Universidade do Minho
Escola de Engenharia

Cálculo de Programas
Trabalho Prático (2024/25)

Lic. em Ciências da Computação

Grupo G15

a108473	André Filipe Dourado Pinheiro
a105532	Killian Alexandre Ferreira Oliveira
a108398	Pedro Dong Mo

junho 2025

Preâmbulo

Na UC de Cálculo de Programas pretende-se ensinar a programação de computadores como uma disciplina científica. Para isso parte-se de um repertório de *combinadores* que formam uma álgebra da programação e usam-se esses combinadores para construir programas *composicionalmente*, isto é, agregando programas já existentes.

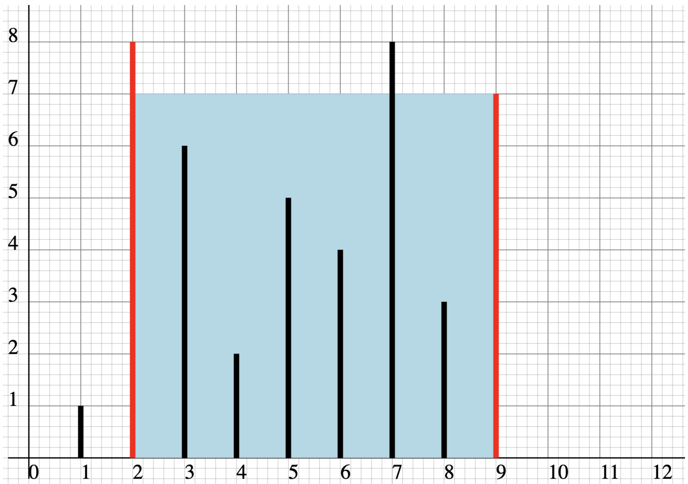
Na sequência pedagógica dos planos de estudo dos cursos que têm esta disciplina, opta-se pela aplicação deste método à programação em Haskell (sem prejuízo da sua aplicação a outras linguagens funcionais). Assim, o presente trabalho prático coloca os alunos perante problemas concretos que deverão ser implementados em Haskell. Há ainda um outro objectivo: o de ensinar a documentar programas, a validá-los e a produzir textos técnico-científicos de qualidade.

Antes de abordarem os problemas propostos no trabalho, os grupos devem ler com atenção o anexo A onde encontrarão as instruções relativas ao *software* a instalar, etc.

Valoriza-se a escrita de *pouco* código que corresponda a soluções simples e elegantes que utilizem os combinadores de ordem superior estudados na disciplina.

Problema 1

Esta questão aborda um problema que é conhecido pela designação '*Container With Most Water*' e que se formula facilmente através do exemplo da figura seguinte:



A figura mostra a sequência de números

$$hghts = [1, 8, 6, 2, 5, 4, 8, 3, 7]$$

representada sob a forma de um histograma. O que se pretende é obter a maior área rectangular delimitada por duas barras do histograma, área essa marcada a azul na figura. (A "metáfora" *container with most water* sugere que as barras selecionadas delimitam um *container* com água.)

Pretende-se definida como um catamorfismo, anamorfismo ou hilomorfismo uma função em Haskell

$$mostwater :: [Int] \rightarrow Int$$

que deverá dar essa área. (No exemplo acima tem-se $mostwater\ [1, 8, 6, 2, 5, 4, 8, 3, 7] = 49$.) A resolução desta questão deverá ser acompanhada de diagramas elucidativos.

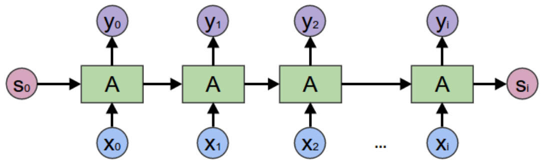


Figure 1: RNN vista como instância de um *accumulating map* [?].

Problema 2

Um dos problemas prementes da Computação na actualidade é conseguir, por engenharia reversa, interpretar as redes neuronais (RN) geradas artificialmente sob a forma de algoritmos compreensíveis por humanos.

Já foram dados passos que, nesse sentido, explicam vários padrões de RNs em termos de combinadores funcionais [?]. Em particular, já se mostrou como as RNNs (*Recurrent Neural Networks*) podem ser vistas como instâncias de *accumulating maps*, que em Haskell correspondem às funções *mapAccumR* e *mapAccumL*, conforme o sentido em que a acumulação se verifica (cf. figura 1).

A RNN que a figura 1 mostra diz-se *'one-to-one'* [?]. Há contudo padrões de RNNs mais gerais: por exemplo, o padrão *'many-to-one'* que se mostra na figura 2 extraída de [?].

Se *mapAccumR* e *mapAccumL* juntam *maps* com *folds*, pretendemos agora combinadores que a isso acrescentem *filter*, por forma a selecionar que etapas da computação geram ou não *outputs* — obtendo-se assim o efeito *'many-to-one'*. Ter-se-á, para esse efeito:

$$
\begin{array}{l}
mapAccumRfilter :: ((a,s) \to Bool) \to ((a,s) \to (c,s)) \to ([a],s) \to ([c],s) \\
mapAccumLfilter :: ((a,s) \to Bool) \to ((a,s) \to (c,s)) \to ([a],s) \to ([c],s)
\end{array}
$$

Pretende-se a implementação de *mapAccumRfilter* e *mapAccumLfilter* sob a forma de ana / cata ou hilomorfismos em Haskell, acompanhadas por diagramas.

Como caso de uso, sugere-se o que se dá no anexo F que, inspirado em [?], recorre à biblioteca `Data.Matrix`.

Problema 3

Umas das fórmulas conhecidas para calcular o número π é a que se segue,

$$\pi = \sum_{n=0}^{\infty} \frac{(n!)^2 2^{n+1}}{(2n+1)!} \tag{1}$$

correspondente à função π_{calc} cuja implementação em Haskell, paramétrica em n, é dada no anexo F.

Pretende-se uma implementação eficiente de (1) que, derivada por recursividade mútua, não calcule factoriais nenhuns:

$$\pi_{loop} = \cdots \cdot \text{for } loop\ inic\ \textbf{where}\ \cdots$$

Sugestão: recomenda-se a **regra prática** que se dá no anexo E para problemas deste género, que podem envolver várias decomposições por recursividade mútua em $\mathbb{N}_0$.

