Universidade do Minho
Escola de Engenharia

Cálculo de Programas

Trabalho Prático (2024/25)

Lic. em Engenharia Informática

Grupo G05

a103993 Júlia Bughi Corrêa da Costa
a104171 Gabriel Pereira Ribeiro
a104613 Luís Pinto da Cunha

janeiro 2025

Preâmbulo

Em 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

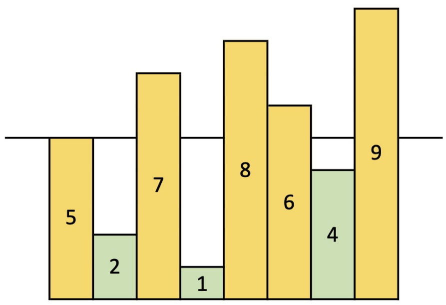
Esta questão aborda um problema que é conhecido pela designação '*H-index of a Histogram*' e que se formula facilmente:

> *O h-index de um histograma é o maior número n de barras do histograma cuja altura é maior ou igual a n.*

Por exemplo, o histograma

$$h = [5, 2, 7, 1, 8, 6, 4, 9]$$

que se mostra na figura



tem *hindex* $h = 5$ pois há 5 colunas maiores que 5. (Não é 6 pois maiores ou iguais que seis só há quatro.)

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

$$hindex :: [Int] \rightarrow (Int, [Int])$$

tal que, para $(i, x) = hindex\ h$, i é o H-index de h e x é a lista de colunas de h que para ele contribuem.

A proposta de *hindex* deverá vir acompanhada de um **diagrama** ilustrativo.

Problema 2

Pelo teorema fundamental da aritmética, todo número inteiro positivo tem uma única factorização prima. For exemplo,

```
primes 455
[5,7,13]
primes 433
[433]
primes 230
[2,5,23]
```

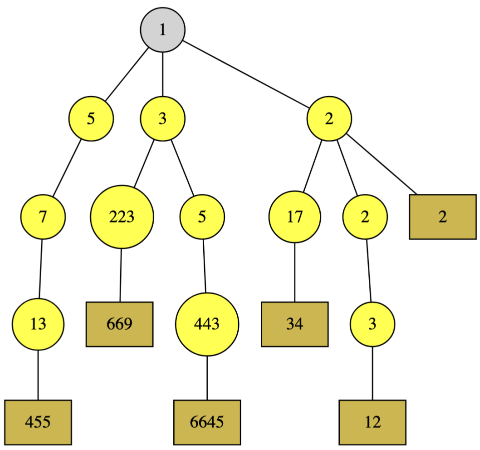
1. Implemente como anamorfismo de listas a função

$$primes :: \mathbb{Z} \to [\mathbb{Z}]$$

que deverá, recebendo um número inteiro positivo, devolver a respectiva lista de factores primos.

A proposta de *primes* deverá vir acompanhada de um **diagrama** ilustrativo.

2. A figura mostra a "*árvore dos primos*" dos números $[455, 669, 6645, 34, 12, 2]$.



Com base na alínea anterior, implemente uma função em Haskell que faça a geração de uma tal árvore a partir de uma lista de inteiros:

$$prime\_tree :: [\mathbb{Z}] \to Exp\ \mathbb{Z}\ \mathbb{Z}$$

Sugestão: escreva o mínimo de código possível em *prime_tree* investigando cuidadosamente que funções disponíveis nas bibliotecas que são dadas podem ser reutilizadas.[1]

Problema 3

A convolução $a \star b$ de duas listas a e b — uma operação relevante em computação — está muito bem explicada neste vídeo do canal **3Blue1Brown** do YouTube, a partir de $t = 6:30$. Aí se mostra como, por exemplo:

[1] Pense sempre na sua produtividade quando está a programar — essa atitude será valorizada por qualquer empregador que vier a ter.

