Funções de ordem superior

3.1 Mostre como a lista em compreensão $[f\ x \mid x \leftarrow xs,\ p\ x]$ se pode escrever como combinação das funções de ordem superior *map* e *filter*.

3.2 Usando *foldl*, defina uma função $dec2int :: [Int] \to Int$ que converte uma lista de dígitos decimais num inteiro. Exemplo: $dec2int\ [2, 3, 4, 5] = 2345$.

3.3 A função $zipWith :: (a \to b \to c) \to [a] \to [b] \to [c]$ do prelúdio-padrão é uma variante de *zip* cujo primeiro argumento é uma função usada para combinar cada par de elementos. Podemos definir *zipWith* usando uma lista em compreensão:

$$zipWith\ f\ xs\ ys = [f\ x\ y \mid (x, y) \leftarrow zip\ xs\ ys]$$

Escreva uma definição recursiva de *zipWith*.

3.4 Mostre que pode definir função $isort :: Ord\ a \Rightarrow [a] \to [a]$ para ordenar uma lista pelo método de inserção (ver a Folha 3) usando *foldr* e *insert*.

3.5 As funções *foldl1* e *foldr1* do prelúdio-padrão são variantes de *foldl* e *foldr* que só estão definidas para listas com pelo menos um elemento (i.e. não-vazias). *Foldl1* e *foldr1* têm apenas dois argumentos (uma operação de agregação e uma lista) e o seu resultado é dado pelas equações seguintes.

$$foldl1\ (\oplus)\ [x_1, \ldots, x_n] = (\ldots(x_1 \oplus x_2)\ldots) \oplus x_n$$
$$foldr1\ (\oplus)\ [x_1, \ldots, x_n] = x_1 \oplus (\ldots(x_{n-1} \oplus x_n)\ldots)$$

(a) Mostre que pode definir as funções $maximum, minimum :: Ord\ a \Rightarrow [a] \to a$ do prelúdio-padrão (que calculam, respectivamente, o maior e o menor elemento duma lista não-vazia) usando *foldl1* e *foldr1*.

(b) Mostre que pode definir *foldl1* e *foldr1* usando *foldl* e *foldr*. Sugestão: utilize as funções *head*, *tail*, *last* e *init*.

3.6 A função de ordem superior $until :: (a \to Bool) \to (a \to a) \to a \to a$ está definida no prelúdio-padrão; $until\ p\ f$ é a função que repete sucessivamente a aplicação de f ao argumento até que que p seja verdade. Usando *until*, escreva uma definição não recursiva da função

$$mdc\ a\ b = \mathsf{if}\ b == 0\ \mathsf{then}\ a\ \mathsf{else}\ mdc\ b\ (a`mod`b)$$

que calcula o máximo divisor comum pelo algoritmo de Euclides.

3.7 Sem consultar a especificação do Haskell 98, escreva definições não-recursivas das seguintes funções do prelúdio-padrão:

(a) $(+\!\!+) :: [a] \to [a] \to [a]$, usando *foldr*;

(b) $concat :: [[a]] \to [a]$, usando *foldr*;

(c) $reverse :: [a] \to [a]$, usando *foldr*;

(d) $reverse :: [a] \to [a]$, usando *foldl*;

(e) $elem :: Eq\ a \Rightarrow a \to [a] \to Bool$, usando *any*.

3.8 Pretende-se que resolva este exercício sem usar *words* e *unwords* do prelúdio-padrão (pois *words* = *palavras* e *unwords* = *despalavras*).

(a) Escreva uma definição da função $palavras :: String \to [String]$ que decompõe uma linha de texto em palavras delimitadas por um ou mais espaços. Exemplo: *palavras* "Abra- ca- drabra!" = ["Abra-", "ca-", "dabra!"].

(b) Escreva uma definição da função $despalavras :: [String] \to String$ que concatena uma lista de palavras juntando um espaço entre cada uma. Note que *despalavras* não é a função inversa de *palavras*; encontre um contra-exemplo que justifique esta afirmação.

