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

Departamento de Informática
Universidade do Minho

Junho de 2018

Grupo nr.	(37)
a78679	Diana Ribeiro Barbosa
a78806	José Pedro Ferreira de Oliveira
a77377	Pedro Henrique Moreira Gomes Fernandes

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.

type $MagicNo = String$
type $Time = Int$ -- em milisegundos
type $Entity = String$
type $Value = Int$

Neste contexto, implemente as seguintes funções:

1. Defina a função $allTransactions :: Blockchain \to Transactions$, 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 $ledger :: Blockchain \to Ledger$, 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 $isValidMagicNr :: Blockchain \to Bool$, 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.

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

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:

[2] Cf. módulo $Data.Matrix$.

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

(a) Matriz de exemplo bm.

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

(b) Quadtree de exemplo qt.

Figure 1: Exemplos de representações de bitmaps.

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

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

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

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

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

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 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"
```

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

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

[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.

Figure 2: Manipulação de uma figura bitmap utilizando quadtrees.

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

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

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

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

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

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

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

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

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

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

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

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

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

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

Problema 3

O cálculo das combinações de n k-a-k,

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

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,

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

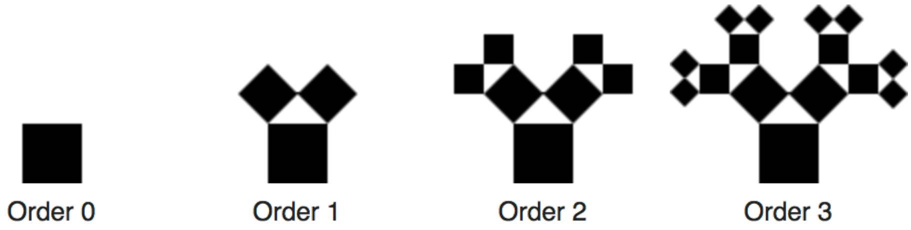


Figure 3: Passos de construção de uma árvore de Pitágoras de ordem 3.

onde

$$
\begin{aligned}
&h\ k\ d = \frac{f\ k\ d}{g\ d}\\
&f\ k\ d = \frac{(d+k)!}{k!}\\
&g\ d = d!
\end{aligned}
$$

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

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

e

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

A partir daqui alguém derivou a seguinte implementação:

$$\left(\begin{array}{c} n \\ k \end{array}\right) = h\ k\ (n-k)\ \textbf{where}\ h\ k\ n = \textbf{let}\ (a, \_, b, \_) = \mathsf{for}\ loop\ (base\ k)\ n\ \textbf{in}\ a\ /\ b$$

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* $\left(\begin{array}{c} n \\ k \end{array}\right)$ *coincide com a sua especificação (1):*

$$prop3\ (NonNegative\ n)\ (NonNegative\ k) = k \leqslant n \Rightarrow \left(\begin{array}{c} n \\ k \end{array}\right) \equiv n!\ /\ (k! * (n-k)!)$$

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:

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

1. Defina a função $\mathit{generatePTree} :: \mathit{Int} \to \mathit{PTree}$, 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:*

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

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

   $$\mathit{prop4b}\ (\mathit{SmallNat}\ n) = (\mathit{isBalancedFTree} \cdot \mathit{generatePTree})\ n$$

2. Defina a função $\mathit{drawPTree} :: \mathit{PTree} \to [\mathit{Picture}]$, 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

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

um tipo dado.[6] A lista $[\mathit{Pink}, \mathit{Green}, \mathit{Red}, \mathit{Blue}, \mathit{Green}, \mathit{Red}, \mathit{Green}, \mathit{Pink}, \mathit{Blue}, \mathit{White}]$ 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":

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

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:

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

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

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

onde $\mathit{bagOfMarbles}$ é o saco de berlindes referido acima, obtendo-se:

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

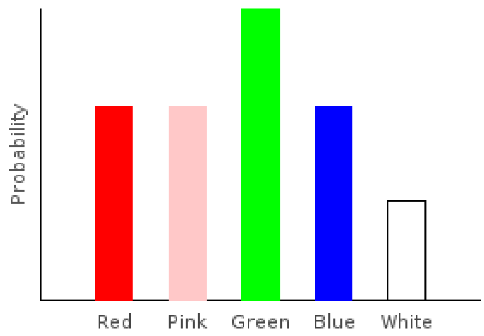


Figure 4: Distribuição de berlindes num saco.

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

Mais ainda, se quisermos saber o total de berlindes em $bagOfMarbles$ basta calcular $\mathsf{fmap}\ (!)\ bagOfMarbles$ 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,

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

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

<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

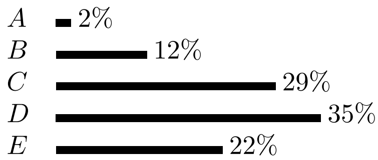
A Mónade para probabilidades e estatística

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

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

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*:

```
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:

