Cálculo de Programas
Algebra of Programming

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

2022/23 - Ficha *(Exercise sheet)* nr. 9

1. 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*

$$k = \mathsf{in} \cdot \mathsf{F}\ k \cdot g \tag{F1}$$

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

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

$$k = [\!((id + \langle f, id \rangle) \cdot \mathsf{out}_{\mathbb{N}_0})\!] \tag{F2}$$

   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

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

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

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

---

2. 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*

$$concat = (\!|\, [\mathsf{nil}\ , \mathsf{conc}]\, |\!) \tag{F3}$$

   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*

$$\mathsf{length} \cdot concat = \mathsf{sum} \cdot \mathsf{map\ length} \tag{F4}$$

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:*

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

---

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

$$\mathsf{length} = \mathsf{sum} \cdot (\mathsf{map}\ \underline{1}) \tag{F5}$$

$$\mathsf{length} = \mathsf{length} \cdot (\mathsf{map}\ f) \tag{F6}$$

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. 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:*

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

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

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 nos nós *(Trees whose data of type A are stored in their nodes)*:

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

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

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

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

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

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

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

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

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

$$\mathsf{T} = \mathsf{Expr}\ V\ O \qquad \left\{ \begin{array}{l} \mathsf{B}\ (Z,X,Y) = Z + X \times Y^* \\ \mathsf{B}\ (h,g,f) = h + g \times \mathsf{map}\ f \end{array} \right. \qquad \mathsf{in} = [Var\ , Op]$$

Haskell: **data** $\mathsf{Expr}\ v\ o = Var\ v \mid Op\ (o, [\mathsf{Expr}\ v\ o])$

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

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

$$\mathsf{T}\ f = (\!|\ \mathsf{in} \cdot \mathsf{B}\ (f, id)\ |\!)$$

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

---

5. 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:*

$$depth \cdot \mathsf{LTree}\ f = depth \qquad \text{(F7)}$$