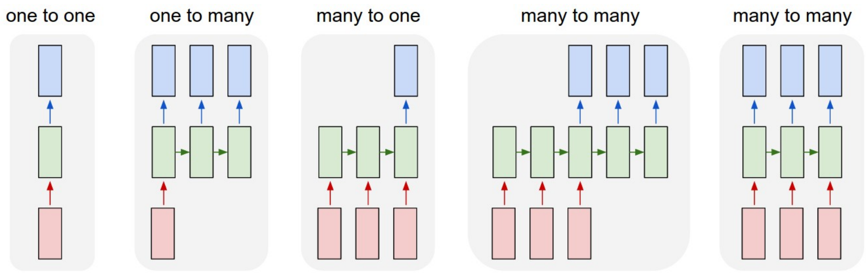


Figure 2: Várias tipologias de RNNs [?].

Problema 4

Considere-se a matriz e o vector que se seguem:

$$mat = \begin{bmatrix} 1 & -1 & 2 \\ 0 & -3 & 1 \end{bmatrix} \tag{2}$$

$$vec = \begin{bmatrix} 2 \\ 1 \\ 0 \end{bmatrix} \tag{3}$$

Em Haskell, podemos tornar explícito o espaço vectorial a que (3) pertence definindo-o da forma seguinte,

$vec :: Vec\ X$
$vec = V\ [(X_1, 2), (X_2, 1), (X_3, 0)]$

assumindo definido o tipo

$\mathbf{data}\ Vec\ a = V\ \{outV :: [(a, Int)]\}\ \mathbf{deriving}\ (Ord)$

e o "tipo-dimensão":

$\mathbf{data}\ X = X_1 \mid X_2 \mid X_3\ \mathbf{deriving}\ (Eq, Show, Ord)$

Da mesma forma que *tipamos* vec, também o podemos fazer para a matrix mat (2), cujas colunas podem ser indexadas por X também e as linhas por $Bool$, por exemplo:

$mat :: X \to Vec\ Bool$
$mat\ X_1 = V\ [(False, 1), (True, 0)]$
$mat\ X_2 = V\ [(False, -1), (True, -3)]$
$mat\ X_3 = V\ [(False, 2), (True, 1)]$

Quer dizer, matrizes podem ser encaradas como funções que dão vectores como resultado. Mais ainda, a multiplicação de mat por vec pode ser obtida correndo, simplesmente

$vec' = vec \ggg\!= mat$

obtendo-se $vec' = V\ [(False, 1), (True, -3)]$ do tipo $Vec\ Bool$. Finalmente, se for dada a matrix

$neg :: Bool \to Vec\ Bool$
$neg\ False = V\ [(False, 0), (True, 1)]$
$neg\ True = V\ [(False, 1), (True, 0)]$

então a multiplicação de neg por mat mais não será que a matriz

$$neg \bullet mat$$

também do tipo $X \to Vec\ Bool$.

Obtém-se assim uma *álgebra linear tipada*. Contudo, para isso é preciso mostrar que Vec é um **mónade**, e é esse o tema desta questão, em duas partes:

- Instanciar Vec na class $Functor$ em Haskell:

  instance $Functor\ Vec$ **where**
  $\mathsf{fmap}\ f = ...$

- Instanciar Vec na class $Monad$ em Haskell:

  instance $Monad\ Vec$ **where**
  $x \ggg f = ...$
  $return\ a = ...$

Anexos

A Natureza do trabalho a realizar

Este trabalho teórico-prático deve ser realizado por grupos de 3 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 em **todos** os exercícios do trabalho, para assim poderem responder a qualquer questão colocada na *defesa oral* do relatório.

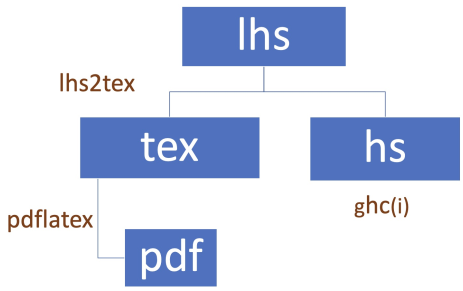
Para cumprir de forma integrada os objectivos do trabalho vamos recorrer a uma técnica de programação dita "literária" [**?**], cujo princípio base é o seguinte:

> *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 `cp2425t.pdf` que está a ler é já um exemplo de programação literária: foi gerado a partir do texto fonte `cp2425t.lhs`[1] que encontrará no material pedagógico desta disciplina descompactando o ficheiro `cp2425t.zip`.

Como se mostra no esquema abaixo, de um único ficheiro (*lhs*) gera-se um PDF ou faz-se a interpretação do código Haskell que ele inclui:



[1] O sufixo 'lhs' quer dizer *literate Haskell*.

Vê-se assim que, para além do GHCi, serão necessários os executáveis pdflatex e lhs2TeX. Para facilitar a instalação e evitar problemas de versões e conflitos com sistemas operativos, é recomendado o uso do Docker tal como a seguir se descreve.

B Docker

Recomenda-se o uso do container cuja imagem é gerada pelo Docker a partir do ficheiro `Dockerfile` que se encontra na diretoria que resulta de descompactar `cp2425t.zip`. Este container deverá ser usado na execução do GHCi e dos comandos relativos ao LaTeX. (Ver também a `Makefile` que é disponibilizada.)

Após instalar o Docker e descarregar o referido zip com o código fonte do trabalho, basta executar os seguintes comandos:

```
$ docker build -t cp2425t .
$ docker run -v ${PWD}:/cp2425t -it cp2425t
```

NB: O objetivo é que o container seja usado *apenas* para executar o GHCi e os comandos relativos ao LaTeX. Deste modo, é criado um *volume* (cf. a opção `-v ${PWD}:/cp2425t`) que permite que a diretoria em que se encontra na sua máquina local e a diretoria `/cp2425t` no container sejam partilhadas.

Pretende-se então que visualize/edite os ficheiros na sua máquina local e que os compile no container, executando:

```
$ lhs2TeX cp2425t.lhs > cp2425t.tex
$ pdflatex cp2425t
```

lhs2TeX é o pre-processador que faz "pretty printing" de código Haskell em LaTeX e que faz parte já do container. Alternativamente, basta executar

```
$ make
```

para obter o mesmo efeito que acima.