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

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

De maneira a iniciar a resolução do problema foram definidas as seguintes funções:

$$\mathit{inBlockchain} = [\mathit{Bc}, \mathit{Bcs}]$$

Através do isomorfismo **in.out = id** foi possivel obter o out para Blockchain de acordo com a prova seguinte:

$$
\begin{array}{ll}
 & \mathit{out} \cdot \mathbf{in} = \mathit{id}\\
\equiv & \quad \{\ \text{Definição de in para Blockchain}\ \}\\
 & \mathit{out} \cdot [\mathit{Bc}, \mathit{Bcs}] = \mathit{id}\\
\equiv & \quad \{\ \text{Lei 20 Fusão-+}\ \}\\
 & [\mathit{out} \cdot \mathit{Bc}, \mathit{out} \cdot \mathit{Bcs}] = \mathit{id}\\
\equiv & \quad \{\ \text{Lei 27 Eq-+}\ \}\\
 & \left\{ \begin{array}{l} \mathit{out} \cdot \mathit{Bc} = i_1\\ \mathit{out} \cdot \mathit{Bcs} = i_2 \end{array} \right.\\
\equiv & \quad \{\ \text{Lei 73, Lei 74}\ \}\\
 & \left\{ \begin{array}{l} \mathit{out}\ (\mathit{Bc}\ a) = i_1\ a\\ \mathit{out}\ (\mathit{Bcs}\ (a, b)) = i_2\ (a, b) \end{array} \right.\\
\Box &
\end{array}
$$

$$
\begin{array}{l}
\mathit{outBlockchain}\ (\mathit{Bc}\ a) = i_1\ a\\
\mathit{outBlockchain}\ (\mathit{Bcs}\ (a, b)) = i_2\ (a, b)
\end{array}
$$

O diagrama do catamorfismo de Blockchain apresenta-se de seguida, e através deste foi possivel deduzir as restantes funções.

$$
\begin{tikzcd}
Blockchain \arrow[r, "out"] \arrow[d, "k=(\!|g\ \cdot|\!)_A"'] & Block + Block \times Blockchain \arrow[d, "id+id\times k"] \\
A & Block + Block \times A \arrow[l, "g"]
\end{tikzcd}
$$

$recBlockchain\ f = id + (id \times f)$
$cataBlockchain\ g = g \cdot (recBlockchain\ (cataBlockchain\ g)) \cdot outBlockchain$
$anaBlockchain\ g = inBlockchain \cdot (recBlockchain\ (anaBlockchain\ g)) \cdot g$
$hyloBlockchain\ f\ g = cataBlockchain\ f \cdot anaBlockchain\ g$

De seguida são apresentadas as soluções obtidas pelo grupo para cada uma das alíneas do problema 1.

C.1 allTransactions

A função **allTransactions** calcula a lista de todas as Transactions presentes numa Blockchain, deste modo foi definido um catamorfismo de Blockchain que para o caso base de um Block - (MagicNo,(Time,Transactions)) utiliza a função π_2 para extrair a lista de Transactions. Para o caso de um Blockchain - (Block,Blockchain) utiliza de novo a função π_2 para extrair a lista de Transactions do Block e a função *conc* para concatenar a lista de transações ao resultado recursivo do resto da Blockchain.
A solução é apresentada a seguir.

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

C.2 Ledger

A função *Ledger* calcula o valor que cada entidade detém numa Blockchain, para defini-la utilizamos três catamorfismos e um conjunto de funções auxiliares que serão descritas mais abaixo. O processo é seguinte:

- Obter a lista de transações através do catamorfismo **allTransactions**
- Obter um par com a lista das entidades (obtida através da função **getEntities**) e a lista das transações
- Aplicar um catamorfismo **cataSaldo** à lista das transações, obtendo um par da lista das entidades e do catamorfismo aplicado às transações
- Obter um par com a lista das entidades e uma lista com o saldo de cada entidade
- Obter uma lista de pares (Entidade,Saldo).

A solução é apresentada de seguida:

$ledger = \widehat{zip} \cdot \langle \widehat{\pi_1, \cdot swap} \rangle \cdot (id \times cataSaldo) \cdot \langle getEntities, id \rangle \cdot allTransactions$

O catamorfismo de listas **cataSaldo** dada uma lista de transações e uma entidade calcula o saldo dessa entidade.

$cataSaldo :: [\,Transaction] \rightarrow String \rightarrow Int$
$cataSaldo\ transactions\ entity = (cataList\ [\underline{0}, addInt \cdot ((getSaldo\ entity) \times id)])\ transactions$

$getSaldo :: String \rightarrow Transaction \rightarrow Int$
$getSaldo\ e\ (a, (b, c)) \mid e \equiv a = -b$
$\quad \mid e \equiv c = b$
$\quad \mid otherwise = 0$

Foi utilizada a função addInt porque a add pré-definida trabalha com Integer

$$addInt :: (Int, Int) \rightarrow Int$$
$$addInt\ (a, b) = a + b$$

O catamorfismo de listas **cataEntities** que dá a lista de todas as entidades presentes em todas as transações

$$cataEntities = cataList\ [nil, \mathsf{conc} \cdot ((pairToList \cdot \langle \pi_1, \pi_2 \cdot \pi_2 \rangle) \times id)]$$

Esta função elimina as entidades repetidas da lista produzida pelo catamorfismo anterior através da função **rmDuplicates**.

$$getEntities :: (Ord\ a) \Rightarrow [(a, (b, a))] \rightarrow [a]$$
$$getEntities = rmDuplicates \cdot cataEntities$$
$$rmDuplicates :: (Ord\ a) \Rightarrow [a] \rightarrow [a]$$
$$rmDuplicates = \mathsf{map}\ \ head \cdot group \cdot sort$$
$$pairToList :: (a, a) \rightarrow [a]$$
$$pairToList\ (x, y) = [x, y]$$

