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

Departamento de Informática
Universidade do Minho

Julho de 2018

Grupo nr.	23
a81946	Carlos Castro
a81302	Daniel Costa
a80494	Luís Macedo

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 **??** 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 )      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 **??**.

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

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

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

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 **??**.

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:

$$
\begin{array}{l}
whitePx = PixelRGBA8\ 255\ 255\ 255\ 255 \\
blackPx = PixelRGBA8\ 0\ 0\ 0\ 255 \\
redPx \quad = PixelRGBA8\ 255\ 0\ 0\ 255
\end{array}
$$

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 **??**:

```
> 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 **??**, **??** e **??**:

```
> 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.

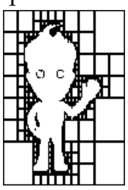
(d) Inversão de cores.

(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 **??**, **??**, **??** e **??**:

```
> 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 **??** e **??**:

```
> 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

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

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

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

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 (**??**):*

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

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 **??**).

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:

$$marbleWeight :: Marble \to Int$$
$$marbleWeight\ Red\ = 3$$
$$marbleWeight\ Pink = 2$$
$$marbleWeight\ Green = 3$$
$$marbleWeight\ Blue\ = 6$$
$$marbleWeight\ White = 2$$

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.

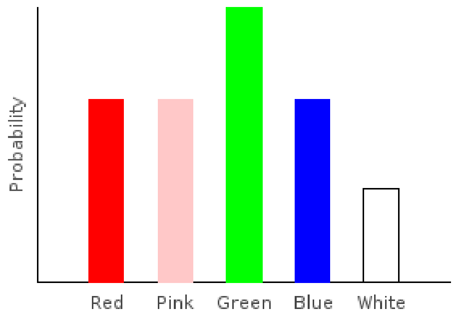


Figura 4: Distribuição de berlindes num saco.

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 **??**.

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

$$\begin{array}{l}\mathbf{instance}\ Functor\ Bag\ \mathbf{where}\\ \quad \mathsf{fmap}\ f = B \cdot \mathsf{map}\ (f \times id) \cdot unB\\ \mathbf{instance}\ Monad\ Bag\ \mathbf{where}\\ \quad x \gg\!= f = (\mu \cdot \mathsf{fmap}\ f)\ x\ \mathbf{where}\\ \quad return = singletonbag\end{array}$$

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

<u>Teste unitário</u> 2 *Lei* $\mu \cdot return = id$*:*

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

<u>Teste unitário</u> 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

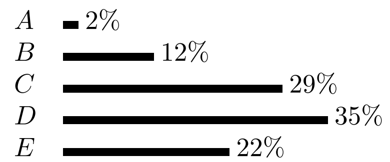
A Mónade para probabilidades e estatística

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

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

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*:

```
instance (Show a, Ord a, Eq a) => Show (Bag a) where
  show = showbag . consol . unB where
    showbag = concat .
      (++[" }"]) . ("{ ":) .
      (intersperse " , ") .
      sort .
      (map f) where f (a, b) = (show a) ++ " |-> " ++ (show b)
unB (B x) = x
```

Igualdade de *bags*:

```
instance (Eq a) => Eq (Bag a) where
  b ≡ b' = (unB b) `lequal` (unB b')
    where lequal a b = isempty (a ⊖ b)
      ominus a b = a ++ neg b
      neg x = [(k, -i) | (k, i) <- x]
```

Ainda sobre o mónade *Bag*:

$$\textbf{instance}\ \mathit{Applicative}\ \mathit{Bag}\ \textbf{where}$$
$$\mathit{pure} = \mathit{return}$$
$$(<*>) = \mathit{aap}$$

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):

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

Outras funções auxiliares:

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

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

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

Na função *allTransactions*, tal como é pedido no enunciado, usou-se um catamorfismo de forma a ter recursividade. A função que o catamorfismo recebe terá que ser um *Either*, onde o primeiro argumento será para o *Bc* e o segundo argumento será para o *Bcs*. O primeiro argumento do *Either* serve para receber o elemento *Transactions* do *Bc*, enquanto que o segundo argumento concatena um par onde o primeiro elemento é do tipo *Transactions* do *Block*, e o segundo é id para o *Blockchain* em *Bcs*.

