Cálculo de Programas
Algebra of Programming

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

2024/25 - Ficha *(Exercise sheet)* nr. 10

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

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

   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.

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

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

   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:*

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

---

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:

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

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.*

---

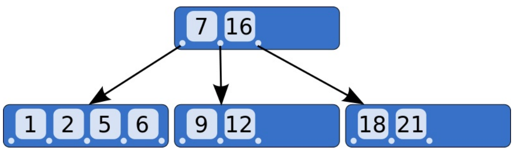
7. **Questão prática** — Este problema não irá ser abordado em sala de aula. Os alunos devem tentar resolvê-lo em casa e, querendo, publicarem a sua solução no canal **#geral** do Slack, com vista à sua discussão com colegas.
Dão-se a seguir os requisitos do problema.

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:

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

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