C.3 isValidMagicNr

A função **isValidMagicNr** verifica se todos os números mágicos numa blockchain são únicos. Para definir esta função utilizamos um catamorfismo **cataNrMagico** que para o caso base de um Block, aplica um $\langle \pi_1, nil \rangle$ de maneira a obter o número mágico desse Block e aplica a função nil ao lado direito do par para obter uma lista vazia, o resultado de π_1 (número mágico) vai ser posteriormente inserido na lista através da função *cons*. Para o caso de um Blockchain, é usada novamente a função *cons* para inserir o número mágico à cabeça da lista resultado recursivo para o resto da Blockchain. Neste momento temos uma lista com todos os números mágicos da Blockchain, é altura de verificar se existem alguns repetidos utilizamos para isso uma função **checkDuplicates** que dada uma lista verifica se existem ou não elementos repetidos. A solução é apresentada de seguida.

$$isValidMagicNr = checkDuplicates \cdot cataNrMagico$$

-- catamorfismo que transforma um blockchain

$$cataNrMagico :: Blockchain \rightarrow [String]$$
$$cataNrMagico = cataBlockchain\ [cons \cdot \langle \pi_1, nil \rangle, cons \cdot (\pi_1 \times id)]$$

-- verifica os se existem número mágicos repetidos

$$checkDuplicates :: (Ord\ a) \Rightarrow [a] \rightarrow Bool$$
$$checkDuplicates\ x = ((rmDuplicates\ x) \equiv x)$$

Problema 2

De maneira a iniciar a resolução do problema 2, foi necessário primeiro definir funções que nos permitam criar catamorfismos e anamorfismos para a estrutura de dados do problema em causa.

$$pairToCell\ (a, (b, c)) = Cell\ a\ b\ c$$
$$pairToBlock\ (a, (b, (c, d))) = Block\ a\ b\ c\ d$$

$$inQTree = [pairToCell, pairToBlock]$$

Através do isomorfismo **in.out = id** foi possivel obter o out para Blockchain de acordo com a prova seguinte:

$$
\begin{aligned}
& outQTree \cdot inQTree = id \\
\equiv \quad & \{ \text{ Definição de inQTree } \} \\
& outQTree \cdot [toCell, toBlock] = id \\
\equiv \quad & \{ \text{ Lei 20 Fusão-+ } \} \\
& [outQTree \cdot toCell, outQTree \cdot toBlock] = id \\
\equiv \quad & \{ \text{ Lei 17 Universal-+ } \} \\
& \begin{cases} id \cdot i_1 = outQTree \cdot toCell \\ id \cdot i_2 = outQTree \cdot toBlock \end{cases} \\
\equiv \quad & \{ \text{ Lei 73, Lei 74 } \} \\
& \begin{cases} outQTree\ toCell\ (a, (b, c)) = i_1\ (a, (b, c)) \\ outQTree\ toBlock\ (a, (b, (c, d))) = i_2\ (a, (b, (c, d))) \end{cases} \\
\square &
\end{aligned}
$$

$outQTree\ (Cell\ a\ b\ c) = i_1\ (a, (b, c))$
$outQTree\ (Block\ a\ b\ c\ d) = i_2\ (a, (b, (c, d)))$

O diagrama do catamorfismo de QTree apresenta-se de seguida, e através deste foi possivel deduzir as restantes funções.

$$
\begin{tikzcd}
QTree\ A \arrow[r, "out"] \arrow[d, "k=cataQTree\ g"'] & (A, (Int, Int)) + (QTree\ A, (QTree\ A, (QTree\ A, QTree\ A))) \arrow[d, "recQTree(k)"] \\
QTreeA & (A, (Int, Int)) + (QTree\ A, (QTree\ A, (QTree\ A, QTree\ A))) \arrow[l, "g"]
\end{tikzcd}
$$

$baseQTree\ g\ f = (g \times id) + (f \times (f \times (f \times f)))$
$recQTree\ f = baseQTree\ id\ f$
$cataQTree\ g = g \cdot (recQTree\ (cataQTree\ g)) \cdot outQTree$
$anaQTree\ g = inQTree \cdot (recQTree\ (anaQTree\ g)) \cdot g$
$hyloQTree\ g1\ g2 = cataQTree\ g1 \cdot anaQTree\ g2$

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

C.4 rotateQTree

Esta função roda 90 graus uma QTree, para tal é necessário alterar a posição dos blocos e o formato das células. Foi feito um catamorfismo de QTree, em que no caso base de uma Cell é usada a função **rotateCell** que roda 90 graus uma Cell, ou seja, troca o elementos do par que define a dimensão da matriz.

Para o caso de um Block troca-se a posição das QTree de maneira a ser feita uma rotação de 90 graus, através da função **rotateBlock**, A solução encontrada está apresentada de seguida.

$rotateQTree = cataQTree\ [rotateCell, rotateBlock]$
$rotateCell\ (a, (b, c)) = Cell\ a\ c\ b$
$rotateBlock\ (a, (b, (c, d))) = Block\ c\ a\ d\ b$

Assumindo que A, B, C e D são QTree que pertencem a um bloco a rotação de 90 graus leva a um reposionamento das QTree de acordo com o diagrama seguinte.

$$\begin{bmatrix} A & B \\ C & D \end{bmatrix} = \begin{bmatrix} C & A \\ D & B \end{bmatrix}$$

C.5 scaleQTree

A função scaleQTree recalcula a dimensão de uma QTree de acordo com um fator, para tal é necessário multiplicar os elementos que definem o tamanho de cada célula por esse fator. A solução é apresentada de seguida.

$$\begin{aligned}
&scaleQTree\ a = cataQTree\ [scaleCell\ a, pairToBlock] \\
&scaleCell\ mult\ (x, (y, z)) = Cell\ x\ (mult * y)\ (mult * z)
\end{aligned}$$

