Raciocinar sobre programas

9.1 Considere a definição recursiva dos naturais e da adição.

$$\text{data } \texttt{Nat} = \texttt{Zero} \ \mid \ \texttt{Succ}\,\texttt{Nat}$$

$$\begin{aligned} &\texttt{Zero} + y = y \\ &\texttt{Succ}\ x + y = \texttt{Succ}\ (x + y) \end{aligned}$$

Prove a associatividade da adição: $x + (y + z) = (x + y) + z$ para todo x, y, z.
Sugestão: use indução sobre x.

9.2 Usando indução sobre a lista xs, prove a associatividade da concatenação: $(xs{+\!\!+}ys){+\!\!+}zs = xs{+\!\!+}(ys{+\!\!+}zs)$.
Sugestão: a prova é análoga à do exercício anterior.

9.3 Prove a distributividade de `reverse` sobre $+\!\!+$:

$$\texttt{reverse}\ (xs{+\!\!+}ys) = \texttt{reverse}\ ys +\!\!+ \texttt{reverse}\ xs$$

usando indução sobre a lista xs.
Sugestão: será útil usar o resultado do exercício anterior (associatividade de $+\!\!+$). Tenha em atenção que as listas xs, ys aparecem por ordem contrária no lado direito da igualdade!

9.4 (T) Considere as definições das funções de ordem-superior `map` e $\circ$ (composição de duas funções):

$$\begin{aligned} &\texttt{map}\ f\ [\,] = [\,] \\ &\texttt{map}\ f\ (x : xs) = f\ x : \texttt{map}\ f\ xs \end{aligned}$$

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

Usando indução sobre listas, mostre que $\texttt{map}\ f\ (\texttt{map}\ g\ xs) = \texttt{map}\ (f \circ g)\ xs$.

9.5 (T) Usando as seguintes definições das funções `take`, `drop` $:: \mathit{Int} \to [a] \to [a]$ e a definição canónica de $+\!\!+$, mostre que $\texttt{take}\ n\ xs +\!\!+ \texttt{drop}\ n\ xs = xs$.

$$\begin{aligned} &\texttt{take}\ 0\ xs = [\,] \\ &\texttt{take}\ n\ [\,] \quad \mid \quad n > 0 = [\,] \\ &\texttt{take}\ n\ (x : xs) \quad \mid \quad n > 0 = x : \texttt{take}\ (n-1)\ xs \end{aligned}$$

$$\begin{aligned} &\texttt{drop}\ 0\ xs = xs \\ &\texttt{drop}\ n\ [\,] \quad \mid \quad n > 0 = [\,] \\ &\texttt{drop}\ n\ (x : xs) \quad \mid \quad n > 0 = \texttt{drop}\ (n-1)\ xs \end{aligned}$$

Sugestão: use indução sobre n e análise de casos da lista xs.

9.6 Usando indução sobre listas, prove que $\texttt{length}\ (\texttt{map}\ f\ xs) = \texttt{length}\ xs$, isto é, a função map preserva o comprimento da lista.

9.7 Usando indução sobre listas, prove que $\texttt{sum}\ (\texttt{map}\ (1+)\ xs) = \texttt{length}\ xs + \texttt{sum}\ xs$. Recorde que a notação $(1+)$ representa a função que adiciona 1 a um número.

9.8 Usando indução sobre listas, mostre que

$$\texttt{map}\ f\ (xs{+\!\!+}ys) = \texttt{map}\ f\ xs{+\!\!+}\texttt{map}\ f\ ys$$

para quaisquer funções f e listas finitas xs e ys.

9.9 Usando indução sobre listas, mostre que

$$\texttt{map}\ f\ (\texttt{reverse}\ xs) = \texttt{reverse}\ (\texttt{map}\ f\ xs)$$

Sugestão: use a propriedade provada no exerício 9.8.

9.10 Considere a seguinte definição da função $\texttt{inserir} :: \mathit{Int} \to [\mathit{Int}] \to [\mathit{Int}]$ que insere um valor numa lista crescente de inteiros mantendo a ordenação.

$$\begin{aligned}
&\texttt{inserir}\ x\ [\,] = [x]\\
&\texttt{inserir}\ x\ (y : ys)\ |\ x \leq y = x : y : ys\\
&\texttt{inserir}\ x\ (y : ys)\ |\ x > y = y : \texttt{inserir}\ x\ ys
\end{aligned}$$

Usando indução sobre listas, prove que $\texttt{length}\ (\mathit{insert}\ x\ xs) = 1 + \texttt{length}\ xs$.

9.11 Considere a declaração dum tipo recursivo para árvores binárias anotadas:

$$\text{data } \texttt{Arv}\ a = \texttt{Folha}\ \ |\ \ \texttt{No}\ a\ (\texttt{Arv}\ a)\ (\texttt{Arv}\ a)$$

Usando indução sobre árvores, mostre que o *número de folhas* é sempre mais um do que o *número de nós intermédios*.
Sugestão: começe por definir duas funções recursivas para calcular o número de folhas e nós intermédios duma árvore.

9.12 (T) Considere a função para listar por ordem os elementos duma árvore binária:

$$\begin{aligned}
&\texttt{listar} :: \texttt{Arv}\ a \to [a]\\
&\texttt{listar}\ \texttt{Folha} = [\,]\\
&\texttt{listar}\ (\texttt{No}\ x\ \mathit{esq}\ \mathit{dir}) = \texttt{listar}\ \mathit{esq} +\!\!+ [x] +\!\!+ \texttt{listar}\ \mathit{dir}
\end{aligned}$$

Empregue a técnica para eliminar concatenações apresentada na aula teórica para derivar uma versão mais eficiente desta função.
Sugestão: sintetize uma definição recursiva da função auxiliar $\texttt{listarAcc} :: \texttt{Arv}\ a \to [a] \to [a]$ tal que $\texttt{listarAcc}\ t\ xs = \texttt{listar}\ t +\!\!+ xs$.