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

A função *ledger* começa por criar um lista de *(Entity, (Value, Entity))*, separa esses pares na forma de *(Entity, Value)* e dá sort. Apos estes passos é criada uma lista de listas, onde as sublista contêm os pares com a mesma *Entity*. Por último, as sublistas são substituidas por um par do tipo *(Entity, Value)*, onde o *Value* é a soma de todos os segundos elementos dos pares da sublista.

$$\mathit{ledger} = \mathit{groupTransactions} \cdot \mathit{groupBy}\ (\lambda(e_1, \_)\ (e_2, \_) \rightarrow e_1 \equiv e_2) \cdot \mathit{sort} \cdot \mathit{splitTransactions} \cdot \mathit{allTransactions}$$
$$\textbf{where}$$

$$
\begin{array}{l}
splitTransactions = cataList \ \$ \ [nil, cons \cdot \langle\langle e_1, v\rangle, cons \cdot \langle\langle e_2, negate \cdot v\rangle, \pi_2\rangle\rangle] \\
\quad \textbf{where} \\
\qquad e_1 = \pi_2 \cdot \pi_2 \cdot \pi_1 \\
\qquad e_2 = \pi_1 \cdot \pi_1 \\
\qquad v = \pi_1 \cdot \pi_2 \cdot \pi_1 \\
groupTransactions = \mathsf{map} \ \langle \pi_1 \cdot head, sum \cdot \mathsf{map} \ \pi_2 \rangle
\end{array}
$$

A função *isValidMagicNr* usa duas funções auxiliares, a *getMagicNrs* que recebe um *Blockchain* e devolve uma lista de *MagicNo*, e a *isSingle* que recebe uma lista de listas de *MagicNo* e devolve *True* se o tamanho de de todas as sublistas e *False* se existir pelo menos uma das sublistas que o seu tamanho seja diferente de 1. A função *isValidMagicNr* recebe um *Blockchain*, cria uma lista com todos os *MagicNo* do *Blockchain*, ordena-a, e transforma-a numa lista com sublistas, onde essas sublistas contêm *MagicNo* iguais e por fim verifica se cada uma das sublistas têm tamanho 1.

$$
\begin{array}{l}
isValidMagicNr = isSingle \cdot group \cdot sort \cdot getMagicNrs \\
\quad \textbf{where} \\
\qquad getMagicNrs = cataBlockchain \ \$ \ [singl \cdot \pi_1, cons \cdot (\pi_1 \times id)] \\
\qquad isSingle = all \ ((1 \equiv) \cdot \mathsf{length} \ )
\end{array}
$$

Problema 2

$$
\begin{array}{l}
inQTree = [uncurry2 \ Cell, uncurry3 \ Block] \\
\quad \textbf{where} \ uncurry3 \ f = \lambda(a, (b, (c, d))) \rightarrow f \ a \ b \ c \ d \\
\qquad uncurry2 \ f = \lambda(x, (y, z)) \rightarrow f \ x \ y \ z \\
outQTree \ (Cell \ a \ x \ y) = i_1 \ (a, (x, y)) \\
outQTree \ (Block \ t1 \ t2 \ t3 \ t4) = i_2 \ (t1, (t2, (t3, t4))) \\
baseQTree \ f \ g = (f \times id) + (g \times (g \times (g \times g))) \\
recQTree \ f = baseQTree \ id \ f \\
cataQTree \ a = a \cdot (recQTree \ (cataQTree \ a)) \cdot outQTree \\
anaQTree \ f = inQTree \cdot (recQTree \ (anaQTree \ f)) \cdot f \\
hyloQTree \ a \ c = cataQTree \ a \cdot anaQTree \ c \\
\textbf{instance} \ Functor \ QTree \ \textbf{where} \\
\quad \mathsf{fmap} \ f = cataQTree \ (inQTree \cdot baseQTree \ f \ id)
\end{array}
$$

