



Universidade do Minho
Escola de Engenharia

Cálculo de Programas

Trabalho Prático (2024/25)

Lic. em Ciências da Computação

Grupo G99

xxxxxxx	Nome
xxxxxxx	Nome
xxxxxxx	Nome

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 (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 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 seleccionadas 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.

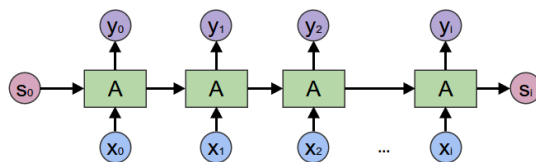


Figure 1: [RNN](#) vista como instância de um *accumulating map* [3].

Problema 2

Um dos problemas prementes da Computação na actualidade é conseguir, por engenharia reversa, interpretar as redes neurais ([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 [3]. 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' [1]. Há contudo padrões de [RNNs](#) mais gerais: por exemplo, o padrão 'many-to-one' que se mostra na figura 2 extraída de [1].

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

$$\text{mapAccumRfilter} :: ((a, s) \rightarrow \text{Bool}) \rightarrow ((a, s) \rightarrow (c, s)) \rightarrow ([a], s) \rightarrow ([c], s)$$

$$\text{mapAccumLfilter} :: ((a, s) \rightarrow \text{Bool}) \rightarrow ((a, s) \rightarrow (c, s)) \rightarrow ([a], s) \rightarrow ([c], s)$$

Pretendem-se as implementações 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 E que, inspirado em [1], recorre à biblioteca [Data.Matrix](#).

Problema 3

A fornecer na segunda edição deste enunciado

Problema 4

A fornecer na segunda edição deste enunciado

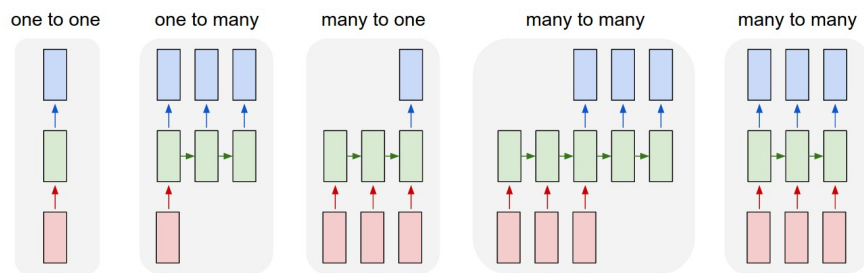


Figure 2: Várias tipologias de RNNs [1].

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.

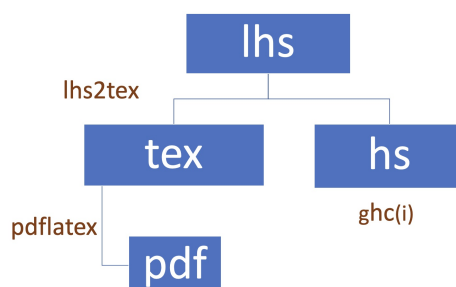
Para cumprir de forma integrada os objectivos do trabalho vamos recorrer a uma técnica de programação dita “[literária](#)” [2], cujo princípio base é o seguinte:

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:



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

¹ O sufixo ‘lhs’ quer dizer *literate Haskell*.

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 [L^AT_EX](#). (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 [L^AT_EX](#). 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-processor que faz “pretty printing” de código [Haskell](#) em [L^AT_EX](#) 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 [F](#) 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 [Bib_TE_X](#)) 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 [E](#) 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 ler¹ onde se obtém o efeito seguinte:²

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

Os diagramas podem ser produzidos recorrendo à *package* [xymatrix](#), por exemplo:

$$\begin{array}{ccc}
 \mathbb{N}_0 & \xleftarrow{\text{in}} & 1 + \mathbb{N}_0 \\
 \downarrow \langle g \rangle & & \downarrow id + \langle g \rangle \\
 B & \xleftarrow{g} & 1 + B
 \end{array}$$

E Código fornecido

Teste relativo à figura da página [1](#):

test₁ = mostwater hghts

Problema 2

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

test_{2a} = mapAccumLfilter ((>10) · π₁) f (odds 12, 0)
test_{2b} = mapAccumRfilter ((>10) · π₁) f (odds 12, 0)

¹ Procure e.g. por "sec:diagramas".