Por outro lado, o mesmo ficheiro `cp2425t.lhs` é executável e contém o "kit" básico, escrito em Haskell, para realizar o trabalho. Basta executar

```
$ ghci cp2425t.lhs
```

Abra o ficheiro `cp2425t.lhs` no seu editor de texto preferido e verifique que assim é: todo o texto que se encontra dentro do ambiente

```
\begin{code}
...
\end{code}
```

é seleccionado pelo GHCi para ser executado.

C Em que consiste o TP

Em que consiste, então, o *relatório* a que se referiu acima? É a edição do texto que está a ser lido, preenchendo o anexo G 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 cp2425t.aux
$ makeindex cp2425t.idx
```

e recompilar o texto como acima se indicou. (Como já se disse, pode fazê-lo correndo simplesmente `make` no container.)

No anexo F disponibiliza-se algum código Haskell relativo aos problemas que são colocados. Esse anexo deverá ser consultado e analisado à medida que isso for necessário.

Deve ser feito uso da programação literária para documentar bem o código que se desenvolver, em particular fazendo diagramas explicativos do que foi feito e tal como se explica no anexo D que se segue.

D Como exprimir cálculos e diagramas em LaTeX/lhs2TeX

Como primeiro exemplo, estudar o texto fonte (lhs) do que está a ler[1] onde se obtém o efeito seguinte:[2]

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

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

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

E Regra prática para a recursividade mútua em $\mathbb{N}_0$

Nesta disciplina estudou-se como fazer programação dinâmica por cálculo, recorrendo à lei de recursividade mútua.[3]

Para o caso de funções sobre os números naturais ($\mathbb{N}_0$, com functor $\mathsf{F}\ X = 1 + X$) pode derivar-se da lei que foi estudada uma *regra de algibeira* que se pode ensinar a programadores que não tenham estudado Cálculo de Programas. Apresenta-se de seguida essa regra, tomando como exemplo o cálculo do ciclo-for que implementa a função de Fibonacci, recordar o sistema:

[1] Procure e.g. por `"sec:diagramas"`.

[2] Exemplos tirados de [?].

[3] Lei (3.95) em [?], página 110.

$$
\begin{array}{l}
\mathit{fib}\ 0 = 1 \\
\mathit{fib}\ (n+1) = f\ n \\
f\ 0 = 1 \\
f\ (n+1) = \mathit{fib}\ n + f\ n
\end{array}
$$

Obter-se-á de imediato

$$
\begin{array}{l}
\mathit{fib}' = \pi_1 \cdot \mathsf{for}\ \mathit{loop}\ \mathit{init}\ \mathbf{where} \\
\quad \mathit{loop}\ (\mathit{fib}, f) = (f, \mathit{fib} + f) \\
\quad \mathit{init} = (1, 1)
\end{array}
$$

usando as regras seguintes:

- O corpo do ciclo *loop* terá tantos argumentos quanto o número de funções mutuamente recursivas.
- Para as variáveis escolhem-se os próprios nomes das funções, pela ordem que se achar conveniente.[1]
- Para os resultados vão-se buscar as expressões respectivas, retirando a variável n.
- Em *init* coleccionam-se os resultados dos casos de base das funções, pela mesma ordem.

Mais um exemplo, envolvendo polinómios do segundo grau $ax^2 + bx + c$ em $\mathbb{N}_0$. Seguindo o método estudado nas aulas[2], de $f\ x = ax^2 + bx + c$ derivam-se duas funções mutuamente recursivas:

$$
\begin{array}{l}
f\ 0 = c \\
f\ (n+1) = f\ n + k\ n \\
k\ 0 = a + b \\
k\ (n+1) = k\ n + 2\ a
\end{array}
$$

Seguindo a regra acima, calcula-se de imediato a seguinte implementação, em Haskell:

$$
\begin{array}{l}
f'\ a\ b\ c = \pi_1 \cdot \mathsf{for}\ \mathit{loop}\ \mathit{init}\ \mathbf{where} \\
\quad \mathit{loop}\ (f, k) = (f + k, k + 2 * a) \\
\quad \mathit{init} = (c, a + b)
\end{array}
$$

F Código fornecido

Problema 1

Teste relativo à figura da página 1:

$$
\mathit{test}_1 = \mathit{mostwater}\ \mathit{hghts}
$$

Problema 2

Testes relativos a *mapAccumLfilter* e *mapAccumRfilter* em geral (comparar os *outputs*)

[1] Podem obviamente usar-se outros símbolos, mas numa primeira leitura dá jeito usarem-se tais nomes.

[2] Secção 3.17 de [?] e tópico Recursividade mútua nos vídeos de apoio às aulas teóricas.

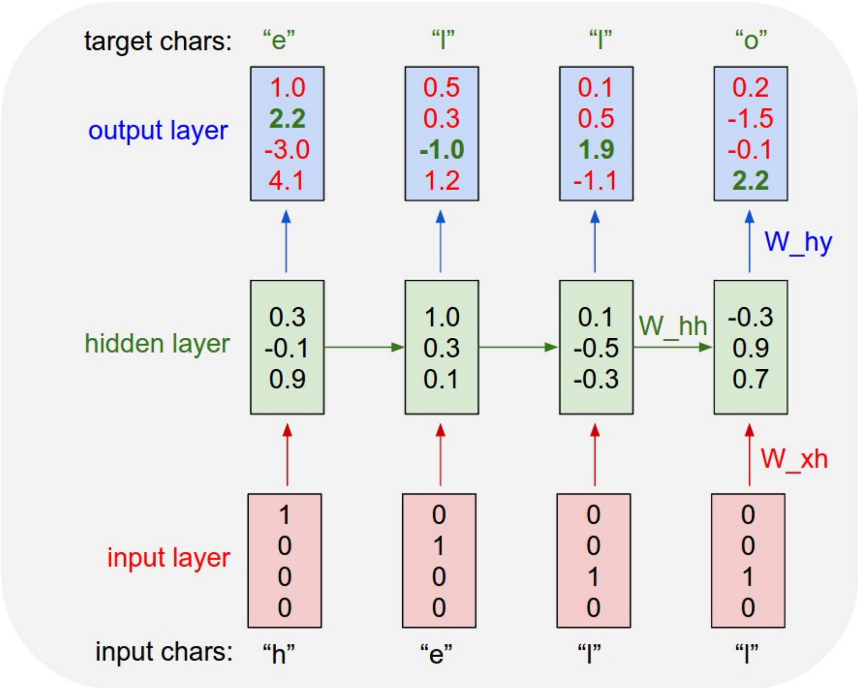


Figure 3: Exemplo *char seq* extraído de [?].

$$\begin{aligned} test_{2a} &= mapAccumLfilter\ ((>10) \cdot \pi_1)\ f\ (odds\ 12, 0) \\ test_{2b} &= mapAccumRfilter\ ((>10) \cdot \pi_1)\ f\ (odds\ 12, 0) \end{aligned}$$

onde:

$$\begin{aligned} odds\ n &= \mathsf{map}\ \ ((1+) \cdot (2*))\ [0\,..\,n-1] \\ f\ (a, s) &= (s, a + s) \end{aligned}$$

Teste

$$test_{2c} = mapAccumLfilter\ true\ step\ ([x_1, x_2, x_3, x_4], h_0)$$

baseado no exemplo de Karpathy [?] que a figura 3 mostra, usando os dados seguintes:

- Estado inicial:

$$h_0 = fromList\ 3\ 1\ [1.0, 1.0, 1, 0]$$

- *Step function*:

$$step\ (x, h) = (\alpha\ (wy * h), \alpha\ (wh * h + wx * x))$$

- Função de activação:

$$\alpha = \mathsf{fmap}\ \sigma\ \mathbf{where}\ \sigma\ x = (tanh\ x + 1)\ /\ 2$$

- *Input layer*:

$$\begin{aligned} inp &= [x_1, x_2, x_3, x_4] \\ x_1 &= fromList\ 4\ 1\ [1.0, 0, 0, 0] \\ x_2 &= fromList\ 4\ 1\ [0, 1.0, 0, 0] \\ x_3 &= fromList\ 4\ 1\ [0, 0, 1.0, 0] \\ x_4 &= x_3 \end{aligned}$$

- Matrizes exemplo:

$$wh = fromList\ 3\ 3\ [0.4, -0.2, 1.6, -3.1, 1.4, 0.1, 5.4, -2.7, 0.1]$$
$$wy = fromList\ 4\ 3\ [2.1, 1.1, 0.8, 1.3, -6.4, -3.4, -2.7, -3.8, -1.3, -0.5, -0.9, -0.4]$$
$$wx = fromLists\ [[0.0, -51.9, 0.0, 0.0], [0.0, 26.6, 0.0, 0.0], [-16.7, -5.5, -0.1, 0.1]]$$

NB: Podem ser definidos e usados outros dados em função das experiências que se queiram fazer.

Problema 3

Fórmula (1) em Haskell:

