Cálculo de Programas

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

2022/23 - Ficha nr.° 1 (revisões de PF)

1. A **composição** de funções define-se, em Haskell, tal como na matemática:

$$(f \cdot g)\ x = f\ (g\ x) \tag{F1}$$

   (a) Calcule $(f \cdot g)\ x$ para os casos seguintes:

$$\begin{cases} f\ x = 2 * x \\ g\ x = x + 1 \end{cases} \quad \begin{cases} f = \mathsf{succ} \\ g\ x = 2 * x \end{cases} \quad \begin{cases} f = \mathsf{succ} \\ g = \mathsf{length} \end{cases} \quad \begin{cases} g\ (x, y) = x + y \\ f = \mathsf{succ} \cdot (2*) \end{cases}$$

   Anime as composições funcionais acima num interpretador de Haskell.

   (b) Mostre que $(f \cdot g) \cdot h = f \cdot (g \cdot h)$, quaisquer que sejam f, g e h.

   (c) A função $id :: a \to a$ é tal que $id\ x = x$. Mostre que $f \cdot id = id \cdot f = f$ qualquer que seja f.

2. Recorde o *problema do telemóvel antigo* da primeira aula teórica,

   *(...) For each **list of calls** stored in the mobile phone (eg. numbers dialled, SMS messages, lost calls), the **store** operation should work in a way such that **(a)** the more recently a **call** is made the more accessible it is; **(b)** no number appears twice in a list; **(c)** only the most recent 10 entries in each list are stored.*

   para o qual se propôs a seguinte solução, que usa a composição de funções, uma por cada requisito do problema :

$$store\ c = \underbrace{\mathsf{take}\ 10}_{(c)} \cdot \underbrace{(c{:})}_{(a)} \cdot \underbrace{\mathsf{filter}\ (\neq c)}_{(b)} \tag{F2}$$

   (a) Usando a definição (**??**) tantas vezes quanto necessário, avalie as expressões

$$store\ 7\ [1\,..\,10]$$
$$store\ 11\ [1\,..\,10]$$

   (b) Suponha que alguém usou a mesma abordagem ao problema, mas enganou-se na ordem das etapas:

$$store\ c = (c{:}) \cdot \mathsf{take}\ 10 \cdot \mathsf{filter}\ (\neq c)$$

   Qual é o problema desta solução? Que requisitos (a,b,c) viola?

   (c) E se o engano for como escreve a seguir?

$$store\ c = \mathsf{filter}\ (\neq c) \cdot (c{:}) \cdot \mathsf{take}\ 10$$

   Conclua que a composição não é mesmo nada comutativa — a ordem entre as etapas de uma solução composicional é importante!

(d) Voltando a agora à definição *certa* (**??**), suponha que submete ao seu interpretador de Haskell a expressão:

$$\textit{store}\ \texttt{"Maria"}\ [\texttt{"Manuel"}, \texttt{"Tia Irene"}, \texttt{"Maria"}, \texttt{"Augusto"}]$$

Que espera do resultado? Vai dar erro? Tem que mexer no código para funcionar? Que propriedade da linguagem é evidenciada neste exemplo?

3. Complete a codificação abaixo (em Haskell) das funções $\mathsf{length} :: [a] \to \mathbb{Z}$ e $\mathsf{reverse} :: [a] \to [a]$ que conhece da disciplina de Programação Funcional (PF) e que, respectivamente, calculam o comprimento da lista de entrada e a invertem:

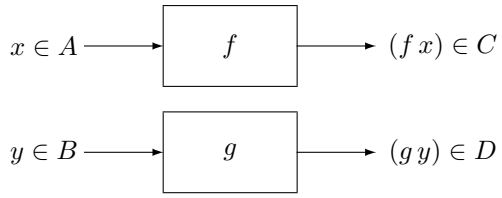
$$\begin{array}{ll} \mathsf{length}\ [\,] & = \ldots \\ \mathsf{length}\ (x : xs) & = \ldots \\ \mathsf{reverse}\ [\,] & = \ldots \\ \mathsf{reverse}\ (x : xs) & = \ldots \end{array}$$

4. Codifique em Haskell a função filter que foi usada na questão **??**.

5. Nas alíneas anteriores explorou-se o conceito de composição **sequencial**. Queremos agora um combinador que corra duas funções f e g em **paralelo**, isto é, ao mesmo tempo:

$$(f \times g)\ (x, y) = (f\ x, g\ y) \tag{F3}$$

cf. o diagrama de blocos:



(a) Demonstre as igualdades:

$$id \times id = id \tag{F4}$$

$$(f \times g) \cdot (h \times k) = f \cdot h \times g \cdot k \tag{F5}$$

(b) Suponha agora definidas as funções de projecção

$$\left\{ \begin{array}{l} \pi_1\ (x, y) = x \\ \pi_2\ (x, y) = y \end{array} \right. \tag{F6}$$

Demonstre as igualdades seguintes envolvendo esses operadores:

$$\pi_1 \cdot (f \times g) = f \cdot \pi_1 \tag{F7}$$

$$\pi_2 \cdot (f \times g) = g \cdot \pi_2 \tag{F8}$$

$$f \times g = \langle f \cdot \pi_1, g \cdot \pi_2 \rangle \tag{F9}$$

6. Apresente definições em Haskell das seguintes funções que estudou em PF:

$\mathsf{uncurry} :: (a \to b \to c) \to (a, b) \to c$ (que emparelha os argumentos de uma função)
$\mathsf{curry} :: ((a, b) \to c) \to a \to b \to c$ (que faz o efeito inverso da anterior)
$\mathsf{flip}\ \cdot :: (a \to b \to c) \to b \to a \to c$ (que troca a ordem dos argumentos de uma função)