Cálculo de Programas
Algebra of Programming

UNIVERSIDADE DO MINHO
Lic. em Engenharia Informática (3º ano)
Lic. Ciências da Computação (2º ano)

2023/24 - Ficha *(Exercise sheet)* nr. 10

1. A função $concat$, extraída do *Prelude* do Haskell, é o catamorfismo de listas

The concat function, taken from the Haskell Prelude, *is the list-catamorphism*

$$concat = (\![[\mathsf{nil}\ , \mathsf{conc}]]\!) \tag{F1}$$

onde $\mathsf{conc}\ (x, y) = x + y$ e $\mathsf{nil}\ \_ = [\,]$. Apresente justificações para a prova da propriedade

where $\mathsf{conc}\ (x, y) = x + y$ *and* $\mathsf{nil}\ \_ = [\,]$. *Provide justifications for proof of property*

$$\mathsf{length} \cdot concat = \mathsf{sum} \cdot \mathsf{map\ length} \tag{F2}$$

que a seguir se apresenta, onde é de esperar que as leis de *fusão*-cata e *absorção*-cata desempenhem um papel importante:

*which is presented below, where the cata-*fusion *and cata-*absorption *laws are expected to play an important role:*

$$
\begin{array}{ll}
& \mathsf{length} \cdot concat = \mathsf{sum} \cdot \mathsf{map\ length} \\
\equiv & \quad \{ \quad \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots \quad \} \\
& \mathsf{length} \cdot (\![[\mathsf{nil}\ , \mathsf{conc}]]\!) = (\![[\underline{0}\ , \mathsf{add}]]\!) \cdot \mathsf{map\ length} \\
\equiv & \quad \{ \quad \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots \quad \} \\
& \mathsf{length} \cdot (\![[\mathsf{nil}\ , \mathsf{conc}]]\!) = (\![[\underline{0}\ , \mathsf{add}] \cdot (id + \mathsf{length} \times id)]\!) \\
\Leftarrow & \quad \{ \quad \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots \quad \} \\
& \mathsf{length} \cdot [\mathsf{nil}\ , \mathsf{conc}] = [\underline{0}\ , \mathsf{add} \cdot (\mathsf{length} \times id)] \cdot (id + id \times \mathsf{length}) \\
\equiv & \quad \{ \quad \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots \quad \} \\
& \left\{ \begin{array}{l} \mathsf{length} \cdot \mathsf{nil} = \underline{0} \\ \mathsf{length} \cdot \mathsf{conc} = \mathsf{add} \cdot (\mathsf{length} \times id) \cdot (id \times \mathsf{length}) \end{array} \right. \\
\equiv & \quad \{ \quad \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots \quad \} \\
& \mathsf{length} \cdot \mathsf{conc} = \mathsf{add} \cdot (\mathsf{length} \times \mathsf{length}) \\
\equiv & \quad \{ \quad \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots \quad \} \\
& true \\
\square &
\end{array}
$$

2. O diagrama genérico de um catamorfismo de gene g sobre o *tipo paramétrico* $\mathsf{T}\ X \cong \mathsf{B}\ (X, \mathsf{T}\ X)$ cuja base é o bifunctor B, bem como a sua propriedade universal, são representados a seguir:

The generic diagram of a catamorphism with gene g over the parametric type $\mathsf{T}\ X \cong \mathsf{B}\ (X, \mathsf{T}\ X)$ *with base* B*, as well as its universal property, are represented below:*

$$\begin{tikzcd}
\mathsf{T}\ X \arrow[d, "\llparenthesis\, g\, \rrparenthesis"'] & \mathsf{B}\ (X, \mathsf{T}\ X) \arrow[l, "\mathsf{in}"'] \arrow[d, "\mathsf{B}\ (id, \llparenthesis\, g\, \rrparenthesis) = \mathsf{F}\ \llparenthesis\, g\, \rrparenthesis"] \\
B & \mathsf{B}\ (X, B) \arrow[l, "g"]
\end{tikzcd}$$

$$k = \llparenthesis\, g\, \rrparenthesis \quad \equiv \quad k \cdot \mathsf{in} = g \cdot \underbrace{\mathsf{B}\ (id, k)}_{\mathsf{F}\ k}$$

De seguida, apresenta-se uma revisão do inventário de tipos indutivos da questão 6 da ficha anterior, recorrendo agora aos seus functores de base:

Next, a review of the inventory of inductive types of question 6 of the previous exercise sheet is given, now using its base-functors:

