Cálculo de Programas
Algebra of Programming

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

2025/26 - Ficha *(Exercise sheet)* nr. 10

1. Considere o anamorfismo $r = [\!(g)\!]$ em que

Consider the anamorphism $r = [\!(g)\!]$ where

$$\begin{array}{l} g\ [\,] = i_1\ () \\ g\ x = i_2\ (\mathsf{last}\ x, \mathsf{init}\ x) \end{array}$$

onde $\mathsf{last}\ \ x$ dá o último elemento da lista x e $\mathsf{init}\ x$ dá x sem esse último elemento. O que faz a função r? Responda informalmente desenhando o diagrama de r.

in which $\mathsf{last}\ \ x$ *gives the last element of list* x *and* $\mathsf{init}\ \ x$ *gives* x *without this last element. What does* r *do? Answer informally by drawing the diagram of* r.

---

2. Considere a função:

Let function

$$x \ominus y = \mathbf{if}\ x \leqslant y\ \mathbf{then}\ 0\ \mathbf{else}\ 1 + x \ominus (y+1)$$

Quais os valores das expressões $(3 \ominus 2) \ominus 3$ e $(3 \ominus 4) + 4$? Codifique $\widehat{\ominus} : \mathbb{N}_0 \times \mathbb{N}_0 \to \mathbb{N}_0$ como um anamorfismo de naturais e faça o respectivo diagrama.

be given. Evaluate $(3 \ominus 2) \ominus 3$ *and* $(3 \ominus 4) + 4$ *and encode* $\widehat{\ominus} : \mathbb{N}_0 \times \mathbb{N}_0 \to \mathbb{N}_0$ *as an anamorphism over* $\mathbb{N}_0$*. Draw the corresponding diagram.*

---

3. 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*

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

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:*

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

---

4. 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* $\mathsf{F}\ f = \mathsf{B}\ (id, f) = id + id \times f$*.)*

---

5. Seja dado o catamorfismo

Let catamorphism

$$depth = (\!|\, [\mathsf{one}\,, \mathsf{succ} \cdot umax] \,|\!) \tag{F5}$$

que dá a profundidade de árvores do tipo LTree, onde $umax\ (a, b) = max\ a\ b$ e $\mathsf{one} = \underline{1}$. 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* $umax\ (a, b) = max\ a\ b$ *and* $\mathsf{one} = \underline{1}$*. Show, by cata-absorption, that the depth of a tree* t *is not changed when you apply a function* f *to all its leaves:*

$$depth \cdot \mathsf{LTree}\ f = depth \tag{F6}$$

---

6. O algoritmo "bubble-sort" é o ciclo-for

The "bubble-sort" algorithm is a for-loop:

$$
\begin{array}{l}
bSort\ xs = \mathsf{for}\ \mathsf{bubble}\ xs\ (\mathsf{length}\ xs)\ \mathbf{where} \\
\quad \mathsf{bubble}\ (x : y : xs) \\
\qquad \mid x > y = y : \mathsf{bubble}\ (x : xs) \\
\qquad \mid \mathsf{otherwise} = x : \mathsf{bubble}\ (y : xs) \\
\quad \mathsf{bubble}\ x = x
\end{array}
$$

cujo corpo de ciclo é um hilomorfismo $\mathsf{bubble} = [\![conquer, divide]\!]$. Recordando a hilo-factorização que se fez na aula teórica para a função *fib*, identifique os genes *divide* e *conquer* desse hilomorfismo.

Its loop-body is a hylomorphism $\mathsf{bubble} = [\![conquer, divide]\!]$. *Recalling the theory classes (cf. hylo-factorization of fib) identify the genes divide and conquer of this hylomorphism.*

---

7. Considere a seguinte sintaxe concreta em Haskell para um tipo que descreve pontos no espaço tridimensional:

Consider the following concrete syntax in Haskell for a type that describes 3D-points:

$$
\mathbf{data}\ Point\ a = Point\ \{ x :: a, y :: a, z :: a \}\ \mathbf{deriving}\ (Eq, Show)
$$

Pelo GHCi apura-se:

GHCi tells:

$$
Point :: a \to a \to a \to Point\ a
$$

Raciocinando apenas em termos de tipos, conjecture a definição de in na seguinte conversão dessa sintaxe concreta para abstracta:

Reasoning only in terms of types, conjecture the definition of in *in the following conversion from concrete to abstract syntax:*

$$
\begin{tikzcd}
Point\ A \arrow[r, bend left, "\mathsf{out} = \langle \langle x, y \rangle, z \rangle"] \arrow[r, phantom, "\cong"] & (A \times A) \times A \arrow[l, bend left, "\mathsf{in} = \cdots"]
\end{tikzcd}
$$

---

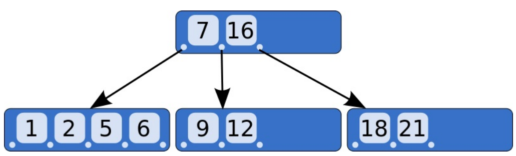
8. **Questão prática** — Este problema não irá ser abordado em sala de aula. Os alunos devem tentar resolvê-lo em casa e, querendo, publicarem a sua solução no canal **#geral** do Slack, com vista à sua discussão com colegas.
Dão-se a seguir os requisitos do problema.

Open assignment *— This assignment will not be addressed in class. Students should try to solve it at home and, whishing so, publish their solutions in the* ***#geral*** *Slack channel, so as to trigger discussion among other colleagues.*
The requirements of the problem are given below.

Problem requirements*:*
A "B-tree"is a generalization of the binary trees of the BTree *module to more than two subtrees per node:*

$$
\mathbf{data}\ B\_tree\ a = Nil \mid Block\ \{ leftmost :: B\_tree\ a, block :: [(a, B\_tree\ a)] \}
$$

For instance, the B-tree



is represented by the data type above as follows:

$$
\begin{array}{l}
t = \mathit{Block}\ \{ \\
\quad \mathit{leftmost} = \mathit{Block}\ \{ \\
\quad\quad \mathit{leftmost} = \mathit{Nil}, \\
\quad\quad \mathit{block} = [(1, \mathit{Nil}), (2, \mathit{Nil}), (5, \mathit{Nil}), (6, \mathit{Nil})]\}, \\
\quad \mathit{block} = [\\
\quad\quad (7, \mathit{Block}\ \{ \\
\quad\quad\quad \mathit{leftmost} = \mathit{Nil}, \\
\quad\quad\quad \mathit{block} = [(9, \mathit{Nil}), (12, \mathit{Nil})]\}), \\
\quad\quad (16, \mathit{Block}\ \{ \\
\quad\quad\quad \mathit{leftmost} = \mathit{Nil}, \\
\quad\quad\quad \mathit{block} = [(18, \mathit{Nil}), (21, \mathit{Nil})]\}) \\
\quad\quad]\}
\end{array}
$$

Write a Haskell library for data type B_Tree *following the same structure as the others already available, e.g. BTree.hs.*

□