² Exemplos tirados de [\[4\]](#).

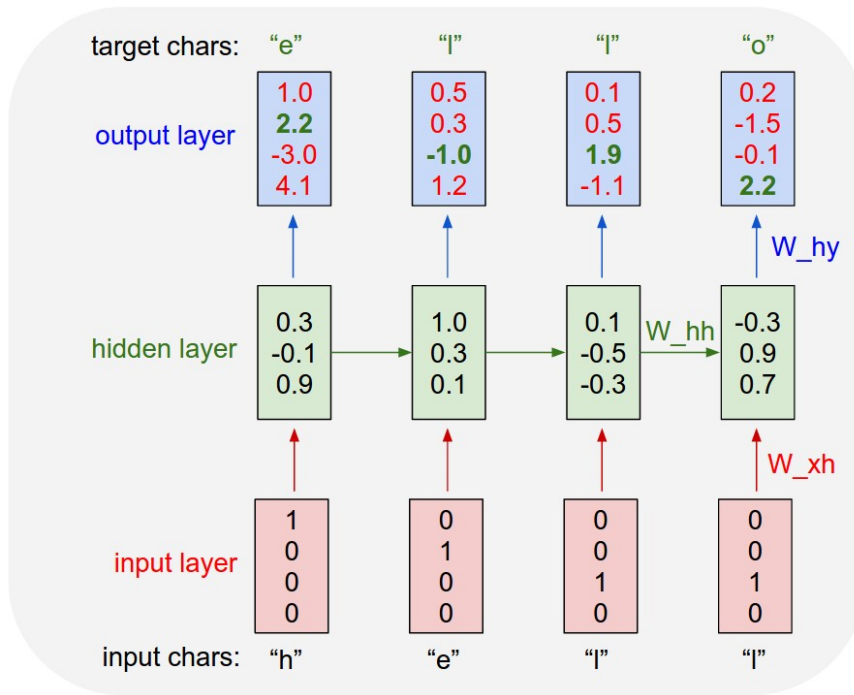


Figure 3: Exemplo *char seq* extraído de [1].

onde:

$$\text{odds } n = \text{map } ((1+) \cdot (2*)) [0..n-1]$$

$$f(a, s) = (s, a + s)$$

Teste

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

baseado no exemplo de Karpathy [1] que a figura 3 mostra, usando os dados seguintes:

- Estado inicial:

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

- Step function:

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

- Função de activação:

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

- Input layer:

$$\text{inp} = [x_1, x_2, x_3, x_4]$$

$$x_1 = \text{fromList } 4 \ 1 \ [1.0, 0, 0, 0]$$

$$x_2 = \text{fromList } 4 \ 1 \ [0, 1.0, 0, 0]$$

$$x_3 = \text{fromList } 4 \ 1 \ [0, 0, 1.0, 0]$$

$$x_4 = x_3$$

- 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.

F 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

mostwater = \perp

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:

```
outListAcc ([], s) = i1 ((), s)
outListAcc ((a : x), s) = i2 (a, (x, s))
⟦g⟧ = g · recList ⟦g⟧ · outListAcc
```

O que resulta neste diagrama:

$$\begin{array}{ccc} A^* \times S & \xrightarrow{\text{outListAcc}} & (1 \times S) + A \times (A^* \times S) \\ \text{myMapAccumR } f \downarrow & & \downarrow \text{id} + \text{id} \times (\text{myMapAccumR } f) \\ C^* \times S & \xleftarrow{g} & (1 \times S) + A \times (C^* \times S) \end{array}$$

Falta apenas definir o gene deste catamorfismo.

myMapAccumR f = $\llbracket [\text{myMapAccumR1}, \text{myMapAccumR2 } f] \rrbracket$

