

# 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*

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

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*):

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

Haskell: `data LTree a = Leaf a | Fork (LTree a, LTree a)`

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

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

Haskell: `data BTree a = Empty | Node (a, (BTree a, BTree a))`

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

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

Haskell: `data FTree b a = Unit b | Comp (a, (FTree b a, FTree b a))`

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

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

Haskell: `data Expr v o = Var v | Term (o, [Expr v o])`

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

- $maximum = \llbracket g \rrbracket$  — devolve a maior folha de uma árvore de tipo (2a).
- $inorder = \llbracket g \rrbracket$  — faz a travessia in-order de uma árvore de tipo (2b).
- $mirror = \llbracket g \rrbracket$  — espelha uma árvore de tipo (2b), i.e., roda-a de  $180^\circ$ .
- $rep\ a = \llbracket g \rrbracket$  — substitui todas as folhas de uma árvore de tipo (2a) por um mesmo valor  $a \in A$ .
- $convert = \llbracket g \rrbracket$  — converte árvores de tipo (2c) em árvores de tipo (2b) eliminando os  $B$ s que estão na primeira.
- $vars = \llbracket g \rrbracket$  — 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:

- $maximum = \llbracket g \rrbracket$  — returns the largest leaf of a tree of type (2a).
- $inorder = \llbracket g \rrbracket$  — performs a traversal of a type tree (2b).
- $mirror = \llbracket g \rrbracket$  — mirrors a tree of type (2b), i.e., rotates it  $180^\circ$ .
- $rep\ a = \llbracket g \rrbracket$  — replaces all leaves of a tree of type (2a) by the same value  $a \in A$ .
- $convert = \llbracket g \rrbracket$  — converts trees of type (2c) into trees of type (2b) eliminating the  $B$ s that can be found in the first.
- $vars = \llbracket g \rrbracket$  — 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

$$\llbracket g \rrbracket \cdot \llbracket in \cdot k \rrbracket = \llbracket g \cdot m \rrbracket \quad (F1)$$

se verifica desde que

holds wherever

$$m \cdot F\ f = F\ f \cdot k \quad (F2)$$

se verifique também, para qualquer  $f$ .

also holds, for any  $f$ .

4. Seja definido o catamorfismo

Let the following catamorphism be defined,

$$mirror = \llbracket in \cdot (id + swap) \rrbracket$$

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

which mirrors trees of type  $LTree$ . (a) Show that  $k = m = (id + swap)$  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  $BTrees$

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

entregando no final uma versão da função em que não ocorrem os nomes das funções `map`, `cons`, `singl`, `nil`, `conc` e `lstr`. Pode usar  $\text{map } f \ x = [f \ a \mid a \leftarrow x]$  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  $\text{map } f \ x = [f \ a \mid a \leftarrow x]$  as pointwise definition of `map` in lists.*

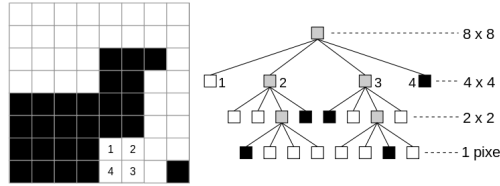
6. Converta o catamorfismo *vars* do exercício 2 numa função em Haskell sem quaisquer combinadores *pointfree*.

*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, wishing 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:

```
data QTree = Pixel | Blocks (QTree) (QTree) (QTree) (QTree)
```

Having chosen for this type the base functor

$$F \ Y = \text{Pixel} + Y^2 \times Y^2 \quad (\text{F3})$$

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

$$\begin{array}{ccc} \text{QTree } A & \xrightarrow{\text{out}_{\text{QTree}}} & F(\text{QTree}) \\ & \cong & \\ & \xleftarrow{\text{in}_{\text{QTree}}} & \end{array}$$

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 90° clockwise.

□