O seguinte diagrama demonstra o catamorfismo de QTree utilizado, em que g = $[scaleCell, pairToBlock]$

$$\begin{tikzcd}
Int \times QTree\ A \arrow[r, "out"] \arrow[d, "k = cataQTree\ g"'] & Int \times (A, (Int, Int)) + (QTree\ A, (QTree\ A, (QTree\ A, QTree\ A))) \arrow[d, "recQTree\ (k)"] \\
QTree\ A & Int \times (A, (Int, Int)) + (QTree\ A, (QTree\ A, (QTree\ A, QTree\ A))) \arrow[l, "g"]
\end{tikzcd}$$

C.6 invertQTree

A função invertQTree inverte as cores de uma QTree

$$\begin{aligned}
&invertQTree = cataQTree\ [invertCell, pairToBlock] \\
&invertCell\ ((PixelRGBA8\ r\ g\ b\ a), (n, m)) = Cell\ (PixelRGBA8\ (255 - r)\ (255 - g)\ (255 - b)\ a)\ n\ m
\end{aligned}$$

O seguinte diagrama demonstra o catamorfismo de QTree utilizado, em que g = $[invertCell, pairToBlock]$.

$$\begin{tikzcd}
QTree\ A \arrow[r, "out"] \arrow[d, "k = cataQTree\ g"'] & (A, (Int, Int)) + (QTree\ A, (QTree\ A, (QTree\ A, QTree\ A))) \arrow[d, "recQTree(k)"] \\
QTreeA & (A, (Int, Int)) + (QTree\ A, (QTree\ A, (QTree\ A, QTree\ A))) \arrow[l, "g"]
\end{tikzcd}$$

C.7 compressQTree

A função **compressQTree** corta as folhas da árvore de maneira a reduzir a sua profundidade dado um determinado nivel, para a definição desta função foi feito um anamorfismo de QTree. A solução obtida é apresentada a seguir.

$$\begin{aligned}
&compressQTree\ a\ b = (anaQTree\ geneForCompression)\ (a, b) \\
&geneForCompression\ (x, (Cell\ a\ b\ c)) = i_1\ (a, (b, c)) \\
&geneForCompression\ (x, block@(Block\ a\ b\ c\ d)) \\
&\quad \mid x \geqslant (depthQTree\ block) = i_1\ ((attributeValue\ block), ((\pi_1\ (sizeQTree\ block)), (\pi_2\ (sizeQTree\ block)))) \\
&\quad \mid otherwise = i_2\ (((x, a), ((x, b), ((x, c), (x, d)))))
\end{aligned}$$

A função **attributeValue** é usada para obter um valor qualquer de uma QTree, foi criada para contornar a necessidade de atribuir um valor a uma Cell que foi criada a partir de um Block.

$$\begin{aligned}
&attributeValue :: QTree\ a \rightarrow a \\
&attributeValue\ (Cell\ x\ y\ z) = x \\
&attributeValue\ (Block\ x\ y\ z\ k) = attributeValue\ x
\end{aligned}$$

O gene do anamorfismo é explicado a seguir:

- Caso receba uma Cell, coloca-a do lado esquerdo usando i_1, pois chegamos a uma folha da árvore que não deve ser cortada.

- Caso receba um Block, e a sua profundidade é menor ou igual ao nível de compressão dado como argumento, elimina esse bloco convertendo-o para uma Cell e por consequência elimina os seus filhos. De seguida, executa aplica i_1 para colocar a nova célula do lado esquerdo do par.
- Caso receba um Block, e a sua profundidade é maior que o nível de compressão dado como argumento, coloca-o do lado direito usando i_2, sem efetuar qualquer alteração.

O seguinte diagrama demonstra o anamorfismo de QTree utilizado:

$$
\begin{tikzcd}
QTree\ A & (A,(Int,Int)) + (QTree\ A,(QTree\ A,(QTree\ A, QTree\ A))) \arrow[l, "inQTree"'] \\
Int \times QTree\ A \arrow[u, "f"] \arrow[r, "g"'] & (A,(Int,Int)) + (Int \times (QTree\ A,(Int \times QTree\ A,(Int \times QTree\ A, Int \times QTree\ A)))) \arrow[u, "id+f \times f \times f \times f"']
\end{tikzcd}
$$

C.8 outlineQTree

A função **outlineQTree** apresenta o contorno de uma malha poligonal explicada no enunciado. Inicialmente é preciso verificar se a célula, após aplicada a função dada, é de valor *True*. Em caso positivo, utiliza-se a função outlineBlock que, dado um tamanho de bloco, procede ao contorno do mesmo.

O catamorfismo de QTree definido transforma uma QTree na respetiva QTree de Booleanos. Após esta conversão, basta utilizar a função **qt2bm** para converter para uma Matrix, como pedido no enunciado.

A solução obtido é apresentada a seguir.

$$
\begin{array}{l}
outlineQTree\ f = qt2bm \cdot (cataQTree\ [outlineCell\ f, pairToBlock]) \\
outlineCell\ f\ (a,(b,c)) = \mathbf{if}\ (f\ a)\ \mathbf{then}\ (outlineBlock\ b\ c)\ \mathbf{else}\ (Cell\ (f\ a)\ b\ c) \\
outlineBlock\ a\ b = Block \\
\quad (Block\ (Cell\ True\ 1\ 1) \\
\qquad (Cell\ True\ (a-2)\ 1) \\
\qquad (Cell\ True\ 1\ (b-2)) \\
\qquad (Cell\ False\ (a-2)\ (b-2))) \\
\quad (Cell\ True\ 1\ (b-1)) \\
\quad (Cell\ True\ (a-1)\ 1) \\
\quad (Cell\ True\ 1\ 1)
\end{array}
$$

