Cálculo de Programas

2.° ano das Licenciaturas em
Engenharia Informática e Ciências da Computação
UNIVERSIDADE DO MINHO

2014/15 - Ficha nr.° 1 (revisões de PF)

1. Codifique em Haskell as funções $\mathsf{length} :: [a] \to \mathsf{Int}$ 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.

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

   $$\mathbf{data}\ \mathsf{Maybe}\ a = \mathsf{Nothing} \mid \mathsf{Just}\ a$$

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

3. 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)
   $\mathsf{flip} :: (a \to b \to c) \to b \to a \to c$ (que troca a ordem dos argumentos de uma função)

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

   $$\mathbf{data}\ \mathsf{LTree}\ a = \mathsf{Leaf}\ a \mid \mathsf{Fork}\ (\mathsf{LTree}\ a, \mathsf{LTree}\ a)$$

   Codifique as funções seguintes:

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

5. 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:
   $f\ x = 2 * x$, $g\ x = x + 1$
   $f = \mathsf{succ}$, $g\ x = 2 * x$
   $f = \mathsf{succ}$, $g = \mathsf{length}$
   $g\ (x, y) = x + y$, $f = \mathsf{succ} \cdot (2*)$

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

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

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

   $$\begin{aligned}&\mathsf{foldr} :: (a \to b \to b) \to b \to [a] \to b\\ &\mathsf{foldr}\ g\ z\ [\,] \quad\ = z\\ &\mathsf{foldr}\ g\ z\ (x : xs) = x\ `g`\ (\mathsf{foldr}\ g\ z\ xs)\end{aligned}$$

(a) Defina length :: $[a] \to$ Int usando foldr.

(b) O que faz a função f = foldr (:) []? **Sugestão:** comece por copiar a definição dada e faça literalmente as substituições g := (:) e z := []. De seguida substitua foldr (:) [] por f. Obtém assim uma definição explícita de f, sem recorrer ao combinador dado, que é mais fácil de inspeccionar.

7. Re-defina a função

$$\begin{aligned}&\mathsf{concat} :: [[a]] \to [a]\\ &\mathsf{concat} = \mathsf{foldr}\ (+\!\!+)\ []\end{aligned}$$

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

8. Diga por palavras suas o que faz a função

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

e escreva-a sob a forma de um foldr.

9. Considere a função m seguinte:

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

(a) Reescreva-a usando o combinador foldr.

(b) Reescreva-a sem usar o combinador foldr.

(c) Qual o tipo da expressão $m\ (\lambda x \to [x])$? E o que faz essa expressão?

(d) Abreviando a funçao $\lambda x \to [x]$ pela designação $singl$, averigue qual o resultado das expressões

$$\mathbf{let}\ s = m\ singl\ \texttt{"Calculo de Programas"}\ \mathbf{in}\ \mathsf{concat}\ s$$

e

$$\mathsf{concat}\ (singl\ \texttt{"Calculo de Programas"})$$

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