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 (conjunto de leis universais e seus corolários) e usam-se esses combinadores para construir programas *composicionalmente*, isto é, agregando programas já existentes.

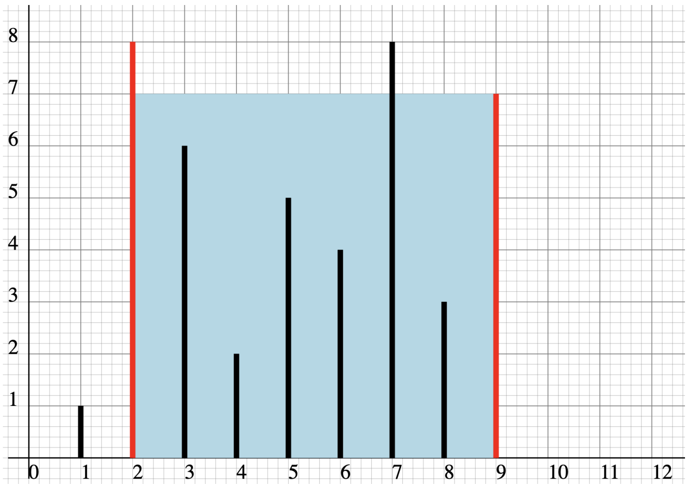
Na sequência pedagógica dos planos de estudo dos cursos que têm esta disciplina, opta-se pela aplicação deste método à programação em Haskell (sem prejuízo da sua aplicação a outras linguagens funcionais). Assim, o presente trabalho prático coloca os alunos perante problemas concretos que deverão ser 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 selecionadas delimitam um *container* com água.)

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

$$mostwater :: [Int] \to Int$$

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.

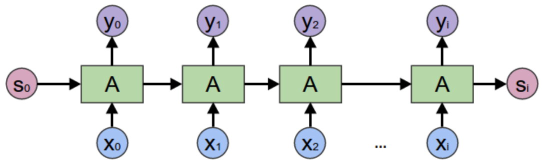


Figure 1: RNN vista como instância de um *accumulating map* [?].

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:

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

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 [?], recorre à biblioteca `Data.Matrix`.

Problema 3

A fornecer na segunda edição deste enunciado

Problema 4

A fornecer na segunda edição deste enunciado

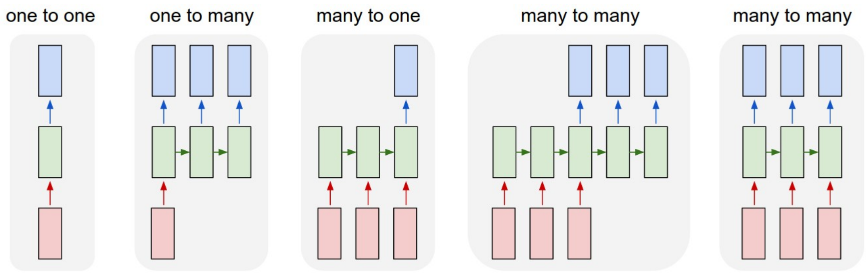


Figure 2: Várias tipologias de RNNs [?].

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.

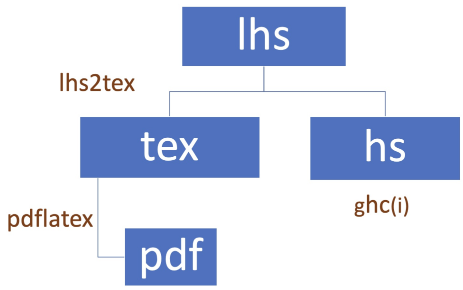
Para cumprir de forma integrada os objectivos do trabalho vamos recorrer a uma técnica de programação dita "literária" [?], cujo princípio base é o seguinte:

> *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:



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 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 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 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 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 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 Código fornecido

Teste relativo à figura da página 1:

$$test_1 = mostwater\ hghts$$

Problema 2

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

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

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

[2] Exemplos tirados de [?].