O seguinte diagrama demonstra o catamorfismo de QTree utilizado, em que g = $[outLineCell\ f, pairToBlock]$

$$
\begin{tikzcd}
QTree\ A \arrow[r, "out"] \arrow[d, "k=cataQTree\ g"'] & f \times (A,(Int,Int)) + (QTree\ A,(QTree\ A,(QTree\ A, QTree\ A))) \arrow[d, "recQTree(k)"] \\
QTreeA & f \times (A,(Int,Int)) + (QTree\ A,(QTree\ A,(QTree\ A, QTree\ A))) \arrow[l, "g"]
\end{tikzcd}
$$

Problema 3

Como sugerido no enunciado a abordagem seguida foi fazer o $\langle f, l \rangle$ e $\langle g, s \rangle$, para isso deduzimos através das definições em pointwise as funções. Finalmente aplicamos a lei da recursividade múltipla para os dois splits obtidos e obtemos as definições da base e loop para o for.

$$\begin{cases} fk\ 0 = 1 \\ fk\ (d+1) = (d+k+1) * fk\ d \end{cases} \begin{cases} lk\ 0 = 1 \\ lk\ (d+1) = lk\ d + 1 \end{cases}$$

$\equiv$ { Lei 73 (x2), Lei 74 (x4), Definiçao de (d+k+1), Lei 76 (x2), Lei 78 }

$$\begin{cases} fk \cdot \underline{0} = \underline{1} \\ fk \cdot \mathsf{succ} = mul \cdot \langle lk, fk \rangle \end{cases} \begin{cases} lk \cdot \underline{0} = \underline{(k+1)} \\ lk \cdot \mathsf{succ} = \mathsf{succ} \cdot lk \end{cases}$$

$\equiv$ { Lei 27 eq+ }

$$\begin{cases} [fk \cdot \underline{0}, fk \cdot \mathsf{succ}] = [\underline{1}, mul \cdot \langle lk, fk \rangle] \\ [lk \cdot \underline{0}, lk \cdot \mathsf{succ}] = [\underline{(k+1)}, \mathsf{succ} \cdot lk] \end{cases}$$

$\equiv$ { Definição de in dos naturais, Lei Fusão-+ (x2), Lei Absorção-+ (x2) }

$$\begin{cases} fk \cdot \mathbf{in} = [\underline{1}, mul] \cdot (id + \langle lk, fk \rangle) \\ lk \cdot \mathbf{in} = [\underline{(k+1)}, \mathsf{succ}] \cdot (id + lk) \end{cases}$$

$\equiv$ { Definição de swap e Lei 7 Cancelamento-x }

$$\begin{cases} fk \cdot \mathbf{in} = ([\underline{1}, mul] \cdot swap) \cdot (id + \langle fk, lk \rangle) \\ lk \cdot \mathbf{in} = ([\underline{(k+1)}, \mathsf{succ} \cdot \pi_2] \cdot (id + \langle fk, lk \rangle)) \end{cases}$$

$\equiv$ { Lei 50 Fokkinga }

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

□

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

$\equiv$ { Lei 73 (x2), Lei 74 (x4), Lei 76 (x2), Definição de (d+1) e Lei 78 }

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

$\equiv$ { Lei 27 eq-+ }

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

$\equiv$ { Definição de in dos naturais, Lei Fusão (x2), Lei Absorção (x2) }

$$\begin{cases} g \cdot \mathbf{in} = [\underline{1}, mul] \cdot (id + \langle s, g \rangle) \\ s \cdot \mathbf{in} = ([\underline{(1)}, \mathsf{succ}] \cdot \pi_1) \cdot (id + \langle s, g \rangle) \end{cases}$$

$\equiv$ { Propriedade do swap (x2) e Lei 7 Cancelamento-x }

$$\begin{cases} g \cdot \mathbf{in} = [\underline{1}, mul \cdot swap] \cdot (id + \langle g, s \rangle) \\ s \cdot \mathbf{in} = ([\underline{1}, \mathsf{succ} \cdot \pi_1 \cdot swap] \cdot (id + \langle g, s \rangle)) \end{cases}$$

$\equiv$ { Lei 50 Fokkinga }

$$\langle g, s \rangle = (\!|\langle [\underline{1}, mul \cdot swap], [\underline{1}, \mathsf{succ} \cdot \pi_1 \cdot swap] \rangle \cdot |\!)_A$$

□

$$\begin{cases} (\!| i \cdot |\!)_A = (\!| \langle [\underline{1}, mul \cdot swap], [\underline{(k+1)}, \mathsf{succ} \cdot \pi_2] \rangle \cdot |\!)_A \\ (\!| j \cdot |\!)_A = (\!| \langle [\underline{1}, mul \cdot swap], [\underline{1}, \mathsf{succ} \cdot \pi_1 \cdot swap] \rangle \cdot |\!)_A \end{cases}$$

$\equiv \quad \{$ Lei 51 Banana-split $\}$

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

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

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

$\equiv \quad \{$ Lei da troca (x2), Def de funtor (F p1) e (F p2), Lei 75 Definição Constante, 3.90 apontamentos $\}$

$$\langle (\!| i \cdot |\!)_A, (\!| j \cdot |\!)_A \rangle = (\!| \langle [\langle \underline{1}, \underline{(k+1)} \rangle, \langle mul \cdot swap, \mathsf{succ} \cdot \pi_2 \rangle] \cdot F\ \pi_1, [\langle \underline{1}, \underline{(k+1)} \rangle, \langle mul \cdot swap, \mathsf{succ} \cdot \pi_2 \rangle] \cdot F\ \pi_2 \rangle \cdot |\!)_A$$

