

Tipos algébricos

- Novos tipos de dados
- Tipos algébricos
- Exemplos

Novos tipos de dados

- Tipos básicos:
 - Bool, Char, Int, Integer, Float, Double
- Tipos Compostos:
 - tuplas: (t1, t2, ..., tn)
 - listas: [t]
 - funções: $t1 \rightarrow t2$
- Novos tipos: como definir?
 - dias da semana, estações do ano, figuras geométricas, árvores, tipos cujos elementos são inteiros ou strings – ...

Novos tipos de dados

- Até agora foram apresentados vários tipos intrínsecos da linguagem, como valores booleanos, caracteres e números.
- Porém existem certos problemas computacionais que são mais difíceis de serem modelados com esses valores,
- como por exemplo, os meses, as cores, etc.
- Podemos deixar com os programadores a definição de novos tipos ?

Tipos algébricos

- Sim, podemos definir um tipo **Meses** da seguinte maneira:
- `data Meses = Jan | Fev | Mar | Abr | Mai | Jun | Jul | Ago | Set | Out | Nov | Dez`
- **Meses** é chamado de **construtor de tipo**.
- Ele é formado por doze valores que são chamados de **construtores (constructors) de dados**.
- Os construtores de dados são todos os valores que um tipo pode assumir.

Tipos algébricos

- Uma definição de tipo começa sempre com a palavra reservada **data**
- depois vem o **construtor de tipo** (nome do tipo **Meses**), que deve começar com letra maiúscula
 - note que todos os tipos em Haskell começam com letra maiúscula
- e seus **construtores de dados**
 - que também começam com letra maiúscula (...**Jun** | **Jul** | **Ago**...).

Tipos algébricos

- Um exemplo de tipo algébrico já conhecido é o tipo Bool:
 - `data Bool = True | False`
 - Qual o construtor do tipo?
 - Quais são os construtores de dados?
-
- Por isso que afirmamos que os valores True e False são Bool:
 - `True::Bool`
 - `False::Bool`

Tipos algébricos

- Também podemos criar um tipo algébrico com construtores de dados que tenham argumentos.
- Esses argumentos podem representar diversos tipos:
- Exemplo:
- `data Pessoas = Pessoa String Int`
- Onde `String` e `Int` são os argumentos do construtor de dados `Pessoa`
- `String` pode servir para representar um nome e `Int` para representar uma idade, por exemplo.

Tipos algébricos

- As funções que manipulam os tipos algébricos podem ser definidas por casamento de padrão:
- `mostraPessoa :: Pessoas -> String`
- `mostraPessoa (Pessoa nom idade) = "Nome: " ++ nom ++
" Idade: " ++ show idade`
- Exemplo de uso:
- `Haskell > mostraPessoa (Pessoa "Adian Ribeiro" 25)`

Nome: Adian Ribeiro Idade: 25

Tipos algébricos

- Também podemos definir o tipo Pessoas da seguinte maneira:
- `type Nome = String`
- `type Idade = Int`
- `data Pessoas = Pessoa Nome Idade`

Tipos algébricos

- Um construtor de dados é utilizado para
 - construir valores do tipo definido, funcionando como uma função (eventualmente, constante) que recebe argumentos (do tipo indicado para o construtor), e constrói um valor do novo tipo de dados;
 - decompor um valor do tipo em seus componentes, através de casamento de padrão
- Construtores de dados são funções especiais, pois não tem nenhuma definição (algoritmo) associada.

Tipos algébricos

- Uma declaração de tipo algébrico é da forma:

Nome do *construtor do tipo*,
iniciado por letra maiúscula.

Parâmetros (*variáveis de tipo*).

```
data Nome p1 ... pn = Construtor t1 ... tm | ...  
...  
| Construtor t1 ... tk
```

Nome dos *construtores de dados* do tipo
(ou simplesmente, construtores).
Iniciado por letra maiúscula.

Tipo dos argumentos de cada
construtor (*as variáveis de tipo dos
parâmetros podem ocorrer neste tipos*).