(a) Árvores com informação de tipo A nas folhas *(Trees with data in their leaves)* :

$$\mathsf{T} = \mathsf{LTree}\ A \qquad \begin{cases} \mathsf{B}\ (X, Y) = X + Y^2 \\ \mathsf{B}\ (g, f) = g + f^2 \end{cases} \qquad \mathsf{in} = [\mathit{Leaf}\ , \mathit{Fork}]$$

Haskell: $\mathbf{data}\ \mathsf{LTree}\ a = \mathit{Leaf}\ a \mid \mathit{Fork}\ (\mathsf{LTree}\ a, \mathsf{LTree}\ a)$

(b) Árvores com informação de tipo A nos nós *(Trees whose data of type A are stored in their nodes)*:

$$\mathsf{T} = \mathsf{BTree}\ A \qquad \begin{cases} \mathsf{B}\ (X, Y) = 1 + X \times Y^2 \\ \mathsf{B}\ (g, f) = id + g \times f^2 \end{cases} \qquad \mathsf{in} = [\underline{\mathit{Empty}}\ , \mathit{Node}]$$

Haskell: $\mathbf{data}\ \mathsf{BTree}\ a = \mathit{Empty} \mid \mathit{Node}\ (a, (\mathsf{BTree}\ a, \mathsf{BTree}\ a))$

(c) Árvores com informação nos nós e nas folhas *(Full trees — data in both leaves and nodes)*:

$$\mathsf{T} = \mathsf{FTree}\ B\ A \qquad \begin{cases} \mathsf{B}\ (Z, X, Y) = Z + X \times Y^2 \\ \mathsf{B}\ (h, g, f) = h + g \times f^2 \end{cases} \qquad \mathsf{in} = [\mathit{Unit}\ , \mathit{Comp}]$$

Haskell: $\mathbf{data}\ \mathsf{FTree}\ b\ a = \mathit{Unit}\ b \mid \mathit{Comp}\ (a, (\mathsf{FTree}\ b\ a, \mathsf{FTree}\ b\ a))$

(d) Árvores de expressão *(Expression trees)*:

$$\mathsf{T} = \mathsf{Expr}\ V\ O \qquad \begin{cases} \mathsf{B}\ (Z, X, Y) = Z + X \times Y^* \\ \mathsf{B}\ (h, g, f) = h + g \times \mathsf{map}\ f \end{cases} \qquad \mathsf{in} = [\mathit{Var}\ , \mathit{Term}]$$

Haskell: $\mathbf{data}\ \mathsf{Expr}\ v\ o = \mathit{Var}\ v \mid \mathit{Term}\ (o, [\mathsf{Expr}\ v\ o])$

Partindo da definição *genérica* de map associado ao tipo T,

Starting from the generic *definition of* map *associated with the type* T*,*

$$\mathsf{T}\ f = \llparenthesis\, \mathsf{in} \cdot \mathsf{B}\ (f, id)\, \rrparenthesis$$

calcule $\mathit{fmap}\ f = \mathsf{T}\ f$ para $\mathsf{T} := \mathsf{BTree}$, entregando o resultado em Haskell sem combinadores *pointfree*. (Repare-se que se tem sempre $\mathsf{F}\ k = \mathsf{B}\ (id, k)$.)

derive $\mathit{fmap}\ f = \mathsf{T}\ f$ for $\mathsf{T} := \mathsf{BTree}$, delivering the result in Haskell without point-free *combinators. (Note that we always have $\mathsf{F}\ k = \mathsf{B}\ (id, k)$.)*

---

3. Recorra à lei da absorção-cata, entre outras, para verificar as seguintes propriedades sobre listas

Use the cata-absorption law, among others, to prove the following properties about lists

$$\mathsf{length} = \mathsf{sum} \cdot (\mathsf{map}\ \underline{1}) \tag{F3}$$

$$\mathsf{length} = \mathsf{length} \cdot (\mathsf{map}\ f) \tag{F4}$$

onde length, sum e map são catamorfismos de listas que conhece. (Recorda-se que o bifunctor de base para listas é $\mathsf{B}\ (f, g) = id + f \times g$, de onde se deriva $\mathsf{F}\ f = \mathsf{B}\ (id, f) = id + id \times f$.)

where length, sum *and* map *they are list-catamorphisms you know. (Remember that the basic bifunctor for lists is* $\mathsf{B}\ (f, g) = id + f \times g$, *yielding* $\mathsf{F}\ f = \mathsf{B}\ (id, f) = id + id \times f$.*)*

