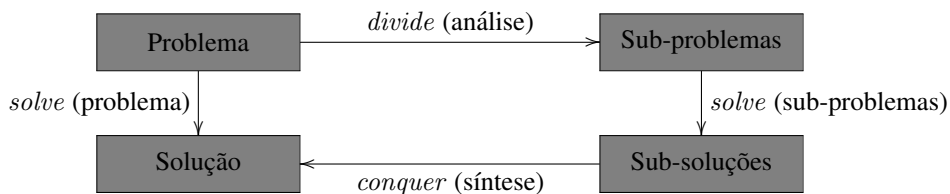


# Cálculo de Programas

2.º ano da Licenciatura em Engenharia Informática da  
Universidade do Minho

2010/11 - Ficha nr.º 10

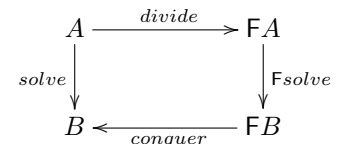
1. O desenho que se segue



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 para assim resolver o problema inicial.

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  $F$ :

$solve = \llbracket conquer, divide \rrbracket$   
 $= (\llbracket conquer \rrbracket) \cdot \llbracket divide \rrbracket$  a que corresponde o diagrama



isto é, tem-se a equação

$$solve = conquer \cdot F \text{ solve} \cdot divide \tag{1}$$

Mostre que

- para  $divide = \text{out}$  se tem  $solve = (\llbracket conquer \rrbracket)$
- para  $conquer = \text{in}$  se tem  $solve = \llbracket divide \rrbracket$ .

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

T	F X	Serialização	Ordenação	Inversão	Factorial	Quadrado	Outros
$\mathbb{N}_0$	$1 + X$						$(a^*)$ , $(\text{'div' } b)$
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</i> , <i>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) A divisão inteira  $x \text{ 'div' } y$  obtém-se da equação (1) acima definindo

```

div = flip aux where
  aux y = [conquer, divide y]
  conquer = in
  divide y x
    | x < y = i1 ()
    | x ≥ y = i2 (x - y)

```

Resolva a equação e introduza variáveis por forma a deduzir a versão habitual do algoritmo:

$$\begin{aligned}
 x \text{ 'div' } y \mid x < y &= 0 \\
 \mid x \geq y &= (x - y) \text{ 'div' } y + 1
 \end{aligned}$$

- (b) 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)

```

- (c) 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.