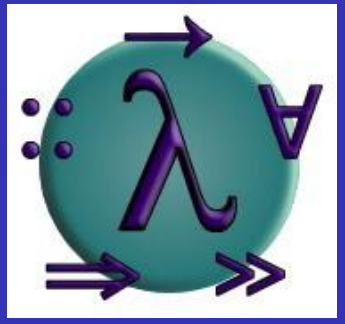
Programação Funcional



Capítulo 5

Casamento de Padrão

José Romildo Malaquias

Departamento de Computação
Universidade Federal de Ouro Preto

2012.1

Tópicos

Casamento de padrão

- **Padrão** é uma frase que permite analisar um valor e associar variáveis aos dados que compõem o valor.
- **Casamento de padrão** é uma operação envolvendo um padrão e uma expressão que faz a correspondência (*casamento*) entre o padrão e o valor da expressão.
- O casamento de padrão pode suceder ou falhar, dependendo da forma do padrão e da expressão envolvidos.
- Quando o casamento de padrão sucede as **variáveis** que ocorrem no padrão são associadas aos componentes correspondentes do valor.
- Em um casamento de padrão, o padrão e a expressão devem ser do mesmo tipo.
- Existem várias **formas de padrão**. Na seqüência algumas delas serão apresentadas.

Padrão constante

- O **padrão constante** é simplesmente uma constante.
- O **casamento** sucede se e somente se o padrão for idêntico ao valor.
- Nenhuma associação de **variável** é produzida.
- **Exemplos**:

padrão	valor	casamento
`10`	`10`	✓
`10`	`28`	×
`10`	`'P'`	*erro de tipo*
`'P'`	`'P'`	✓
`'P'`	`'q'`	×
`'P'`	`True`	*erro de tipo*
`True`	`True`	✓
`True`	`False`	×
`True`	`65`	*erro de tipo*

✓: sucede

×: falha

Padrão variável

- O **padrão variável** é simplesmente um identificador de variável de valor (e como tal deve começar com letra minúscula).
- O **casamento** sucede sempre.
- A **variável** é associada ao valor.
- **Exemplos**:

padrão	valor	casamento
`x`	`10`	✓ `x` ↦ `10`
`alfa`	`563.1223`	✓ `alfa` ↦ `563.1223`
`letra`	`'K'`	✓ `letra` ↦ `'K'`
`nomeCliente`	`"Ana Maria"`	✓ `nomeCliente` ↦ `"Ana Maria"`
`pessoa`	`("Ana",'F',16)`	✓ `pessoa` ↦ `("Ana",'F',16)`
`notas`	`[5.6,7.1,9.0]`	✓ `notas` ↦ `[5.6,7.1,9.0]`

Padrão curinga

- O **padrão curinga** é escrito como um sublinhado (_).
- O **casamento** sucede sempre.
- Nenhuma associação de **variável** é produzida.
- _ é também chamado de **variável anônima**, pois casa com qualquer valor sem dar nome ao valor.
- **Exemplos**:

padrão	valor	casamento
_	10	✓
_	28	✓
_	'P'	✓
_	()	✓
_	(18,3,2012)	✓
_	"Ana Maria"	✓
_	[5.6,7.1,9.0]	✓

Padrão tupla

- Uma **tupla** de padrões também é um padrão

  ($padrao_1$, ... , $padrao_n$)

- O **casamento** sucede se e somente se cada um dos padrões casar com o componente correspondente do valor.
- Se as aridades do padrão tupla e do valor tupla forem diferentes, então ocorre um erro de tipo.

Padrão tupla (cont.)

- **Exemplos**:

padrão	valor	casamento
`(18,True)`	`(18,True)`	✓
`(97,True)`	`(18,True)`	×
`(18,False)`	`(18,True)`	×
`(18,'M')`	`(18,True)`	*erro de tipo*
`(18,True,'M')`	`(18,True)`	*erro de tipo*
`()`	`()`	✓
`(x,y)`	`(5,9)`	✓ $x \mapsto 5, y \mapsto 9$
`(d,_,a)`	`(5,9,2012)`	✓ $d \mapsto 5, a \mapsto 2012$
`(x,y,z)`	`(5,9)`	*erro de tipo*
`(18,m,a)`	`(18,3,2012)`	✓ $m \mapsto 3, a \mapsto 2012$
`(d,5,a)`	`(18,3,2012)`	×
`(nome,sexo,_)`	`("Ana",'F',18)`	✓ $nome \mapsto$ `"Ana"`, $sexo \mapsto$ `'F'`
`(_,_,idade)`	`("Ana",'F',18)`	✓ $idade \mapsto 18$
`(_,(_,fam),9)`	`('F',("Ana","Dias"),9)`	✓ $fam \mapsto$ `"Dias"`
`(_,(_,fam),5)`	`('F',("Ana","Dias"),9)`	×