$[1,2,3] \star [4,5,6] = [4,13,28,27,18]$

A solução abaixo, proposta pelo chatGPT,

$$
\begin{array}{l}
\mathit{convolve} :: \mathit{Num}\ a \Rightarrow [a] \rightarrow [a] \rightarrow [a] \\
\mathit{convolve}\ \mathit{xs}\ \mathit{ys} = [\mathit{sum}\ \$\ \mathit{zipWith}\ (*)\ (\mathit{take}\ n\ (\mathit{drop}\ i\ \mathit{xs}))\ \mathit{ys} \mid i \leftarrow [0\mathinner{..}(\mathit{length}\ \mathit{xs} - n)]] \\
\quad \textbf{where}\ n = \mathit{length}\ \mathit{ys}
\end{array}
$$

está manifestamente errada, pois $\mathit{convolve}\ [1,2,3]\ [4,5,6] = [32]$ (!).

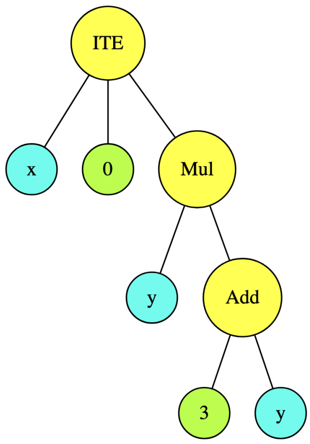
Proponha, explicando-a devidamente, uma solução sua para *convolve*. Valorizar-se-á a economia de código e o recurso aos combinadores *pointfree* estudados na disciplina, em particular a triologia *ana-cata-hilo* de tipos disponíveis nas bibliotecas dadas ou a definir.

Problema 4

Considere-se a seguinte sintaxe (abstrata e simplificada) para **expressões numéricas** (em b) com variáveis (em a),

$$
\begin{array}{l}
\textbf{data}\ \mathit{Expr}\ b\ a = V\ a \mid N\ b \mid T\ \mathit{Op}\ [\mathit{Expr}\ b\ a]\ \textbf{deriving}\ (\mathit{Show}, \mathit{Eq}) \\
\textbf{data}\ \mathit{Op} = \mathit{ITE} \mid \mathit{Add} \mid \mathit{Mul} \mid \mathit{Suc}\ \textbf{deriving}\ (\mathit{Show}, \mathit{Eq})
\end{array}
$$

possivelmente condicionais (cf. *ITE*, i.e. o operador condicional "if-then-else"). Por exemplo, a árvore mostrada a seguir



representa a expressão

$$\mathit{ite}\ (V\ \texttt{"x"})\ (N\ 0)\ (\mathit{multi}\ (V\ \texttt{"y"})\ (\mathit{soma}\ (N\ 3)\ (V\ \texttt{"y"}))) \tag{1}$$

– i.e. $\textbf{if}\ x\ \textbf{then}\ 0\ \textbf{else}\ y * (3 + y)$ – assumindo as "helper functions":

$$
\begin{array}{l}
\mathit{soma}\ x\ y = T\ \mathit{Add}\ [x,y] \\
\mathit{multi}\ x\ y = T\ \mathit{Mul}\ [x,y] \\
\mathit{ite}\ x\ y\ z = T\ \mathit{ITE}\ [x,y,z]
\end{array}
$$

No anexo E propõe-se uma base para o tipo Expr (*baseExpr*) e a correspondente algebra *inExpr* para construção do tipo *Expr*.

1. Complete as restantes definições da biblioteca *Expr* pedidas no anexo F.
2. No mesmo anexo, declare $\mathit{Expr}\ b$ como instância da classe *Monad*. **Sugestão**: relembre os exercícios da ficha 12.

3. Defina como um catamorfismo de *Expr* a sua versão monádia, que deverá ter o tipo:

$$mcataExpr :: Monad\ m \Rightarrow (a + (b + (Op, m\ [c])) \rightarrow m\ c) \rightarrow Expr\ b\ a \rightarrow m\ c$$

4. Para se avaliar uma expressão é preciso que todas as suas variáveis estejam instanciadas. Complete a definição da função

$$let\_exp :: (Num\ c) \Rightarrow (a \rightarrow Expr\ c\ b) \rightarrow Expr\ c\ a \rightarrow Expr\ c\ b$$