```
πcalc n = (sum · map f) [0 .. n] where
  f n = fromIntegral (n! * n! * (g n)) / fromIntegral (d n)
  g n = 2 ↑ (n + 1)
  d n = (2 * n + 1)!
```

Problema 4

Se pedirmos ao GHCi que nos mostre o vector *vec* obteremos:

```
{ X1 |-> 2 , X2 |-> 1 }
```

Este resultado aparece mediante a seguinte instância de *Vec* na classe *Show*:

```
instance (Show a, Ord a, Eq a) ⇒ Show (Vec a) where
  show = showbag · consol · outV where
    showbag = concat ·
      (++[" }"]) · ("{ ":) ·
      (intersperse " , ") ·
      sort ·
      (map f) where f (a, b) = (show a) ++ " |-> " ++ (show b)
```

Outros detalhes da implementação de *Vec* em Haskell:

```
instance Applicative Vec where
  pure = return
  (< * >) = aap
instance (Eq a) ⇒ Eq (Vec a) where
  b ≡ b′ = (outV b) ‘lequal‘ (outV b′)
    where lequal a b = isempty (a ⊖ b)
      a ⊖ b = a ++ b̄
      x̄ = [(k, −i) | (k, i) ← x]
```

Funções auxiliares:

```
consol :: (Eq b) ⇒ [(b, Int)] → [(b, Int)]
consol = filter nzero · map (id × sum) · col where nzero (_, x) = x ≢ 0

