Listas por compreensão

3.1 Defina uma função `divprop` $:: Integer \to [Integer]$ usando uma lista em compreensão para calcular a lista de *divisores próprios* de um inteiro positivo (i.e. inferiores ao número dado). Exemplo: `divprop` $10 = [1, 2, 5]$.

3.2 Um inteiro positivo n diz-se *perfeito* se for igual à soma dos seus divisores (excluindo o próprio n). Defina uma função `perfeitos` $:: Integer \to [Integer]$ que calcula a lista de todos os números perfeitos até um limite dado como argumento. Exemplo: `perfeitos` $500 = [6, 28, 496]$. *Sugestão:* utilize a solução do exercício 3.1.

3.3 Um trio (x, y, z) de inteiros positivos diz-se *pitagórico* se $x^2 + y^2 = z^2$. Defina a função `pitagoricos` $:: Integer \to [(Integer, Integer, Integer)]$ que calcule todos os trios pitagóricos cujas componentes não ultrapassem o argumento. Por exemplo: `pitagoricos` $10 = [(3, 4, 5), (4, 3, 5), (6, 8, 10), (8, 6, 10)]$.

3.4 Defina uma função `primo` $:: Integer \to Bool$ que testa primalidade: n é primo se tem exactamente dois divisores, a saber, 1 e n. *Sugestão*: utilize a função do exercício 3.1 para obter a lista dos divisores próprios.

3.5 Usando uma função `binom` da folha 1 que calcula coeficientes binomiais, escreva uma definição da função `pascal` $:: Integer \to [[Integer]]$ que calcula o triângulo de Pascal até à linha n:

$$\begin{array}{ccccc} & & \binom{0}{0} & & \\ & \binom{1}{0} & & \binom{1}{1} & \\ & \iddots & \vdots & \ddots & \\ \binom{n}{0} & \cdots & \binom{n}{k} & \cdots & \binom{n}{n} \end{array}$$

3.6 Podemos representar uma relação binária em conjuntos de inteiros como um par `Rel = ([Int], [(Int,Int)])`; o primeiro elemento do par é a lista dos inteiros no conjunto; o segundo elemento do par é a lista de pares na relação.

(a) Defina uma função `reflexiva :: Rel -> Bool` que verifica se a relação (V, R) é reflexiva, isto é, se $(x, x) \in R$ para todo $x \in V$.

(b) Defina uma função `simetrica :: Rel -> Bool` que verifica se a relação (V, R) é simétrica, isto é, se $(x, y) \in R \implies (y, x) \in R$.

(c) Defina uma função `transitiva :: Rel -> Bool` que verifica se a relação (V, R) é transitiva, isto é, se $(x, y) \in R \wedge (y, z) \in R \implies (x, z) \in R$.

Sugestão: cada uma destas funções se podem definir apenas numa linha usando listas em compreensão e funções do Prelúdio.

Definições recursivas

3.7 Escreva novas definições recursivas de funções equivalentes às do prelúdio de Haskell. Por exemplo: defina uma função `myand` equivalente a `and`, `myor` equivalente a `or`, etc.

(a) `and` $:: [Bool] \to Bool$ — testar se todos os valores são `True`;

(b) `or` $:: [Bool] \to Bool$ — testar se algum valor é `True`;

(c) `concat` $:: [[a]] \to [a]$ — concatenar uma lista de listas;

(d) `replicate` $:: Int \to a \to [a]$ — produzir uma lista com n elementos iguais;

(e) $(!!) :: [a] \to Int \to a$ — selecionar o n-ésimo elemento duma lista;

(f) `elem` $:: Eq\ a \Rightarrow a \to [a] \to Bool$ — testar se um valor ocorre numa lista.

3.8 Mostre que as funções do prelúdio-padrão `concat`, `replicate` e (!!) podem também ser definidas sem recursão usando listas em compreensão.

3.9 Defina uma função `forte` $:: String \to Bool$ para verificar se uma palavra-passe dada numa cadeia de carateres é forte segundo os seguintes critérios: deve ter 8 carateres ou mais e pelo menos uma letra maiúscula, uma letra minúscula e um algarismo.

Sugestão: use a função `or` $:: [Bool] \to Bool$ e listas em compreensão.

3.10 Neste exercício pretende-se implementar um teste de primalidade mais eficiente do que o do exercício 3.4.

(a) Escreva uma função `mindiv` $:: Int \to Int$ cujo resultado é o menor divisor próprio do argumento (i.e. o menor divisor superior a 1). Note que se $n = p \times q$, então p e q são ambos divisores de n; se $p \geq \sqrt{n}$, então $q \leq \sqrt{n}$ pelo que o menor divisor será sempre $\leq \sqrt{n}$. Assim *não necessitamos de tentar candidatos a divisores superiores à* $\sqrt{n}$.

(b) Utilize `mindiv` para definir um teste de primalidade mais eficiente do que o exercício 3.4: n é primo se $n > 1$ e o seu menor divisor próprio for igual a n.

3.11 A função `nub` $:: Eq\ a \Rightarrow [a] \to [a]$ do módulo *Data.List* elimina ocorrências de elementos repetidos numa lista ("nub" em inglês significa *essência*). Por exemplo: `nub` "banana" = "ban".

Escreva uma definição recursiva para esta função. Sugestão: use uma lista em compreensão com uma guarda para eliminar elementos duma lista.

3.12 Escreva uma definição da função `intersperse` $:: a \to [a] \to [a]$ do módulo *Data.List* que intercala um valor entre os elementos duma lista. Exemplo: `intersperse` '-' "banana" = "b-a-n-a-n-a".