CURSO: Ciência da Computação

PERÍODO: 4o.

DISCIPLINA: Técnicas Alternativas de Programação

CÓDIGO: CI062

PROFESSOR: Andrey

AULA: 18

1 Apresentação

Na aula de hoje vamos apresentar e discutir e exercitar os conceitos de manipulação de listas em Haskell. A aula de hoje é baseada em [de Sá et al., 2006]

2 Desenvolvimento

2.1 Listas

Haskell possui um mecanismo para a manipulação de listas incorporado á linguagem que favorece o seu uso em programas.

As listas em Haskell são uma coleção de objetos do mesmo tipo colocados em sequência. Para acessar um elemento da lista, você deve percorrer a lista até achá-lo.

Em Haskell, as listas são decompostas em duas partes básicas:

- cabeça (head) - primeiro elemento da lista;
- cauda (tail) - uma lista com os elementos restantes.

Por exemplo:

```
Letras :: [Char]
Letras = ['a', 'z', 'j', 'c']
```

```
Inteiros :: [Int]
Inteiros = [5,23,4,66]
```

Operador ':' - Define uma lista em termos de sua cabeça e sua cauda. Serve para separar o elemento inicial do resto da lista e para adicionar um elemento no início da lista.

Exemplo:

```
Prelude> 0:[1,2]
[0,1,2]
```

```
Prelude> [1,2,3] == (1:[2,3])
True
```

2.2 Exemplos de manipulação de listas

Comprimento de uma lista

```
comp [] = 0
comp (a:x) = 1 + comp x
```

Soma dos elementos de uma lista.

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

Retornar os 'n' primeiros elementos de uma lista.

```
primeiros 0 _ = []
primeiros _ [] = []
primeiros n (a:x) = a:(primeiros (n-1) x)
```

Verificar se um elemento pertence à lista.

```
pertence p [] = False
pertence p (a:x)
  | p == a = True
  | otherwise = a:(insere c x)
```

Encontrar o maior elemento da lista.

```
maior [a] = a
maior (a:c) | a > (maior c) = a
             | otherwise = maior c
```

2.3 Ordenação de uma lista

Para ordenar os elementos de uma lista vamos escolher o algoritmo Selection Sort. Para implementá-lo, vamos precisar das funções menor e remove_menor.

```
menor [a] = a
menor (a:c) | a < (menor c) = a
            | otherwise = menor c
```

```
remove_menor [a] = [ ]
remove_menor (a:c) | a == (menor (a:c)) = c
                   | otherwise = a: (remove_menor c)
```

Agora vamos implementar a função ordena do algoritmo Selection Sort.

```
ordena [ ] = [ ]
ordena [a] = [a]
ordena l = (menor l):(ordena (remove_menor l))
```

2.4 Compreensão de uma lista

A compreensão de uma lista é a especificação de uma lista através da sua definição matemática.

$$A = \{x^2 | x \in \mathbb{N} \wedge \text{par } x\}$$

Isto pode ser feito em Haskell da seguinte forma:

```
constroi_lista = [x*x | x <- [9 .. 39], even x]
```

2.5 Concatenação de listas

A concatenação de listas em Haskell pode ser feita usando-se o operador '++'. A segunda lista é conectada ao final da primeira. Exemplo:

```
Prelude> [2,5] ++ [9,8]
[2,5,9,8]
```

3 Atividades

- 1:** Construa uma função que inverta uma lista.
- 2:** Construa uma função que remova o primeiro elemento de uma lista.
- 3:** Construa uma função que remova o último elemento de uma lista.
- 4:** Construa uma função que some os elementos pares de uma lista.
- 5:** Construa uma função que some os elementos em posições pares de uma lista. (A primeira posição da lista é a posição 1).
- 6:** Construa uma função que receba uma lista e um número 'n' e insira um elemento na n-ésima posição da lista.

Referências

[de Sá et al., 2006] de Sá, C. C., da Silva, and Ferreira, M. (2006). *Haskell: uma abordagem prática*. Novatec Editora.