Programação Funcional — BCC222

Aula 2

Avaliação

Lucília Camarão de Figueiredo
Departamento de Ciência da Computação
Universidade Federal de Ouro Preto

Parte I

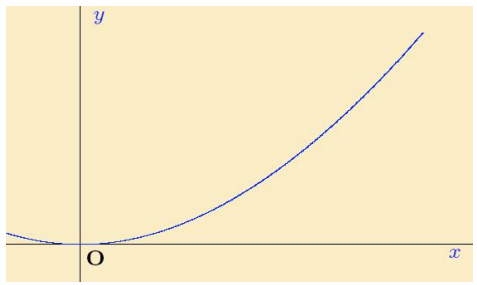
Regra de Leibniz

O que é uma função?

- Uma receita para gerar uma saída a partir de dados de entrada:
  "Multiplicar um número por ele próprio"
- Um conjunto de pares (entrada, saída):
  (1,1) (2,4) (3,9) (4,16) (5,25) ...
- Uma equação

$$f\ x = x^2$$

- Um gráfico relacionando entradas e saídas (apenas para números):



Operações sobre números

```
C:\Users\lucilia>ghci
GHCi, version 6.10.4:  www.haskell.org/ghc/ :?  for help
Loading package ghc-prim ...  linking ...  done.
Loading package integer ...  linking ...  done.
Loading package base ...  linking ...  done.
Prelude>3 + 3
6
Prelude>3 * 3
9
```

Funções sobre números

lect02.hs

```
square ::  Integer -> Integer
square x = x * x

pyth ::  Integer -> Integer -> Integer
pyth a b = square a + square b
```

Testando suas funções

```
C:\Users\lucilia>ghci lect02b.hs
GHCi, version 6.10.4:  www.haskell.org/ghc/ :?  for help
Loading package ghc-prim ...  linking ...  done.
Loading package integer ...  linking ...  done.
Loading package base ...  linking ...  done.
[1 of 1] Compiling Main ( lect02b.hs, interpreted )
Ok, modules loaded:  Main.
*Main> square 3
9
*Main> pyth 3 4
25
```

Mais alguns testes

```
*Main> square 0
0
*Main> square 1
1
*Main> square 2
4
*Main> square 3
9
*Main> square 4
16
*Main> square (-3)
9
*Main> square 10000000000
100000000000000000000
```

Declaração e avaliação

Declaração (file lect02a.hs)

```
square :: Integer -> Integer
square x = x * x

pyth :: Integer -> Integer -> Integer
pyth a b = square a + square b
```

Avaliação

```
% ghci lect02a.hs
GHCi, version 6.10.4: www.haskell.org/ghc/ :? for help
Loading package ghc-prim ... linking ... done.
Loading package integer ... linking ... done.
Loading package base ... linking ... done.
[1 of 1] Compiling Main ( lect02b.hs, interpreted )
Ok, modules loaded: Main.
*Main> pyth 3 4
25
*Main>
```

Regra de Leibniz

```
square ::  Integer -> Integer
square x = x * x

pyth ::  Integer -> Integer -> Integer
pyth a b = square a + square b
```

"Equals may be substituted for equals."
– Gottfried Wilhelm Leibniz (1646–1716)

```
   pyth 3 4
=
   square 3 + square 4
=
   3*3 + 4*4
=
   9 + 16
=
   25
```

Operações numéricas são funções

```
(+) ::  Integer -> Integer -> Integer
(*) ::  Integer -> Integer -> Integer
```

```
Main*> 3+4
7
Main*> 3*4
12
```

```
3 + 4    é o mesmo que    (+) 3 4
3 * 4    é o mesmo que    (*) 3 4
```

```
Main*> (+) 3 4
7
Main*> (*) 3 4
12
```

Precedencia e parenteses

Aplicação de função tem *precedencia* sobre operadores infixados.

```
    square 3 + square 4
=
    (square 3) + (square 4)
```

Multiplicação tem *precedencia* sobre adição.

```
    3*3 + 4*4
=
    (3*3) + (4*4)
```

Associatividade

Adição é *associativa*.

```
    3 + (4 + 5)
=
    3 + 9
=
    12
=
    7 + 5
=
    (3 + 4) + 5
```

Adição é *associativa à esquerda*.

```
    3 + 4 + 5
significa
    (3 + 4) + 5
```