Na função *rotateQTree* o catamorfismo recebe um *Either*, onde o primeiro argumento é aplicado quando a *QTree* argumento é uma *Cell*e o segundo argumento quando a *QTree* é um *Block*. No caso de:

- Uma *Cell* é invocada a função auxiliar *f* que transforma um par do tipo *(a,(x,y))* para *Cell a x y*;
- Um *Block* é invocada a função auxiliar *g* que transforma um par do tipo *(a,(b,(c,d)))* para *Block c a d b* de forma a fazer a rotação;

$$
\begin{array}{l}
rotateQTree = cataQTree \ \$ \ [f, g] \ \textbf{where} \\
\quad f \ (k, (x, y)) = Cell \ k \ y \ x \\
\quad g \ (a, (b, (c, d))) = Block \ c \ a \ d \ b
\end{array}
$$

A função *scaleQTree* é um catamorfismo que aplica (id × (n*) × (n*)) ao primeiro elemento e *id* ao segundo elemnto do *Either* argumento, que em seguida são transformados para o tipo *QTree*.

$$
scaleQTree \ n = cataQTree \ \$ \ inQTree \cdot ((id \times ((n*) \times (n*)) + id))
$$

A *invertQTree* é um função map que aplica a função lambda recebida como argumeto a todas as *Cells*. A função lambda faz a subtração entre o numero 255 e os diferentes valores do tipo *PixelRGBA8*, excepto o último que se refere à trasparencia.

$$
invertQTree = \mathsf{fmap} \ \$ \ \lambda(PixelRGBA8 \ r \ g \ b \ a) \rightarrow (PixelRGBA8 \ (255 - r) \ (255 - g) \ (255 - b) \ a)
$$

Na função *compressQTree*, tal como é pedido no enunciado, usou-se um catamorfismo. Inicialmente, calcula-se a nova profundidade, diferença entre a profundidade da *QTree* e o n dado. Usando um for, percorre a *QTree* até chegar à nova profundidade. Depois executa a função *compress*. A função *compress* percorre a *QTree* até ser uma *Cell*. Caso não o seja, procura no primeiro *Block* da *QTree*. Por fim, devolve a *Cell* com a soma do seu tamanho e das células à sua volta.

$$
\begin{array}{l}
compressQTree\ n = (\mathsf{for}\ \widehat{recTree\ compress}) \cdot \langle \widehat{max} \cdot \langle subtract\ n \cdot depthQTree, \underline{0} \rangle, id \rangle \\
\quad \mathbf{where} \\
\qquad recTree\ f = inQTree \cdot (recQTree\ f) \cdot outQTree \\
\qquad compress = cataQTree\ [f, g] \\
\qquad\quad \mathbf{where} \\
\qquad\qquad f = inQTree \cdot i_1 \\
\qquad\qquad g\ (Cell\ a\ xa\ ya, (Cell\ \_\ xb\ \_, (Cell\ \_\ \_\ yc, Cell\ \_\ \_\ \_))) = Cell\ a\ (xa + xb)\ (ya + yc) \\
\qquad\qquad g\ a = inQTree \cdot i_2\ \$\ a
\end{array}
$$

A função *outlineQTree* transforma uma *Cell* numa matrix de *Bool* com o mesmo tamanho, onde as bordas dependem da função argumento e no interior é tudo *False*. No caso do *Block* são aplicadas operações de matrizes de forma a criar toda a matriz que representa a imagem.

$$
\begin{array}{l}
outlineQTree\ h = cataQTree\ [f, g]\ \mathbf{where} \\
\quad f\ (k, (x, y)) = matrix\ y\ x\ (\lambda(b, a) \rightarrow (p2p\ (False, (h\ k))\ (b \equiv y \vee b \equiv 1 \vee a \equiv x \vee a \equiv 1))) \\
\quad g\ (a, (b, (c, d))) = (a \updownarrow b) \leftrightarrow (c \updownarrow d)
\end{array}
$$

