

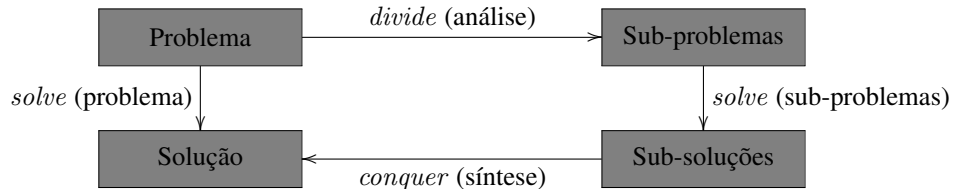
Cálculo de Programas

2.º ano

Lic. Ciências da Computação e Mestrado Integrado em Engenharia Informática
UNIVERSIDADE DO MINHO

2018/19 - Ficha nr.º 11

1. O desenho que se segue



descreve aquela que deve ser a principal competência de um programador: saber *dividir* um problema complexo em partes e juntar as respectivas sub-soluções para assim resolver o problema inicial.

No Cálculo de Programas, esse desenho é captado pelo conceito de *hilomorfismo*,

$$\begin{array}{ccc}
 A & \xrightarrow{\text{divide}} & F A \\
 \text{solve} \downarrow & & \downarrow F \text{ solve} \\
 B & \xleftarrow{\text{conquer}} & F B
 \end{array} \quad (F1)$$

$\text{solve} = \text{conquer} \cdot (F \text{ solve}) \cdot \text{divide}$

tendo-se¹

$$\begin{array}{ccc}
 A & \xrightarrow{\text{divide}} & F A \\
 \downarrow \llbracket \text{divide} \rrbracket & & \downarrow F \llbracket \text{divide} \rrbracket \\
 T & \begin{array}{c} \xrightarrow{\text{out}} \\ \cong \\ \xleftarrow{\text{in}} \end{array} & F T \\
 \downarrow \llbracket \text{conquer} \rrbracket & & \downarrow F \llbracket \text{conquer} \rrbracket \\
 B & \xleftarrow{\text{conquer}} & F B
 \end{array} \quad (F2)$$

$\text{solve} = \llbracket \text{conquer} \rrbracket \cdot \llbracket \text{divide} \rrbracket$

onde T é o tipo indutivo que tem F como base.² Apresente justificações para os passos seguintes da

¹Normalmente escreve-se $\text{solve} = \llbracket \text{conquer}, \text{divide} \rrbracket$.

²Esta estrutura intermédia é designada normalmente por estrutura de dados **virtual** por “não ser ver” quando o algoritmo executa, ficando escondida no ‘heap’ do sistema de ‘run-time’ da linguagem recursiva que está a ser utilizada.

demonstração do princípio da *hilo-factorização*, isto é, da equivalência entre (F1) e (F2):

$$\begin{aligned}
& solve = (\llbracket conquer \rrbracket) \cdot (\llbracket divide \rrbracket) \\
\equiv & \{ \dots\dots\dots \} \\
& solve = (conquer \cdot F(\llbracket conquer \rrbracket) \cdot out) \cdot (in \cdot F(\llbracket divide \rrbracket) \cdot divide) \\
\equiv & \{ \dots\dots\dots \} \\
& solve = conquer \cdot F(\llbracket conquer \rrbracket) \cdot F(\llbracket divide \rrbracket) \cdot divide \\
\equiv & \{ \dots\dots\dots \} \\
& solve = conquer \cdot F(\llbracket conquer \rrbracket \cdot (\llbracket divide \rrbracket)) \cdot divide \\
\equiv & \{ \dots\dots\dots \} \\
& solve = conquer \cdot F solve \cdot divide
\end{aligned}$$

□

2. O algoritmo de “quick-sort” foi definido nas aulas teóricas como o hilomorfismo $qSort = (\llbracket inord \rrbracket) \cdot (\llbracket qsep \rrbracket)$ sobre árvores BTree, cujo catamorfismo recorre à função:

$$inord = [\text{nil}, f] \text{ where } f(x, (l, r)) = l \uparrow [x] \uparrow r$$

Para um certo isomorfismo α , $h = (\llbracket inord \cdot \alpha \rrbracket) \cdot (\llbracket qsep \rrbracket)$ ordenará a lista de entrada por ordem inversa.

(a) Identifique α , justificando informalmente. (b) Seja agora $h' = (\llbracket inord \rrbracket) \cdot (\llbracket \alpha \cdot qsep \rrbracket)$. Demonstre que $h' = h$.

3. Um mónade é um functor T equipado com duas funções μ e u ,

$$A \xrightarrow{u} T A \xleftarrow{\mu} T(T A)$$

que satisfazem (para além das naturais, ie. “grátis”) as propriedades

$$\mu \cdot u = id = \mu \cdot T u$$

$$\mu \cdot \mu = \mu \cdot T \mu$$

— identifique-as no formulário — com base nas quais se pode definir a *composição monádica*:

$$f \bullet g = \mu \cdot T f \cdot g.$$

(Identifique-a também no formulário.) Demonstre os factos seguintes:

$$\mu = id \bullet id \tag{F3}$$

$$f \bullet u = f \quad \wedge \quad f = u \bullet f \tag{F4}$$

$$(f \cdot g) \bullet h = f \bullet (T g \cdot h) \tag{F5}$$

$$T f = (u \cdot f) \bullet id \tag{F6}$$

4. A função $discollect : (A \times B^*)^* \rightarrow (A \times B)^*$ que apareceu (sem ser definida) numa questão da ficha nº2 não é mais do que

$$discollect = lstr \bullet id \tag{F7}$$

— onde $lstr(a, x) = [(a, b) \mid b \leftarrow x]$ — no mónade das listas, $T A = A^*$,

$$A \xrightarrow{singl} A^* \xleftarrow{concat} (A^*)^*$$

onde $u = singl$ e $\mu = concat = (\llbracket [\text{nil}, \text{conc}] \rrbracket)$. Recordando a lei de absorção-cata (para listas), derive uma definição recursiva para $discollect$ que não use nenhum dos combinadores ‘point-free’ estudados nesta disciplina.