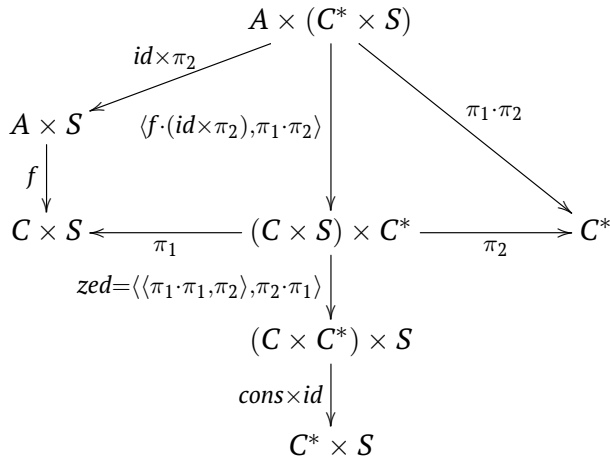
Seja o diagrama do *myMapAccumR1*:

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

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

myMapAccumR1 = *nil* × *id*

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



assim:

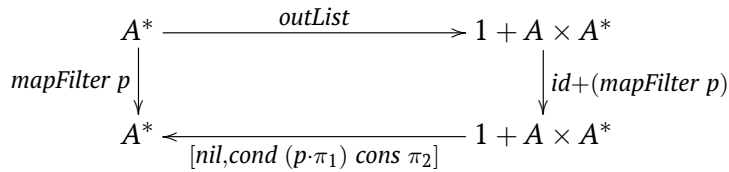
$$\begin{aligned}
 zed &= \langle \langle \pi_1 \cdot \pi_1, \pi_2 \rangle, \pi_2 \cdot \pi_1 \rangle \\
 myMapAccumR2 f &= (cons \times id) \cdot zed \cdot \langle f \cdot (id \times \pi_2), \pi_1 \cdot \pi_2 \rangle
 \end{aligned}$$

e tambem podemos definir zed como $zed = assocl \cdot (id \times swap) \cdot assocr$.

Com isto, resulta:

$$myMapAccumR f = ([myMapAccumR1, myMapAccumR2 f])$$

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



$$myfilter p = ([nil, cond (p \cdot \pi_1) cons \pi_2])$$

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

$$\begin{aligned}
 outListAcc' ([], s) &= i_1 ((), s) \\
 outListAcc' (x, s) &= i_2 (last x, (init x, s)) \\
 \langle g \rangle &= g \cdot recList \langle g \rangle \cdot outListAcc' \\
 addToLast &= (flip (++) \cdot \widehat{singleton}) \\
 myMapAccumL1 &= nil \times id \\
 myMapAccumL2 f &= (addToLast \times id) \cdot zed \cdot \langle f \cdot (id \times \pi_2), \pi_1 \cdot \pi_2 \rangle \\
 myMapAccumL f &= ([myMapAccumL1, myMapAccumL2 f])
 \end{aligned}$$

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

Agora podemos definir *filtermapAccumR* e *filtermapAccumL* inspirado com as funções anteriores

$$\begin{array}{ccc}
 A^* \times S & \xrightarrow{\text{outListAcc}} & (1 \times S) + A \times (A^* \times S) \\
 \text{mapAccumRfilter } f \, p \downarrow & & \downarrow \text{id} + \text{id} \times (\text{mapAccumRfilter } f \, p) \\
 C^* \times S & \xleftarrow{g} & (1 \times S) + A \times (C^* \times S)
 \end{array}$$

$$\text{mapAccumRfilter } p \, f = [\text{mapAccumRfilter1}, \text{mapAccumRfilter2 } p \, f]$$

$$\text{mapAccumRfilter1} = \text{nil} \times \text{id}$$

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

$$\begin{array}{ccccc}
 1 & \xrightarrow{i_1} & 1 + A \times A^* & \xleftarrow{i_2} & A \times A^* \\
 & \searrow \text{nil} & \downarrow [\text{nil}, [\text{cons}, \pi_2] \cdot (\text{grd } (p \cdot \pi_1))] & & \downarrow \text{grd } (p \cdot \pi_1) \\
 & & A^* & \xleftarrow{[\text{cons}, \pi_2]} & A \times A^* + A \times A^*
 \end{array}$$