Problema 3

$$
\begin{array}{ll}
 & s \cdot \mathbf{in} = [\underline{1}, \mathsf{succ}\ \cdot s] \\
\equiv & \quad \{ \text{ Identidade, Cancelamento-}\times \ \} \\
 & s \cdot \mathbf{in} = [\underline{1} \cdot id, \mathsf{succ}\ \cdot \pi_2 \cdot \langle g, s \rangle] \\
\equiv & \quad \{ \text{ Absorção-+ (22) } \} \\
 & s \cdot \mathbf{in} = [\underline{1}, \mathsf{succ}\ \cdot \pi_2] \cdot (id + \langle g, s \rangle) \\
\equiv & \quad \{ \text{ Functor } \} \\
 & s \cdot \mathbf{in} = [\underline{1}, \mathsf{succ}\ \cdot \pi_2] \cdot F\ \langle g, s \rangle
\end{array}
$$

$$
\begin{tikzcd}
\mathbb{N}_0 \arrow[d, "s"'] & 1 + \mathbb{N}_0 \arrow[l, "\mathsf{in}"'] \arrow[d, "F\ \langle g, s \rangle"] \\
\mathbb{N}_0 & 1 + \mathbb{N}_0 * \mathbb{N}_0 \arrow[l, "{[\underline{1}, \mathsf{succ}\ \cdot \pi_2]}"]
\end{tikzcd}
$$

$$
\begin{array}{ll}
 & g \cdot \mathbf{in} = [\underline{1}, mul \cdot \langle g, s \rangle] \\
\equiv & \quad \{ \text{ Identidade } \} \\
 & g \cdot \mathbf{in} = [\underline{1} \cdot id, mul \cdot \langle g, s \rangle] \\
\equiv & \quad \{ \text{ Absorção-+ (22) } \} \\
 & g \cdot \mathbf{in} = [\underline{1}, mul] \cdot (id + \langle g, s \rangle) \\
\equiv & \quad \{ \text{ Functor } \} \\
 & g \cdot \mathbf{in} = [\underline{1}, mul] \cdot F\ \langle g, s \rangle
\end{array}
$$

$$
\begin{tikzcd}
\mathbb{N}_0 \arrow[d, "g"'] & & 1 + \mathbb{N}_0 \arrow[ll, "\mathsf{in}"'] \arrow[d, "F\ \langle g, s\rangle"] \\
\mathbb{N}_0 & & 1 + \mathbb{N}_0 * \mathbb{N}_0 \arrow[ll, "{[\underline{1}, mul]}"]
\end{tikzcd}
$$

$$
\begin{aligned}
& \left\{ \begin{array}{l} g \cdot \mathbf{in} = [\underline{1}, mul] \cdot F\ \langle g, s\rangle \\ s \cdot \mathbf{in} = [\underline{1}, \mathsf{succ}\ \cdot \pi_2] \cdot F\ \langle g, s\rangle \end{array} \right. \\
\equiv \quad & \quad \{ \text{ Fokkinga (50) } \} \\
& \langle g, s\rangle = \llparenthesis \langle [\underline{1}, mul], [\underline{1}, \mathsf{succ}\ \cdot \pi_2] \rangle \rrparenthesis \\
\equiv \quad & \quad \{ \text{ Lei da Troca (28) } \} \\
& \langle g, s\rangle = \llparenthesis [\langle \underline{1}, \underline{1}\rangle, \langle mul, \mathsf{succ}\ \cdot \pi_2\rangle] \rrparenthesis
\end{aligned}
$$

$$
\begin{aligned}
& f\ k \cdot \mathbf{in} = [\underline{1}, mul \cdot \langle f\ k, l\ k\rangle] \\
\equiv \quad & \quad \{ \text{ Identidade } \} \\
& f\ k \cdot \mathbf{in} = [\underline{1} \cdot id, mul \cdot \langle f\ k, l\ k\rangle] \\
\equiv \quad & \quad \{ \text{ Absorção-+ (22) } \} \\
& f\ k \cdot \mathbf{in} = [\underline{1}, mul] \cdot (id + \langle f\ k, l\ k\rangle) \\
\equiv \quad & \quad \{ \text{ Functor } \} \\
& f\ k \cdot \mathbf{in} = [\underline{1}, mul] \cdot F\ \langle f\ k, l\ k\rangle
\end{aligned}
$$