Tópicos

Expressão **case**

- **Expressão `case`** é uma forma de expressão que realiza casamento de padrão entre o valor de uma expressão e um ou mais padrões.
- Uma expressão `case` é da forma:

```
case exp of
  padrao_1 -> res_1
  ...
  padrao_n -> res_n
```

onde

  - exp, $res_1, \ldots, res_n$ são expressões
  - $padrao_1, \ldots, padrao_n$ são padrões
  - exp, $padrao_1, \ldots, padrao_n$ devem ser do mesmo tipo
  - $res_1, \ldots, res_n$ devem ser do mesmo tipo, que determina o tipo da expressão case
  - os padrões devem estar alinhados na mesma coluna (regra de *layout*)

Avaliação de expressões `case`

- É feito o casamento de padrão do valor de *exp* com os padrões, na seqüência em que foram escritos, até que se obtenha sucesso ou se esgotem os padrões
- O primeiro padrão cujo casamento suceder é escolhido
- O resultado final da expressão case é dado pela expressão associada ao padrão escolhido
- O resultado é avaliado em um ambiente estendido com as associações de variáveis resultantes do casamento de padrão
- Se a expressão não casar com nenhum padrão, a avaliação da expressão case resulta em um erro

Exemplos de expressões `case`

A expressão

```
case 3 - 2 + 1 of
  0 -> "zero"
  1 -> "um"
  2 -> "dois"
  3 -> "tres"
```

resulta em `"dois"`, pois o valor da expressão `3-2+1` é `2`, que casa com o terceiro padrão `2`, selecionando `"dois"` como resultado.

Exemplos de expressões `case` (cont.)

A expressão

```
case 23 > 10 of
  True  -> "beleza!"
  False -> "oops!"
```

resulta em `"beleza!"`, pois o valor da expressão `23 > 10` é **`True`**, que casa com o primeiro padrão **`True`**, selecionando `"beleza!"` como resultado.

Exemplos de expressões `case` (cont.)

A expressão

```
case toUpper (head "masculino") of
  'F' -> 10.2
  'M' -> 20.0
```

resulta em `20.0`, pois o valor da expressão `toUpper (head "masculino")` é `'M'`, que casa com o segundo padrão `'M'`, selecionando `20.0` como resultado.

Exemplos de expressões `case` (cont.)

A expressão

```
case head "masculino" of
  'F' -> 10.2
  'M' -> 20.0
```

resulta em um erro em tempo de execução, pois o valor da expressão `head "masculino"` não casa com nenhum dos padrões.

Exemplos de expressões `case` (cont.)

A expressão

```
case toUpper (head "masculino") of
  'F' -> "mulher"
  'M' -> 20.0
```

está incorreta, pois os resultados `"mulher"` e `20.0` não são do mesmo tipo.

Exemplos de expressões `case` (cont.)

A expressão

```
case head "Masculino" == 'F' of
  True  -> "mulher"
  1     -> "homem"
```

está incorreta, pois os padrões `True` e `1` não são do mesmo tipo.

Exemplos de expressões `case` (cont.)

A expressão

```
case head "Masculino" of
  True  -> "mulher"
  False -> "homem"
```

está incorreta, pois a expressão `head "Masculino"` e os padrões **`True`** e **`False`** não são do mesmo tipo.

Exemplos de expressões `case` (cont.)

A expressão

```
case toUpper (head "masculino") of
   'F' -> 10.0
 'M' -> 20.0
```

está incorreta, uma vez que não segue a regra de *layout* (os padrões não estão na mesma coluna).

Exemplos de expressões `case` (cont.)

A expressão

```
case 3 - 2 + 1 of
  x -> 11 * x
```

resulta em `22`, pois o valor da expressão `3 - 2 + 1` é `2`, que casa com o primeiro padrão `x`, associando a variável `x` com o valor `2`, e selecionando `11 * x` como resultado

Exemplos de expressões `case` (cont.)

