Cálculo Funcional, chegamos ao seguinte catamorfismo:

$$nMag = cataBlockchain\ [singl \cdot \pi_1, \mathsf{conc} \cdot (singl \cdot \pi_1 \times id)]$$

Foi ainda necessário elaborar uma função, *isValidAux*, que associasse a um determinado número mágico os seguintes números mágicos presentes na block chain. Assim, criamos uma lista de túplos em que o primeiro elemento representa o número mágico e o segundo uma lista com os seguintes.

$$\begin{cases} isValidAux\ [\,] = [\,] \\ isValidAux\ (h : t) = (h, t) : isValidAux\ t \end{cases}$$

$\equiv$ { Definição nil e cons, Def-comp, Igualdade Extensional, Def-split }

$$\begin{cases} isValidAux \cdot nil = nil \\ isValidAux \cdot cons = cons \cdot \langle id, isValidAux \cdot \pi_2 \rangle \end{cases}$$

$\equiv$ { Eq-+, Absorção-+, Fusão-+, Natural-id }

$$isValidAux \cdot [nil, cons] = [nil, cons] \cdot (id + \langle id, isValidAux \cdot \pi_2 \rangle)$$

$\equiv$ { Definição inList, Functor-+, Absorção-x }

$$isValidAux \cdot inList = inList \cdot (id + id \times isValidAux) \cdot (id + \langle id, \pi_2 \rangle)$$

$\equiv$ { Isomorfismo outList = inList, F f = id + id ¿¡ f }

$$outList \cdot isValidAux = (F\ \ isValidAux) \cdot (id + \langle id, \pi_2 \rangle) \cdot outList$$

$\equiv$ { Universal-ana }

$$isValidAux = anaList\ ((id + \langle id, \pi_2 \rangle) \cdot outList)$$

$$\begin{tikzcd}
[MagicNo] \arrow[d, "isValidAux"'] & 1 + (MagicNo \times [MagicNo]) \times [MagicNo] \arrow[l, "f"'] \arrow[d, "id+id\times isValidAux"] \\
[MagicNo \times [MagicNo]] & 1 + (MagicNo \times [MagicNo]) \times [MagicNo \times [MagicNo]] \arrow[l, "inList"]
\end{tikzcd}$$

A *isValidMagicNr* verifica se nenhum dos números mágicos é repetido para tal, averiguamos se nenhum dos primeiros elementos dos túplos se encontra na lista associada. Desta forma, recorremos ao anamorfismo de listas obtido da *isValidAux* que será aplicado à lista dos números mágicos proveniente da *nMag*.

$$\begin{aligned} & isValidMagicNr = (all\ elemMag) \cdot (anaList\ ((id + \langle id, \pi_2 \rangle) \cdot outList)) \cdot nMag \\ & \quad \textbf{where}\ elemMag = \widehat{notElem} \end{aligned}$$

Problema 2

$$\begin{aligned}
& uncurryQTree\ f = \lambda(x, (y, (z, w))) \rightarrow f\ x\ y\ z\ w \\
& uncurryQTree2\ f = \lambda(x, (z, w)) \rightarrow f\ x\ z\ w \\
& inQTree = [uncurryQTree2\ Cell, uncurryQTree\ Block] \\
& outQTree\ (Cell\ a\ b\ c) = i_1\ (a, (b, c)) \\
& outQTree\ (Block\ a\ b\ c\ d) = i_2\ (a, (b, (c, d))) \\
& baseQTree\ f\ g = (f \times id) + (g \times (g \times (g \times g))) \\
& recQTree\ f = baseQTree\ id\ f \\
& cataQTree\ g = g \cdot (recQTree\ (cataQTree\ g)) \cdot outQTree \\
& anaQTree\ f = inQTree \cdot (recQTree\ (anaQTree\ f)) \cdot f \\
& hyloQTree\ f\ g = cataQTree\ f \cdot anaQTree\ g \\
& \textbf{instance}\ Functor\ QTree\ \textbf{where} \\
& \quad \mathsf{fmap}\ g = cataQTree\ (inQTree \cdot baseQTree\ g\ id)
\end{aligned}$$

Para obtermos o anamorfismo da *rotateQTree* que permite rodar uma *quadtree* partimos da seguinte função em Haskell:

$$\begin{cases} rotateQTree\ (Cell\ a\ b\ c) = (Cell\ a\ c\ b) \\ rotateQTree\ (Block\ a\ b\ c\ d) = (Block\ c\ a\ d\ b) \end{cases}$$