Exemplo: cor

- Definição de um novo tipo para representar cores:
- `data Cor = Azul | Amarelo | Verde | Vermelho`
- O construtor de tipo é `Cor`.
- A esse gênero de tipo algébrico damos o nome de tipo enumerado

Exemplo: cor

- Podemos agora definir funções envolvendo cores:

```
fria :: Cor -> Bool
fria Azul    = True
fria Verde   = True
fria _       = False
```

```
quente :: Cor -> Bool
quente Amarelo = True
quente Vermelho = True
quente _       = False
```

Exemplo: cor

- Exemplo de uso das funções:
- fria Amarelo ———> False
- fria Azul ———> True
- quente Amarelo ———> True
- quente Azul ———> False

Exemplo: cor

- Os construtores de dados deste tipo são:
- Azul :: Cor
- Amarelo :: Cor
- Verde :: Cor
- Vermelho :: Cor

Exemplo: booleanos

- O tipo Bool da biblioteca padrão é um tipo algébrico:
- `data Bool = True | False`
- O construtor de tipo é Bool.
- Os construtores de dados deste tipo são:
- `True :: Bool`
- `False :: Bool`
- O tipo Bool também é um exemplo de um tipo enumerado

Exemplo: booleanos

- Exemplo de uso do tipo:
- `data Bool = True | False`
- `not :: Bool -> Bool`
- `not True = False`
- `not False = True`

Exemplo: coordenadas cartesianas

- Definição de um novo tipo para representar coordenadas cartesianas:
- `data CCart = Coord Double Double`
- O construtor de tipo é `CCart`.
- O construtor de dados deste tipo pode ser visto como uma função:
- `Coord :: Double -> Double -> CCart`

Exemplo: coordenadas cartesianas

- Podemos agora definir funções envolvendo coordenadas:
- `data CCart = Coord Double Double`
- `somaVet :: CCart -> CCart -> CCart`
- `somaVet (Coord x1 y1) (Coord x2 y2) = Coord (x1+x2) (y1+y2)`

Exemplo: horário

- Definição de um novo tipo para representar horários:
- `data Horario = AM Int Int Int | PM Int Int Int`
- O construtor de tipo é `Horario`.
- Os construtores de dados do tipo `Horario` são:
- `AM :: Int -> Int -> Int -> Horario`
- `PM :: Int -> Int -> Int -> Horario`
- e podem ser vistos como uma etiqueta (tag) que indica de que forma os argumentos a que são aplicados devem ser entendidos.

Exemplo: horário

- Os valores AM 5 10 30 , PM 5 10 30 e (5,10,30) não contém a mesma informação.
- Os construtores AM e PM tem um papel essencial na interpretação que fazemos destes termos.
- Podemos agora definir funções envolvendo horários:
- `data Horario = AM Int Int Int | PM Int Int Int`
- `totalSegundos :: Horario -> Int`
- `totalSegundos (AM h m s) = (h*60 + m)*60 + s`
- `totalSegundos (PM h m s) = ((h+12)*60 + m)*60 + s`

Exemplo: formas geométricas

- Definição de um novo tipo para representar formas geométricas:
- `data Figura = Circulo Double`
 `| Retangulo Double Double`
- O construtor de tipo é `Figura`.
- Os construtores de dados deste tipo são:
- `Circulo :: Double -> Figura`
- `Retangulo :: Double -> Double -> Figura`

Exemplo: formas geométricas

- com eles é possível construir todo e qualquer valor do tipo Figura:
- `a :: Figura`
- `a = Circulo 2.3` -- um círculo de raio 2.3
- `b :: Figura`
- `b = Retangulo 2.8 3.1` -- um retângulo de base 2.8 e altura 3.1
- `lfig :: [Figura]`
- `lfig = [Retangulo 5 3, Circulo 5.7, Retangulo 2 2]`

Exemplo: formas geométricas

- Podemos definir funções envolvendo os tipos algébricos.

- ```
data Figura = Circulo Double
 | Retangulo Double Double
```

- ```
area :: Figura -> Double
```

- ```
area (Circulo r) = pi * r^2
```

- ```
area (Retangulo x y) = x * y
```

- ```
area (Circulo 2.5) ———> 19.634954084936208
```

- ```
area (Retangulo 2 5.1)  ———> 10.2
```


