

Ficha 1

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

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

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

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

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 $id :: a \rightarrow a$ é tal que $id x = x$. Mostre que $f \cdot id = id \cdot f = f$ qualquer que seja f .

1-

a) i) $(f \cdot g) x = f (g x) = f (x + 1) = 2(x + 1) = 2x + 2$

ii) $(f \cdot g) x = f (2x) = 2x + 1$

iii) $(f \cdot g) x = f (\text{length } x) = \text{length} + 1$

iv) $(f \cdot g) (x, y) = f (x + y) = \text{succ} (2x + 2y)$

b) $(f \cdot g) \cdot h = f \cdot (g \cdot h) \Leftrightarrow \checkmark(72)$

$\Rightarrow \forall x :: (f \cdot g) \cdot h (x) = f \cdot (g \cdot h) (x) \Leftrightarrow \checkmark(73)$

$\Rightarrow (f \cdot g) (h x) = f (g \cdot h x) \Leftrightarrow \checkmark(73)$

$\Rightarrow f (g (h x)) = f (g (h x)) \Leftrightarrow$

$\hookrightarrow \text{True}$

$$c) f \cdot id = id \cdot f = f \quad \downarrow \quad (72)$$

$$\forall x :: f \cdot id(x) = id \cdot f(x) = f(x) \quad \downarrow \quad (73)$$

$$f(id(x)) = id(f(x)) = f(x) \quad \downarrow \quad (74)$$

$$f(x) = f(x) = f(x)$$

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

(...) For each *list of calls* stored in the mobile phone (eg. numbers dialed, 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 :

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

(a) Usando a definição (F1) 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:

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

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

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

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

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* (F2), suponha que submete ao seu interpretador de Haskell a expressão:

$$store\ "Maria"\ ["Manuel", "Tia Irene", "Maria", "Augusto"]$$

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

Q- a) store $\text{filter } f [1..10]$ $\xrightarrow{\text{filter } \neq c} [1, 2, 3, 4, 5, 6, 8, 9, 10]$
 $\xrightarrow{c:} [7, 1, 2, 3, 4, 5, 6, 8, 9, 10] \xrightarrow{\text{take } 10} [7, 1, 2, 3, 4, 5, 6, 8, 9, 10]$

$\xrightarrow{c:} \text{Store } 11 [1..10] \xrightarrow{\text{filter } \neq c} [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11]$
 $\xrightarrow{c:} [11, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10] \xrightarrow{\text{take } 10} [11, 1, 2, 3, 4, 5, 6, 7, 8, 9]$

b) viola requisito c, lista fica com 11 elementos.

c) viola requisito a e c.

d) Q: ["Maria", "Manuel", "Tia Irene", "Augusto"]

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

$\text{length} [] = \dots$

$\text{length} (x : xs) = \dots$

$\text{reverse} [] = \dots$

$\text{reverse} (x : xs) = \dots$

3- $\text{length} [] = 0;$

$\text{length} (x : xs) = 1 + \text{length} xs;$

$\text{reverse} [] = [];$

$\text{reverse} (x : xs) = \text{reverse} xs : x$

4. Codifique em Haskell a função filter que foi usada na questão 2.

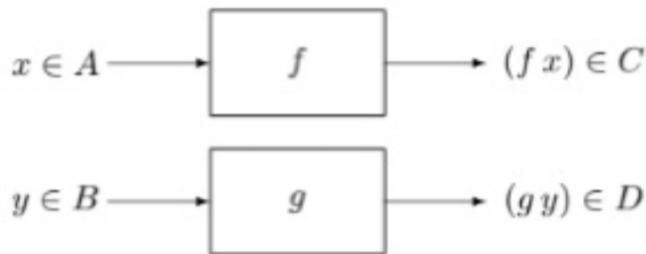
4- $\text{filter} (f) [] = [];$

$\text{filter} (f) (x : xs) = \text{if } f x \text{ then } x : \text{filter } f xs$
 $\text{else } \text{filter } f xs$

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:

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

cf. o diagrama de blocos:



(a) Demonstre as igualdades:

$$id \times id = id \quad (\text{F4})$$

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

(b) Suponha agora definidas as funções de projecção

$$\begin{cases} \pi_1(x, y) = x \\ \pi_2(x, y) = y \end{cases} \quad (\text{F6})$$