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

$$\langle (\!| i \cdot |\!)_A, (\!| j \cdot |\!)_A \rangle = (\!| [\langle \langle \underline{1}, \underline{(k+1)} \rangle, \langle \underline{1}, \underline{1} \rangle \rangle, \langle \langle mul \cdot swap, \mathsf{succ} \cdot \pi_2 \rangle \cdot \pi_1, \langle mul \cdot swap, \mathsf{succ} \cdot \pi_1 \cdot swap \rangle \cdot \pi_2 \rangle] \cdot |\!)_A)$$

$\equiv \quad \{$ Definição for b i $\}$

$$\begin{cases} b = \langle \langle mul \cdot swap, \mathsf{succ} \cdot \pi_2 \rangle \cdot \pi_1, \langle mul \cdot swap, \mathsf{succ} \cdot \pi_1 \cdot swap \rangle \cdot \pi_2 \rangle \\ i = \langle \langle \underline{1}, \underline{(k+1)} \rangle, \langle \underline{1}, \underline{1} \rangle \rangle \end{cases}$$

$\square$

$untuple\ ((i, j), (k, z)) = (i, j, k, z)$
$tuple\ (i, j, k, z) = ((i, j), (k, z))$
$loop = untuple \cdot \langle \langle mul \cdot swap \cdot \pi_1, \mathsf{succ} \cdot \pi_2 \cdot \pi_1 \rangle \cdot tuple, \langle mul \cdot swap \cdot \pi_2, \mathsf{succ} \cdot \pi_1 \cdot swap \cdot \pi_2 \rangle \cdot tuple \rangle$
$base = untuple \cdot \langle \langle one, \mathsf{succ} \rangle, \langle one, one \rangle \rangle$

Problema 4

De maneira a resolver o problema 4, foi necessário definir as funções que facilitam a manipulação do tipo de dados *FTree*:

$inFTree$ usa os construtores de $FTree$, usando uma função auxiliar $toComp$ de maneira a poder converter um par recebido.

$inFTree = [Unit, toComp]$
$toComp\ (a, (b, c)) = Comp\ a\ b\ c$

$outFTree$ foi derivada de uma maneira semelhante à out das Blockchain (Problema 1):

$outFTree\ (Unit\ a) = i_1\ a$
$outFTree\ (Comp\ a\ b\ c) = i_2\ (a, (b, c))$

As restantes funções são:

$baseFTree\ f\ g\ h = g + (f \times (h \times h))$
$recFTree\ f = baseFTree\ id\ id\ f$
$cataFTree\ g = g \cdot (recFTree\ (cataFTree\ g)) \cdot outFTree$
$anaFTree\ g = inFTree \cdot (recFTree\ (anaFTree\ g)) \cdot g$
$hyloFTree\ f\ g = cataFTree\ f \cdot anaFTree\ g$

A partir da lei 47 (Def-map-cata) do formulário desta unidade curricular ficou definido o bifunctor:

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

generatePTree

A função $generatePTree$ deve gerar, para um dado valor inteiro de entrada, a árvore de Pitágoras de ordem correspondente, composta por quadrados com escalas adequadas a cada nível. Esta função será definida como um anamorfismo.

Para este efeito, partimos do diagrama do anamorfismo de $FTree$, uma vez que dada a definição de $PTree$ com **type** $PTree = FTree\ Square\ Square$, é possível inferir que a estrutura geral será idêntica, sendo apenas definidos os tipos que a $FTree$ utiliza. Dada a definição de $Square$ com **type** $Square = Float$, os tipos A e B abaixo definidos corresponderão, naturalmente, a esse mesmo tipo.

O diagrama é então o seguinte:

$$
\begin{tikzcd}
FTree\ A\ B /{}_{/}{}_{-}\, out & B + A \times (FTree\ A\ B \times FTree\ A\ B) /^{\prime}_{-} \mathbf{in} \arrow[l] \\
C \arrow[u, "f"] \arrow[r, "g"'] & B + A \times (C \times C) \arrow[u, "id + id \times (f \times f)"']
\end{tikzcd}
$$

Numa primeira tentativa, a ideia para o anamorfismo partia de um valor numérico inteiro de entrada que correspondia à ordem pretendida para a árvore de Pitágoras. Esse inteiro seria diminuído a cada iteração, ocorrendo o caso de paragem para esta computação quando esse valor atingisse o 0.

Este anamorfismo inicial servia-se então do seguinte gene:
$genePTree = (id + \langle \pi_2, \langle \pi_1, \pi_1 \rangle \rangle) \cdot (id + (pred \times id)) \cdot (id + \langle id, orderMultiplier \rangle) \cdot (fromIntegral + id) \cdot oneToLeft$

No entanto, o anamorfismo inicialmente sugerido tinha como resultado uma árvore de Pitágoras de dimensões invertidas, o que obrigou a que se partisse do valor de ordem mínimo para a árvore, 0, e se iterasse consecutivamente até ser atingido o valor de ordem pretendido para a árvore a construir.

Com este intuito, surgiu uma segunda versão para o anamorfismo, que define o tipo dos valores de entrada como o tuplo (Int, Int). O primeiro elemento deste par representa o valor de ordem da iteração atual e o segundo elemento corresponde ao valor de ordem final, que foi definido para a árvore a ser criada.