Parte II

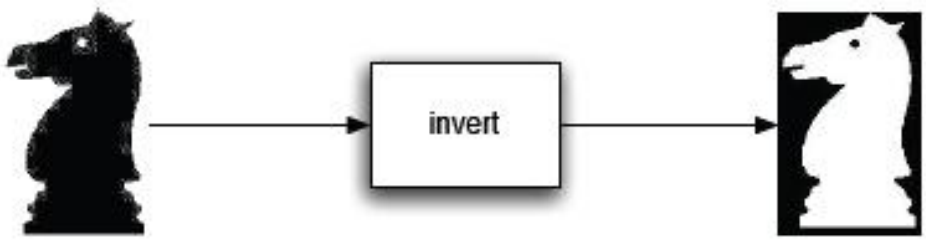
Xadrez

Tipos de dados

- Integers: 42, -69
- Floats: 3.14
- Characters: ’h’
- Strings: "hello"
- Pictures: 

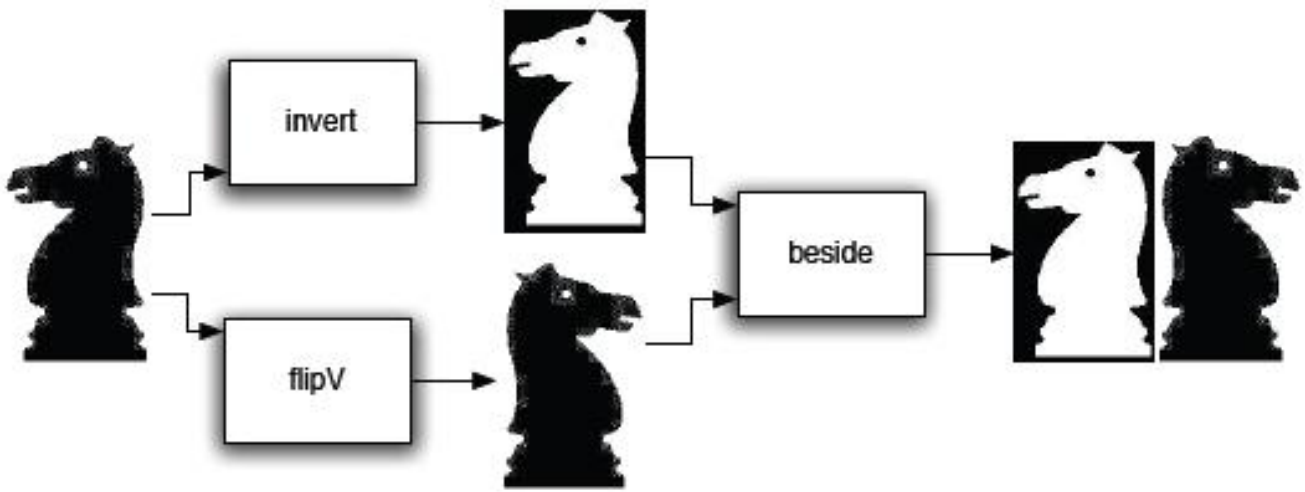
Aplicando uma função

```
invert ::  Picture -> Picture
knight ::  Picture

invert knight
```