3.9 A função do prelúdio *scanl* é uma variante do *foldl* que produz a lista com os valores acumulados:

$$scanl\ f\ z\ [x_1, x_2, \ldots] = [z,\ f\ z\ x_1,\ f\ (f\ z\ x_1)\ x_2, \ldots]$$

Por exemplo:

$$scanl\ (+)\ 0\ [1,2,3] = [0,\ 0+1,\ 0+1+2,\ 0+1+2+3] = [0,1,3,6]$$

Em particular, para listas finitas xs temos que $last\ (scanl\ f\ z\ xs) = foldl\ f\ z\ xs$.

Escreva uma definição recursiva de *scanl*; deve usar outro nome para evitar colidir com a definição do prelúdio.

Lazy evaluation, listas e listas infinitas

3.10 Usando a definição da lista infinita *primos* apresentada nas aulas teóricas, escreva uma função $factores :: Int \to [Int]$ para factorizar um inteiro positivo em primos. Exemplo: $factores\ 100 = [2,2,5,5]$ porque $100 = 2 \times 2 \times 5 \times 5$.

3.11 Considere duas séries (i.e. somas infinitas) que convergem para π:

$$\pi = \frac{4}{1} - \frac{4}{3} + \frac{4}{5} - \frac{4}{7} + \cdots \tag{1}$$

$$\pi = 3 + \frac{4}{2\times3\times4} - \frac{4}{4\times5\times6} + \frac{4}{6\times7\times8} - \cdots \tag{2}$$

Escreva duas funções `calcPi1, calcPi2 :: Int -> Double` que calculam um valor aproximado de π usando o número de parcelas dado como argumento; investigue qual das séries converge mais depressa para π.

Sugestão: construa listas infinitas para os numeradores e denominadores dos termos separadamente e combine-as usando *zip*/*zipWith*.

3.12 Escreva uma definição recursiva duma função $intercalar :: a \to [a] \to [[a]]$ tal que $intercalar\ x\ ys$ que obtém todas formas possíveis de intercalar x com os elementos em ys. Exemplo: $intercalar\ 1\ [2,3] = [[1,2,3],\ [2,1,3],\ [2,3,1]]$.

3.13 Escreva uma definição da função $perms :: [a] \to [[a]]$ que obtém todas as permutações de uma lista (a ordem das permutações não é importante). Exemplo: $perms\ [1,2,3] = [[1,2,3],[2,1,3],[2,3,1],[1,3,2],[3,1,2],[3,2,1]]$.

Sugestão: utilize a função *intercalar* do exercício anterior para obter todas as formas de intercalar um elemento numa lista.

3.14 Podemos tornar a cifra de César da Folha 2 um pouco mais difícil de quebrar usando uma palavra chave em vez de um deslocamento único. Começamos por repetir a palavra-chave (por exemplo, "LUAR") ao longo do texto da mensagem; cada letra da chave de 'A' a 'Z' fazemos corresponder um índice de deslocamento de 0 a 25 (e.g., "LUAR" corresponde aos deslocamentos 11, 20, 0 e 17).

A	T	A	Q	U	E	D	E	M	A	D	R	U	G	A	D	A
L	U	A	R	L	U	A	R	L	U	A	R	L	U	A	R	L
L	N	A	H	F	Y	D	V	X	U	D	I	F	A	A	U	L

Escreva uma função $cifraChave :: String \to String \to String$ que implemente esta variante da cifra de César.

3.15 Neste exercício pretende-se definir o triângulo de Pascal completo como uma *lista infinita* $pascal :: [[Int]]$ das linhas do triângulo.

(a) Escreva uma definição de *pascal* usando a função *binom* do Exercício 5 da Folha 1. Note que que $pascal\,!!\,n\,!!\,k = binom\ n\ k$, para quaisquer n e k tais que $n > 0$ e $0 \leq k \leq n$.

(b) Escreva outra definição que evite o cálculo de factoriais usando as seguintes propriedades de coeficientes binomiais:

$$\binom{n}{0} = \binom{n}{n} = 1 \qquad \binom{n+1}{k+1} = \binom{n}{k} + \binom{n}{k+1} \quad (\text{se } n > k)$$