Exemplo: formas geométricas

- Podemos definir funções envolvendo os tipos algébricos.
- `data Figura = Circulo Double
 | Retangulo Double Double`
- `quadrado :: Double -> Figura`
- `quadrado lado = Retangulo lado lado`
- `area (quadrado 2.5) ——-> 6.25`

Exemplo: sentido de movimento

- Definição de um novo tipo para representar direções de movimento:
- `data Sentido = Esquerda | Direita | Acima | Abaixo`
- O construtor de tipo é `Sentido`.
- Os construtores de dados deste tipo, todos constantes, são:
- `Esquerda :: Sentido`
- `Direita :: Sentido`
- `Acima :: Sentido`
- `Abaixo :: Sentido`

Exemplo: sentido de movimento

- Quando os construtores de dados são constantes, (ou seja, não tem argumentos), dizemos que o tipo é uma enumeração.
- Neste exemplo os únicos valores do tipo Sentido são
- Direita, Esquerda, Acima e Abaixo.

Exemplo: sentido de movimento

- Podemos definir funções envolvendo o tipo algébrico:
- `data Sentido = Esquerda | Direita | Acima | Abaixo`
- `type Pos = (Double, Double)`
- `move :: Sentido -> Pos -> Pos`
- `move Esquerda (x,y) = (x-1,y)`
- `move Direita (x,y) = (x+1,y)`
- `move Acima (x,y) = (x ,y+1)`
- `move Abaixo (x,y) = (x ,y-1)`

Exemplo: sentido de movimento

- $\text{moves} :: [\text{Sentido}] \rightarrow \text{Pos} \rightarrow \text{Pos}$
- $\text{moves } [] \text{ } p = p$
- $\text{moves } (s:ss) \text{ } p = \text{moves } ss \text{ } (\text{move } s \text{ } p)$
- $\text{moves } [\text{Direita}, \text{Acima}, \text{Acima}, \text{Abaixo}, \text{Acima}, \text{Direita}, \text{Acima}]$
 $(0,0)$
 $\implies (2.0, 3.0)$

Exemplo: listas

- Um tipo algébrico pode ser polimórfico.
- O tipo $\text{Lista } a$ é um tipo algébrico polimórfico:
- $\text{data Lista } a = \text{Nil} \mid \text{Cons } a (\text{Lista } a)$
- Os construtores de dados são:
- $\text{Nil} :: \text{Lista } a$
um construtor constante representando a lista vazia
- $\text{Cons} :: a \rightarrow \text{Lista } a \rightarrow \text{Lista } a$
um construtor para listas não vazias, formadas por uma cabeça e uma cauda.

Exemplo: listas

- Exemplo:
- a lista do tipo `Lista Int` formada pelos elementos 3, 7 e 1 é representada por `Cons 3 (Cons 7 (Cons 1 Nil))`.
- O construtor de tipo `Lista` está parametrizado com uma variável de tipo a , que poderá ser substituída por um tipo qualquer.
- É neste sentido que se diz que `Lista` é um construtor de tipo.

Exemplo: listas

- Operações com lista:
- comprimento :: Lista $a \rightarrow$ Int
- comprimento Nil = 0
- comprimento (Cons _ xs) = 1 + comprimento xs

Exemplo: listas

- Operações com lista:
- $\text{elemento} :: \text{Eq } a \Rightarrow a \rightarrow \text{Lista } a \rightarrow \text{Bool}$
- $\text{elemento } _ \text{Nil} = \text{False}$
- $\text{elemento } x (\text{Cons } y \text{ xs}) = x == y \parallel \text{elemento } x \text{ xs}$

Exemplo: listas

- O tipo `Lista a` deste exemplo é similar ao tipo `[a]` da biblioteca padrão do Haskell:
- `data [a] = [] | a : [a]`
- Observe apenas que Haskell usa:
 - uma notação especial para o construtor de tipo: `[a]`
 - uma notação especial para o construtor de lista vazia: `[]`
 - um identificador simbólico com status de operador infix para o construtor de lista não vazia: `(:)`