$$
\begin{tikzcd}
\mathbb{N}_0 \arrow[d, "f\ k"'] & & 1 + \mathbb{N}_0 \arrow[ll, "\mathsf{in}"'] \arrow[d, "F\ \langle f\ k, l\ k\rangle"] \\
\mathbb{N}_0 & & 1 + \mathbb{N}_0 * \mathbb{N}_0 \arrow[ll, "{[\underline{1}, mul]}"]
\end{tikzcd}
$$

$$
\begin{aligned}
& l\ k \cdot \mathbf{in} = [\mathsf{succ}\ \cdot k, \mathsf{succ}\ \cdot l\ k] \\
\equiv \quad & \quad \{ \text{ Identidade, Cancelamento-}\times \text{ } \} \\
& l\ k \cdot \mathbf{in} = [\mathsf{succ}\ \cdot k \cdot id, \mathsf{succ}\ \cdot \pi_2 \cdot \langle f\ k, l\ k\rangle] \\
\equiv \quad & \quad \{ \text{ Absorção-+ (22) } \} \\
& l\ k \cdot \mathbf{in} = [\mathsf{succ}\ \cdot k, \mathsf{succ}\ \cdot \pi_2] \cdot (id + \langle f\ k, l\ k\rangle) \\
\equiv \quad & \quad \{ \text{ Functor } \} \\
& l\ k \cdot \mathbf{in} = [\mathsf{succ}\ \cdot k, \mathsf{succ}\ \cdot \pi_2] \cdot F\ \langle f\ k, l\ k\rangle
\end{aligned}
$$

$$
\begin{tikzcd}
\mathbb{N}_0 \arrow[d, "l\ k"'] & & 1 + \mathbb{N}_0 \arrow[ll, "\mathsf{in}"'] \arrow[d, "F\ \langle f\ k, l\ k\rangle"] \\
\mathbb{N}_0 & & 1 + \mathbb{N}_0 * \mathbb{N}_0 \arrow[ll, "{[\mathsf{succ}\ \cdot k, \mathsf{succ}\ \cdot \pi_2]}"]
\end{tikzcd}
$$

$$
\left\{\begin{array}{l}
f\ k \cdot \mathbf{in} = [\underline{1}, mul] \cdot F\ \langle f\ k, l\ k\rangle \\
l\ k \cdot \mathbf{in} = [\mathsf{succ}\ \cdot k, \mathsf{succ}\ \cdot \pi_2] \cdot F\ \langle f\ k, l\ k\rangle
\end{array}\right.
$$

$$
\begin{array}{ll}
\equiv & \quad \{\ \text{Fokkinga (50)}\ \} \\
& \langle f\ k, l\ k\rangle = \llparenthesis \langle [\underline{1}, mul], [\mathsf{succ}\ \cdot k, \mathsf{succ}\ \cdot \pi_2]\rangle \rrparenthesis \\
\equiv & \quad \{\ \text{Lei da Troca (28)}\ \} \\
& \langle f\ k, l\ k\rangle = \llparenthesis [\langle \underline{1}, \mathsf{succ}\ \cdot k\rangle, \langle mul, \mathsf{succ}\ \cdot \pi_2\rangle] \rrparenthesis
\end{array}
$$

