Cálculo de Programas

2.° ano das Licenciaturas em
Engenharia Informática e Ciências da Computação
UNIVERSIDADE DO MINHO

2014/15 - Ficha nr.° 9

1. Recorra à lei da absorção-cata, entre outras, para verificar a seguintes propriedades sobre listas

$$\mathsf{length} = \mathsf{sum} \cdot (\mathsf{map}\ (\underline{1})) \tag{F1}$$
$$\mathsf{length} = \mathsf{length} \cdot (\mathsf{map}\ f) \tag{F2}$$

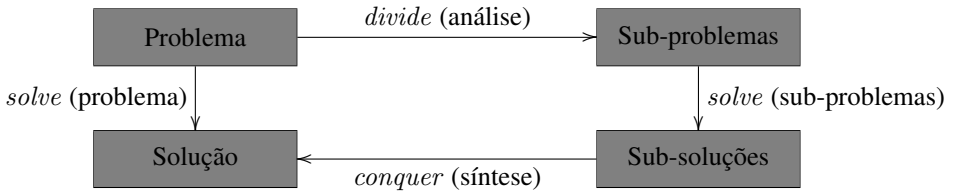
   onde length, sum e map são catamorfismos de listas que conhece, e sabendo que o functor de base para listas é $\mathsf{B}\ (f, g) = id + f \times g$.

2. Na sequência da questão anterior, mostre que $\mathsf{map}\ f$ tem as propriedades de functor, isto é:

$$\mathsf{map}\ id = id \tag{F3}$$
$$(\mathsf{map}\ f) \cdot (\mathsf{map}\ g) = \mathsf{map}\ (f \cdot g) \tag{F4}$$

   Sugestão: recorra à absorção-cata na prova da segunda propriedade.

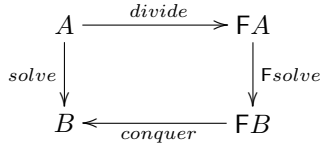
3. O desenho que se segue



   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 conceito de *hilomorfismo*,

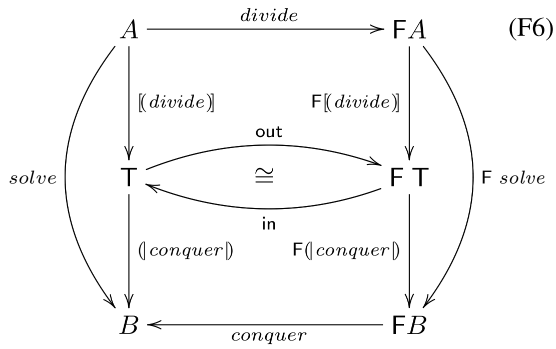
$$solve = conquer \cdot (\mathsf{F}\ solve) \cdot divide \tag{F5}$$



   que normalmente se escreve $solve = [\![conquer, divide]\!]$ e se factoriza na composição do anamor-

fismo $[\![(divide)]\!]$ com o catamorfismo $(\![conquer]\!)$,

$$[\![conquer, divide]\!] = (\![conquer]\!) \cdot [\![(divide)]\!]$$



(F6)

mediados por uma estrutura de dados do tipo T associado ao functor F.[1]

(a) Apresente justificações para os passos seguintes da demonstração do princípio da *hilo-factorização*, isto é, da equivalência entre (F5) e (F6):

$$
\begin{aligned}
& solve = [\![conquer, divide]\!] \\
\equiv \quad & \{ \quad \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots \quad \} \\
& solve = (\![conquer]\!) \cdot [\![(divide)]\!] \\
\equiv \quad & \{ \quad \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots \quad \} \\
& solve = (conquer \cdot \mathsf{F}\ (\![conquer]\!) \cdot \mathsf{out}.) \cdot (\mathsf{in} \cdot \mathsf{F}\ [\![(divide)]\!] \cdot divide) \\
\equiv \quad & \{ \quad \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots \quad \} \\
& solve = conquer \cdot \mathsf{F}\ (\![conquer]\!) \cdot \mathsf{F}\ [\![(divide)]\!] \cdot divide \\
\equiv \quad & \{ \quad \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots \quad \} \\
& solve = conquer \cdot \mathsf{F}\ ((\![conquer]\!) \cdot [\![(divide)]\!]) \cdot divide \\
\equiv \quad & \{ \quad \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots \quad \} \\
& solve = conquer \cdot \mathsf{F}\ solve \cdot divide
\end{aligned}
$$

(b) Desenhe o diagrama (F6) correspondente ao hilomorfismo $f = [\![[g\ , h], (p \rightarrow i_1, (i_2 \cdot k))]\!]$, identificando o tipo T. Mostre ainda que f satisfaz a propriedade seguinte:

$$f = p \rightarrow g, (h \cdot f \cdot k)$$

4. As chamadas “rose trees”, que se podem definir em Haskell por

$$\mathbf{data}\ \mathsf{Rose}\ a = \mathsf{Rose}\ (a, [\mathsf{Rose}\ a])$$

são árvores generalizadas em que cada nó tem um número arbitrário (mas finito) de sub-árvores. Defina para este tipo

- os isomorfismos in e out que o caracterizam
- o functor de base B e o da recursividade F
- o catamorfismo que conta o número de nós de uma “rose tree”.

[1] Esta 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.