Haskell:
Valores e Tipos

UFRN, 2018

Valores e Tipos

- Em Haskell, computações são feitas pela avaliação de expressões e produção de valores.
- Valores:
  - Argumentos para funções, retorno como resultado, conteúdo de estruturas de dados, etc.
  - Cada valor tem um tipo associado.
- Expressões:
  - Valores atômicos, valores estruturados, funções.
  - Denotam valores ou tipos.
  - Expressões de tipo: Int, Char, Int->Int, [Int], (Char, Int)
- Sistema de tipos Haskell define a relação entre tipos e valores estaticamente.

Tipos Polimórficos

- Descrevem famílias de tipos.
- São como variáveis de tipos.
- Um exemplo de função polimórfica (contagem dos elementos de uma lista):

```
length :: [a] -> Integer
length []     =  0
length (x:xs) =  1 + length xs
```

- A função `length` pode ser aplicada a uma lista contendo elementos de qualquer tipo: [Integer], [Char], [[Integer]].

Tipos Definidos pelo Usuário

- Valores booleanos são um importante tipo predefinido em Haskell:

```
data Bool = False | True
```

  onde `Bool` é um construtor de tipo e `False` e `True` são construtores de dados.
- Podemos definir novos tipos em Haskell usando uma declaração semelhante:

```
data Color = Red | Green | Blue| Indigo
```

- `Bool` e `Color` são tipos enumerados.

Tipos Definidos pelo Usuário

- Tipos com um único construtor de dados, ex:

```
data Point a = Pt a a
```

- `Point` é um tipo polimórfico e o construtor de dados Pt é binário:

```
Pt :: a -> a -> Point a
Pt 2.0  3.0   :: Point Float
Pt 'a'  'b'   :: Point Char
Pt True False :: Point Bool
```

- Enquanto `Bool` e `Color` correspondem a *uniões*, `Point` corresponde a uma tupla.

Construtores

- Construtor de Tipos: avaliados em tempo de compilação e faz parte do sistema de tipos.
- Construtor de Dados: avaliados em tempo e execução.
- Como construtores de tipos e construtores de dados estão em espaços de nomes diferentes, podemos ter:

```
data Point a = Point a a
```

Tipos Recursivos

- Tipos podem ser recursivos, como em uma árvore binária:

```
data Tree a = Leaf a |
              Branch (Tree a) a (Tree a)
```

onde `Tree` é um construtor de tipo, `Leaf` e `Branch` são construtores de dados tais que:

```
Branch:: Tree a -> a -> Tree a -> Tree a
Leaf  :: a -> Tree a
```

Exemplo de Função com Tipo Recursivo

- Função que retorna uma lista com todos os elementos das folhas de uma árvore, da esquerda para a direita:

```
folhas :: Tree a -> [a]
folhas (Leaf x)                 = [x]
Folhas (Branch left _ right)= folhas left
                              ++
                              folhas right
```

Tipos Sinônimos

- Haskell fornece meios para definir tipos sinônimos, criados com a declaração type:

```
type String  = [Char]
type Person  = (Name,Address)
type Name    = String
data Address = None | Addr String
```

- Tipos sinônimos não definem novos tipos, simplesmente dão novos nomes para tipos já existentes.