$$
\begin{array}{ll}
& \langle \langle f\ k, l\ k\rangle, \langle g, s\rangle\rangle \\
\equiv & \quad \{\ \text{Banana-split (51)}\ \} \\
& \llparenthesis ([\langle \underline{1}, \mathsf{succ}\ \cdot k\rangle, \langle mul, \mathsf{succ}\ \cdot \pi_2\rangle] \times [\langle \underline{1}, \underline{1}\rangle, \langle mul, \mathsf{succ}\ \cdot \pi_2\rangle]) \cdot \langle F\ \pi_1, F\ \pi_2\rangle \rrparenthesis \\
\equiv & \quad \{\ \text{Absorção-}\times\text{ (11)}\ \} \\
& \llparenthesis \langle [\langle \underline{1}, \mathsf{succ}\ \cdot k\rangle, \langle mul, \mathsf{succ}\ \cdot \pi_2\rangle] \cdot F\ \pi_1, [\langle \underline{1}, \underline{1}\rangle, \langle mul, \mathsf{succ}\ \cdot \pi_2\rangle] \cdot F\ \pi_2\rangle \rrparenthesis \\
\equiv & \quad \{\ \text{Functor}\ \} \\
& \llparenthesis \langle [\langle \underline{1}, \mathsf{succ}\ \cdot k\rangle, \langle mul, \mathsf{succ}\ \cdot \pi_2\rangle] \cdot (id + \pi_1), [\langle \underline{1}, \underline{1}\rangle, \langle mul, \mathsf{succ}\ \cdot \pi_2\rangle] \cdot (id + \pi_2)\rangle \rrparenthesis \\
\equiv & \quad \{\ \text{Absorção-+ (22)}\ \} \\
& \llparenthesis \langle [\langle \underline{1}, \mathsf{succ}\ \cdot k\rangle, \langle mul, \mathsf{succ}\ \cdot \pi_2\rangle \cdot \pi_1], [\langle \underline{1}, \underline{1}\rangle, \langle mul, \mathsf{succ}\ \cdot \pi_2\rangle \cdot \pi_2]\rangle \rrparenthesis \\
\equiv & \quad \{\ \text{Fusão-}\times\text{ (9)}\ \} \\
& \llparenthesis \langle [\langle \underline{1}, \mathsf{succ}\ \cdot k\rangle, \langle mul \cdot \pi_1, \mathsf{succ}\ \cdot \pi_2 \cdot \pi_1\rangle], [\langle \underline{1}, \underline{1}\rangle, \langle mul \cdot \pi_2, \mathsf{succ}\ \cdot \pi_2 \cdot \pi_2\rangle]\rangle \rrparenthesis \\
\equiv & \quad \{\ \text{Lei da troca (28)}\ \} \\
& \llparenthesis [\langle \langle \underline{1}, \mathsf{succ}\ \cdot k\rangle, \langle \underline{1}, \underline{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] \rrparenthesis
\end{array}
$$

$$
\begin{array}{ll}
& \mathsf{for}\ g\ i = \llparenthesis [\underline{i}, g] \rrparenthesis \\
\equiv & \quad \{\ \} \\
& (a, \_, b, \_) = \llparenthesis [\underline{(base\ k)}, loop] \rrparenthesis \\
\equiv & \quad \{\ \} \\
& \left\{\begin{array}{l}
base\ k = \langle \langle \underline{1}, \mathsf{succ}\ \cdot k\rangle, \langle \underline{1}, \underline{1}\rangle\rangle \\
loop = \langle \langle mul \cdot \pi_1, \mathsf{succ}\ \cdot \pi_2 \cdot \pi_1\rangle, \langle mul \cdot \pi_2, \mathsf{succ}\ \cdot \pi_2 \cdot \pi_2\rangle\rangle
\end{array}\right.
\end{array}
$$

$base\ k = (1, \mathsf{succ}\ \ k, 1, 1)$
$loop = pack \cdot \langle \langle mul \cdot \pi_1, \mathsf{succ}\ \cdot \pi_2 \cdot \pi_1\rangle, \langle mul \cdot \pi_2, \mathsf{succ}\ \cdot \pi_2 \cdot \pi_2\rangle\rangle \cdot unpack$
  where
    $unpack\ (a, b, c, d) = ((a, b), (c, d))$
    $pack\ ((a, b), (c, d)) = (a, b, c, d)$

Problema 4

$inFTree = [Unit, uncurry2\ \ Comp]$
  where $uncurry2\ f = \lambda(x, (y, z)) \rightarrow f\ x\ y\ z$
$outFTree\ (Unit\ c) = i_1\ c$
$outFTree\ (Comp\ a\ t1\ t2) = i_2\ (a, (t1, t2))$
$baseFTree\ f\ g\ h = g + (f \times (h \times h))$

$$recFTree\ f = baseFTree\ id\ id\ f$$
$$cataFTree\ a = a \cdot (recFTree\ (cataFTree\ a)) \cdot outFTree$$
$$anaFTree\ f = inFTree \cdot (recFTree\ (anaFTree\ f)) \cdot f$$
$$hyloFTree\ a\ c = cataFTree\ a \cdot anaFTree\ c$$

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

A função *generatePTree* é um anamorfismo que cria uma árvore de folhas/nodos de tamanho$\sqrt{2}^n$, sendo n a profundidade da folha/nodo, logo as folhas terão sempre tamanho 1 (n = 0).

