Princípios de Programação

Exercícios

Universidade de Lisboa
Faculdade de Ciências
Departamento de Informática
Licenciatura em Engenharia Informática

2019/2020

XI Raciocínio sobre programas

1. Escreva leis que relacionem as funções abaixo com as duas funções que revelam a constituição de uma lista: `length` e `(!!)`.

   (a) `last :: [a] -> a`

   (b) `(++) :: [a] -> [a] -> [a]`

   (c) `map :: (a -> b) -> [a] -> [b]`

   (d) `tail :: [a] -> [a]`

   (e) `take :: Int -> [a] -> [a]`

   (f) `drop :: Int -> [a] -> [a]`

   (g) `init :: [a] -> [a]`

   (h) `zip :: [a] -> [b] -> [(a,b)]`

   (i) `unzip :: [(a,b)] -> ([a],[b])`

2. Considere a função `filter :: (a -> Bool) -> [a] -> [a]`.

   (a) Escreva uma lei que relacione a `filter` e `length`.

   (b) Escreva uma lei que relacione quaiquer dois elementos da lista devolvida por `filter`.

3. Mostre, recorrendo à definição, os seguintes resultados:

   (a) `length [x] = 1`

   (b) `[x] ++ xs = x:xs`

   (c) `reverse [x] = [x]`

4. Mostre que o operador `++` é
   (a) associativo e que
   (b) tem `[]` como elemento neutro.
5. Mostre por indução as seguintes equivalências:
   (a) `length (reverse xs)= length xs`
   (b) `reverse (xs ++ ys)= reverse ys ++ reverse xs`
   (c) `concat (xss ++ yss)= concat xss ++ concat yss`
   (d) `sum (xs ++ ys)= sum xs + sum ys`
   (e) `sum (reverse xs)= sum xs`
6. Mostre que para todas as listas finitas `ps` se tem:

```
zip (fst (unzip ps)) (snd (unzip ps)) = ps
```

7. Recorrendo ao princípio da extensionalidade, mostre que o operador composição, `.`, é associativo e que tem a função **`id`** (isto é `\x -> x`) como elemento neutro.
8. Mostre as seguintes leis sobre **`take`** e **`drop`**:
   (a) `take m . take n = take (m `min` n)`
   (b) `drop m . drop n = drop (m + n)`
   (c) `take m . drop n = drop n . take (m + n)`
9. Mostre as seguintes leis sobre **`map`** e **`filter`**:
   (a) `map f . map g = map (f . g)`
   (b) `map f . tail = tail . map f`
   (c) `map f . reverse = reverse . map f`
   (d) `map f . concat = concat . map (map f)`
   (e) `filter p . map f = map f . filter (p . f)`
   (f) `filter p . filter q = filter (p `and` q)`
   (g) `filter p . concat = concat . map (filter p)`