

Cálculo de Programas

2.º ano

Lic. Ciências da Computação e Mestrado Integrado em Engenharia Informática
UNIVERSIDADE DO MINHO

2016/17 - Ficha nr.º 1 (revisões de PF)

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

$$(f \cdot g) x = f (g x)$$

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

$$f x = 2 * x, g x = x + 1$$

$$f = \text{succ}, g x = 2 * x$$

$$f = \text{succ}, g = \text{length}$$

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

- (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 .

2. Codifique em Haskell as funções $\text{length} :: [a] \rightarrow \text{Int}$ 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.

3. Recorde o tipo que se usa em Haskell para representar valores opcionais:

```
data Maybe a = Nothing | Just a
```

Defina a função $\text{catMaybes} :: [\text{Maybe } a] \rightarrow [a]$ que extrai o conteúdo útil da lista de entrada.

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

```
uncurry :: (a -> b -> 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)
```

5. Considere a seguinte declaração de um tipo de *árvores binárias*, em Haskell:

```
data LTree a = Leaf a | Fork (LTree a, LTree a)
```

Codifique as funções seguintes:

- $\text{flatten} :: \text{LTree } a \rightarrow [a]$ — que deverá dar a lista de elementos da árvore argumento
- $\text{mirror} :: \text{LTree } a \rightarrow \text{LTree } a$ — que deverá espelhar a árvore argumento
- $\text{fmap} :: (b \rightarrow a) \rightarrow \text{LTree } b \rightarrow \text{LTree } a$ — que deverá transformar cada folha x da árvore argumento na folha $f x$.

6. Atente na definição seguinte de um dos combinadores emblemáticos da linguagem Haskell, que já conhece de PF:

$$\begin{aligned} \text{foldr} &:: (a \rightarrow b \rightarrow b) \rightarrow b \rightarrow [a] \rightarrow b \\ \text{foldr } g \ z \ [] &= z \\ \text{foldr } g \ z \ (x : xs) &= x \ 'g'\ (\text{foldr } g \ z \ xs) \end{aligned}$$

- (a) Defina $\text{length} :: [a] \rightarrow \text{Int}$ usando foldr .
- (b) O que faz a função $f = \text{foldr } (\cdot) \ []$? **Sugestão:** comece por copiar a definição dada e faça literalmente as substituições $g := (\cdot)$ e $z := []$. De seguida substitua $\text{foldr } (\cdot) \ []$ por f . Obtem assim uma definição explícita de f , sem recorrer ao combinador foldr , que é mais fácil de inspeccionar.
7. Re-defina a função

$$\begin{aligned} \text{concat} &:: [[a]] \rightarrow [a] \\ \text{concat} &= \text{foldr } (+) \ [] \end{aligned}$$

sem recorrer ao combinador foldr (Sugestão: faça como na questão 6).

8. Diga por palavras suas o que faz a função

$$\begin{aligned} f &:: [\text{Int}] \rightarrow [\text{Int}] \\ f \ s &= [a + 1 \mid a \leftarrow s, a > 0] \end{aligned}$$

e escreva-a sob a forma de um foldr .

9. Considere a função m seguinte:

$$\begin{aligned} m &:: (a \rightarrow b) \rightarrow [a] \rightarrow [b] \\ m \ f \ [] &= [] \\ m \ f \ (h : t) &= (f \ h) : m \ f \ t \end{aligned}$$

- (a) Reescreva-a usando o combinador foldr .
- (b) Reescreva-a sem usar o combinador foldr .
- (c) Qual o tipo da expressão $m \ (\lambda x \rightarrow [x])$? E o que faz essa expressão?
- (d) Abreviando a função $\lambda x \rightarrow [x]$ pela designação *singl*, averigue qual o resultado das expressões

$$\text{let } s = m \ \textit{singl} \ \text{"Calculo de Programas"} \ \text{in } \text{concat } s$$

e

$$\text{concat } (\textit{singl} \ \text{"Calculo de Programas"})$$

correndo-as mentalmente. Tente generalizar o que apurou nesse exercício mental.