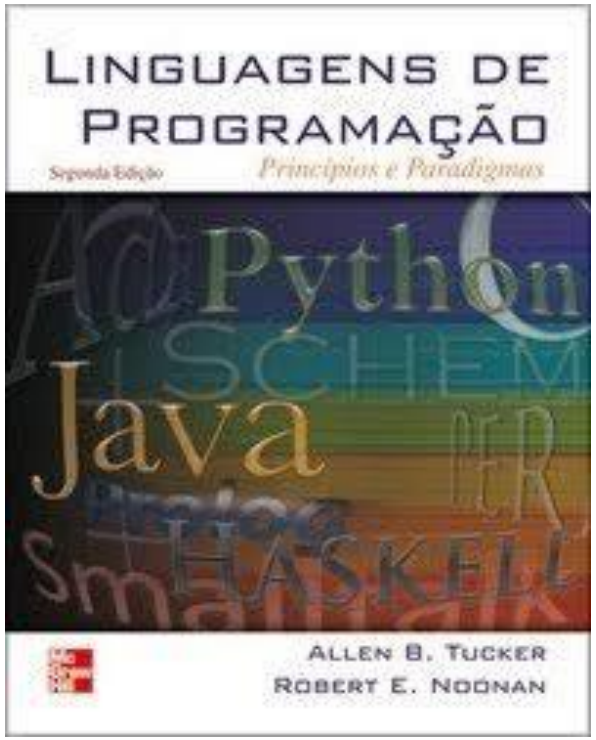
Paradigma Funcional
Listas em Haskell

Fabio Lubacheski
fabio.lubacheski@mackenzie.br

Leituras recomendadas



TUCKER, A. B.; NOONAN, R. E. **Linguagens de programação: Princípios e Paradigmas**. Capítulo 14 sessões 14.1 e 14.3

Listas em Haskell

- A estrutura de dados fundamental no de Haskell é a lista.
- Listas são coleções de elementos de um **mesmo tipo** e podem ser definidas pela enumeração de seus elementos, exemplo:

  ```
  [0,1,2,3,4,5,6,7,8,9,10]
  ```

  ou

  ```
  [0..10]
  ```

- A acima é definida pela convenção matemática familiar usando reticências (`..`), para omitir todos os elementos intermediários quando o padrão for óbvio.
- Abaixo como gerar uma lista de números pares de 0 até 10

  ```
  [0,2..10]
  ```

Listas em Haskell

- Alternativamente, uma lista pode ser definida por intermédio de algo chamado gerador:

```
[2*x | x <- [0..10]]
```

- Isso significa, literalmente, "*a lista de todos os valores* `2*x` *tais que* `x` *é um elemento* na lista `[0..10]`".
- O operador `<-` representa o símbolo matemático $\in$, que representa a participação na lista.

- Em Haskell podemos ter uma lista infinita, a lista não é criada é apenas uma **promessa** de lista, por exemplo:

```
[2*x | x <- [0,1..]]
```

Lazy Evaluation

- As listas infinitas são possíveis Haskell em razão dos seus parâmetros só serem avaliados quando for necessário (**avaliação lenta** ou **avaliação preguiçosa** ou **lazy evaluation**).

- As listas infinitas são armazenadas na forma não-avaliadas, ou seja, o n-ésimo elemento, não importa quão grande seja o valor de n, pode ser calculado sempre que for necessário.

- Os benefícios da avaliação preguiçosa incluem o aumento do desempenho ao evitar cálculos desnecessários, evitando condições de erro na avaliação de expressões compostas, a habilidade em construir estruturas de dados infinitas.

Geradores de listas com funções

- O gerador é muito parecido com a notação de conjuntos na matemática, anexando **condições** a definição da lista

  $\{ x \mid x \in \mathbb{N}, x \text{ é par} \}$

  ```
  [ x | x <- [0,1..], mod x 2 == 0]
  ```

- Além disso, o gerador pode se usado em funções. A função a seguir calcula os fatores (divisores) de um número:

  ```
  fatores n = [ f | f <- [1..n], mod n f == 0]
  ```

- A função pode ser lida assim: os fatores de `n` são todos os números `f` no intervalo de `1` até `n`, tal que `f` divide `n` e o resto seja igual a zero. Observe que a expressão