---

4. Seja dado o catamorfismo

Let catamorphism

$$depth = (\![[\mathsf{one}\ , \mathsf{succ} \cdot umax]]\!)$$

que dá a profundidade de árvores do tipo LTree, onde $umax\ (a, b) = max\ a\ b$. Mostre, por absorção-cata, que a profundidade de uma árvore t não é alterada quando aplica uma função f a todas as suas folhas:

be given, which gives the depth of trees of type LTree, *where* $umax\ (a, b) = max\ a\ b$. *Show, by cata-absorption, that the depth of a tree* t *is not changed when you apply a function* f *to all its leaves:*

$$depth \cdot \mathsf{LTree}\ f = depth \tag{F5}$$

---

5. Um *anamorfismo* é um *"catamorfismo ao contrário"*, i.é uma função $k : A \to \mathsf{T}$ tal que

An anamorphism *is a* "reverse catamorphism", *i.e. a function* $k : A \to \mathsf{T}$ *such that*

$$k = \mathsf{in} \cdot \mathsf{F}\ k \cdot g \tag{F6}$$

escrevendo-se $k = [\!(g)\!]$. Mostre que o anamorfismo de listas

One writes $k = [\!(g)\!]$. *Show that the list-anamorphism*

$$k = [\!((id + \langle f, id \rangle) \cdot \mathsf{out}_{\mathbb{N}_0})\!] \tag{F7}$$

descrito pelo diagrama

depicted in diagram

$$\begin{tikzcd}
{\mathbb{N}_0}^* & & & 1 + \mathbb{N}_0 \times {\mathbb{N}_0}^* \arrow[lll, "\mathsf{in}"'] \\
\mathbb{N}_0 \arrow[u, "k"] \arrow[r, "\mathsf{out}_{\mathbb{N}_0}"'] & 1 + \mathbb{N}_0 \arrow[rr, "id + \langle f, id \rangle"'] & & 1 + \mathbb{N}_0 \times \mathbb{N}_0 \arrow[u, "id + id \times k"']
\end{tikzcd}$$

é a função

is the function

$$\begin{aligned}
&k\ 0 = [\,] \\
&k\ (n + 1) = (2\ n + 1) : k\ n
\end{aligned}$$

para $f\ n = 2\ n + 1$. (Que faz esta função?)

for $f\ n = 2\ n + 1$. *(What does this function do?)*

---

6. Mostre que o anamorfismo que calcula os sufixos de uma lista

Show that the anamorphism that computes the suffixes of a list

$$suffixes = [\!(g)\!]\ \textbf{where}\ g = (id + \langle \mathsf{cons}, \pi_2 \rangle) \cdot \mathsf{out}$$

é a função:

is the function:

$$\begin{aligned}
&suffixes\ [\,] = [\,] \\
&suffixes\ (h : t) = (h : t) : suffixes\ t
\end{aligned}$$

---

7. O formulário desta disciplina apresenta duas definições alternativas para o functor $\mathsf{T}\ f$, uma como *catamorfismo* e outra como *anamorfismo*. Identifique-as e acrescente justificações à seguinte prova de que essas definições são equivalentes:

The reference sheet of this course presents two alternative definitions for functor $\mathsf{T}\ f$, *one as a* catamorphism *and another as an* anamorphism. *Identify them and fill in justifications in the following proof that such definitions are equivalent:*

$$
\begin{array}{ll}
& \mathsf{T}\ f = \llparenthesis\, \mathsf{in} \cdot \mathsf{B}\ (f, id)\, \rrparenthesis \\
\equiv & \qquad \{ \quad \dots\dots\dots\dots\dots\dots\dots\dots\dots\dots \quad \} \\
& \mathsf{T}\ f \cdot \mathsf{in} = \mathsf{in} \cdot \mathsf{B}\ (f, id) \cdot \mathsf{F}\ (\mathsf{T}\ f) \\
\equiv & \qquad \{ \quad \dots\dots\dots\dots\dots\dots\dots\dots\dots\dots \quad \} \\
& \mathsf{T}\ f \cdot \mathsf{in} = \mathsf{in} \cdot \mathsf{B}\ (id, \mathsf{T}\ f) \cdot \mathsf{B}\ (f, id) \\
\equiv & \qquad \{ \quad \dots\dots\dots\dots\dots\dots\dots\dots\dots\dots \quad \} \\
& \mathsf{out} \cdot \mathsf{T}\ f = \mathsf{F}\ (\mathsf{T}\ f) \cdot \mathsf{B}\ (f, id) \cdot \mathsf{out} \\
\equiv & \qquad \{ \quad \dots\dots\dots\dots\dots\dots\dots\dots\dots\dots \quad \} \\
& \mathsf{T}\ f = \llbracket\, \mathsf{B}\ (f, id) \cdot \mathsf{out}\, \rrbracket \\
\Box &
\end{array}
$$