A expressão

```
case mod 256 10 of
  7 -> 0
  n -> n * 1000
```

resulta em `6000`, pois o valor da expressão `mod 256 10` é `6`, que casa com o segundo padrão `n`, associando a variável `n` com o valor `6`, e selecionando `n * 1000` como resultado

Exemplos de expressões `case` (cont.)

A expressão

```
case mod 257 10 of
  7 -> 0
  n -> n * 1000
```

resulta em `0`, pois `7` é o primeiro padrão que casa com o valor da expressão `mod 257 10`.

Já a expressão

```
case mod 257 10 of
  n -> n * 1000
  7 -> 0
```

resulta em `7000`, pois `n` é o primeiro padrão que casa com o valor da expressão `mod 257 10`.

Exemplos de expressões `case` (cont.)

A expressão

```
case 46 - 2*20 of
  0 -> "zero"
  1 -> "um"
  2 -> "dois"
  3 -> "tres"
  4 -> "quatro"
  _ -> "maior que quatro"
```

resulta em `"maior que quatro"`, pois `_` é o primeiro padrão que casa com o valor da expressão `46 - 2*20`.

Exemplos de expressões `case` (cont.)

A expressão

```
case (3+2,3-2) of
  (0,0) -> 10
  (_,1) -> 20
  (x,2) -> x^2
  (x,y) -> x*y - 1
```

resulta em `20`, pois `(_,1)` é o primeiro padrão que casa com o valor da expressão `(3+2,3-2)`.

Expressão `case` com guardas

- Em uma expressão case cada padrão pode ser acompanhado de uma seqüência de cláusulas.
- Cada cláusula é introduzida por uma barra vertical (`|`) e consiste em uma condição (**guarda**) e uma expressão (resultado), separados por `->`.
- Para que o resultado de uma cláusula seja escolhido é necessário que o casamento de padrão suceda, e que a guarda correspondente seja verdadeira.

Exemplo de expressão `case` com guardas

A expressão

```
case ("Paulo Roberto",'M',28,69.3) of
  (_,_,idade,peso)   | idade < 18 -> 2*peso
                     | idade < 21 -> 3*peso
  (_,'F',_,peso)                  -> peso
  (_,'M',idade,peso) | idade < 40 -> peso + 10
                     | otherwise  -> 0.9 * peso
```

resulta em `79.3`, pois a tupla `("Paulo Roberto",'M',28,69.3)`

- casa com o primeiro padrão, porém nenhuma guarda é satisfeita
- não casa com o segundo padrão
- casa com o terceiro padrão, e a primeira guarda é satisfeita, logo o resultado é dado por `peso + 10`

Tópicos

Definindo funções com casamento de padrão

- Uma **definição de função** é formada por uma seqüência de equações.
- Os **parâmetros** usados em uma equação para representar os argumentos são padrões.
- Em uma **aplicação de função** o resultado é dado pela primeira equação cujos parâmetros casam com os respectivos argumentos, e cuja guarda (se houver) é verdadeira.
- Se em todas as equações os casamentos de padrão falharem ou todas as guardas forem falsas, ocorre um erro de execução.
- Geralmente o uso de padrões para especificar os argumentos torna a definição da função mais clara.

Definindo funções com casamento de padrão (cont.)

- **Exemplo**:

```
not :: Bool -> Bool
not False = True
not True  = False
```

A função `not` mapeia `False` a `True`, e `True` a `False`.

```
not False     ⇝ True
not (even 6) ⇝ False
```

Definindo funções com casamento de padrão (cont.)

- **Exemplo**:

```
(&&) :: Bool -> Bool -> Bool
True  && True  = True
True  && False = False
False && True  = False
False && False = False
```

```
True && True  ⇝ True
False && True ⇝ False
2>3 && odd 4  ⇝ False
```

Definindo funções com casamento de padrão (cont.)

- **Exemplo**:

```
(&&) :: Bool -> Bool -> Bool
True && True = True
_    && _    = False
```

```
True && True  ⇝ True
False && True ⇝ False
2>3 && 2<3    ⇝ False
```

Definindo funções com casamento de padrão (cont.)

