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. 9

1. A igualdade que se segue *The following equality*

$$f \cdot \mathsf{length} = (\!|\, [\mathsf{zero}\ , (2+) \cdot \pi_2]\, |\!)$$

verifica-se para $f = (2*)$ ou $f = (2+)$? Use a lei de fusão-cata para justificar, por cálculo, a sua resposta.

holds for $f = (2*)$ or $f = (2+)$? Use the cata-fusion law to justify, by calculation, your answer.

---

2. 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)* :

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

   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)*:

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

   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)*:

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

   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)*:

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

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

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 (2a).
- $inorder = (\!|\, g \,|\!)$ — faz a travessia inorder de uma árvore de tipo (2b).
- $mirror = (\!|\, g \,|\!)$ — espelha uma árvore de tipo (2b), i.e., roda-a de 180º.
- $rep\ a = (\!|\, g \,|\!)$ — substitui todas as folhas de uma árvore de tipo (2a) por um mesmo valor $a \in A$.
- $convert = (\!|\, g \,|\!)$ — converte árvores de tipo (2c) em árvores de tipo (2b) eliminando os Bs que estão na primeira.
- $vars = (\!|\, g \,|\!)$ — lista as variáveis de uma árvore expressão de tipo (2d).

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 (2a).*
- $inorder = (\!|\, g \,|\!)$ *— performs a traversal of a type tree (2b).*
- $mirror = (\!|\, g \,|\!)$ *— mirrors a tree of type (2b), i.e., rotates it 180º.*
- $rep\ a = (\!|\, g \,|\!)$ *— replaces all leaves of a tree of type (2a) by the same value $a \in A$.*
- $convert = (\!|\, g \,|\!)$ *— converts trees of type (2c) into trees of type (2b) eliminating the Bs that can be found in the first.*
- $vars = (\!|\, g \,|\!)$ *— lists the variables of an expression tree of type (2d).*

---

3. Mostre por fusão-cata que a propriedade genérica

Show by cata-fusion that the following generic property of catamorphisms

$$(\!|\, g \,|\!) \cdot (\!|\, \mathsf{in} \cdot k \,|\!) = (\!|\, g \cdot m \,|\!) \tag{F1}$$

se verifica desde que

holds wherever

$$m \cdot \mathsf{F}\, f = \mathsf{F}\, f \cdot k \tag{F2}$$

se verifique também, para qualquer f.

also holds, for any f.

---

4. Seja definido o catamorfismo

Let the following catamorphism be defined,

$$mirror = (\!|\, \mathsf{in} \cdot (id + \mathsf{swap}) \,|\!)$$

que espelha árvores de tipo LTree. (a) Mostre que $k = m = (id + \mathsf{swap})$ satisfazem a condição (F2) da questão anterior; (b) Mostre que

which mirrors trees of type LTree. *(a) Show that $k = m = (id+\mathsf{swap})$ satisfy the condition (F2) of the previous question; (b) Show that*

$$mirror \cdot mirror = id$$

resulta da correspondente aplicação de (F1).

result of the corresponding application of (F1).

---

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

Derive the pointwise *version of the following catamorphism of* BTree*s*

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

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.*

---

6. Converta o catamorfismo $vars$ do exercício 2 numa função em Haskell sem quaisquer combinadores *pointfree*.

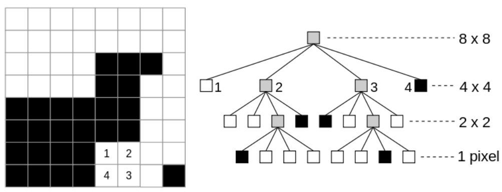
Unfold catamorphism $vars$ (exercise 2) towards a function in Haskell without any pointfree combinator.

---

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**: The figure below*



(Source: Wikipedia) shows how an image (in this case in black and white) is represented in the form of a quaternary tree (vulg. quadtree*) by successive divisions of the 2D space into four regions, until reaching the resolution of one pixel.*
Let the following Haskell definition of a quadtree *be given, for a given type $Pixel$ predefined:*

$$\mathbf{data}\ \mathsf{QTree} = Pixel \mid Blocks\ (\mathsf{QTree})\ (\mathsf{QTree})\ (\mathsf{QTree})\ (\mathsf{QTree})$$

Having chosen for this type the base functor

$$\mathsf{F}\ Y = Pixel + Y^2 \times Y^2 \tag{F3}$$

where Y^2 abbreviates $Y \times Y$, as usual, define the usual construction and decomposition functions of this type, cf.:

$$\begin{tikzcd} \mathsf{QTree}\ A \arrow[rr, bend left, "\mathsf{out}_{\mathsf{QTree}}"] \arrow[rr, phantom, "\cong"] & & \mathsf{F}\ (\mathsf{QTree}) \arrow[ll, bend left, "\mathsf{in}_{\mathsf{QTree}}"] \end{tikzcd}$$

Then, write the Haskell code of Quad.hs, a Haskell library similar to others already available, e.g. LTree.hs. Finally, implement as a QTree *catamorphism the operation that rotates an image 90º clockwise.*
□