$\equiv$ { Def-comp, Def-split, Igualdade Extencional, Absorção-x, definição swap }

$$\begin{cases} rotateQTree \cdot Cell = Cell \cdot (id \times swap) \\ rotateQTree \cdot Block = Block.rotateQTree \uparrow 4 \cdot \langle \pi_1 \cdot \pi_2 \cdot \pi_2, \langle \pi_1, \langle \pi_2 \cdot \pi_2 \cdot \pi_2, \pi_1 \cdot \pi_2 \rangle \rangle \rangle \end{cases}$$

$\equiv$ { Eq-+, Fusão-+, Definição inQTree }

$$rotateQTree \cdot inQTree = [Cell \cdot (id \times swap), Block.rotateQTree \uparrow 4 \cdot \langle \pi_1 \cdot \pi_2 \cdot \pi_2, \langle \pi_1, \langle \pi_2 \cdot \pi_2 \cdot \pi_2, \pi_1 \cdot \pi_2 \rangle \rangle \rangle]$$

$\equiv$ { Definição inQTree, Absorção-+ }

$$rotateQTree \cdot inQTree = inQTree \cdot (id \times swap) + rotateQTree \uparrow 4 \cdot \langle \pi_1 \cdot \pi_2 \cdot \pi_2, \langle \pi_1, \langle \pi_2 \cdot \pi_2 \cdot \pi_2, \pi_1 \cdot \pi_2 \rangle \rangle \rangle$$

$\equiv$ { Definição $F\ rotateQTree = id \times id + rotateQTree \uparrow 4$, Functor-+, Functor-x, Natural-id }

$$rotateQTree \cdot inQTree = inQTree \cdot F\ rotateQTree \cdot ((id \times swap) + \langle \pi_1 \cdot \pi_2 \cdot \pi_2, \langle \pi_1, \langle \pi_2 \cdot \pi_2 \cdot \pi_2, \pi_1 \cdot \pi_2 \rangle \rangle \rangle)$$

$\equiv$ { Isomorfismo outQTree = inQTree }

$$outQTree \cdot rotateQTree = F\ rotateQTree \cdot (id \times swap + \langle \pi_1 \cdot \pi_2 \cdot \pi_2, \langle \pi_1, \langle \pi_2 \cdot \pi_2 \cdot \pi_2, \pi_1 \cdot \pi_2 \rangle \rangle \rangle) \cdot outQTree$$

$\equiv$ { Universal-ana }

$$rotateQTree = anaQTree\ ((id \times swap + \langle \pi_1 \cdot \pi_2 \cdot \pi_2, \langle \pi_1, \langle \pi_2 \cdot \pi_2 \cdot \pi_2, \pi_1 \cdot \pi_2 \rangle \rangle \rangle) \cdot outQTree)$$

Pelas Leis do Cálculo Funcional foi possível representar a função pelo seguinte anamorfismo:

$$rotateQTree = anaQTree\ ((id \times swap + \langle \pi_1 \cdot \pi_2 \cdot \pi_2, \langle \pi_1, \langle \pi_2 \cdot \pi_2 \cdot \pi_2, \pi_1 \cdot \pi_2 \rangle \rangle \rangle) \cdot outQTree)$$

Aplicamos as Leis do Cálculo Funcional a esta função em Haskell que permite redimensonar uma *quadtree*, isto é, multiplica o seu tamanho pelo valor recebido.

$$\begin{cases} scaleQTree\ s\ (Cell\ a\ b\ c) = (Cell\ a\ (s * b)\ (s * c)) \\ scaleQTree\ s\ (Block\ a\ b\ c\ d) = Block\ (scaleQtree\ a)\ (scaleQTree\ b)\ (scaleQTree\ c)\ (scaleQTree\ d) \end{cases}$$

$\equiv$ { Def-comp, Igualdade Extencional, Def-x }

$$\begin{cases} scaleQTree\ s \cdot Cell = Cell \cdot (id \times (*s) \times (*s)) \\ scaleQTree\ s \cdot Block = Block.scaleQTree \uparrow 4 \end{cases}$$

$\equiv$ { Eq-+, Fusão-+, Definição inQTree }

$$scaleQTree\ s \cdot inQTree = [Cell \cdot (id \times (*s) \times (*s)), Block.scaleQTree \uparrow 4]$$

$\equiv$ { Definição inQTree, Absorção-+ }

$$scaleQTree\ s \cdot inQTree = inQTree \cdot ((id \times (*s) \times (*s)) + scaleQTree \uparrow 4)$$

$\equiv$ { Definição $F\ scaleQTree = id \times id + scaleQTree \uparrow 4$, Functor-+, Functor-x }

$$scaleQTree\ s \cdot inQTree = inQTree \cdot F\ scaleQTree \cdot (id \times (*s) \times (*s) + id)$$