Demonstre as igualdades seguintes envolvendo esses operadores:

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

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

$$f \times g = \langle f \cdot \pi_1, g \cdot \pi_2 \rangle \quad (\text{F9})$$

5- F4) $\boxed{id \times id = id} \quad (\Rightarrow)$

\downarrow (72)

$\Rightarrow \forall x, y : id \times id(x, y) = id(x, y) \quad (\Rightarrow)$

\downarrow (78)

$\therefore (id x, id y) = id(x, y) \quad \downarrow$ (74)

$\therefore (x, y) = (x, y) \quad (\Rightarrow) \text{True}$

F5) $(f \times g) \cdot (h \times k) = \{ \cdot h \times g \cdot k \} \quad (\Rightarrow) \downarrow$ (72)

73,78 $\left\{ \begin{array}{l} \therefore \forall (x, y) : (f \times g) \cdot (h \times k)(x, y) = f \cdot h \times g \cdot k(x, y) \end{array} \right. \quad (\Rightarrow)$

78,73 $\left\{ \begin{array}{l} \therefore (f \times g)(h \times k(x, y)) = (f \cdot h(x), g \cdot k(y)) \end{array} \right.$

$$(\Leftarrow) f \times g (h(x), h(y)) = (f(h(x)), g(h(y))) \quad (\Leftarrow)$$

$$(\Leftarrow) ((h(x)), g(h(y))) = ((f(h(x)), g(h(y))))$$

b)

$$F7 - \Pi_1 \cdot (f \times g) = f \cdot \Pi_1 \quad (\Leftarrow) \quad) \quad (72)$$

$$(\Leftarrow) \forall x, y : \Pi_1 \cdot (f \times g)(x, y) = f \cdot \Pi_1(x, y) \quad (\Leftarrow)$$

$$(\Leftarrow) \Pi_1(f \times g(x, y)) = f(\Pi_1(x, y)) \quad (\Leftarrow) \quad) \quad (78, 80)$$

$$(\Leftarrow) \Pi_1(f(x), g(y)) = f(x) \quad (\Leftarrow) \quad) \quad (80)$$

$$(\Leftarrow) f(x) = f(x) \quad (\Leftarrow) \quad \text{True}$$

$$F8 - \Pi_2 \cdot (f \times g) = g \cdot \Pi_2 \quad (\Leftarrow) \quad) \quad (72)$$

$$(\Leftarrow) \forall x, y : \Pi_2 \cdot (f \times g)(x, y) = g \cdot \Pi_2(x, y) \quad (\Leftarrow)$$

$$(\Leftarrow) \Pi_2(f \times g(x, y)) = g(\Pi_2(x, y)) \quad (\Leftarrow) \quad) \quad (78, 80)$$

$$(\Leftarrow) \Pi_2(f(x), g(y)) = g(y) \quad (\Leftarrow) \quad) \quad (80)$$

$$(\Leftarrow) g(y) = g(y) \quad (\Leftarrow) \quad \text{True}$$

$$Fg - f \times g = \langle f \cdot \Pi_1, g \cdot \Pi_2 \rangle \Leftrightarrow \downarrow (72)$$

$$(\exists x, y :: f \times g(x, y) = \langle f \cdot \Pi_1, g \cdot \Pi_2 \rangle (x, y) \vdash) \quad (78, 77)$$

$$\text{④ } (f(x), g(y)) = (\epsilon \cdot \pi_1(x, y), g \cdot \pi_2(x, y)) \in J(73, 80)$$

$$(-) \quad (f(x), g(y)) = (f(x), g(y))$$

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

`uncurry :: ($a \rightarrow b \rightarrow c$) → (a, b) → c` (que emparelha os argumentos de uma função)

`curry :: ((a, b) → c) → a → b → c` (que faz o efeito inverso da anterior)

`flip ::(a → b → c) → b → a → c` (que troca a ordem dos argumentos de uma função)

6-

`Uncurry f (a,b) = f a b`

$$\text{curry } f \ a \ b = f(a,b)$$

$$\text{flip } f \circ b \circ a = f \circ a \circ b$$