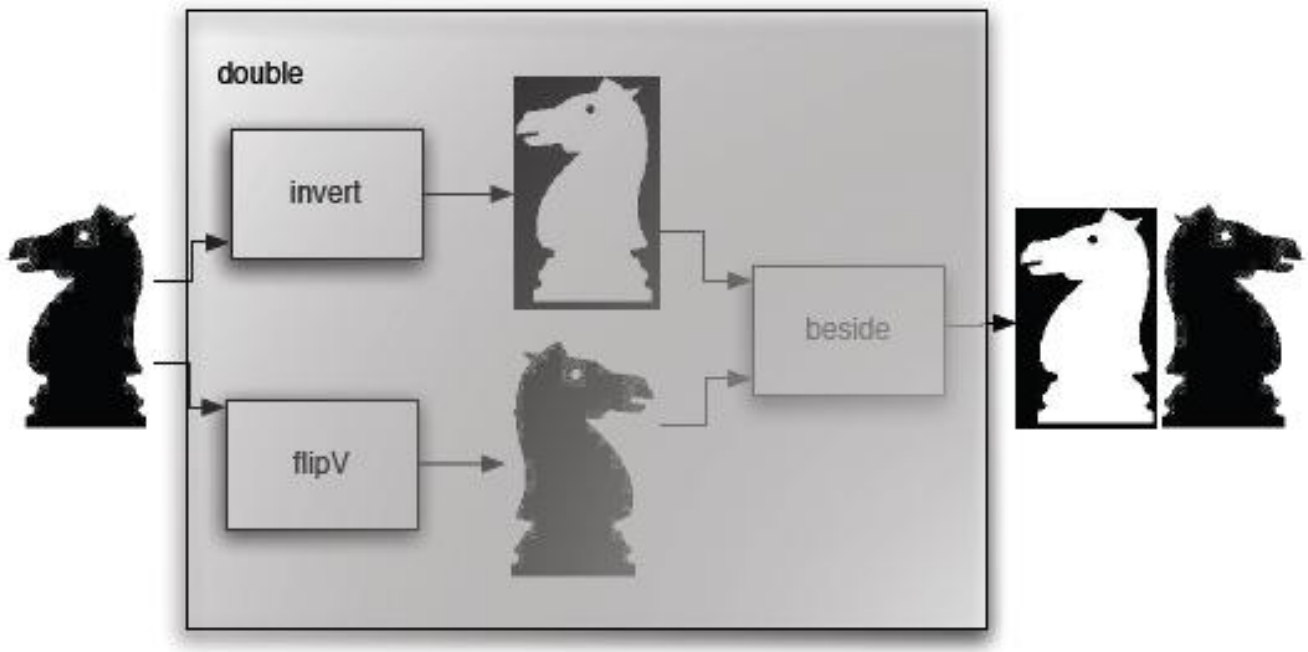
Compondo funções

```
beside ::  Picture -> Picture -> Picture
flipV ::  Picture -> Picture
invert ::  Picture -> Picture
knight ::  Picture

beside (invert knight) (flipV knight)
```



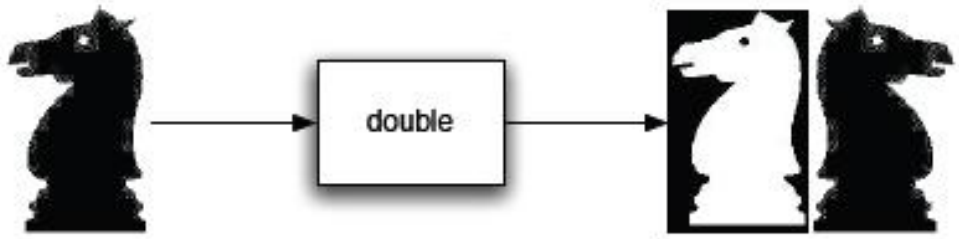
Definindo uma nova função

```
double ::  Picture -> Picture
double p = beside (invert p) (flipV p)

double knight
```



Definindo uma nova função

```
double ::  Picture -> Picture
double p = beside (invert p) (flipV p)

double knight
```