Para que seja obtido o par acima descrito é necessário aplicar um *split* ao valor inteiro de entrada. Este *split* será dado por $\langle \underline{0}, id \rangle$ e quando aplicado ao valor de entrada, permite que a seguir seja aplicado o anamorfismo pretendido, ficando assim definida a função $generatePTree$:

$$generatePTree = anaFTree\ genePTree \cdot \langle \underline{0}, id \rangle$$

A função $genePTree$ será o gene do anamorfismo:

$$genePTree = (id + (id \times \langle id, id \rangle)) \cdot (id + (id \times (\mathsf{succ} \times id))) \cdot (id + \langle orderMultiplier \cdot \pi_1, id \rangle) \\ \cdot ((orderMultiplier \cdot \pi_1) + id) \cdot checkComplete$$

$$
\begin{tikzcd}
Int \times Int \arrow[d, "checkComplete"] \\
Int \times Int + Int \times Int \arrow[d, "(orderMultiplier \cdot \pi_1) + id"] \\
Float + Int \times Int \arrow[d, "id + \langle orderMultiplier \cdot \pi_1, id \rangle"] \\
Float + Float \times (Int \times Int) \arrow[d, "id + id \times (\mathsf{succ} \times id)"] \\
Float + Float \times (Int \times Int) \arrow[d, "id + id \times \langle id, id \rangle"] \\
Float + Float \times ((Int \times Int) \times (Int \times Int))
\end{tikzcd}
$$

Serão aqui apresentadas as funções a que o anamorfismo recorre.

Uma delas, *orderMultiplier*, retorna o multiplicador de uma *PTree* para um dado número de ordem. Dado o valor de escala definido pelo enunciado, de $\frac{\sqrt{2}}{2}$, sabe-se então que o valor de escala a aplicar nos quadrados a adicionar numa dada ordem é dado por $(\frac{\sqrt{2}}{2})^o$, sendo o o número representante da ordem.

$$
\begin{array}{l}
\mathit{orderMultiplier} :: \mathit{Int} \rightarrow \mathit{Float} \\
\mathit{orderMultiplier}\ a = (((\mathit{sqrt}\ 2)\ /\ 2) \uparrow a)
\end{array}
$$

Por sua vez, a função *checkComplete* executa i_1 sobre um par de inteiros se estes forem iguais ou i_2 se forem diferentes. Esta função é útil para a determinação da última iteração do anamorfismo.

A primeira guarda, na qual surge $b<0 = i_1\ (a, 0)$, verifica se o valor em b é negativo para evitar um número infinito de iterações quando é pedida uma BTree com ordem negativa. Nesse caso, a ordem assumida toma o valor 0. Nas restantes guardas é efetuado o que havia sido definido em cima.

$$
\begin{array}{l}
\mathit{checkComplete} :: (\mathit{Int}, \mathit{Int}) \rightarrow (\mathit{Int}, \mathit{Int}) + (\mathit{Int}, \mathit{Int}) \\
\mathit{checkComplete}\ (a, b) \\
\quad \mid b < 0 = i_1\ (a, 0) \\
\quad \mid a \equiv b = i_1\ (a, b) \\
\quad \mid \mathit{otherwise} = i_2\ (a, b)
\end{array}
$$

drawPTree

Não foi desenvolvida uma definição para a função *drawPTree*.

$$
\mathit{drawPTree} = \bot
$$

Verificam-se com isto as propriedades *QuickCheck* relativas a este problema, como se pode verificar de seguida:

```
*Main> quickCheck (prop4a 14)
+++ OK, passed 1 tests.
*Main> quickCheck (prop4b 14)
+++ OK, passed 1 tests.
```

Problema 5

As funções que se pretendem ver desenvolvidas para a primeira alínea deste enunciado conferem funcionalidades essenciais aos mónades, evidenciando as suas propriedades de multiplicação, no caso da função μ e de unidade, no caso da função *singletonbag* (ou u).

A primeira será uma função polimórfica que permitirá reduzir em uma unidade o nível de monadificação a uma entrada que esteja num nível de monadificação igual ou superior a 2, ou seja, o seu tipo poderá ser dado por:

$$
T\ A \xleftarrow[\mu]{\qquad\qquad} T\ (T\ A)
$$

Neste caso, está em uso o mónade *Bag* e com as reduções dos níveis de monadificação será necessário ajustar os valores de multiplicidade do conteúdo da *Bag* resultante. A função μ fica então definida por:

-- mulMults :: ([(a, Int)], Int) -¿ [(a, Int)]
-- (unB . (fmap unB)) b3 == map (unB ¿¡ id) (unB b3)

$$
\begin{array}{l}
\mathit{mulMults}\ ([\,], c) = [\,] \\
\mathit{mulMults}\ (((a, b) : t), c) = (a, b * c) : (\mathit{mulMults}\ (t, c)) \\
\mu = B \cdot \mathit{concat} \cdot (\mathsf{map}\ \ \mathit{mulMults}) \cdot \mathsf{map}\ \ (\mathit{unB} \times \mathit{id}) \cdot \mathit{unB}
\end{array}
$$

A ação da função *unB* remove a monadificação do seu argumento, o que coloca os pares (Elemento, Multiplicidade) exatamente na forma de tuplo. Desta forma será possível utilizar os valores de multiplicidade dos elementos menos aninhados, os segundos elementos dos pares, que serão relevantes para a redução do nível de monadificação.

$$[(a, Int)] \xleftarrow{unB} Bag\ a$$

De seguida será necessário tornar utilizáveis os valores de multiplicidade dos elementos do nível seguinte de aninhamento. Para isto, ao resultado da aplicação de unB, uma lista de tuplos (pares (Elemento, Multiplicidade)), será necessário remover a monadificação aos seus elementos (os primeiros elementos dos tuplos da lista que resultado da aplicação de unB), "expondo" as suas multiplicidades, por ação de unB, e deixando intacta a multiplicidade do nível superior (os segundos elementos dos tuplos da lista anteriormente mencionada), por ação de id. Estas funções serão mapeadas e aplicadas paralelamente a cada elemento da lista e ficam desta forma utilizáveis todos os valores de multiplicidade necessários para o processo de redução do nível de monadificação.