- **Exemplo**:

```
(&&) :: Bool -> Bool -> Bool
True  && b = b
False && _ = False
```

```
True && True  ⇝ True
2>3 && 2<3    ⇝ False
2<3 && even 5 ⇝ False
```

Definindo funções com casamento de padrão (cont.)

- **Exemplo**:

```
(&&) :: Bool -> Bool -> Bool
b && b = b
_ && _ = False
```

está incorreto, pois não é possível usar uma variável mais de uma vez nos padrões (**princípio da linearidade**).

Definindo funções com casamento de padrão (cont.)

- **Exemplos**:

```
fst      :: (a,b) -> a
fst (x,_) = x
```

```
snd      :: (a,b) -> b
snd (_,y) = y
```

```
fst (1+2,1-2)         ⇝ 3
snd (div 5 0,even 9)  ⇝ False
```

Exercícios

Exercício 1

Dê três possíveis definições para o operador lógico ou (`||`), utilizando casamento de padrão.

Exercícios (cont.)

Exercício 2

Redefina a seguinte versão do operador lógico e (&&) usando expressões condicionais ao invés de casamento de padrão:

```
True && True = True
_    && _    = False
```

Exercício 3

Redefina a seguinte versão do operador lógico e (&&) usando expressões condicionais ao invés de casamento de padrão:

```
True  && b = b
False && _ = False
```

Comente sobre o diferente número de expressões condicionais necessárias em comparação com o exercício 2.

Exercícios (cont.)

Exercício 4

Defina uma função que recebe dois pontos no espaço e retorna a distância entre eles. Considere que um ponto no espaço é representado por uma tripla de números que são as coordenadas do ponto. Use casamento de padrão.

Exercício 5

Analise a seguinte definição e apresente uma definição alternativa mais simples desta função.

```
opp :: (Int,(Int,Int)) -> Int
opp z = if fst z == 1
        then fst (snd z) + snd (snd z)
        else if fst z == 2
             then fst (snd z) - snd (snd z)
             else 0
```

Tópicos

Definições locais com **where**

- A palavra reservada **where** é utilizada para introduzir definições locais cujo contexto (ou **escopo**) é a expressão no lado direito e as guardas (quando houver) de uma equação.
- **Exemplo:**

```
f       :: Fractional a => (a,a) -> a
f (x,y) = (a + 1) * (a + 2) where a = (x+y)/2
```

```
f (2,3) ⇝ 15.75
f (5,1) ⇝ 20.0
```

Definições locais com **where** (cont.)

- Quando há duas ou mais definições locais, elas podem ser escritas em diferentes estilos.
- **Exemplos:**

```
f (x,y) = (a+1)*(b+2)
  where { a = (x+y)/2; b = (x+y)/3 }
```

```
f (x,y) = (a+1)*(b+2)
  where a = (x+y)/2; b = (x+y)/3
```

```
f (x,y) = (a+1)*(b+2)
  where a = (x+y)/2
        b = (x+y)/3
```

Neste último exemplo foi usada a regra de *layout*, que dispensa os símbolos `;`, `{` e `}` mas exige que cada definição local esteja alinhada em uma mesma coluna.

Definições locais com **where** (cont.)

- **Exemplo:**

```
square x = x * x

g x y | x <= 10 = x + a
      | x >  10 = x - a
  where
    a = square (y+1)
```

O escopo de a inclui os dois possíveis resultados, determinados pelas guardas.

Definições locais com **where** (cont.)

- As definições locais podem ser de funções e de variáveis, fazendo uso de padrões.
- **Exemplo:**

```
h y = 3 + f y + f a + f b
  where
    c = 10
    (a,b) = (3*c,f 2)
    f x = x + 7*c
```

```
h 5 ⇝ 320
```

O escopo de a inclui os dois possíveis resultados, determinados pelas guardas.

Definições locais com **where** (cont.)

- Exemplo: análise do índice de massa corporal

```
analisaIMC peso altura
  | 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!"
  where
    imc = peso / height ^2
```

Ou ainda:

```
analisaIMC peso altura
  | imc <= magro  = "Voce esta abaixo do peso, seu emo!"
  | imc <= normal = "Voce parece normal. Deve ser feio!"
  | imc <= gordo  = "Voce esta gordo! Perca algum peso!"
  | otherwise     = "Voce esta uma baleia. Parabens!"
  where
    imc    = peso / height ^2
    magro  = 18.5
    normal = 25.0
    gordo  = 30.0
```

Definições locais com **where** (cont.)