$\equiv$ { Isomorfismo outQTree = inQTree }

$$outQTree \cdot scaleQTree\ s = F\ scaleQTree \cdot (id \times (*s) \times (*s) + id) \cdot outQTree$$

$\equiv$ { Universal-ana }

$$scaleQTree\ s = anaQTree\ ((id \times (*s) \times (*s) + id) \cdot outQTree)$$

Deste modo obtemos o anamorfismo:

$$scaleQTree\ s = anaQTree\ (((id \times ((*s) \times (*s))) + id) \cdot outQTree)$$

Para obtermos o catamorfismo da invertQTree, que inverte as cores de uma quadtree, aplicamos a função *invertP* que inverte cada *pixel*.

$$invertP\ (PixelRGBA8\ a\ b\ c\ d) = PixelRGBA8\ (255 - a)\ (255 - b)\ (255 - c)\ d$$
$$invertQTree = \mathsf{fmap}\ invertP$$

A função *compressQTree* pretende comprimir a quadtree cortando folhas da árvore para reduzir a sua profundidade num dado número de níveis. Para tal, começamos por calcular o número de níveis que a quadtree deverá ter de modo a no último nível aplicarmos o método de substituição elaborado.

$$
\begin{array}{l}
compressQTree\ n\ tree = compressAux\ ((depthQTree\ tree) - n)\ tree \\
compressAux\ n\ (Cell\ a\ b\ c) = Cell\ a\ b\ c \\
compressAux\ n\ (Block\ a\ b\ c\ d) = \\
\quad \mathbf{if}\ (n > 1)\ \mathbf{then}\ Block\ (r\ (n-1)\ a)\ (r\ (n-1)\ b)\ (r\ (n-1)\ c)\ (r\ (n-1)\ d) \\
\quad \mathbf{else}\ Cell\ (getF\ a)\ (sumX\ (Block\ a\ b\ c\ d))\ (sumY\ (Block\ a\ b\ c\ d)) \\
\quad\quad \mathbf{where}\ r = compressAux \\
getF\ (Cell\ a\ b\ c) = a \\
getF\ (Block\ a\ b\ c\ d) = getF\ a \\
sumX\ (Cell\ a\ b\ c) = b \\
sumX\ (Block\ a\ b\ c\ d) = sumX\ a + sumX\ b \\
sumY\ (Cell\ a\ b\ c) = c \\
sumY\ (Block\ a\ b\ c\ d) = sumY\ a + sumY\ c
\end{array}
$$

Função, auxiliar da *outlineQTree*, que verifica e altera as células da quadtree tendo em conta se estas são de fundo.

$$
\begin{array}{ll}
 & \left\{ \begin{array}{l} outAux\ p\ Cell\ (a,(b,c)) = outLC\ p\ (a,(b,c)) \\ outAux\ p\ Block\ (a\ b\ c\ d) = Block\ (outAux\ p\ a)\ (outAux\ p\ b)\ (outAux\ p\ c)\ (outAux\ p\ d) \end{array} \right. \\
\equiv & \quad \{ \text{Def-comp, Def-x, Igualdade extencional} \} \\
 & \left\{ \begin{array}{l} outAux\ p \cdot Cell = outLC\ p \\ outAux\ p \cdot Block = Block.outAux \uparrow 4 \end{array} \right. \\
\equiv & \quad \{ \text{Eq-+, Fusão-+, Functor-id-x, Natural-id} \} \\
 & outAux\ p \cdot [Cell, Block] = [outLC\ p \cdot (id \times id), Block.outAux \uparrow 4] \\
\equiv & \quad \{ \text{Definição inQTree, Absorção-+, Cancelamento-+} \} \\
 & outAux\ p \cdot inQTree = [outLC\ p, inQTree \cdot i_2] \cdot (id \times id + outAux \uparrow 4) \\
\equiv & \quad \{ \text{Universal-cata} \} \\
 & outAux\ p = cataQTree\ ([outLC\ p, inQTree \cdot i_2])
\end{array}
$$

Aplicando as Leis do Cálculo funcional chegamos ao catamorfismo:

$$outAux\ p = cataQTree\ [outLC\ p, inQTree \cdot i_2]$$

A função *outLC* verifica se um pixel é de fundo, em caso afirmativo constrói um bloco em que as células do interior são Falsas e do exterior Verdadeiras. Caso contrário, fica tudo a Falso.

$$
\begin{array}{ll}
 & outLC\ p\ (a,(x,y)) = \mathbf{if}\ p\ a\ \mathbf{then}\ outLB\ x\ y\ \mathbf{else}\ Cell\ False\ x\ y \\
\equiv & \quad \{ \text{Def-const, Def-cond} \} \\
 & outLC\ p\ (a,(x,y)) = p \rightarrow \underline{(outLB\ x\ y)}\ \underline{(Cell\ False\ x\ y)}\ a
\end{array}
$$

$$outLC\ p\ (a,(x,y)) = cond\ p\ \underline{(outLB\ x\ y)}\ \underline{(Cell\ False\ x\ y)}\ a$$