$$[([(a, Int)], Int)] \xleftarrow{\mathsf{map}\ (unB \times id)} [(Bag\ a, Int)]$$

Será agora necessário remover a multiplicidade menos aninhada, e no entanto garantir a manutenção da correção das multiplicidades no mónade Bag resultante. Para isto, por ação de multiplicação das multiplicidades é obtido esse efeito. Para isto é usada a função $mulMults$, que será mapeada a cada elemento da lista resultante anterior. Para isto pretendemos que o 2º elemento dos tuplos menos aninhados sejam multiplicados pelo 2º elemento do 1º elemento dos tuplos menos aninhados, que por sua vez, como se pode inferir e verificar pelos exemplos aqui explicitados, será um tuplo também, que se apresenta na forma (Elemento, Multiplicidade). A lista de resultado ignora por completo o 2º elemento do tuplo principal, o que está de acordo com o pretendido.

$$[[(a, b)]] \xleftarrow{\mathsf{map}\ \ mulMults} [([(a, b)], b)]$$

À lista de listas resultante será aplicada a função $concat$, que permitirá unificar os conteúdos das listas interiores e a partir da lista resultante é construído o mónade resultado, por ação de B, obtendo-se assim um mónade num grau imediatamente inferior de monadificação.

Segue-se a função que permite exibir a propriedade de unidade. A já mencionada $singletonbag$ ou u "encapsula" valores de entrada, conferindo-lhes um grau superior (em uma unidade) de monadificação. Neste caso está em uso o mónade Bag e, assim sendo, a função $singletonbag$ poderá tomar a definição que se segue. A um único elemento que se pretenda colocar num Bag, será necessário colocá-lo na forma adequada para que possa ser monadificado por ação de B, ou seja, numa lista de tuplos de 2 elementos, já que B é do tipo:

$$Bag\ a \xleftarrow{B} [(a, Int)]$$

Para qualquer elemento que se pretenda encapsular, o valor da sua multiciplidade será de 1. Dessa forma o tuplo será algo como o par (Elemento, 1). Para este efeito entra em ação a função $s \backslash\_\ tuple$, que simplesmente forma o par adequado.

$$(a, 1) \xleftarrow{s\_tuple} a$$

Obtendo o par adequado, será agora apenas necessário colocá-lo numa lista, o que será efetuado pela função $singl$.

$$[a] \xleftarrow{singl} a$$

À lista de saída da aplicação da função $singl$ será apenas necessário conferir-lhe monadificação, pelo que é novamente utilizado B.

$$s\_tuple\ a = (a, 1)$$
$$singletonbag = B \cdot singl \cdot s\_tuple$$

Resta apenas a função $dist$, que para um qualquer Bag apresenta as percentagens de distribuição dos seus conteúdos, recorrendo ao mónade Dist.

$$\mathsf{Dist}\ a \xleftarrow[dist]{} Bag\ a$$

Com base na composição de funções, a função $dist$ apresenta a definição do código que se segue. Por ação da função unB, a partir do mónade de entrada obtém-se a lista de pares (Elemento, Int), sendo o valor inteiro a sua multiplicidade do elemento no Bag que acabou de ser removido. A cada um desses pares, com recurso ao mapeamento da função $repMarbles$, é obtida uma lista, que representa explicitamente a ideia definida em cada par (p.e. no caso do elemento ser uma $Marble$, seria possível um dos elementos da lista original ser algo como (Blue, 3), que por ação da função $repMarbles$, dá origem a [Blue, Blue, Blue]), utilizando a versão *uncurried* da função pré-definida $replicate$ e o par (Marble,Int) com a ordem inversa por ação da função $swap$, ou seja, um par (Int, Marble). As listas aninhadas são de seguida concatenadas e é aplicada a função $uniform$, que obterá a distribuição e os valores de probabilidade adequados.

$$repMarbles = \widehat{replicate} \cdot swap$$
$$dist = uniform \cdot concat \cdot \mathsf{map}\ \ repMarbles \cdot unB$$

A segunda alínea deste problema pretende apenas demonstrar a correção das funções desenvolvidas, por verificação da validade das propriedades de multiplicação (à esquerda) e unidade (à direita) já mencionadas e que são referidas nas notas teóricas desta unidade curricular.

$$\begin{tikzcd}
T\ (T\ A) \arrow[d, "\mu"] & T\ (T\ (T\ A)) \arrow[l, "\mu"'] \arrow[d, "T\ \mu"] \\
T\ A & T\ (T\ A) \arrow[l, "\mu"]
\end{tikzcd}
\qquad
\begin{tikzcd}
T\ (T\ A) \arrow[d, "\mu"] & T\ A \arrow[l, "u"'] \arrow[d, "T\ u"] \arrow[dl, "id"] \\
T\ A & T\ (T\ A) \arrow[l, "\mu"]
\end{tikzcd}$$

Com tudo isto, os resultados dos testes $test5a$ e $test5b$ são os seguintes:

```
*Main> quickCheck (bagOfMarbles == muB (return bagOfMarbles))
+++ OK, passed 1 tests.
*Main> quickCheck ((muB . muB) b3 == (muB .fmap muB) b3)
+++ OK, passed 1 tests.
```

D Como exprimir cálculos e diagramas em LaTeX/lhs2tex

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

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

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 [?].