Cálculo de Programas
Algebra of Programming

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

2026/27 — Ficha *(Worksheet)* nr.º 8

1. Considere o seguinte inventário de quatro tipos de árvores:

   Consider the following inventory of four types of trees:

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

   $$\mathsf{T} = \mathsf{LTree}\ A \qquad \begin{cases} \mathsf{F}\ X = A + X^2 \\ \mathsf{F}\ f = id + f^2 \end{cases} \qquad \mathsf{in} = [Leaf\ , Fork]$$

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

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

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

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

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

   $$\mathsf{T} = \mathsf{FTree}\ B\ A \qquad \begin{cases} \mathsf{F}\ X = B + A \times X^2 \\ \mathsf{F}\ f = id + id \times f^2 \end{cases} \qquad \mathsf{in} = [Unit\ , Comp]$$

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

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

   $$\mathsf{T} = \mathsf{Expr}\ V\ O \qquad \begin{cases} \mathsf{F}\ X = V + O \times X^* \\ \mathsf{F}\ f = id + id \times \mathsf{map}\ f \end{cases} \qquad \mathsf{in} = [Var\ , Term]$$

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

   Defina o gene g para cada um dos catamorfismos seguintes desenhando, para cada caso, o diagrama correspondente:

   - $maximum = (\!|\, g \,|\!)$ — devolve a maior folha de uma árvore de tipo (1a).
   - $inorder = (\!|\, g \,|\!)$ — faz a travessia inorder de uma árvore de tipo (1b).
   - $mirror = (\!|\, g \,|\!)$ — espelha uma árvore de tipo (1b), i.e., roda-a de 180º.
   - $rep\ a = (\!|\, g \,|\!)$ — substitui todas as folhas de uma árvore de tipo (1a) por um mesmo valor $a \in A$.

   Define the "gene" g for each of the following catamorphisms by drawing, for each case, the corresponding diagram:

   - $maximum = (\!|\, g \,|\!)$ — *returns the largest leaf of a tree of type (1a).*
   - $inorder = (\!|\, g \,|\!)$ — *performs a traversal of a type tree (1b).*
   - $mirror = (\!|\, g \,|\!)$ — *mirrors a tree of type (1b), i.e., rotates it 180º.*
   - $rep\ a = (\!|\, g \,|\!)$ — *replaces all leaves of a tree of type (1a) by the same value $a \in A$.*

- $convert = (\!|\, g \,|\!)$ — converte árvores de tipo (1c) em árvores de tipo (1b) eliminando os Bs que estão na primeira.
- $vars = (\!|\, g \,|\!)$ — lista as variáveis de uma árvore expressão de tipo (1d).

- $convert = (\!|\, g \,|\!)$ — *converts trees of type (1c) into trees of type (1b) eliminating the Bs that can be found in the first.*
- $vars = (\!|\, g \,|\!)$ — *lists the variables of an expression tree of type (1d).*

---

2. Derive a versão *pointwise* do seguinte catamorfismo de BTrees,

   Derive the pointwise *version of the following catamorphism of* BTrees

$$\begin{aligned} tar &= (\!|\, [\mathsf{singl} \cdot \mathsf{nil}\ , g] \,|\!)\ \mathbf{where} \\ g &= \mathsf{map}\ \mathsf{cons} \cdot lstr \cdot (id \times \mathsf{conc}) \\ lstr\ (b, x) &= [(b, a) \mid a \leftarrow x] \end{aligned}$$

   entregando no final uma versão da função em que não ocorrem os nomes das funções map, cons, singl, nil, conc e $lstr$. Pode usar map $f\ x = [f\ a \mid a \leftarrow x]$ como definição *pointwise* de map em listas.

   eventually delivering a version of the function in which the function names map, cons, singl, nil, conc *do not occur. and* $lstr$. *You can use* map $f\ x = [f\ a \mid a \leftarrow x]$ *as* pointwise *definition of* map *in lists.*

---

3. Converta o catamorfismo $vars$ do exercício 1 numa função em Haskell sem quaisquer combinadores *pointfree*.

   Unfold catamorphism $vars$ *(exercise 1) towards a function in Haskell without any pointfree combinator.*

---

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

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

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

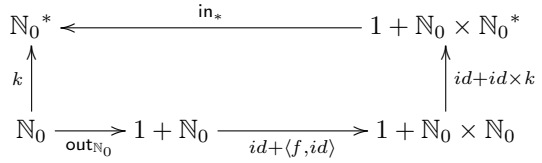
   escrevendo-se $k = [\!(g)\!]$. Mostre que o anamorfismo de listas

   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



   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?)*

---

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

   Show that the anamorphism that computes the suffixes of a list

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

é a função:

is the function:

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

---

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

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

---

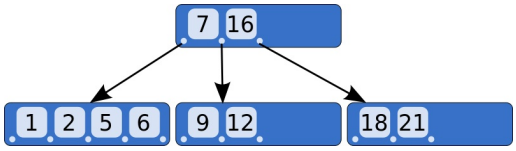
7. **Questão prática** — Este problema não irá ser abordado em sala de aula. Os alunos devem tentar resolvê-lo em casa e, querendo, publicarem a sua solução no canal **#all-cp2627** do Slack, com vista à sua discussão com colegas. Dão-se a seguir os requisitos do problema.

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* ***#all-cp2627*** *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:

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

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