isempty :: Eq a ⇒ [(a, Int)] → Bool
isempty = all (≡ 0) · map π2 · consol

col :: (Eq a, Eq b) ⇒ [(a, b)] → [(a, [b])]
col x = nub [k ↦ [d′ | (k′, d′) ← x, k′ ≡ k] | (k, d) ← x] where a ↦ b = (a, b)
```

G Soluções dos alunos

Os alunos devem colocar neste anexo as suas soluções para os 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 ao anexo, bem como diagramas e/ou outras funções auxiliares que sejam necessárias.

Importante: Não pode ser alterado o texto deste ficheiro fora deste anexo.

Problema 1

Vamos utilizar a estratégia de *divide and conquer*, isto é, vamos construir um anamorfismo que devolve uma lista com as maiores áreas possíveis e um catamorfismo para obter a maior dessas áreas.

Para fazer isto de forma otimal, começemos por calcular a área com base no primeiro e último elemento de uma lista não vazia l.

Sendo assim, o cálculo da área é dado pela função:

$$area\ l = (min\ (head\ l)\ (last\ l)) * ((length\ l) - 1)$$

A seguir, descartamos o menor elemento entre primeiro e último elemento da lista, de forma a encontrar um valor maior. Para isso temos dois casos:

1. Se head $l \geqslant$ last l, então isto significa que calculámos a água armazenada para o recipiente de altura last$l$, por isso descartamos o last$l$.
2. Se head $l <$ last l, então isto significa que calculámos a água armazenada para o recipiente de altura head l, por isso descartamos o head l.

Recursivamente, isto pode ser escrito como

$$\begin{cases} bestAreas\ [\,] = [\,] \\ bestAreas\ l = (area\ l) : (\textbf{if}\ head\ l < last\ l\ \textbf{then}\ bestAreas\ (tail\ l)\ \textbf{else}\ bestAreas\ (init\ l)) \end{cases}$$

Tornando esta definição point-free, temos

$$\begin{cases} bestAreas\ [\,] = [\,] \\ bestAreas\ l = i_2\ (area\ l) : (\textbf{if}\ head\ l < last\ l\ \textbf{then}\ bestAreas\ (tail\ l)\ \textbf{else}\ bestAreas\ (init\ l)) \end{cases}$$

$\equiv$ { definição pointwise, 72, 73 }

$$\begin{cases} bestAreas \cdot nil = nil \\ bestAreas \cdot id = cons \cdot \langle area, cond\ (\widehat{(<)} \cdot \langle head, last\rangle)\ (bestAreas \cdot tail)\ (bestAreas \cdot init)\rangle \end{cases}$$

$\equiv$ { 1ª Lei de fusão do condicional }

$$\begin{cases} bestAreas \cdot nil = nil \\ bestAreas \cdot id = cons \cdot \langle area, bestAreas \cdot (cond\ (\widehat{(<)} \cdot \langle head, last\rangle)\ tail\ init)\rangle \end{cases}$$

$\equiv$ { Eq+, Fusão+ }

$$bestAreas \cdot [nil, id] = [nil, cons \cdot \langle area, bestAreas \cdot (cond\ (\widehat{(<)} \cdot \langle head, last\rangle)\ tail\ init)\rangle]$$

$\equiv$ { definição do isomorfimo $inid = [nil, id]$ e Absorção-+ }

$$
\begin{array}{ll}
& \mathit{bestAreas} \cdot \mathit{inid} = [\mathit{nil}, \mathit{cons}] \cdot (\mathit{id} + \langle \mathit{area}, \mathit{bestAreas} \cdot (\mathit{cond}\ (\widehat{(<)} \cdot \langle \mathit{head}, \mathit{last} \rangle)\ \mathit{tail}\ \mathit{init}) \rangle) \\
\equiv & \quad \{ \text{ isomorfimo } \textit{inid/outid} \text{ e definição do isomorfimo } \mathbf{in} = [\mathit{nil}, \mathit{cons}] \ \} \\
& \mathit{bestAreas} = \mathbf{in} \cdot (\mathit{id} + \langle \mathit{area}, \mathit{bestAreas} \cdot (\mathit{cond}\ (\widehat{(<)} \cdot \langle \mathit{head}, \mathit{last} \rangle)\ \mathit{tail}\ \mathit{init}) \rangle) \cdot \mathit{outid} \\
\equiv & \quad \{ \text{ Absorção-x, funtor-+ } \} \\
& \mathit{bestAreas} = \mathbf{in} \cdot (\mathit{id} + (\mathit{id} \times \mathit{bestAreas})) \cdot (\mathit{id} + \langle \mathit{area}, \mathit{cond}\ (\widehat{(<)} \cdot \langle \mathit{head}, \mathit{last} \rangle)\ \mathit{tail}\ \mathit{init} \rangle) \cdot \mathit{outid} \\
\equiv & \quad \{ \text{ Universal-ana } \} \\
& \mathit{bestAreas} = [\!(\ (\mathit{id} + \langle \mathit{area}, \mathit{cond}\ (\widehat{(<)} \cdot \langle \mathit{head}, \mathit{last} \rangle)\ \mathit{tail}\ \mathit{init} \rangle) \cdot \mathit{outid}\ )\!] \\
\square &
\end{array}
$$

Portanto, o *divide* desse anamorfismo é $(\mathit{id} + \langle \mathit{area}, \mathit{cond}\ (\widehat{(<)} \cdot \langle \mathit{head}, \mathit{last} \rangle)\ \mathit{tail}\ \mathit{init} \rangle) \cdot \mathit{out}$. Resultando assim no diagrama do anamorfismo

$$
\begin{tikzcd}
\mathbb{N}_0{}^* \arrow[r, "\mathit{divide}"'] \arrow[d, "\mathit{bestAreas} = [\!(\mathit{divide})\!]"'] & 1 + \mathbb{N}_0 \times \mathbb{N}_0{}^* \arrow[d, "\mathit{id} + \mathit{id} \times \mathit{bestAreas}"] \\
\mathbb{N}_0{}^* & 1 + \mathbb{N}_0 \times \mathbb{N}_0{}^* \arrow[l, "\mathbf{in}"]
\end{tikzcd}
$$

Além disso, já vimos nas aulas que o catamorfismo de listas

$$\mathit{maxList} = (\!|\ [\mathit{zero}, \widehat{\mathit{max}}]\ |\!)$$