Função responsável por construir um bloco com as células do interior a Falso e as exteriores a Verdadeiro.

$$
\begin{array}{l}
outLB\ x\ y = Block \\
\quad (Block\ (Cell\ True\ 1\ 1) \\
\quad\quad (Cell\ True\ (x-2)\ 1) \\
\quad\quad (Cell\ True\ 1\ (y-2)) \\
\quad\quad (Cell\ False\ (x-2)\ (y-2))) \\
\quad (Cell\ True\ 1\ (y-1)) \\
\quad (Cell\ True\ (x-1)\ 1) \\
\quad (Cell\ True\ 1\ 1)
\end{array}
$$

Nesta função pretende-se converter uma quadtree numa matriz monocromática, para tal convertemos a quadtree proveniente da *outAux* na matriz pretendida.

$$outlineQTree\ p = qt2bm \cdot (outAux\ p)$$

Problema 3

Começamos por escrever f k como um catamorfismo:

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

$\equiv$ { Igualdade extensional, Def-comp, Definição succ e mul, Def-split, Def-const }

$$\left\{\begin{array}{l} f\ k \cdot \underline{0} = \underline{1} \\ f\ k \cdot succ = mul \cdot \langle fk, lk \rangle \end{array}\right.$$

$\equiv$ { Definição inNat, Eq-+, Fusão-+ }

$$f\ k \cdot \mathsf{in} = [\underline{1}, mul \cdot \langle f\ k, l\ k \rangle]$$

$\equiv$ { Absorção-+, Natural-id }

$$f\ k \cdot \mathsf{in} = [\underline{1}, mul] \cdot (id \times \langle f\ k, l\ k \rangle)$$

E fazemos o mesmo para l k:

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

$\equiv$ { Igualdade extensional, Def-comp, Definição succ, Def-const }

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

$\equiv$ { Cancelamento-x }

$$\left\{\begin{array}{l} l\ k \cdot \underline{0} = \underline{(k+1)} \\ l\ k \cdot succ = succ \cdot \pi_2 \cdot \langle fk, lk \rangle \end{array}\right.$$

$\equiv$ { Definição inNat, Eq-+, Fusão-+ }

$$l\ k \cdot \mathsf{in} = [\underline{(k+1)}, succ \cdot \pi_2 \cdot \langle fk, lk \rangle]$$

$\equiv$ { Absorção-+ }

$$l\ k \cdot \mathsf{in} = [\underline{(k+1)}, succ \cdot \pi_2] \cdot (id \times \langle fk, lk \rangle)$$

$$\left\{\begin{array}{l} fk \cdot \mathbf{in} = [\underline{1}, mul] \cdot F\ \langle g, s \rangle \\ lk \cdot \mathbf{in} = [\underline{(k+1)}, succ \cdot \pi_2] \cdot F\ \langle g, s \rangle \end{array}\right.$$

$\equiv$ { Fokkinga }

$$\langle fk, lk \rangle = (\!|\langle [\underline{1}, mul], [\underline{(k+1)}, succ \cdot \pi_2] \rangle|\!)$$

Aplicando a lei da recursividade múltipla verificamos que:

$$\langle fk, lk \rangle = (\!|\langle [\underline{1}, mul], [\underline{(k+1)}, succ \cdot \pi_2] \rangle|\!)$$

Começamos por escrever g como um catamorfismo:

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

$\equiv$ { Igualdade extensional, Def-comp, definição mul, def-split, Def-const }

$$\left\{\begin{array}{l} g \cdot \underline{0} = \underline{1} \\ g \cdot succ = mul \cdot \langle g, s \rangle \end{array}\right.$$

$\equiv$ { Definição inNat, Eq-+, Fusão-+ }

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

$$\equiv \quad \{ \text{ Absorção-+ } \}$$

$$g \cdot \mathsf{in} = [\underline{1}, mul] \cdot (id \times \langle g, s \rangle)$$

