Cálculo de Programas
Algebra of Programming

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

2023/24 - Ficha nr.° 1

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

   *Function **composition** is defined, in Haskell, just as in mathematics:*

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

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

Evaluate $(f \cdot g)\ x$ for the following cases:

$$\left\{ \begin{array}{l} f\ x = 2 * x \\ g\ x = x + 1 \end{array} \right. \quad \left\{ \begin{array}{l} f = \mathsf{succ} \\ g\ x = 2 * x \end{array} \right. \quad \left\{ \begin{array}{l} f = \mathsf{succ} \\ g = \mathsf{length} \end{array} \right. \quad \left\{ \begin{array}{l} g\ (x, y) = x + y \\ f = \mathsf{succ} \cdot (2*) \end{array} \right.$$

Anime as composições funcionais acima num interpretador de Haskell.

Animate the above functional compositions in a Haskell interpreter.

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

   Show that the equality $(f \cdot g) \cdot h = f \cdot (g \cdot h)$ holds for all f, g and h.

3. 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.

   The function $id :: a \rightarrow a$ is such that $id\ x = x$. Show that $f \cdot id = id \cdot f = f$ for all f.

4. Recorde o *problema do telemóvel antigo* da primeira aula teórica:

   Remember the old cell-phone problem *from the first theoretical class:*

> *(...) 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.*

Nessa aula foi proposta a seguinte solução, que usa a composição de funções, uma por cada requisito do problema:

In the class, the following solution was proposed, which uses function composition, one function for each requirement of the problem:

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

(a) Usando a definição (F1) tantas vezes quanto necessário, avalie as expressões

(a) Using definition (F1) as many times as needed, evaluate the expressions

$$store\ 7\ [1\mathinner{..}10]$$
$$store\ 11\ [1\mathinner{..}10]$$

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

Suppose someone used the same approach to the problem, but got the order of the steps wrong:

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

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

What is the problem with this solution? Which requirements (a,b,c) does it violate?

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

What if the mistake is as written below?

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

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

Conclude that composition is not commutative at all — the order between the steps of a compositional solution is important!

5. Voltando a agora à definição *certa* (F2), suponha que submete ao seu interpretador de Haskell a expressão:

   Returning to definition (F2), suppose you submit the following expression to your Haskell interpreter:

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

   Que espera do resultado? Vai dar erro? Tem que mexer em (F2) para funcinar? Que propriedade da linguagem é evidenciada neste exemplo?

   What is the outcome you expect? Will it be an error? Do I need to change (F2) for the above to work? What property of the Haskell programming language is made evident in this example?

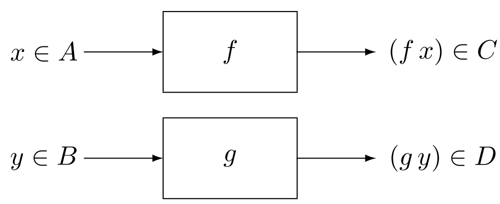
6. Nas alíneas anteriores explorou-se o conceito de composição **sequencial**. Queremos agora um combinador que corra duas funções f e g em **paralelo**, isto é, ao mesmo tempo:

   *In the previous questions, the concept of **sequential** composition was explored. We now want a combinator that runs two functions f and g in **parallel**, that is, at the same time:*

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

   cf. o diagrama de blocos:

   check the following block diagram:



   Demonstre as igualdades:

   Prove the following equalities:

$$id \times id = id \tag{F4}$$
$$(f \times g) \cdot (h \times k) = f \cdot h \times g \cdot k \tag{F5}$$

7. Supondo agora definidas as funções de projecção

   Assuming the following projection functions,

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

demonstre as igualdades seguintes envolvendo esses operadores:

prove the following equalities involving such operators:

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

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

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

   ***(Revision of functional programming background)** Complete the code below (in Haskell) of functions* length :: $[a] \rightarrow \mathbb{Z}$ *and* reverse :: $[a] \rightarrow [a]$ *that you know from the Functional Programming (PF) course and that, respectively, calculate the length of the input list and reverse it:*

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

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

9. (**Revisões de PF**) Apresente definições em Haskell das seguintes funções que estudou em PF:

   ***(Revision of functional programming background)** Give Haskell definitions for the following functions that you studied in the Functional Programming course (1st year):*

$$\begin{array}{l} \mathsf{uncurry}\ :: (a \rightarrow b \rightarrow c) \rightarrow (a, b) \rightarrow c \\ \mathsf{curry}\ :: ((a, b) \rightarrow c) \rightarrow a \rightarrow b \rightarrow c \\ \mathsf{flip}\ \cdot :: (a \rightarrow b \rightarrow c) \rightarrow b \rightarrow a \rightarrow c \end{array}$$