Funções e operações com listas

- Uma lista Haskell tem duas partes: o primeiro elemento ou a *head* da lista, e a listados demais elementos ou seu final (*tail*). As funções `head` e `tail` retornam essas duas partes, exemplo:

```
>head [0,1,2,3,4,5,6,7,8,9,10]
=> 0
>tail [0,1,2,3,4,5,6,7,8,9,10]
=> [1,2,3,4,5,6,7,8,9,10]
```

- A função especial `null` para testar quanto a uma lista vazia:

```
>null []
=> True
```

Funções e operações com listas

`++ ::` concatena duas listas.

```
[1,2]++[3,4]++[5] => [1,2,3,4,5]
```

`!! :: x !! n` obtem o n-ésimo elemento da lista x

```
[0,2,4,6,8,10] !! 5 => 10
```

`length ::` devolve número de elementos em uma lista

```
length [0,2,4,6,8,10] => 6
```

`take ::` devolve os n elementos do início de uma lista

```
take 4 [0,2,4,6,8,10] => [0,2,4,6]
```

`drop ::` retira n elementos do início de uma lista

```
drop 3 [0,2,4,6,8,10]  => [6,8,10]
```

Funções e operações com listas

`reverse ::` inverte a ordem dos elementos de uma lista
`reverse [0,2,4,6,8,10] => [10,8,6,4,2,0]`

`elem ::` verifica se um elemento ocorre em uma lista
`elem 8 [0,2,4,6,8,10] = True`

`sum ::` soma os elementos de uma lista
`sum [0,2,4,6,8,10] => 30`

`product ::` multiplica os elementos de uma lista
`product [1,3,5,7] => 105`

Funções com listas

- As funções apresentadas resolvem problemas com listas de **modo iterativo**, pois são funções internas do Haskell.
- Por exemplo, para escrever uma **função iterativa** que soma todos de uma lista seria, note que a função recebe uma lista como paramêtro:

```
soma x = sum x
```

- Um versão **recursiva** da função sem a utilização da função interna sum poderia ser assim:

```
soma [] = 0
soma (a:x) = a + soma x
```

Funções com listas

- Explicando a função recursiva `soma`:

```
soma [] = 0
soma (a:x) = a + soma x
```

A primeira linha define a função `soma` para o caso da base da recursão quando a lista está vazia é devolvido o valor `0`. A segunda linha define a função `soma` para os outros casos, nos quais a lista é não está vazia, a expressão `a:x` define um padrão que separa o `head` (`a`) do resto, `tail` (`x`) da lista. Abaixo uma outra forma de escrever a função recursiva:

```
somaIf x = if length x == 0
           then 0 else (head x ) + somaIf (tail x)
```

Exercicios

1) Escreva uma função que usa um gerador para verificar se um número é primo. A função recebe como argumento um número natural maior que 1, se o número informado é primo é devolvido `True` e caso contrário `False`.

2) Dada uma lista de inteiros, escreva uma função que calcule a multiplicação de todos os seus elementos. Não se utilize da função `product`.

3) Dada uma lista de inteiros e um valor inteiro `m`, escreva uma função que devolve uma lista com todos os elementos menores ou igual a `m`.

   Sugestão: use um gerador para resolver esse problema.

Exercicios

4) Dada uma lista de inteiros, escreva uma função que devolve uma lista com todos os elementos menores ou igual ao primeiro elemento da lista. (partição do QuickSort). Sugestão: use um gerador para resolver esse problema.

5) Dada uma lista de inteiros, escreva uma função que devolve a quantidade de elementos **pares** na lista.

6) Dada uma lista de inteiros, escreva uma função que devolve a média dos elementos **ímpares** na lista.

Pra finalizar .....

- Vocês acreditam que o QuickSort abaixo funciona?

```
particao1 p lista = [ menor | menor <- lista, menor <= p ]
particao2 p lista = [ maior | maior <- lista, maior > p ]

quicksort [] = []
quicksort (p:lista)=(quicksort (particao1 p  lista)) ++ [p]
++ (quicksort (particao2 p lista))
```

- Para testar ...

```
 quicksort [5,4,3,2,1]
=> [1,2,3,4,5]
```

Fim