- Definições locais com `where` não são compartilhadas entre corpos de funções de diferentes padrões nas equações.
- **Exemplo:**

```
saudacao           :: String -> String
saudacao "Joana"    = saudacaoLegal ++ " Joana!"
saudacao "Ferando"  = saudacaoLegal ++ " Fernando!"
saudacao nome       = saudacaoInfeliz ++ " " ++ nome
  where
    saudacaoLegal   = "Ola! Que bom encontrar voce, "
    saudacaoInfeliz = "Oh! Pfft. E voce, "
```

Esta definição de função está incorreta. Para corrigi-la, transforme as definições locais de `saudacaoLegal` e `saudacaoInfeliz` em definições globais.

Expressoes **let**

- Uma **expressão let** é formada por uma lista de declarações mutuamente recursivas, e por uma expressão:

```
let definições in expressão
```

- O escopo das declarações é a expressão e o lado direito das declarações. Portanto os nomes introduzidos nas declarações só podem ser usados nas próprias declarações e na expressão.
- O tipo da expressão let é o tipo da expressão que aparece no seu corpo.
- O valor da expressão let é o valor da expressão que aparece no seu corpo, calculado em um contexto que inclui as variáveis introduzidas nas declarações.
- A expressão let se estende à direita tanto quanto possível.

Expressoes **let** (cont.)

- **Exemplos:**

```
let a = 5 in (a - 1)*a + 1
⇝ 21
```

Expressoes **let** (cont.)

```
let { y = 1 + 2; z = 4 + 6 } in y + z
⇝ 21
```

Expressoes **let** (cont.)

```
let r = 3; s = 6 in r^2 + s^2
⇝ 45
```

Expressoes **let** (cont.)

```
let a = 1
    b = -5
    c = 6
    r = sqrt (b^2 - 4*a*c)
    k = a/2
in ((-b + r)/k,(-b - r)/k)
⇝ (12.0,8.0)
```

Expressoes **let** (cont.)

```
f a b = let y   = a*b
            g x = (x+y)/y
        in g (2*a) + g (3*b)
```

```
f 3 4
⇝ 3.5
```

Expressoes **let** (cont.)

```
4 * let x = 5-2 in x * x
⇝ 36
```

Expressoes **let** (cont.)

```
(let x = 5-2 in x * x) ^ 2
⇝ 81
```

Expressoes **let** (cont.)

```
[ let square x = x*x in (square 5,square 3, square 2) ]
⇝ [(25,9,4)]
```

Expressoes **let** (cont.)

```
( let a = 100; b = 200; c = 300 in a*b*c
, let foo = "Hey "; bar = "there!" in foo ++ bar
)
⇝ (6000000,"Hey there!")
```

Expressoes **let** (cont.)

```
let (a,b,c) = (1,2,3) in a + b + c
⇝ 6
```

Expressoes **let** (cont.)

```
let a = 1
in let b = 2
   in a + b
⇝ 3
```

Expressoes **let** (cont.)

```
let x = 7 in (let x = "foo" in x, x)
⇝ ("foo",7)
```

Expressoes **let** (cont.)

```
let c = 10
    (a,b) = (3*c, f 2)
    f x = x + 7*c
in f a + f b
⇝ 242
```

Expressoes **let** (cont.)

- **Exemplo:**
  Cálculo da área da superfície de um cilindro:

```
areaSuperfCil    :: Double -> Double -> Double
areaSuperfCil r h = let areaLado = 2 * pi * r * h
                        areaBase = pi * r^2
                    in areaLado + 2*areaBase
```

Diferenças entre `let` e `where`

- Com **where** as definições são colocadas no final, e com **let** elas são colocadas no início.
- **let** é uma expressão e pode ser usada em qualquer lugar onde se espera uma expressão.
- Já **where** não é uma expressão, podendo ser usada apenas para fazer definições locais em uma definição de função.

Exercícios

Exercício 6

Defina uma função para calcular as raízes reais do polinômio

$$ax^2 + bx + c$$

Faça duas versões, usando:

- expressão **let** para calcular o discrimante
- definição local com **where** para calcular o discriminante.

Teste suas funções no GHCi.
Dica: Use a função `error :: String ->` a do prelúdio, que exibe uma mensagem de erro e termina o programa, para exibir uma mensagem quando não houver raízes reais.

Fim