E aplicamos o mesmo método a s:

$$\begin{cases} s\ 0 = 1 \\ s\ (d+1) = s\ d + 1 \end{cases}$$

$$\equiv \quad \{ \text{ Igualdade extensional, Def-comp, definição succ, Def-const } \}$$

$$\begin{cases} s \cdot \underline{0} = \underline{1} \\ s \cdot succ = succ \cdot s \end{cases}$$

$$\equiv \quad \{ \text{ Cancelamento-x } \}$$

$$\begin{cases} s \cdot \underline{0} = \underline{1} \\ s \cdot succ = succ \cdot \pi_2 \cdot \langle g, s \rangle \end{cases}$$

$$\equiv \quad \{ \text{ Definição inNat, Eq-+ e Fusão-+ } \}$$

$$s \cdot \mathsf{in} = [\underline{1}, succ \cdot \pi_2 \cdot \langle g, s \rangle]$$

$$\equiv \quad \{ \text{ Absorção-+ } \}$$

$$s \cdot \mathsf{in} = [\underline{1}, succ \cdot \pi_2] \cdot (id \times \langle g, s \rangle)$$

$$\begin{cases} g \cdot \mathbf{in} = [\underline{1}, mul] \cdot F\ \langle g, s \rangle \\ s \cdot \mathbf{in} = [\underline{1}, succ \cdot \pi_2] \cdot F\ \langle g, s \rangle \end{cases}$$

$$\equiv \quad \{ \text{ Fokkinga } \}$$

$$\langle g, s \rangle = (\!|\langle [\underline{1}, mul], [\underline{1}, succ \cdot \pi_2] \rangle|\!)$$

Do mesmo modo temos que:

$$\langle g, s \rangle = (\!|\langle [\underline{1}, mul], [\underline{1}, succ \cdot \pi_2] \rangle|\!)$$

Combinando os resultados para aplicar a lei de banana-split temos:

$$\langle (\!|\langle [\underline{1}, mul], [\underline{(k+1)}, succ \cdot \pi_2] \rangle|\!), (\!|\langle [\underline{1}, mul], [\underline{1}, succ \cdot \pi_2] \rangle|\!) \rangle$$

$$\equiv \quad \{ \text{ Lei da troca } \}$$

$$\langle (\!|[\langle \underline{1}, \underline{(k+1)} \rangle, \langle mul, succ \cdot \pi_2 \rangle]|\!), (\!|[\langle \underline{1}, \underline{1} \rangle, \langle \underline{1}, succ \cdot \pi_2 \rangle]|\!) \rangle$$

$$\equiv \quad \{ \text{ Banana-split } \}$$

$$(\!|[\langle \underline{1}, \underline{(k+1)} \rangle, \langle mul, succ \cdot \pi_2 \rangle] \times [\langle \underline{1}, \underline{1} \rangle, \langle \underline{1}, succ \cdot \pi_2 \rangle]|\!) \cdot \langle F\ \pi_1, F\ \pi_2 \rangle$$

$$\equiv \quad \{ \text{ F f = id + f } \}$$

$$(\!|[\langle \underline{1}, \underline{(k+1)} \rangle, \langle mul, succ \cdot \pi_2 \rangle] \times [\langle \underline{1}, \underline{1} \rangle, \langle \underline{1}, succ \cdot \pi_2 \rangle]|\!) \cdot \langle id + \pi_1, id + \pi_2 \rangle$$

$$\equiv \quad \{ \text{ Absorção-x } \}$$

$$(\!|\langle [\langle \underline{1}, \underline{(k+1)} \rangle, \langle mul, succ \cdot \pi_2 \rangle] \cdot (id + \pi_1), [\langle \underline{1}, \underline{1} \rangle, \langle mul, succ \cdot \pi_2 \rangle] \cdot (id + \pi_2) \rangle|\!)$$

$$\equiv \quad \{ \text{ Absorção-+, Natural-id } \}$$

$$(\!|\langle [\langle \underline{1}, \underline{(k+1)} \rangle, \langle mul, succ \cdot \pi_2 \rangle \cdot \pi_1], [\langle \underline{1}, \underline{1} \rangle, \langle mul, succ \cdot \pi_2 \rangle \cdot \pi_2] \rangle|\!)$$

$$\equiv \quad \{ \text{ Fusão-x } \}$$

$$(\!|\langle [\langle \underline{1}, \underline{(k+1)} \rangle, \langle mul \cdot \pi_1, succ \cdot \pi_2 \cdot \pi_1 \rangle], [\langle \underline{1}, \underline{1} \rangle, \langle mul \cdot \pi_2, succ \cdot \pi_2 \cdot \pi_2 \rangle] \rangle|\!)$$

$$\equiv \quad \{ \text{ Lei da troca } \}$$

$$(\![[\langle\langle \underline{1}, \underline{(k+1)}\rangle, \langle \underline{1}, \underline{1}\rangle\rangle, \langle\langle mul \cdot \pi_1, succ \cdot \pi_2\rangle, \langle mul \cdot \pi_2, suc \cdot \pi_2 \cdot \pi_2\rangle\rangle]]\!)$$

$$\equiv \quad \{ \ \langle \underline{a}, \underline{s}\rangle = \text{(a,b)} \ \}$$

$$(\![[((1, k+1), (1,1)), \langle\langle mul \cdot \pi_1, succ \cdot \pi_2\rangle, \langle mul \cdot \pi_2, suc \cdot \pi_2 \cdot \pi_2\rangle\rangle]]\!)$$