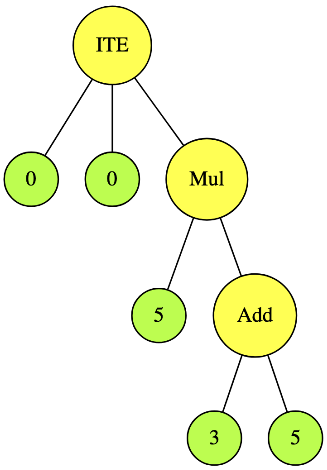
que, dada uma expressão com variáveis em a e uma função que a cada uma dessas variáveis atribui uma expressão ($a \rightarrow Expr\ c\ b$), faz a correspondente substituição.[1] Por exemplo, dada

$$\begin{array}{l} f\ \texttt{"x"} = N\ 0 \\ f\ \texttt{"y"} = N\ 5 \\ f\ \_ = N\ 99 \end{array}$$

ter-se-á

$$let\_exp\ f\ e = T\ ITE\ [N\ 1, N\ 0, T\ Mul\ [N\ 5, T\ Add\ [N\ 3, N\ 1]]]$$

isto é, a árvore da figura a seguir:



5. Finalmente, defina a função de avaliação de uma expressão, com tipo

$$evaluate :: (Num\ a, Ord\ a) \Rightarrow Expr\ a\ b \rightarrow Maybe\ a$$

que deverá ter em conta as seguintes situações de erro:

(a) *Variáveis* — para ser avaliada, x em $evaluate\ x$ não pode conter variáveis. Assim, por exemplo,

$$\begin{array}{l} evaluate\ e = Nothing \\ evaluate\ (let\_exp\ f\ e) = Just\ 40 \end{array}$$

para f e e dadas acima.

(b) *Aridades* — todas as ocorrências dos operadores deverão ter o devido número de subexpressões, por exemplo:

$$\begin{array}{l} evaluate\ (T\ Add\ [N\ 2, N\ 3]) = Just\ 5 \\ evaluate\ (T\ Mul\ [N\ 2]) = Nothing \end{array}$$

[1] Cf. expressões **let** ... **in**....

Sugestão: de novo se insiste na escrita do mínimo de código possível, tirando partido da riqueza estrutural do tipo *Expr* que é assunto desta questão. Sugere-se também o recurso a diagramas para explicar as soluções propostas.

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.

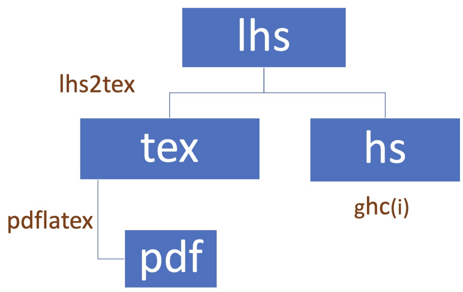
Para cumprir de forma integrada os objectivos do trabalho vamos recorrer a uma técnica de programação dita "literária" [1], cujo princípio base é o seguinte:

> *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 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.)

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

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:

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

E Código fornecido

Problema 1

$$h :: [Int]$$

Problema 4

Definição do tipo:

$$inExpr = [V, [N, \widehat{T}]]$$
$$baseExpr\ g\ h\ f = g + (h + id \times \mathsf{map}\ f)$$

Exemplos de expressões:

$$e = ite\ (V\ \texttt{"x"})\ (N\ 0)\ (multi\ (V\ \texttt{"y"})\ (soma\ (N\ 3)\ (V\ \texttt{"y"})))$$
$$i = ite\ (V\ \texttt{"x"})\ (N\ 1)\ (multi\ (V\ \texttt{"y"})\ (soma\ (N\ (3 / 5))\ (V\ \texttt{"y"})))$$

Exemplo de teste:

$$teste = evaluate\ (let\_exp\ f\ i) \equiv Just\ (26 / 245)$$
$$\quad \mathbf{where}\ f\ \texttt{"x"} = N\ 0; f\ \texttt{"y"} = N\ (1 / 7)$$

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

[2] Exemplos tirados de [2].

