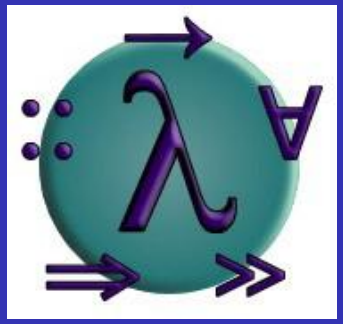
Programação Funcional



Capítulo 4

Expressões Condicionais

José Romildo Malaquias

Departamento de Computação
Universidade Federal de Ouro Preto

2012.1

Tópicos

Redefinindo funções do prelúdio

- O módulo `Prelude` é importado automaticamente em todos os módulos de uma aplicação em Haskell.
- Um nome que já tenha sido definido não pode ser redefinido.
- Como escrever uma definição usando um nome que já é utilizado em algum módulo?
- Podemos omitir alguns nomes ao importar um módulo, usando a declaração `import hiding`.

Redefinindo funções do prelúdio (cont.)

- **Exemplo**:
  Para fazermos nossas próprias definições de `even` e `odd`, que já são definidas no módulo `Prelude`:

```
import Prelude hiding (even, odd)

even n = mod n 2 == 0

odd n = not (even n)
```

Combinando funções

- A maneira mais simples de definir novas funções é simplesmente pela combinação de uma ou mais funções existentes.

Combinando funções (cont.)

- **Exemplo**:
  Verificar se um caracter é um dígito decimal:

```
isDigit  :: Char -> Bool
isDigit c = c >= '0' && c <= '9'
```

Combinando funções (cont.)

- **Exemplo**:
  Verificar se um número inteiro é par:

```
even :: Integral a => a -> Bool
even n = n `mod` 2 == 0
```

Combinando funções (cont.)

- **Exemplo**:
  Dividir uma lista em duas partes:

```
splitAt :: Int -> [a] -> ([a],[a])
splitAt n xs = (take n xs,drop n xs)
```

Combinando funções (cont.)

- **Exemplo**:
  Calcular o recíproco de um número:

```
recip :: Fractional a => a -> a
recip n = 1/n
```

Tópicos

Expressões condicionais

- Uma **expressão condicional** tem a forma

```
if condição then exp1 else exp2
```

onde *condição* é uma expressão booleana (chamada predicado) e exp_1 (chamada consequência) e exp_2 (chamada alternativa) são expressões de um mesmo tipo.

- O valor da expressão condicional é o valor de exp_1 se a condição é verdadeira, ou o valor de exp_2 se a condição é falsa.
- **Exemplos**:

```
if True then 1 else 2          ⇝ 1
if False then 1 else 2         ⇝ 2
if 2>1 then "OK" else "FAIL"   ⇝ "OK"
if even 5 then 3+2 else 3-2    ⇝ 1
```

Expressões condicionais (cont.)

- A expressão condicional é uma **expressão**, portanto sempre tem um valor.
- Assim uma expressão condicional pode ser usada dentro de outra expressão.
- **Exemplos**:

```
5 * (if True then 10 else 20)            ⇝ 50
5 * if True then 10 else 20              ⇝ 50
(if even 2 then 10 else 20) + 1          ⇝ 11
if even 2 then 10 else 20 + 1            ⇝ 10
length (if 2<=1 then "OK" else "FAIL")   ⇝ 4
```

- Observe que uma expressão condicional se extende à direita o quanto for possível.

Expressões condicionais (cont.)

- A cláusula `else` não é opcional em uma expressão condicional. Omiti-la é um erro de sintaxe.
- **Exemplo**:

```
if True then 10 ⇝ ERRO DE SINTAXE
```

- Se fosse possível omiti-la, qual seria o valor da expressão quando a condição fosse falsa?

Expressões condicionais (cont.)

- **Regra de inferência**:

$$\frac{test :: \text{Bool} \qquad e_1 :: a \qquad e_2 :: a}{\text{if } test \text{ then } e_1 \text{ else } e_2 :: a}$$

- Observe que a consequência e a alternativa devem ser do mesmo tipo, que também é o tipo do resultado.
- **Exemplos**:

```
Prelude> :type if True then 10 else 20
if True then 10 else 20 :: Num a => a

Prelude> :type if 4>5 then "ok" else "bad"
if 4>5 then "ok" else "bad" :: [Char]
```

Expressões condicionais (cont.)

```
Prelude> if length [1,2,3] then "ok" else "bad"
<interactive>:0:4:
    Couldn't match expected type ‘Bool’ with actual type ‘Int’
    In the return type of a call of ‘length’
    In the expression: length [1, 2, 3]
    In the expression: if length [1, 2, 3] then "ok" else "bad"

Prelude> if 4>5 then "ok" else 'H'
<interactive>:0:23:
    Couldn't match expected type ‘[Char]’ with actual type ‘Char’
    In the expression: 'H'
    In the expression: if 4 > 5 then "ok" else 'H'
    In an equation for ‘it’: it = if 4 > 5 then "ok" else 'H'
```

Expressões condicionais (cont.)

- Como na maioria das linguagens de programação, funções podem ser definidas usando expressões condicionais.
- **Exemplo**:
  Valor absoluto

```
abs :: Int -> Int
abs n = if n >= 0 then n else -n
```

  `abs` recebe um inteiro `n` e retorna `n` se ele é não-negativo, e `-n` caso contrário.

Expressões condicionais (cont.)