Sabendo que for $b\ i = (\![[\underline{i}, b]]\!)$ temos que:

$base\ k = toPair\ ((1, k+1), (1,1))$
$loop = toPair \cdot \langle\langle mul \cdot \pi_1, succ \cdot \pi_2 \cdot \pi_1\rangle, \langle mul \cdot \pi_2, succ \cdot \pi_2 \cdot \pi_2\rangle\rangle \cdot desPair$

$toPair :: ((Integer, Integer), (Integer, Integer)) \to (Integer, Integer, Integer, Integer)$
$toPair\ ((a,b),(c,d)) = (a,b,c,d)$
$desPair :: (Integer, Integer, Integer, Integer) \to ((Integer, Integer), (Integer, Integer))$
$desPair\ (a,b,c,d) = ((a,b),(c,d))$

Problema 4

$uncurryFTree\ f = \lambda(x,(y,z)) \to f\ x\ y\ z$
$inFTree = [Unit, uncurryFTree\ Comp]$
$outFTree\ (Unit\ b) = i_1\ b$
$outFTree\ (Comp\ a\ b\ c) = i_2\ (a,(b,c))$
$baseFTree\ g\ f\ h = f + (g \times (h \times h))$
$recFTree\ f = baseFTree\ id\ id\ f$
$cataFTree\ g = g \cdot (recFTree\ (cataFTree\ g)) \cdot outFTree$
$anaFTree\ f = inFTree \cdot (recFTree\ (anaFTree\ f)) \cdot f$
$hyloFTree\ f\ g = cataFTree\ f \cdot anaFTree\ g$
instance $Bifunctor\ FTree$ **where**
$\quad bimap\ f\ g = cataFTree\ (inFTree \cdot baseFTree\ f\ g\ id)$

A *generatePTree* pretende gerar uma árvore de pitágoras para uma dada ordem. De modo a solucionar o problema optamos por iniciar a construção da árvore de níveis superiores para os inferiores. Os quadrados constituintes são redimensionados pela escala de $(sqrt\ 2)\ /\ 2$ de níveis inferiores para os superiores, o que corresponderá a uma aumento no tamanho de $sqrt\ 2$ na nossa resolução. Desta forma, elaboramos a seguinte função em Haskell:

$$\left\{ \begin{array}{l} generatePTree\ 0 = Unit\ 1 \\ generatePTree\ (n+1) = Comp\ (sqrt\ (2) \uparrow (n+1))\ (generatePTree\ n)\ (generatePTree\ n) \end{array} \right.$$

$$\equiv \quad \{ \text{ Def-comp, Def-const, Def-split, Igualdade Extencional } \}$$

$$\left\{ \begin{array}{l} generatePTree \cdot \underline{0} = Unit \cdot \underline{1} \\ generatePTree \cdot succ = Comp \cdot \langle (sqrt\ (2) \uparrow) \cdot succ, \langle generatePTree, generatePTree\rangle\rangle \end{array} \right.$$

$$\equiv \quad \{ \text{ Absorção-x } \}$$

$$\left\{ \begin{array}{l} generatePTree \cdot \underline{0} = Unit \cdot \underline{1} \\ generatePTree \cdot succ = Comp \cdot \langle (sqrt\ (2) \uparrow) \cdot succ, (generatePTree \uparrow 2) \cdot \langle id, id\rangle\rangle \end{array} \right.$$

$$\equiv \quad \{ \text{ Absorção-x } \}$$

$$\left\{ \begin{array}{l} generatePTree \cdot \underline{0} = Unit \cdot \underline{1} \\ generatePTree \cdot succ = Comp \cdot ((id \times (generatePTree \uparrow 2)) \cdot \langle (sqrt\ (2) \uparrow) \cdot succ, \langle id, id\rangle\rangle) \end{array} \right.$$

$$\equiv \quad \{ \text{ Fusão-+, Eq-+ } \}$$

$$generatePTree \cdot [\underline{0}, succ] = [Unit \cdot \underline{1}, Comp \cdot ((id \times (generatePTree \uparrow 2) \cdot \langle (sqrt\ (2) \uparrow) \cdot succ, \langle id, id\rangle\rangle))]$$

$$\equiv \quad \{ \text{ Definição inNat, Absorção-+ } \}$$

$$generatePTree \cdot \mathsf{in} = [Unit, Comp] \cdot (\underline{1} + ((id \times (generatePTree \uparrow 2) \cdot \langle (sqrt\ (2) \uparrow) \cdot succ, \langle id, id\rangle\rangle))$$

