Cálculo de Programas
Algebra of Programming

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

2022/23 - Ficha *(Exercise sheet)* nr. 8

1. Mostre que, sempre que F e G são functores, então a sua composição $\mathsf{H} = \mathsf{F} \cdot \mathsf{G}$ é também um functor.

Show that wherever F and G are functors, then their composition $\mathsf{H} = \mathsf{F} \cdot \mathsf{G}$ is also a functor.

---

2. Mostre que a lei da recursividade mútua generaliza a mais do que duas funções, neste caso três:

Show that the mutual recursion law generalizes to more than two functions (three, in the following case):

$$\left\{ \begin{array}{l} f \cdot \mathsf{in} = h \cdot \mathsf{F}\ \langle \langle f, g \rangle, j \rangle \\ g \cdot \mathsf{in} = k \cdot \mathsf{F}\ \langle \langle f, g \rangle, j \rangle \\ j \cdot \mathsf{in} = l \cdot \mathsf{F}\ \langle \langle f, g \rangle, j \rangle \end{array} \right. \equiv \quad \langle \langle f, g \rangle, j \rangle = (\!| \langle \langle h, k \rangle, l \rangle |\!) \tag{F1}$$

---

3. As seguintes funções mutuamente recursivas testam a paridade de um número natural:

The following mutually recursive functions test the parity of a natural number:

$$\left\{ \begin{array}{l} impar\ 0 = \text{FALSE} \\ impar\ (n+1) = par\ n \end{array} \right. \qquad \left\{ \begin{array}{l} par\ 0 = \text{TRUE} \\ par\ (n+1) = impar\ n \end{array} \right.$$

Assumindo o functor $\mathsf{F}\ f = id + f$, mostre que esse par de definições é equivalente ao sistema de equações

Assuming the functor $\mathsf{F}\ f = id + f$, show that this pair of definitions is equivalent to the system of equations

$$\left\{ \begin{array}{l} impar \cdot \mathsf{in} = h \cdot \mathsf{F}\ \langle impar, par \rangle \\ par \cdot \mathsf{in} = k \cdot \mathsf{F}\ \langle impar, par \rangle \end{array} \right.$$

para um dado h e k (deduza-os). De seguida, recorra às leis da recursividade mútua e da troca para mostrar que

for a given h and k (calculate these). Then use the mutual recursion and exchange laws to show that

$$imparpar = \langle impar, par \rangle = \mathsf{for\ swap}\ (\text{FALSE}, \text{TRUE})$$

---

4. A seguinte função em Haskell lista os primeiros n números naturais por ordem inversa:

The following Haskell function lists the n first natural numbers in reverse order:

$$\left\{ \begin{array}{l} insg\ 0 = [\,] \\ insg\ (n+1) = (n+1) : insg\ n \end{array} \right.$$

Mostre que $insg$ pode ser definida por recursividade mútua tal como se segue:

Show that $insg$ can be defined by mutual recursion as follows:

$$\begin{cases} insg\ 0 = [\,] \\ insg\ (n+1) = (fsuc\ n) : insg\ n \end{cases}$$
$$\begin{cases} fsuc\ 0 = 1 \\ fsuc\ (n+1) = fsuc\ n + 1 \end{cases}$$

A seguir, usando a lei de recursividade mútua, derive:

Then, using the law of mutual recursion, derive:

$$insg = \pi_2 \cdot insgfor$$
$$insgfor = \mathsf{for}\ \langle (1+) \cdot \pi_1, \mathsf{cons} \rangle\ (1, [\,])$$

---

5. Considere o par de funções mutuamente recursivas

Consider the pair of mutually recursive functions

$$\begin{cases} f_1\ [\,] = [\,] \\ f_1\ (h : t) = h : (f_2\ t) \end{cases} \qquad \begin{cases} f_2\ [\,] = [\,] \\ f_2\ (h : t) = f_1\ t \end{cases}$$

Mostre por recursividade mútua que $\langle f_1, f_2 \rangle$ é um catamorfismo de listas (onde $\mathsf{F}\ f = id + id \times f$) e desenhe o respectivo diagrama. Que faz cada uma destas funções f_1 e f_2?

Show by mutual recursion that $\langle f_1, f_2 \rangle$ is a list catamorphism (for $\mathsf{F}\ f = id + id \times f$) and draw the corresponding diagram. What do functions f_1 and f_2 actually do?

---

6. 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 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: **data** BTree a = $Empty \mid Node\ (a, (\mathsf{BTree}\ a, \mathsf{BTree}\ a))$

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

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

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

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

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

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

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

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

- $zeros = (\!| g |\!)$ — substitui todas as folhas de uma árvore de tipo (6b) por zero.
- $conta = (\!| g |\!)$ — conta o número de nós de uma árvore de tipo (6a).
- $mirror = (\!| g |\!)$ — espelha uma árvore de tipo (6b), i.e., roda-a de 180º.
- $converte = (\!| g |\!)$ — converte árvores de tipo (6c) em árvores de tipo (6a) eliminando os Bs que estão na primeira.
- $vars = (\!| g |\!)$ — lista as variáveis de uma árvore expressão de tipo (6d).

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

- $zeros = (\!| g |\!)$ *— replaces all leaves of a type tree (6b) with zero.*
- $count = (\!| g |\!)$ *— counts the number of nodes of a type tree (6a).*
- $mirror = (\!| g |\!)$ *— mirrors a tree of type (6b), i.e. rotates it 180º.*
- $convert = (\!| g |\!)$ *— converts trees of type (6c) into trees of type (6a) eliminating the Bs from the first one.*
- $vars = (\!| g |\!)$ *— list the variables of an expression tree of type (6d).*

---

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

Unfold catamorphism $vars$ (exercise 6) towards a function in Haskell without any pointfree combinator.

---

8. Qualquer função $k = \mathsf{for}\ f\ i$ pode ser codificada em sintaxe C escrevendo

Any function $k = \mathsf{for}\ f\ i$ can be encoded in the syntax of C by writing:

```
int k(int n) {
  int r=i;
  int j;
  for (j=1;j<n+1;j++) {r=f(r);}
  return r;
};
```

Escreva em sintaxe C as funções $(a*) = \mathsf{for}\ (a+)\ 0$ e outros catamorfismos de naturais de que se tenha falado nas aulas da UC.

Encode function $(a*) = \mathsf{for}\ (a+)\ 0$ in C and other catamorphisms that have been discussed in the previous classes.