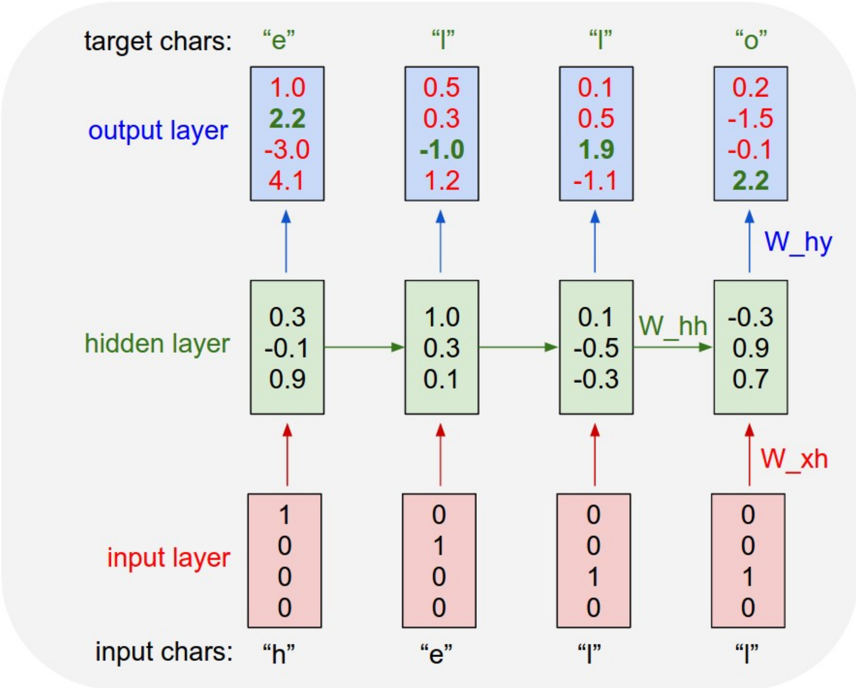


Figure 3: Exemplo *char seq* extraído de [?].

onde:

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

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:

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

- *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.

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

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

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

Tornando esta definição point-free, temos

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

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

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

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

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

$\equiv \quad \{$ Eq+, Fusão+ $\}$

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

$\equiv \quad \{$ definição do isomorfimo $\mathsf{in}_{id} = [nil, id]$ e Absorção-+ $\}$

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

$\equiv \quad \{$ isomorfimo $\mathsf{in}_{id}/\mathsf{out}_{id}$ e definição do isomorfimo $\mathbf{in} = [nil, cons]$ $\}$

$$bestAreas = \mathbf{in} \cdot (id + \langle area, bestAreas \cdot (cond\ (\widehat{(<)} \cdot \langle head, last \rangle)\ tail\ init) \rangle) \cdot \mathsf{out}_{id}$$

$\equiv \quad \{$ Absorção-x, funtor-+ $\}$

$$bestAreas = \mathbf{in} \cdot (id + (id \times bestAreas)) \cdot (id + \langle area, cond\ (\widehat{(<)} \cdot \langle head, last \rangle)\ tail\ init \rangle) \cdot \mathsf{out}_{id}$$

$\equiv \quad \{$ Universal-ana $\}$

$$bestAreas = [\!(\ (id + \langle area, cond\ (\widehat{(<)} \cdot \langle head, last \rangle)\ tail\ init \rangle) \cdot \mathsf{out}_{id}\ )\!]$$

$\square$

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

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

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

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

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

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

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

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

Portanto, temos

$$mostwater = [\![\ conquer,\ divide\ ]\!]$$
$$divide = (id + \langle area, cond\ (\widehat{(<)} \cdot \langle head, last \rangle)\ tail\ init \rangle) \cdot \mathsf{out}_{id}$$

$$
\begin{aligned}
&conquer = [zero, \widehat{max} \cdot (fromIntegral \times fromIntegral)]\\
&area\ l = (min\ (head\ l)\ (last\ l)) * ((length\ l) - 1)\\
&\mathsf{out}_{id}\ [\,] = i_1\ ()\\
&\mathsf{out}_{id}\ (h : t) = i_2\ (h : t)
\end{aligned}
$$

Com diagrama final

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

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:

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

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:

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

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

Com isto, resulta:

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

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

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

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

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

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

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

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*

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