$$
\begin{array}{ll}
\equiv & \quad \{ \text{ Definição inFTree, Functor-+, Natural-id } \} \\
& generatePTree \cdot \mathsf{in} = inFTree \cdot (id + id \times generatePTree \uparrow 2) \cdot (\underline{1} + \langle (sqrt\ (2)\uparrow) \cdot succ, \langle id, id \rangle \rangle) \\
\equiv & \quad \{ \text{ Definição } F\ f = id + id \times f \text{, isomorfismo inFTree = outFTree e inNat = outNat } \} \\
& outFTree \cdot generatePTree = (F\ generatePTree) \cdot (\underline{1} + \langle (sqrt\ (2)\uparrow) \cdot succ, \langle id, id \rangle \rangle) \cdot outNat \\
\equiv & \quad \{ \text{ Universal-ana } \} \\
& generatePTree = anaFTree\ ((\underline{1} + \langle (sqrt\ (2)\uparrow) \cdot succ, \langle id, id \rangle \rangle) \cdot outNat)
\end{array}
$$

Desta forma, a *generatePTree* como anamorfismo é:

$$generatePTree = anaFTree\ ((\underline{1} + \langle (sqrt\ (2)\uparrow) \cdot succ, \langle id, id \rangle \rangle) \cdot outNat)$$

A função auxiliar *ptreeGenerate* gera a árvore de pitágoras de ordem n com o tamanho desejado. A *ptreeDraw* cria a lista de imagens de todos os quadrados necessários para criar a *PTree*. Enquanto que a *transformaL* transforma a lista de imagens que representa os quadrados necessários à construção da *PTree* numa lista de imagens correspondentes a cada frame a apresentar na animação. Estas três funções auxiliam a *drawPTree* que é responsável por gerar a lista de imagens a representar aquando da construção da árvore de Pitágoras.

$$
\begin{array}{l}
ptreeGenerate :: Int \to PTree \\
ptreeGenerate\ n = aux\ n\ 100\ \mathbf{where} \\
\quad aux :: Int \to Float \to PTree \\
\quad aux\ 0\ x = Unit\ x \\
\quad aux\ n\ x = Comp\ x\ (aux\ (n-1)\ (x * (sqrt\ (2)\ /\ 2)))\ (aux\ (n-1)\ (x * (sqrt\ (2)\ /\ 2))) \\
ptreeDraw :: PTree \to Float \to (Float, Float) \to [Picture] \\
ptreeDraw\ (Unit\ a)\ ang\ (x, y) = [Translate\ x\ y\ (Rotate\ ang\ (square\ a))] \\
ptreeDraw\ (Comp\ a\ e\ d)\ ang\ (x, y) = [Translate\ x\ y\ (Rotate\ ang\ (square\ a))] +\!\!+ \\
\quad (ptreeDraw\ e\ (ang - 45)\ (x + somaXLeft, y + somaYLeft)) +\!\!+ \\
\quad (ptreeDraw\ d\ (ang + 45)\ (x + somaXRight, y + somaYRight)) \\
\quad \mathbf{where} \\
\qquad somaX = a\ /\ 2 \\
\qquad angRads = ang * pi\ /\ 180 \\
\qquad branchToGlobal\ angle\ (dx, dy) = (dx * cos\ angle + dy * sin\ angle, dy * cos\ angle - dx * sin\ angle) \\
\qquad (somaXLeft, somaYLeft) = branchToGlobal\ angRads\ (-somaX, a) \\
\qquad (somaXRight, somaYRight) = branchToGlobal\ angRads\ (somaX, a) \\
transformaL :: Int \to [Picture] \\
transformaL\ 0 = [pictures\ (ptreeDraw\ (ptreeGenerate\ 1)\ 0\ (0, 0))] \\
transformaL\ n = (pictures\ (ptreeDraw\ (ptreeGenerate\ (n+1))\ 0\ (0, 0))) : transformaL\ (n-1) \\
drawPTree\ fig = reverse\ (transformaL\ (depthFTree\ fig))
\end{array}
$$

Problema 5

Primeiro é necessário construir um par cujo primeiro elemento é a cor do Marble e o segundo um inteiro. Para tal, recorrendo ao *split* criamos o par desejado. De seguida é criada uma lista com o anterior através do *singl*. Por último, utilizando o construtor B elaboramos a Bag.