em que $\mathit{conquer} = [\mathit{zero}, \widehat{\mathit{max}}]$, é responsável por obter o maior valor numa lista de números não negativos, podendo ser representado pelo seguinte diagrama

$$
\begin{tikzcd}
\mathbb{N}_0{}^* \arrow[r, "\mathit{out}"'] \arrow[d, "\mathit{maxList} = (\!| \mathit{conquer} |\!)"'] & 1 + \mathbb{N}_0 \times \mathbb{N}_0{}^* \arrow[d, "\mathit{id} + (\mathit{id} \times \mathit{maxList})"] \\
\mathbb{N}_0 & 1 + \mathbb{N}_0 \times \mathbb{N}_0 \arrow[l, "\mathit{conquer}"]
\end{tikzcd}
$$

Para evitar conflito entre os tipos *Int* e *Integer*, modificamos o conquer para

$$\mathit{conquer} = [\mathit{zero}, \widehat{\mathit{max}} \cdot (\mathit{fromIntegral} \times \mathit{fromIntegral})]$$

Portanto, temos

$$
\begin{array}{l}
\mathit{mostwater} = [\![\ \mathit{conquer},\ \mathit{divide}\ ]\!] \\
\mathit{divide} = (\mathit{id} + \langle \mathit{area}, \mathit{cond}\ (\widehat{(<)} \cdot \langle \mathit{head}, \mathit{last} \rangle)\ \mathit{tail}\ \mathit{init} \rangle) \cdot \mathit{outid} \\
\mathit{conquer} = [\mathit{zero}, \widehat{\mathit{max}} \cdot (\mathit{fromIntegral} \times \mathit{fromIntegral})] \\
\mathit{area}\ l = (\mathit{min}\ (\mathit{head}\ l)\ (\mathit{last}\ l)) * ((\mathit{length}\ l) - 1) \\
\mathit{outid}\ [\,] = i_1\ () \\
\mathit{outid}\ (h : t) = i_2\ (h : t)
\end{array}
$$

Com diagrama final

$$
\begin{tikzcd}
{\mathbb{N}_0}^* \arrow[r, "outid"'] \arrow[d, "{[\!(\, divide \,)\!]}"'] & 1+(\mathbb{N}_0\times{\mathbb{N}_0}^*) \arrow[d, "id+(id\times divide)"] \\
{\mathbb{N}_0}^* \arrow[d, "{(\!|\, conquer \,|\!)}"'] & 1+(\mathbb{N}_0\times{\mathbb{N}_0}^*) \arrow[l, "\mathbf{in}"] \arrow[d, "id+id\times conquer"] \\
\mathbb{N}_0 & 1+\mathbb{N}_0\times\mathbb{N}_0 \arrow[l, "conquer"]
\end{tikzcd}
$$

Problema 2

Para este problema, iremos primeiro definir o mapAccumR, mapAccumL e o filter com catamorfismos.

$myMapAccumR :: ((a, s) \rightarrow (c, s)) \rightarrow ([a], s) \rightarrow ([c], s)$

e seja:

$$
\begin{array}{l}
outListAcc\ ([\,], s) = i_1\ ((), s) \\
outListAcc\ ((a : x), s) = i_2\ (a, (x, s)) \\
(\!|\, g \,|\!) = g \cdot recList\ (\!|\, g \,|\!) \cdot outListAcc
\end{array}
$$

O que resulta neste diagrama:

$$
\begin{tikzcd}
A^*\times S \arrow[r, "outListAcc"'] \arrow[d, "myMapAccumR\ f"'] & (1\times S)+A\times(A^*\times S) \arrow[d, "id+id\times(myMapAccumR\ f)"] \\
C^*\times S & (1\times S)+A\times(C^*\times S) \arrow[l, "g"]
\end{tikzcd}
$$

Falta apenas definir o gene deste catamorfismo.

myMapAccumR f = $(\!|\, [myMapAccumR1, myMapAccumR2\ f] \,|\!)$

Seja o diagrama do $myMapAccumR1$:

$$
1\times S \xrightarrow{myMapAccumR1} C^*\times S
$$

O caso base do $mapAccumR\ f\ ([\,], n) = ([\,], n)$, logo

$$
myMapAccumR1 = nil \times id
$$

Para o $myMapAccumR2\ f$, iremos resolver passo a passo como está no diagrama a seguir

$$
\begin{tikzcd}
 & A\times(C^*\times S) \arrow[dl, "id\times\pi_2"'] \arrow[dd, "\langle f\cdot(id\times\pi_2),\pi_1\cdot\pi_2\rangle"'] \arrow[ddr, "\pi_1\cdot\pi_2"] & \\
A\times S \arrow[d, "f"'] & & \\
C\times S & (C\times S)\times C^* \arrow[l, "\pi_1"] \arrow[r, "\pi_2"'] \arrow[d, "zed=\langle\langle\pi_1\cdot\pi_1,\pi_2\rangle,\pi_2\cdot\pi_1\rangle"'] & C^* \\
 & (C\times C^*)\times S \arrow[d, "cons\times id"'] & \\
 & C^*\times S &
\end{tikzcd}
$$

assim:

$$\mathit{zed} = \langle \langle \pi_1 \cdot \pi_1, \pi_2 \rangle, \pi_2 \cdot \pi_1 \rangle$$
$$\mathit{myMapAccumR2}\ f = (\mathit{cons} \times \mathit{id}) \cdot \mathit{zed} \cdot \langle f \cdot (\mathit{id} \times \pi_2), \pi_1 \cdot \pi_2 \rangle$$

e tambem podemos definir *zed* como $\mathit{zed} = \mathit{assocl} \cdot (\mathit{id} \times \mathit{swap}) \cdot \mathit{assocr}$ e $\langle f \cdot (\mathit{id} \times \pi_2), \pi_1 \cdot \pi_2 \rangle$ como $(f \times \mathit{id}) \cdot \mathit{assocl} \cdot (\mathit{id} \times \mathit{swap})$

Com isto, resulta:

$$\mathit{myMapAccumR}\ f = (\!|\, [\mathit{myMapAccumR1}, \mathit{myMapAccumR2}\ f]\, |\!)$$

E fazer o mapAccumL é análogo, trocando o funtor assim:

$$\mathit{outListAcc}'\ ([\,], s) = i_1\ ((), s)$$
$$\mathit{outListAcc}'\ (x, s) = i_2\ (\mathit{last}\ x, (\mathit{init}\ x, s))$$
$$(\!|\, g\, |\!) = g \cdot \mathit{recList}\ (\!|\, g\, |\!) \cdot \mathit{outListAcc}'$$
$$\mathit{addToLast} = (\mathit{flip}\ (+\!\!+\widehat{) \cdot (\mathit{sin}}\mathit{gleton}))$$
$$\mathit{myMapAccumL1} = \mathit{nil} \times \mathit{id}$$
$$\mathit{myMapAccumL2}\ f = (\mathit{addToLast} \times \mathit{id}) \cdot \mathit{zed} \cdot \langle f \cdot (\mathit{id} \times \pi_2), \pi_1 \cdot \pi_2 \rangle$$
$$\mathit{myMapAccumL}\ f = (\!|\, [\mathit{myMapAccumL1}, \mathit{myMapAccumL2}\ f]\, |\!)$$