$generatePTree = anaFTree\ f\ \mathbf{where}$
$\quad f\ n = p2p\ (i_2\ (r, ((pred\ n), (pred\ n))), i_1\ 20)\ (n \equiv 0)$
$\quad\quad \mathbf{where}\ r = (sqrt\ 2) \uparrow (fromIntegral\ n) * 20$

A função *drawPTree* converte uma árvore para uma Picture através da função auxiliar drawTree e depois recursivamente vai reduzindo á ordem da árvore por 1 e adicionando as Pictures convertidas à lista devolvida. Esta lista no final é invertida para ficar com a ordem correta. A função auxiliar *drawPTree* . No final as Pictures são juntas numa única Picture.

$drawPTree = cycle \cdot reverse \cdot aux$
$\quad \mathbf{where}$
$\quad\quad aux\ (Unit\ a) = singl\ \$\ drawStep\ (Unit\ a)$
$\quad\quad aux\ a = (drawStep\ a) : aux\ (cutLeaves\ a)$
$\quad\quad\quad \mathbf{where}$
$\quad\quad\quad\quad cutLeaves\ (Unit\ s) = Unit\ s$
$\quad\quad\quad\quad cutLeaves\ (Comp\ s\ (Unit\ \_)\ (Unit\ \_)) = Unit\ s$
$\quad\quad\quad\quad cutLeaves\ (Comp\ s\ l\ r) = Comp\ s\ (cutLeaves\ l)\ (cutLeaves\ r)$
$\quad\quad drawStep = pictures \cdot cataFTree\ [f, g]$
$\quad\quad\quad \mathbf{where}$
$\quad\quad\quad\quad f = singl \cdot square$
$\quad\quad\quad\quad g\ (s, (l, r)) = (f\ s) +\!\!+\ \mathsf{conc}\ (left\ l, right\ r)$
$\quad\quad\quad\quad\quad \mathbf{where}$
$\quad\quad\quad\quad\quad\quad left = \mathsf{map}\ (translate\ (-s\ /\ 2)\ s \cdot rotate\ (-45))$
$\quad\quad\quad\quad\quad\quad right = \mathsf{map}\ (translate\ (s\ /\ 2)\ s \cdot rotate\ 45)$

Problema 5

O *singletonbag* coloca num *Bag* uma lista só com um par. Esse par é constituido pela variável passada à função e o número 1.

O *muB* retira do *Bag*. Depois, multiplica o segundo elemento de cada par da lista por todos os segundos elementos de cada par dentro da lista, que está dentro do *Bag* do primeiro elemento do par.

O *dist* retira do *Bag* e, a cada elemento da lista, replica o valor à esquerda do par, o número de vezes igual ao segundo par. Depois junta as listas de listas numa única lista usando o *concat*. Por fim, usa-se a função *uniform* para converter a lista numa distribuição.

$singletonbag = B \cdot singl \cdot \langle id, \underline{1} \rangle$
$\mu = B \cdot aux \cdot unB$
$\quad \mathbf{where}$
$\quad\quad aux = cataList\ \$\ [nil, \mathsf{conc} \cdot ((\widehat{f} \cdot swap \cdot (unB \times id)) \times id)]$
$\quad\quad\quad \mathbf{where}$
$\quad\quad\quad\quad f\ n = \mathsf{map}\ (id \times (*n))$

$dist = uniform \cdot concat \cdot \mathsf{map}\ (flip\ \widehat{replicate}) \cdot unB$

D Como exprimir cálculos e diagramas em LaTeX/lhs2tex

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

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

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 [?].