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

Para a resolução deste problema, utilizamos uma abordagem em duas fases, *divide* e *conquer*, típica dos hilomorfismos.

$$
\begin{array}{l}
divide :: [Int] \rightarrow () + ((Int, [Int]), [Int]) \\
divide = (id + (\langle\langle head, id\rangle, tail\rangle \cdot cons)) \cdot outList \cdot iSort
\end{array}
$$

$$
\begin{array}{l}
pp1 = \widehat{(\neq)} \cdot \langle \pi_1 \cdot \pi_2, \underline{0} \rangle \\
pp2 = \widehat{(\leqslant)} \cdot \langle \pi_1 \cdot \pi_2, length \cdot \pi_2 \cdot \pi_2 \rangle
\end{array}
$$

$$
\begin{array}{l}
conquer :: () + ((Int, [Int]), (Int, [Int])) \rightarrow (Int, [Int]) \\
conquer = [\langle \underline{0}, nil \rangle, cond\ (\widehat{(\wedge)} \cdot \langle pp1, pp2 \rangle)\ \pi_2\ \pi_1]
\end{array}
$$

$$
hindex = hyloList\ conquer\ divide
$$

$$
\begin{tikzcd}
[\mathbb{Z}] \arrow[r, "iSort \cdot outList"] \arrow[d, "{[\!(\, divide \,)\!]}"'] & 1 + \mathbb{Z} \times [\mathbb{Z}] \arrow[r, "id + (\langle\langle head, id\rangle, tail\rangle \cdot cons)"] & 1 + (\mathbb{Z} \times [\mathbb{Z}]) \times [\mathbb{Z}] \arrow[d, "{id + (id \times id) \times [\!(\, divide \,)\!]}"] \\
{[\mathbb{Z} \times [\mathbb{Z}]]} \arrow[rr, shift left, "out"] \arrow[d, "(\!|\, conquer \,|\!)"'] & & 1 + (\mathbb{Z} \times [\mathbb{Z}]) \times [\mathbb{Z} \times [\mathbb{Z}]] \arrow[ll, shift left, "\mathbf{in}"] \arrow[d, "id + (id \times id) \times (\!|\, conquer \,|\!)"] \\
\mathbb{Z} \times [\mathbb{Z}] & & 1 + (\mathbb{Z} \times [\mathbb{Z}]) \times (\mathbb{Z} \times [\mathbb{Z}]) \arrow[ll, "{[\langle \underline{0}, nil \rangle, cond\ (\widehat{(\wedge)} \cdot \langle pp1, pp2 \rangle)\ \pi_2\ \pi_1]}"]
\end{tikzcd}
$$

Problema 2

Primeira parte:

O *outNat* não é suficiente para este problema, logo desenvolvemos um *outPrimes* que junta os casos de 0, 1 e -1, e no outro caso faz o módulo do número que recebe.

divisorsList cria a lista de divisores de um número até à sua raiz quadrada (otimização). Esta é usada depois na função *isPrime* que verifica se um número é primo. Depois, *nextFactor* verifica qual o próximo fator primo de um número.

Por fim, *primes* é o anamorfismo que gera a lista de primos. A parte $aap\ \cdot \div \cdot\ nextFactor$ recebe um inteiro n e divide-o pelo seu próximo fator primo. *aap* é uma função monádica que calcula $nextFactor\ n$ e depois usa o resultado como argumento a $n \div nextFactor\ n$.

$$
\begin{aligned}
&outPrimes\ 0 = i_1\ ()\\
&outPrimes\ 1 = i_1\ ()\\
&outPrimes\ (-1) = i_1\ ()\\
&outPrimes\ n = i_2\ (abs\ n)
\end{aligned}
$$

$$
\begin{aligned}
&divisorsList :: \mathbb{Z} \to [\mathbb{Z}]\\
&divisorsList\ n = [x \mid x \leftarrow [2\,..\,isqrt\ n], mod\ n\ x \equiv 0]\\
&\quad \textbf{where}\ isqrt = floor \cdot sqrt \cdot fromIntegral
\end{aligned}
$$

$$
\begin{aligned}
&isPrime :: \mathbb{Z} \to Bool\\
&isPrime = [false, null \cdot divisorsList] \cdot outPrimes
\end{aligned}
$$

$$
\begin{aligned}
&nextFactor :: \mathbb{Z} \to \mathbb{Z}\\
&nextFactor\ n = head\ [x \mid x \leftarrow 2 : [3, 5\,..\,n], mod\ n\ x \equiv 0 \wedge isPrime\ x]
\end{aligned}
$$