como estamos começar pelo fim, então também temos de começar a adicionar os elementos pelo fim.

E segue-se o diagrama do *filter* e o seu catamorfismo:

$$\begin{tikzcd}
A^* \arrow[r, "\mathit{outList}"] \arrow[d, "\mathit{myfilter}\ p"'] & 1 + A \times A^* \arrow[d, "\mathit{id} + (\mathit{myfilter}\ p)"] \\
A^* & 1 + A \times A^* \arrow[l, "{[\mathit{nil}, \mathit{cond}\ (p \cdot \pi_1)\ \mathit{cons}\ \pi_2]}"]
\end{tikzcd}$$

$$\mathit{myfilter}\ p = (\!|\, [\mathit{nil}, \mathit{cond}\ (p \cdot \pi_1)\ \mathit{cons}\ \pi_2]\, |\!)$$

Agora podemos definir *mapAccumRfilter* e *mapAccumLfilter* inspirado nas as funções anteriores

$$\begin{tikzcd}
A^* \times S \arrow[r, "outListAcc"'] \arrow[d, "mapAccumRfilter\ p\ f"'] & (1 \times S) + A \times (A^* \times S) \arrow[d, "id + id \times (mapAccumRfilter\ p\ f)"] \\
C^* \times S & (1 \times S) + A \times (C^* \times S) \arrow[l, "g"]
\end{tikzcd}$$

$mapAccumRfilter\ p\ f = (\![[mapAccumRfilter1, mapAccumRfilter2\ p\ f]]\!)$

$$mapAccumRfilter1 = nil \times id$$

e se detalhar mais o diagrama do filter $cond\ p\ f\ g = [f, g] \cdot (grd\ p)$:

$$\begin{tikzcd}
1 \arrow[r, "i_1"] \arrow[dr, "nil"'] & 1 + A \times A^* \arrow[d, "{[nil,[cons,\pi_2] \cdot (grd\ (p \cdot \pi_1))]}"] & A \times A^* \arrow[l, "i_2"'] \arrow[d, "grd\ (p \cdot \pi_1)"] \\
& A^* & A \times A^* + A \times A^* \arrow[l, "{[cons,\pi_2]}"]
\end{tikzcd}$$

Podemos ver que a estrutura recursiva do filter é $A \times A^* + A \times A^*$, e se aplicarmos esta estrutura no nosso diagrama do mapAccumR2 obtemos o mapAccumRfilter2:

$$\begin{tikzcd}
A \times (C^* \times S) \arrow[d, "id \times swap"'] \\
A \times (S \times C^*) \arrow[d, "assocl"'] \\
(A \times S) \times C^* \arrow[d, "grd\ (p \cdot \pi_1)"'] \\
((A \times S) \times C^*) + ((A \times S) \times C^*) \arrow[d, "(f \times id) + (f \times id)"'] \\
((C \times S) \times C^*) + ((C \times S) \times C^*) \arrow[d, "{[(cons \times id) \cdot zed, swap \cdot (\pi_2 \times id)]}"'] \\
C^* \times S
\end{tikzcd}$$

o que resulta

$$\begin{array}{l}
mapAccumRfilter2\ p\ f = \\
\quad [(cons \times id) \cdot zed, swap \cdot (\pi_2 \times id)] \cdot ((f \times id) + (f \times id)) \cdot (grd\ (p \cdot \pi_1)) \cdot assocl \cdot (id \times swap) \\
mapAccumRfilter\ p\ f = (\![[mapAccumRfilter1, mapAccumRfilter2\ p\ f]]\!)
\end{array}$$

e podemos ver que, $[(cons \times id) \cdot zed, swap \cdot (\pi_2 \times id)]$ e $[cons, \pi_2]$ são similares.

Análogamente podemos fazer o *mapAccumLfilter*, com o mesmo funtor do *myMapAccumL*

$$\begin{array}{l}
mapAccumLfilter1 = nil \times id \\
mapAccumLfilter2\ p\ f = \\
\quad [(addToLast \times id) \cdot zed, swap \cdot (\pi_2 \times id)] \cdot ((f \times id) + (f \times id)) \cdot (grd\ (p \cdot \pi_1)) \cdot assocl \cdot (id \times swap) \\
mapAccumLfilter\ p\ f = (\![[mapAccumLfilter1, mapAccumLfilter2\ p\ f]]\!)
\end{array}$$

Problema 3

Reparemos que

$$\pi_n = \sum_{i=0}^{n} \frac{(i!)^2 2^{i+1}}{(2i+1)!} = 2\sum_{i=0}^{n} \frac{(i!)\times(i!)2^i}{(2i+1)!} = 2\sum_{i=0}^{n} \frac{i!\times((\cancel{2i})\times\cancel{(2(i-1))}\times\cdots\times 2\cdot 2\times 2\cdot 1)}{(2i+1)\times(\cancel{2i})\times(2i-1)\times\cancel{(2(i-1))}\times\cdots\times 2\times 1} =$$

$$= 2\sum_{i=0}^{n} \frac{i!}{(2i+1)!!}$$

onde $n!!$ é o fatorial duplo.

Seja $f(n) = \sum_{i=0}^{n} \frac{i!}{(2i+1)!!} = 1 + \sum_{i=0}^{n-1} \frac{(i+1)!}{(2i+3)!!}$, $g(n) = \frac{(n+1)!}{(2n+3)!!}$ e $h(n) = \frac{n+2}{2n+5}$. É fácil reparar que

$$f(n) = 1 + \sum_{i=0}^{n-1} \frac{(i+1)!}{(2i+3)!!} = 1 + \sum_{i=0}^{n-1} g(i) \implies \begin{cases} f(0) = 1 \\ f(n+1) = f(n) + g(n) \end{cases}$$

$$g(n) = \frac{(i+1)!}{(2i+3)!!} = \frac{1}{3}\times\prod_{n=0}^{n-1} h(i) \implies \begin{cases} g(0) = \frac{1}{3} \\ g(n+1) = g(n)\times h(n) \end{cases}$$

e

$$h(n) = \frac{n+2}{2n+5} \implies \begin{cases} h(0) = \frac{2}{5} \\ h(n+1) = \frac{n+3}{2n+7} = \frac{\frac{n+2}{2n+5}-1}{4\frac{n+2}{2n+5}-3} = \frac{h(n)-1}{4h(n)-3} \end{cases}$$

Ao escrever as funções na forma *point-free* e recorrendo ás regras de cálculo usuais, tem-se

$$\begin{aligned}
&f \cdot \mathsf{in} = [\underline{1}, add] \cdot (1 + \langle f, g\rangle) \\
&g \cdot \mathsf{in} = [\underline{1\ /\ 3}, mul] \cdot (1 + \langle g, h\rangle) \\
&h \cdot \mathsf{in} = [\underline{2\ /\ 5}, calc] \cdot (1 + h)\ \mathbf{where} \\
&\quad calc\ n = (n-1)\ /\ (4 * n - 3) \\
&\quad add\ (x, y) = x + y \\
&\quad mul\ (x, y) = x * y
\end{aligned}$$

Em que $\mathbf{in} = [\underline{0}, suc]$.

Recorrendo à lei de absorção $+$ e $\times$ e com auxilío da função assocl, temos

$$\begin{aligned}
&f \cdot \mathsf{in} = [\underline{1}, add \cdot \pi_1 \cdot assocl] \cdot (1 + \langle f, \langle g, h\rangle\rangle) \\
