

Cálculo de Programas

2.º ano das Licenciaturas em
Engenharia Informática (LEI) e Ciências da Computação (LCC)
da Universidade do Minho

2009/10 - Ficha nr.º 10

1. Defina como um catamorfismo a função seguinte, extraída do *Prelude* do Haskell,

```
concat :: [[a]] → [a]  
concat = foldr (+) []
```

e mostre que a propriedade

$$\text{length} \cdot \text{concat} = \text{sum} \cdot \text{map length} \quad (1)$$

se verifica, recorrendo às leis de *fusão- e absorção-cata*

$$f \cdot \langle h \rangle = \langle k \rangle \iff f \cdot h = k \cdot (\mathbf{F} f) \quad (2)$$

$$\langle h \rangle \cdot \mathbf{T} f = \langle h \cdot \mathbf{B}(f, \text{id}) \rangle \quad (3)$$

em que, para listas, se tem

$$\mathbf{B}(f, g) = \text{id} + f \times g \quad (4)$$

$$\mathbf{F} f = \mathbf{B}(\text{id}, f) \quad (5)$$

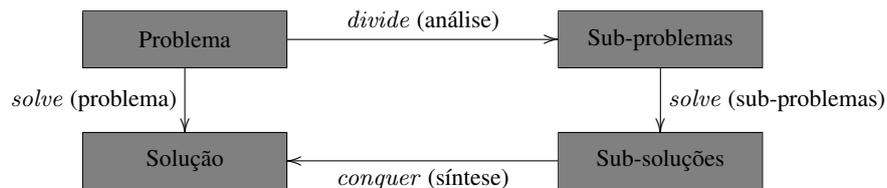
$$\mathbf{T} f = \text{map } f \quad (6)$$

2. A função correspondente a *concat* para árvores é

```
join :: LTree (LTree a) → LTree a  
join = \langle [id, Fork] \rangle
```

que junta uma *árvore de árvores* de tipo *LTree* numa só árvore. Conjecture a propriedade (1) para *join* e demonstre-a.

3. O desenho em baixo descreve aquela que é talvez a principal competência de um bom programador: a capacidade de dividir um problema complexo em partes e a de saber juntar as respectivas sub-soluções:



No Cálculo de Programas, o esquema desenhado acima é captado pelo diagrama de um *hilomorfismo*, cujo ingrediente principal é a fixação do padrão de organização das sub-soluções, captado pelo *functor* polinomial F :

$$\begin{aligned} \text{solve} &= \llbracket \text{conquer}, \text{divide} \rrbracket \\ &= (\text{conquer}) \cdot \llbracket \text{divide} \rrbracket \end{aligned} \quad \text{a que corresponde o diagrama}$$

$$\begin{array}{ccc} A & \xrightarrow{\text{divide}} & FA \\ \text{solve} \downarrow & & \downarrow F\text{solve} \\ B & \xleftarrow{\text{conquer}} & FB \end{array}$$

No quadro que se segue mostra-se a classificação de algumas funções conhecidas de acordo com o respectivo F :

Classe	$F X$	Serialização	Ordenação	Inversão	Factorial	Quadrado	Outros
Listas	$1 + A \times X$		<i>iSort</i>	<i>invl</i>	<i>fac</i>	<i>sq</i>	<i>look</i>
BTree	$1 + A \times X^2$	<i>in/pré/pós</i>	<i>qSort</i>				<i>hanoi, traces</i>
LTree	$A + X^2$	<i>tips</i>	<i>mSort</i>	<i>invLTree</i>	<i>dfac</i>	<i>dsq</i>	<i>fib</i>

- (a) Identifique a linha e coluna onde deve, do quadro acima, colocar o hilomorfismo de *bubble sorting*, identificando para ele os genes *divide* e *conquer*:

```

bSort :: Ord a => [a] -> [a]
bSort [] = []
bSort l = let (x, m) = bubble l
             in x : bSort m

bubble :: Ord a => [a] -> (a, [a])
bubble [x] = (x, [])
bubble (x : l) = let (y, m) = bubble l
                  in if x < y then (x, y : m) else (y, x : m)

```

- (b) Num dos outros algoritmos de ordenação do quadro acima (qual?), o passo de *divide* é a função *sep* que se segue

```

sep :: [a] -> ([a], [a])
sep = ([nil, nil], (cons × id) · assocl · (id × swap))]
  where nil _ = []
        cons (a, b) = a : b

```

cf. biblioteca `LTree.hs`. Mostre, usando a lei de recursividade múltipla para listas ($F f = id + id \times f$) que *sep* se poderia ter escrito (ao nível *pointwise*) sob a forma de duas funções mutuamente recursivas *odds* e *evens*,

```

sep :: [a] -> ([a], [a])
sep = (odds, evens) where
  odds [] = []
  odds (h : t) = h : (evens t)
  evens [] = []
  evens (h : t) = odds t

```

a primeira seleccionando os elementos que ocupam as posições ímpares e a segunda fazendo o mesmo para as pares.