$$
\begin{tikzcd}
a \arrow[r, "\langle id, \underline{1} \rangle"'] \arrow[d, "singletonBag"'] & (a, Int) \arrow[d, "singl"] \\
Bag\ a & {[(a, Int)]} \arrow[l, "B"]
\end{tikzcd}
$$

$$singletonbag = B \cdot singl \cdot \langle id, \underline{1} \rangle$$

Dado que recebemos um saco com vários sacos, recorremos à *unB* para abrir os sacos. Após estarem abertos, verificamos os sacos que são iguais e procedemos à multiplicação do número de Marbles com uma dada cor pelo número de sacos iguais, sendo esta informação armazenada numa lista de túplos. Por fim, utilizando o construtor B, ficamos com uma única Bag com toda a informação.

$$\begin{tikzcd}
\mathit{Bag}\ (\mathit{Bag}\ a) \arrow[r, "\mathsf{fmap}\ \mathit{unB}"'] \arrow[d, "\mu"'] & \mathit{Bag}\ [(a, \mathit{Int})] \arrow[d, "\mathit{retiraBags}"] \\
\mathit{Bag}\ a & {[(a, \mathit{Int})]} \arrow[l, "B"]
\end{tikzcd}$$

$$\begin{array}{l}
\mathit{retiraBags} :: \mathit{Bag}\ [(a, \mathit{Int})] \rightarrow [(a, \mathit{Int})] \\
\mathit{retiraBags} = \mathit{concat} \cdot (\mathsf{map}\ \ \mathit{multiplica}) \cdot \mathit{unB} \\
\mathit{multiplica} :: ([(a, \mathit{Int})], \mathit{Int}) \rightarrow [(a, \mathit{Int})] \\
\mathit{multiplica}\ ([\,], _) = [\,] \\
\mathit{multiplica}\ (((a, b) : t), c) = [(a, b * c)] + \!\! + \ \mathit{multiplica}\ (t, c)
\end{array}$$

$$\mu = B \cdot \mathit{retiraBags} \cdot (\mathsf{fmap}\ \mathit{unB})$$

Para elaborarmos a função *dist* recorremos a três funções auxiliares, a *numM* que devolve o número total de Marbles, a *probM* que constrói uma lista de túplos em que o segundo elemento representa a probabilidade de cada cor e a *sProb* que devolve o pretendido. Note-se que a *Bag* utilizada para calcular a probabilidade não contém elementos repetidos. Assim, a função *dist* retorna a probabilidade de cada cor dos berlindes contidos no Bag recebido.

$$\begin{array}{l}
\mathit{sProb}\ (B\ \ a)\ \mathit{num} = D\ (\mathit{probM}\ \ a\ \ \mathit{num}) \\
\mathit{probM} :: [(a, \mathit{Int})] \rightarrow \mathit{Int} \rightarrow [(a, \mathit{ProbRep})] \\
\mathit{probM}\ [\,]\ _ = [\,] \\
\mathit{probM}\ ((t, n) : h)\ \mathit{num} = [(t, (((\mathit{fromIntegral}\ \ n) \mathbin{/} (\mathit{fromIntegral}\ \ \mathit{num}))))] + \!\! + \ \mathit{probM}\ \ h\ \ \mathit{num} \\
\mathit{numM} :: \mathit{Bag}\ \ a \rightarrow \mathit{Int} \\
\mathit{numM}\ (B\ [\,]) = 0 \\
\mathit{numM}\ (B\ ((t, n) : h)) = n + \mathit{numM}\ \ (B\ \ h)
\end{array}$$

$$\mathit{dist}\ \ a = \mathit{sProb}\ (\mathit{consolidate}\ \ a)\ (\mathit{numM}\ \ a)$$

D Como exprimir cálculos e diagramas em LaTeX/lhs2tex

Estudar o texto fonte deste trabalho para obter o efeito:[7]

$$\begin{array}{ll}
& \mathit{id} = \langle f, g \rangle \\
\equiv & \quad \{\ \text{universal property}\ \} \\
& \left\{ \begin{array}{l} \pi_1 \cdot \mathit{id} = f \\ \pi_2 \cdot \mathit{id} = g \end{array} \right. \\
\equiv & \quad \{\ \text{identity}\ \} \\
& \left\{ \begin{array}{l} \pi_1 = f \\ \pi_2 = g \end{array} \right. \\
\Box &
\end{array}$$

Os diagramas podem ser produzidos recorrendo à *package* LaTeX xymatrix, por exemplo:

$$\begin{tikzcd}
\mathbb{N}_0 \arrow[d, "(\!|g|\!)"'] & 1 + \mathbb{N}_0 \arrow[l, "\mathsf{in}"'] \arrow[d, "id + (\!|g|\!)"] \\
B & 1 + B \arrow[l, "g"]
\end{tikzcd}$$

[7]Exemplos tirados de [**?**].