

Cálculo de Programas

Algebra of Programming

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

2024/25 - Ficha nr.º 1

Revisões de PF (*Functional Programming background*)

1. 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 invertem:

```
length [] = ...  
length (x : xs) = ...  
reverse [] = ...  
reverse (x : xs) = ...
```

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

2. A função $\text{take} :: \mathbb{Z} \rightarrow [a] \rightarrow [a]$ é tal que $\text{take } n \ x$ é o mais longo prefixo da lista x cujo comprimento não excede n . Complete a seguinte formulação de uma propriedade da função take :

$$\text{take } m (\text{take } n \ x) = \text{take } (m \dots n) \ x$$

The function $\text{take} :: \mathbb{Z} \rightarrow [a] \rightarrow [a]$ is such that $\text{take } n \ x$ is the longest prefix of list x whose length does not exceed n . Complete the following statement of a property of take :

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

```
map :: (a → b) → [a] → [b]  
filter :: (a → ℬ) → [a] → [a]  
uncurry :: (a → b → c) → (a, b) → c  
curry :: ((a, b) → c) → a → b → c  
flip · :: (a → b → c) → b → a → c
```

Give Haskell definitions for the following functions that you studied in the Functional Programming course:

4. A composição de funções define-se, em Haskell, tal como na matemática:	<i>Function composition is defined, in Haskell, just as in mathematics:</i>
$(f \cdot g) x = f (g x)$	(F1)
Calcule $(f \cdot g) x$ para os casos seguintes:	<i>Evaluate $(f \cdot g) x$ for the following cases:</i>
$\begin{cases} f x = 2 * x \\ g x = x + 1 \end{cases} \quad \begin{cases} f = \text{succ} \\ g x = 2 * x \end{cases} \quad \begin{cases} f = \text{succ} \\ g = \text{length} \end{cases} \quad \begin{cases} g (x, y) = x + y \\ f = \text{succ} \cdot (2*) \end{cases}$	
Anime as composições funcionais acima num interpretador de Haskell.	<i>Animate the above functional compositions in a Haskell interpreter.</i>
5. Mostre que $(f \cdot g) \cdot h = f \cdot (g \cdot h)$, quaisquer que sejam f, g e h .	<i>Show that the equality $(f \cdot g) \cdot h = f \cdot (g \cdot h)$ holds for all f, g and h.</i>
6. A função $\text{id} :: a \rightarrow a$ é tal que $\text{id } x = x$. Mostre que $f \cdot \text{id} = \text{id} \cdot f = f$ qualquer que seja f .	<i>The function $\text{id} :: a \rightarrow a$ is such that $\text{id } x = x$. Show that $f \cdot \text{id} = \text{id} \cdot f = f$ for all f.</i>
7. Considere o seguinte problema:	<i>Consider the following problem:</i>
<i>(...) For each list of calls stored in an old 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.</i>	
Considere ainda a seguinte proposta de resolução que usa a composição de funções, uma por cada requisito do problema:	<i>Further consider the following solution proposal which uses function composition, one function for each requirement of the problem:</i>
$\text{store } c = \underbrace{\text{take } 10}_{(c)} \cdot \underbrace{\text{nub}}_{(b)} \cdot \underbrace{(c:)}_{(a)}$	(F2)
(a) Usando a definição (F1) tantas vezes quanto necessário, avalie as expressões	<i>(a) Using definition (F1) as many times as needed, evaluate the expressions</i>
$\begin{aligned} \text{store } 7 [1..10] \\ \text{store } 11 [1..10] \end{aligned}$	
(b) Suponha que alguém usou a mesma abordagem ao problema, mas enganou-se na ordem das etapas:	<i>Suppose someone used the same approach to the problem, but got the order of the steps wrong:</i>
$\text{store } c = (c:) \cdot \text{take } 10 \cdot \text{nub}$	
Qual é o problema desta solução? Que requisitos (a,b,c) viola?	<i>What is the problem with this solution? Which requirements (a,b,c) does it violate?</i>
(c) E se o engano for como escreve a seguir?	<i>What if the mistake is as written below?</i>
$\text{store } c = \text{nub} \cdot (c:) \cdot \text{take } 10$	
Conclua que a composição não é mesmo nada comutativa — a ordem entre as etapas de uma solução composicional é importante!	<i>Conclude that composition is not commutative at all — the order between the steps of a compositional solution is important!</i>

8. 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 em (F2) para funcionar? Que propriedade da linguagem é evidenciada neste exemplo?

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

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?