$$
primes = [\!(\ (id + \langle nextFactor, aap\ \cdot \div \cdot\ nextFactor \rangle) \cdot outPrimes\ )\!]
$$

$$
\begin{tikzcd}
\mathbb{Z} \arrow[r, "outPrimes"] \arrow[d, "primes"'] & 1 + \mathbb{Z} \arrow[r, "id + \langle nextFactor{,} aap\ \cdot\div\cdot\ nextFactor \rangle"] & 1 + \mathbb{Z} \times \mathbb{Z} \arrow[d, "id + id \times primes"] \\
{[\mathbb{Z}]} & & 1 + \mathbb{Z} \times [\mathbb{Z}] \arrow[ll, "{[nil, cons]}"]
\end{tikzcd}
$$

Segunda parte:

Para criar a árvore precisamos criar os 'ramos', neste caso num par no formato (fatorização, número) com a função *buildPairs*.

Depois, fornecemos esses 'ramos' à função *untar* que irá criar a árvore do tipo $Exp\ \mathbb{Z}\ \mathbb{Z}$. Essa árvore vem numa lista, então apenas precisamos de usar *head* para a obter.

$$
\begin{aligned}
&buildPairs :: [\mathbb{Z}] \to [([\mathbb{Z}], \mathbb{Z})]\\
&buildPairs = \mathsf{map}\ \langle (:)\ 1 \cdot primes, id \rangle\\
&prime\_tree = head \cdot untar \cdot buildPairs
\end{aligned}
$$

Problema 3

O primeiro passo para a implementação da convolução de 2 listas é acrescentar ($length\ l1 - 1$) zeros ao início da segunda lista para que fique com tamanho igual a $length\ l1 + length\ l_2 - 1$, que é o tamanho da lista resultante da convolução. Depois calculamos as sublistas com *sufixes* para simular o deslizar de uma lista sobre a outra. De seguida são aplicados 2 map , um escrito na forma de catamorfismo outro em anamorfismo para ser possível criar um hilomorfismo. Primeiro, multiplica-se os elementos correspondentes das sublistas e da primeira lista invertida e de seguida reduz-se cada lista à sua soma tendo como resultado a lista correspondente à convolução.

$$
\begin{aligned}
&\mathit{convolve} :: \mathit{Num}\ a \Rightarrow [a] \rightarrow [a] \rightarrow [a] \\
&\mathit{convolve}\ \mathit{l1} = \mathit{hyloList}\ f\ g \cdot \mathit{suffixes} \cdot \mathit{flip}\ \mathit{padZeros}\ (\mathit{length}\ \mathit{l1} - 1) \\
&\quad \textbf{where}\ \mathit{padZeros}\ l = (\!|[\underline{l}, 0:]|\!) \\
&\qquad f = [\mathit{nil}, \mathit{cons} \cdot (\mathit{sum} \times \mathit{id})] \\
&\qquad g = (\mathit{id} + (\mathit{zipWith}\ (*)\ (\mathit{reverse}'\ \mathit{l1}) \times \mathit{id})) \cdot \mathit{outList}
\end{aligned}
$$

Diagrama *padZeros*:

$$
\begin{tikzcd}
\mathbb{N}_0 \arrow[r, "outNat0"] \arrow[d, "padZeros\ l"'] & 1 + \mathbb{N}_0 \arrow[d, "id+(padZeros\ l)"] \\
{[\mathbb{Z}]} & 1 + [\mathbb{Z}] \arrow[l, "{[\underline{l},0:]}"]
\end{tikzcd}
$$

Diagrama *convolve*:

$$
\begin{tikzcd}
{[A]} \arrow[r, "flip\ padZeros\ (length\ l1-1)"] \arrow[drr, "convolve\ l1"'] & {[A]} \arrow[r, "suffixes"] & {[[A]]} \arrow[r, "g"] \arrow[d, "hyloList\ f\ g"] & 1 + [A] \times [[A]] \arrow[d, "id+id\times(hyloList\ f\ g)"] \\
& & {[A]} & 1 + [A] \times [A] \arrow[l, "f"]
\end{tikzcd}
$$