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:

$$\begin{array}{c}
 A \times (C^* \times S) \\
 \downarrow \text{id} \times \text{swap} \\
 A \times (S \times C^*) \\
 \downarrow \text{assocl} \\
 (A \times S) \times C^* \\
 \downarrow \text{grd } (p \cdot \pi_1) \\
 ((A \times S) \times C^*) + ((A \times S) \times C^*) \\
 \downarrow (f \times \text{id}) + (f \times \text{id}) \\
 ((C \times S) \times C^*) + ((C \times S) \times C^*) \\
 \downarrow [(\text{cons} \times \text{id}) \cdot \text{zed}, \text{swap} \cdot (\pi_2 \times \text{id})] \\
 (C \times S) \times C^*
 \end{array}$$

o que resulta

$$\begin{aligned}
 \text{mapAccumRfilter2 } p \, f = & [(\text{cons} \times \text{id}) \cdot \text{zed}, \text{swap} \cdot (\pi_2 \times \text{id})] \cdot ((f \times \text{id}) + (f \times \text{id})) \cdot (\text{grd } (p \cdot \pi_1)) \cdot \text{assocl} \cdot (\text{id} \times \text{swap}) \\
 \text{mapAccumRfilter } p \, f = & \llbracket [\text{mapAccumRfilter1}, \text{mapAccumRfilter2 } p \, f] \rrbracket
 \end{aligned}$$

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

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

$$\text{mapAccumLfilter1} = \text{nil} \times \text{id}$$

$$\text{mapAccumLfilter2 } p \, f =$$

$$\begin{aligned}
 & [(\text{addToLast} \times \text{id}) \cdot \text{zed}, \text{swap} \cdot (\pi_2 \times \text{id})] \cdot ((f \times \text{id}) + (f \times \text{id})) \cdot (\text{grd } (p \cdot \pi_1)) \cdot \text{assocl} \cdot (\text{id} \times \text{swap}) \\
 \text{mapAccumLfilter } p \, f = & \llbracket [\text{mapAccumLfilter1}, \text{mapAccumLfilter2 } p \, f] \rrbracket
 \end{aligned}$$

Index

\LaTeX , [4](#)

bibtex, [4](#)

lhs2TeX, [3–5](#)

makeindex, [4](#)

pdflatex, [3](#)

xymatrix, [5](#)

Combinador “pointfree”

cata

 Naturais, [5](#)

either, [7–9](#)

mapAccumL, [2](#)

mapAccumR, [2](#)

split, [5, 8, 9](#)

Cálculo de Programas, [1, 3](#)

 Material Pedagógico, [3](#)

Docker, [4](#)

 container, [4, 5](#)

Função

π_1 , [5, 8, 9](#)

π_2 , [5, 8, 9](#)

map, [6, 8](#)

uncurry, [8, 9](#)

Haskell, [1–5](#)

 Hackage

 Data.Matrix, [2](#)

 interpretador

 GHCi, [3, 4](#)

 Literate Haskell, [3](#)

Números naturais (\mathbb{N}), [5](#)

Programação

 literária, [3, 5](#)

Rede neuronal, [2](#)

 RNN, [2, 3](#)

References

- [1] A. Karpathy. The unreasonable effectiveness of recurrent neural networks, 2015. Blog: <http://karpathy.github.io/2015/05/21/rnn-effectiveness>, last read: April 21, 2025.
- [2] D.E. Knuth. *Literate Programming*. CSLI Lecture Notes Number 27. Stanford University Center for the Study of Language and Information, Stanford, CA, USA, 1992.
- [3] C. Olah. Neural networks, types, and functional programming, 2015. Blog: <http://colah.github.io/posts/2015-09-NN-Types-FP/>, last read: April 21, 2025.
- [4] J.N. Oliveira. Program Design by Calculation, 2024. Draft of textbook in preparation. First version: 1998. Current version: Sep. 2024. Informatics Department, University of Minho ([pdf](#)).