&g \cdot \mathsf{in} = [\underline{1\ /\ 3}, mul \cdot \pi_2] \cdot (1 + \langle f, \langle g, h\rangle\rangle) \\
&h \cdot \mathsf{in} = [\underline{2\ /\ 5}, calc \cdot \pi_2 \cdot \pi_2] \cdot (1 + \langle f, \langle g, h\rangle\rangle)\ \mathbf{where} \\
&\quad calc\ n = (n-1)\ /\ (4 * n - 3) \\
&\quad add\ (x, y) = x + y \\
&\quad mul\ (x, y) = x * y
\end{aligned}$$

Podemos unir as equações numa só através do Eq-$\times$ e Fusão-$\times$. Portanto, unindo as duas últimas equações e em seguida aplicando a lei da troca, temos

$$
\begin{array}{l}
f \cdot \mathsf{in} = [\underline{1}, add \cdot \pi_1 \cdot assocl] \cdot (1 + \langle f, \langle g, h \rangle \rangle) \\
\langle g, h \rangle \cdot \mathsf{in} = [\langle \underline{1 / 3}, \underline{2 / 5} \rangle, \langle mul \cdot \pi_2, calc \cdot \pi_2 \cdot \pi_2 \rangle] \cdot (1 + \langle f, \langle g, h \rangle \rangle) \ \mathbf{where} \\
\quad calc\ n = (n - 1) / (4 * n - 3) \\
\quad add\ (x, y) = x + y \\
\quad mul\ (x, y) = x * y
\end{array}
$$

Repetindo o mesmo raciocínio de forma análoga, tem-se por fim

$$
\begin{array}{l}
\langle f, \langle g, h \rangle \rangle \cdot \mathsf{in} = [\underline{(1, (1 / 3, 2 / 5))}, \langle add \cdot \pi_1 \cdot assocl, \langle mul \cdot \pi_2, calc \cdot \pi_2 \cdot \pi_2 \rangle \rangle] \cdot (1 + \langle f, \langle g, t \rangle \rangle) \ \mathbf{where} \\
\quad calc\ n = (n - 1) / (4 * n - 3) \\
\quad add\ (x, y) = x + y \\
\quad mul\ (x, y) = x * y
\end{array}
$$

Ora, pela lei de universal-cata, tem-se que a função $\langle f, \langle g, h \rangle \rangle$ é da forma for *loop inic*. Portanto, se $worker = \langle f, \langle g, h \rangle \rangle$, tem-se

$$
\begin{array}{l}
worker = \mathsf{for}\ loop\ inic \\
loop = \langle add \cdot \pi_1 \cdot assocl, \langle mul \cdot \pi_2, calc \cdot \pi_2 \cdot \pi_2 \rangle \rangle \\
inic = \underline{(1, (1 / 3, 2 / 5))} \ \mathbf{where} \\
\quad calc\ n = (n - 1) / (4 * n - 3) \\
\quad add\ (x, y) = x + y \\
\quad mul\ (x, y) = x * y
\end{array}
$$

Finalmente, como $\pi_{loop} = (2*) \cdot \pi_1 \cdot \langle f, \langle g, h \rangle \rangle$, temos

$$
\begin{array}{l}
\pi_{loop} = wrapper \cdot worker \\
worker = \mathsf{for}\ loop\ inic \\
loop = \langle add \cdot \pi_1 \cdot assocl, \langle mul \cdot \pi_2, calc \cdot \pi_2 \cdot \pi_2 \rangle \rangle \ \mathbf{where} \\
\quad calc\ n = (n - 1) / (4 * n - 3) \\
\quad add\ (x, y) = x + y \\
\quad mul\ (x, y) = x * y \\
inic = \underline{(1, (1 / 3, 2 / 5))} \\
wrapper = (2*) \cdot \pi_1
\end{array}
$$

Problema 4

Para fazer o funtor, vamos explorar melhor o in e o out do Vec.

$V :: [(a, Int)] \rightarrow Vec\ a$

$outV :: Vec \rightarrow [(a, Int)]$

Como o $outV$ e V usam lista como input e output, podemos usar funções de listas para auxiliar nas nossas funções de Vec.

Functor:

$\mathsf{fmap} :: (a \rightarrow b) \rightarrow Vec\ a \rightarrow Vec\ b$

Utilizando o $outV$ e o map , podemos definir o seguinte diagrama:

$$\begin{array}{c} Vec\ A \\ \downarrow{\scriptstyle outV} \\ (A \times Int)^* \\ \downarrow{\scriptstyle \mathsf{map}\ (f \times id)} \\ (B \times Int)^* \\ \downarrow{\scriptstyle V} \\ Vec\ B \end{array}$$

O que nos permite definir o fmap assim:

$$\begin{array}{l} \textbf{instance}\ Functor\ Vec\ \textbf{where} \\ \quad \mathsf{fmap}\ f = V \cdot (\mathsf{map}\ \ (f \times id)) \cdot outV \end{array}$$

Monad:

Para o monad, vamos definir o μ (*miu*) e o υ (*return*) para facilitar na definição de outras funções

$return :: a \rightarrow Vec\ a$

$return\ a = V\ [(a, 1)]$

para qualquer a, fazemos um *singleton*, associado com 1, porque é o elemento neutro da multiplicação.

$miu :: Vec\ (Vec\ a) \rightarrow Vec\ a$

ou seja

$$\begin{array}{c} Vec\ (Vec\ A) \\ \downarrow{\scriptstyle outV} \\ (Vec\ A \times Int)^* \\ \downarrow{\scriptstyle \mathsf{map}\ (outV \cdot \widehat{mulV})} \\ (A \times Int)^{**} \\ \downarrow{\scriptstyle concat} \\ A^* \\ \downarrow{\scriptstyle V} \\ Vec\ A \end{array}$$

sendo o $mulV$ o produto escalar de vetores.

$$\begin{array}{l} mulV :: Vec\ a \rightarrow Int \rightarrow Vec\ a \\ mulV\ v\ x = V\ (\mathsf{map}\ \ (id \times (x*))\ (outV\ v)) \end{array}$$

e assim definimos:

$$miu = V \cdot concat \cdot \mathsf{map}\ \ (outV \cdot \widehat{mulV}) \cdot outV$$

falta apenas definir $(\gg\!=) :: Vec\ a \rightarrow (a \rightarrow Vec\ b) \rightarrow Vec\ b$, com o *miu* e fmap definido, fica simples definir:

$$
\begin{tikzcd}
Vec\ A \arrow[d, "\mathsf{fmap}\ f"'] \\
Vec\ (Vec\ B) \arrow[d, "miu"'] \\
Vec\ B
\end{tikzcd}
$$

E assim concluimos que

instance *Monad Vec* **where**
  $x \gg\!= f = miu\ (\mathsf{fmap}\ f\ x)$
  $return\ a = V\ [(a, 0)]$