---

8. Mostre que o catamorfismo de listas $\mathsf{length} = \llparenthesis\, [\mathsf{zero}\ , \mathsf{succ} \cdot \pi_2]\, \rrparenthesis$ é a mesma função que o *anamorfismo* de naturais $\llbracket (id + \pi_2) \cdot \mathsf{out}_{\mathsf{List}} \rrbracket$.

Show that the list catamorphism $\mathsf{length} = \llparenthesis\, [\mathsf{zero}\ , \mathsf{succ} \cdot \pi_2]\, \rrparenthesis$ *is the same function as the* $\mathbb{N}_0$-anamorphism $\llbracket (id + \pi_2) \cdot \mathsf{out}_{\mathsf{List}} \rrbracket$.

---

9. O facto de $\mathsf{length} : A^* \to \mathbb{N}_0$ poder ser definida tanto como *catamorfismo* de listas como *anamorfismo* de naturais (questão 8) pode generalizar-se da forma seguinte: sejam dados dois tipos indutivos

The fact that $\mathsf{length} : A^* \to \mathbb{N}_0$ *is both a list-*catamorphism *and a* $\mathbb{N}_0$-anamorphism *(question 8) generalizes as follows: let two inductive types be given*

$$
\begin{tikzcd}
\mathsf{T}_1 \arrow[r, bend left, "\mathsf{out}_1"] \arrow[r, phantom, "\cong"] & \mathsf{F}\ \mathsf{T}_1 \arrow[l, bend left, "\mathsf{in}_1"]
\end{tikzcd}
\qquad
\begin{tikzcd}
\mathsf{T}_2 \arrow[r, bend left, "\mathsf{out}_2"] \arrow[r, phantom, "\cong"] & \mathsf{G}\ \mathsf{T}_2 \arrow[l, bend left, "\mathsf{in}_2"]
\end{tikzcd}
$$

e $\alpha : \mathsf{F}\ X \to \mathsf{G}\ X$, isto é, α satisfaz a propriedade *grátis*:

and $\alpha : \mathsf{F}\ X \to \mathsf{G}\ X$, *that is,* α *satisfying the* free *property:*

$$
\mathsf{G}\ f \cdot \alpha = \alpha \cdot \mathsf{F}\ f \tag{F8}
$$

Então $\llparenthesis\, \mathsf{in}_2 \cdot \alpha\, \rrparenthesis = \llbracket \alpha \cdot \mathsf{out}_1 \rrbracket$, como se mostra a seguir (complete as justificações):

Then $\llparenthesis\, \mathsf{in}_2 \cdot \alpha\, \rrparenthesis = \llbracket \alpha \cdot \mathsf{out}_1 \rrbracket$, *as shown below (complete the justifications):*

$$
\begin{array}{ll}
& k = (\!| \mathsf{in}_2 \cdot \alpha |\!) \\
\equiv & \qquad \{ \quad \dots\dots\dots\dots\dots\dots\dots\dots\dots\dots\dots\dots\dots\dots \quad \} \\
& k \cdot \mathsf{in}_1 = \mathsf{in}_2 \cdot \alpha \cdot \mathsf{F}\ k \\
\equiv & \qquad \{ \quad \dots\dots\dots\dots\dots\dots\dots\dots\dots\dots\dots\dots\dots\dots \quad \} \\
& \mathsf{out}_2 \cdot k = \mathsf{G}\ k \cdot \alpha \cdot \mathsf{out}_1 \\
\equiv & \qquad \{ \quad \dots\dots\dots\dots\dots\dots\dots\dots\dots\dots\dots\dots\dots\dots \quad \} \\
& k = [\!(\alpha \cdot \mathsf{out}_1)\!] \\
\Box &
\end{array}
$$

Como aplicação imediata deste resultado, redefina

Use the result above to redefine

$$
\left\{
\begin{array}{l}
mirror :: \mathsf{LTree}\ a \rightarrow \mathsf{LTree}\ a \\
mirror = (\!| \mathsf{in} \cdot (id + \mathsf{swap}) |\!)
\end{array}
\right.
\tag{F9}
$$

como um anamorfismo.

as an anamorphism.