- Expressões condicionais podem ser aninhadas.
- **Exemplo**:
  Sinal de um número:

```
signum  :: Int -> Int
signum n = if n < 0
             then -1
             else if n == 0
                    then 0
                    else 1
```

- Em Haskell, expressões condicionais sempre devem ter as duas alternativas, o que evita qualquer possível problema de ambigüidade com expressões condicionais aninhadas.

Tópicos

Equações com guardas

- Funções podem ser definidas através de equações com guardas, onde uma sequência de expressões lógicas chamadas **guardas** é usada para escolher um resultado.
- Uma **equação com guarda** é formada por uma sequência de cláusulas escritas logo após a lista de argumentos. Cada cláusula é introduzida por uma barra vertical (|) e consiste em uma condição chamada **guarda** e uma expressão (resultado), separados por =.

$$
\begin{array}{l}
f\ \ arg_1\ \ \ldots\ \ arg_n \\
\qquad\quad \mid\ guarda_1\ =\ exp_1 \\
\qquad\quad \ldots \\
\qquad\quad \mid\ guarda_m\ =\ exp_m
\end{array}
$$

- Cada guarda deve ser uma expressão lógica.
- Os resultados devem ser todos do mesmo tipo.

Equações com guardas (cont.)

- **Exemplo**:
  Valor absoluto

```
abs n | n >= 0 = n
      | n < 0  = -n
```

  Nesta definição de `abs`, as guardas são

  - `n >= 0`
  - `n < 0`

  e as expressões associadas são

  - `n`
  - `-n`

  respectivamente.

Equações com guardas (cont.)

- Quando a função é aplicada, as guardas são verificadas em sequência. A primeira guarda verdadeira define o resultado.
- Assim no exemplo anterior o teste `n < 0` pode ser substituído pela constante `True`:

```
abs n | n >= 0 = n
      | True   = -n
```

Equações com guardas (cont.)

- A condição `True` pode também ser escrita como `otherwise`.
- Exemplo:

```
abs n | n >= 0    = n
      | otherwise = -n
```

- `otherwise` é uma condição que captura todas as outras situações que ainda não foram consideradas.
- `otherwise` é definida no prelúdio simplesmente como o valor verdadeiro:

```
otherwise :: Bool
otherwise  = True
```

Equações com guardas (cont.)

- Equações com guardas podem ser usadas para tornar definições que envolvem múltiplas condições mais fáceis de ler:
- **Exemplo**:
  Determina o sinal de um número:

```
signum n | n < 0      = -1
         | n == 0     = 0
         | otherwise = 1
```

Equações com guardas (cont.)

- **Exemplo**:
  Analisa o índice de massa corporal

```
analisaIMC imc
  | imc <= 18.5 = "Voce esta abaixo do peso, seu emo!"
  | imc <= 25.0 = "Voce parece normal. Deve ser feio!"
  | imc <= 30.0 = "Voce esta gordo! Perca algum peso!"
  | otherwise   = "Voce esta uma baleia. Parabens!"
```

Equações com guardas (cont.)

- Uma definição pode ser feita com várias equações.
- Se todas as guardas de uma equação forem falsas, a próxima equação é considerada. Se não houver uma próxima equação, ocorre um erro.
- **Exemplo**:

```
minhaFuncao x y | x > y = 1
                | x < y = -1
```

```
minhaFuncao 2 3 ⇝ -1
minhaFuncao 3 2 ⇝ 1
minhaFuncao 2 2 ⇝ ERRO
```

Equações com guardas (cont.)

- Um erro comum cometido por iniciantes é colocar um sinal de igual (=) depois do nome da função e parâmetros, antes da primeira guarda. Isso é um erro de sintaxe.

Exercícios

Em cada um dos exercícios a seguir:

- Defina a função solicitada de acordo com as instruções.
- Especifique o tipo mais geral desta função.
- Teste sua função no GHCi.

Exercícios (cont.)

Exercício 1

Defina uma função chamada `media3` que recebe três valores e retorna a sua média aritmética.

Exercícios (cont.)

Exercício 2

Defina uma função chamada `penultimo` que recebe uma lista e retorna o seu penúltimo elemento.

Exercícios (cont.)

Exercício 3

Defina uma função chamada `maior2` que recebe dois valores e retorna o maior deles. Use expressões condicionais.

Exercícios (cont.)

Exercício 4

Defina uma função chamada `maior2'` que recebe dois valores e retorna o maior deles. Use equações com guardas.

Exercícios (cont.)

Exercício 5

Defina uma função chamada `maior3` que recebe três valores e retorna o maior deles. Use expressões condicionais aninhadas.

Exercícios (cont.)

Exercício 6

Defina uma função chamada `maior3'` que recebe três valores e retorna o maior deles. Use equações com guardas.

Exercícios (cont.)

Exercício 7

Defina uma função chamada `maior3"` que recebe três valores e retorna o maior deles. Não use expressões condicionais e nem equações com guardas. Use a função `maior2` do exercício 3.

Exercícios (cont.)

Exercício 8

Defina uma função chamada `numRaizes` que recebe os três coeficientes de uma equação do segundo grau e retorna a quantidade de raízes reais distintas da equação. Assuma que a equação é não degenerada (isto é, o coeficiente do termo de grau 2 não é zero).

Exercícios (cont.)

Exercício 9

Usando funções da biblioteca, defina a função `halve :: [a] -> ([a],[a])` que divide uma lista em duas metades. Por exemplo:

```
> halve [1,2,3,4,5,6]
([1,2,3],[4,5,6])

> halve [1,2,3,4,5]
([1,2],[3,4,5])
```

Exercícios (cont.)

Exercício 10

Determine o tipo da função definida a seguir e explique o que ela faz.

```
misterio m n p = not (m == n && n == p)
```

Fim