Problema 4

Definição do tipo:

$$
\begin{aligned}
&\mathit{outExpr}\ (V\ a) = i_1\ a \\
&\mathit{outExpr}\ (N\ b) = i_2\ (i_1\ b) \\
&\mathit{outExpr}\ (T\ \mathit{op}\ l) = i_2\ (i_2\ (\mathit{op}, l)) \\
&\mathit{recExpr}\ f = \mathit{baseExpr}\ \mathit{id}\ \mathit{id}\ f
\end{aligned}
$$

Ana + cata + hylo:

$$
\begin{aligned}
&\mathit{cataExpr}\ g = g \cdot \mathit{recExpr}\ (\mathit{cataExpr}\ g) \cdot \mathit{outExpr} \\
&\mathit{anaExpr}\ g = \mathit{inExpr} \cdot \mathit{recExpr}\ (\mathit{anaExpr}\ g) \cdot g \\
&\mathit{hyloExpr}\ h\ g = \mathit{cataExpr}\ h \cdot \mathit{anaExpr}\ g
\end{aligned}
$$

$$
\begin{aligned}
&\textbf{instance}\ \mathit{Functor}\ (\mathit{Expr}\ b)\ \textbf{where} \\
&\quad \mathsf{fmap}\ f = \mathit{cataExpr}\ (\mathit{inExpr} \cdot \mathit{baseExpr}\ f\ \mathit{id}\ \mathit{id}) \\
&\textbf{instance}\ \mathit{Applicative}\ (\mathit{Expr}\ b)\ \textbf{where} \\
&\quad \mathit{pure} = \mathit{return} \\
&\quad (< * >) = \mathit{aap} \\
&\textbf{instance}\ \mathit{Monad}\ (\mathit{Expr}\ b)\ \textbf{where} \\
&\quad \mathit{return} = V \\
&\quad e \gg\!= f = \mathit{cataExpr}\ [f, [N, \widehat{T}]]\ e
\end{aligned}
$$

Maps: *Monad*: Monad: *Let expressions*:

$$let\_exp = flip\ (\gg\!=)$$

let_exp pode ser explicada pelo seguinte diagrama (que também ajuda a perceber $(\gg\!=)$):

$$\begin{tikzcd}
Expr\ C\ A \arrow[r, "outExpr"] \arrow[d, "let_exp\ f"'] & A + (C + Op \times (Expr\ C\ A)) \arrow[d, "id+(id+id\ x\ (\mathsf{map}\ \ (let_exp\ f)))"] \\
Expr\ C\ B & A + (C + Op \times (Expr\ C\ B)) \arrow[l, "{[f, [N, \widehat{T}]]}"]
\end{tikzcd}$$

Catamorfismo monádico:

$$\begin{array}{l}
mcataExpr\ g = g\ .!\ (aux \cdot recExpr\ (mcataExpr\ g) \cdot outExpr) \\
aux :: Monad\ m \Rightarrow a + (b + (Op, [m\ c])) \rightarrow m\ (a + (b + (Op, m\ [c]))) \\
aux = [return \cdot i_1, [return \cdot i_2 \cdot i_1, ret]] \\
\quad \mathbf{where}\ ret = return \cdot i_2 \cdot i_2 \cdot \langle \pi_1, sequence \cdot \pi_2 \rangle
\end{array}$$

Avaliação de expressões:

$$\begin{array}{l}
evaluate = cataExpr\ [nothing, [Just, g]] \\
\quad \mathbf{where}\ g\ (op, vals) = \mathbf{case}\ (op, sequence\ vals)\ \mathbf{of} \\
\qquad (Add, Just\ [x, y]) \rightarrow Just\ (x + y) \\
\qquad (Mul, Just\ [x, y]) \rightarrow Just\ (x * y) \\
\qquad (ITE, Just\ [cond, t, e]) \rightarrow \mathbf{if}\ cond > 0\ \mathbf{then}\ Just\ t\ \mathbf{else}\ Just\ e \\
\qquad _ \rightarrow Nothing
\end{array}$$

Index

References

[1] D.E. Knuth. *Literate Programming*. CSLI Lecture Notes Number 27. Stanford University Center for the Study of Language and Information, Stanford, CA, USA, 1992.

[2] 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).