

Aula Teórico-prática 7

Programação Funcional

LEI 1º ano

Neste conjunto de exercícios vamos apresentar uma definição alternativa para números inteiros positivos usando listas de booleanos (bits). Sempre que achar apropriado use funções de ordem superior, nomeadamente a função `foldr`.

```
type Bit = Bool
bitToInt :: Bit -> Int
bitToInt False = 0
bitToInt True  = 1
```

```
intToBit :: Int -> Bit
intToBit 0 = False
intToBit 1 = True
```

Um número será então representado pela lista dos seus bits, do menos significativo para o mais significativo, i.e., o número 423, cuja representação binária é 110100111, será representado pela lista

```
[True, True, True, False, False, True, False, True, True]
```

Defina funções de conversão entre inteiros (não negativos) e listas de bits.

- `intToBList :: Int -> [Bit]`
- `bListToInt :: [Bit] -> Int`

A adição de números baseia-se na tabuada (da adição) que para o caso binário pode ser descrita pela seguinte função:

```
tabuada :: Bit -> Bit -> Bit -> (Bit, Bit) -- (res, carry)
tabuada False False False = (False, False)
tabuada False False True  = (True,  False)
tabuada False True  False = (True,  False)
tabuada True  False False = (True,  False)
tabuada True  False True  = (False, True )
tabuada True  True  True  = (False, True )
tabuada True  True  False = (False, True )
tabuada True  True  True  = (True,  True )
```

Usando esta função defina a função que calcula a soma de dois números.

- `soma :: [Bit] -> [Bit] -> [Bit]`

Defina agora a função de multiplicação de inteiros, usando um algoritmo semelhante ao que aprendeu para multiplicar números em base 10 (i.e., que envolve n adições quando o multiplicador tem $n + 1$ dígitos).

- `multiplica :: [Bit] -> [Bit] -> [Bit]`

Para testar as funções definidas acima podemos usar a seguinte função.

```
mult :: Int -> Int -> Int
mult x y = let bx = intToBList x
            by = intToBList y
            br = multiplica bx by
            in bListToInt br
```