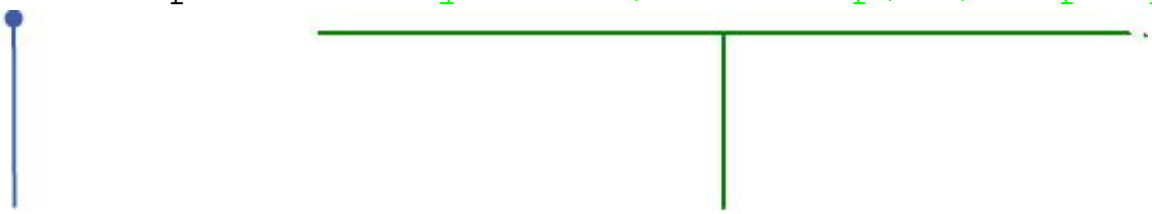
Terminologia

Declaração de tipo (assinatura)

```
makePicture ::  Picture -> Picture
```

Declaração de função

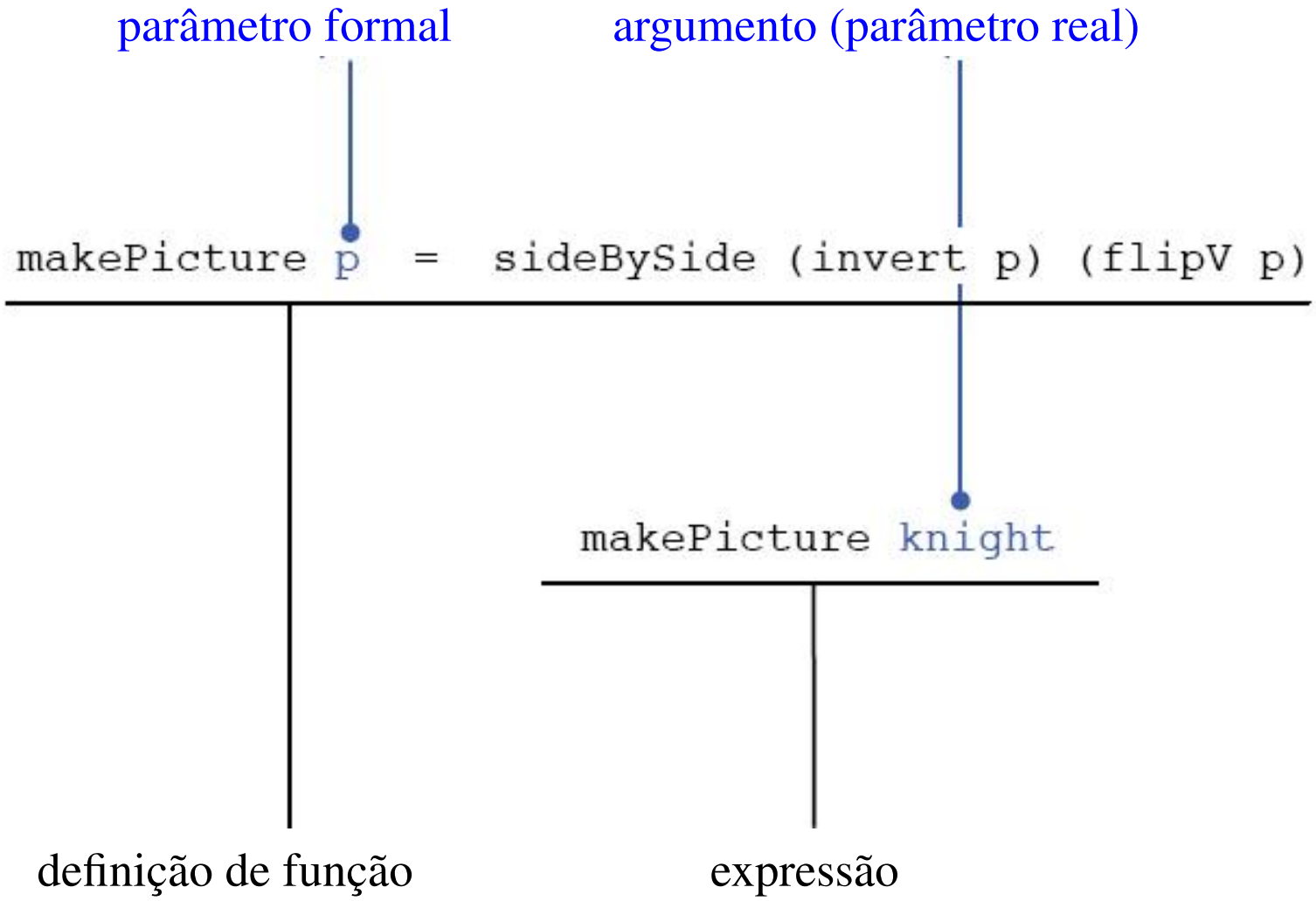
```
makePicture p = sideBySide (invert p) (flipV p)
```



nome da função

corpo da função

Terminologia



Parte III

QuickCheck

Um programa (file lect02a.hs)

```
square ::  Integer -> Integer
square x = x * x

pyth ::  Integer -> Integer -> Integer
pyth a b = square a + square b
```

Executando o programa (file lect02a.hs)

```
% ghci lect02a.hs
GHCi, version 6.10.4:  www.haskell.org/ghc/ :?  for help
Loading package ghc-prim ...  linking ...  done.
Loading package integer ...  linking ...  done.
Loading package base ...  linking ...  done.
[1 of 1] Compiling Main ( lect02b.hs, interpreted )
Ok, modules loaded:  Main.
*Main> square 3
9
*Main> pyth 3 4
25 *Main>
```

Outro programa (file lect02b.hs)

```
import Test.QuickCheck

square ::  Integer -> Integer
square x = x * x

pyth ::  Integer -> Integer -> Integer
pyth a b = square a + square b

prop_square ::  Integer -> Bool
prop_square x =
   square x >= 0

prop_squares ::  Integer -> Integer -> Bool
prop_squares x y =
   square (x+y) == square x + 2*x*y + square y

prop_pyth ::  Integer -> Integer -> Bool
prop_pyth x y =
   square (x+y) == pyth x y + 2*x*y
```

Executando o outro programa (arquivo lect02b.hs)

```
% ghci lect02a.hs
GHCi, version 6.10.4:  www.haskell.org/ghc/ :?  for help
Loading package ghc-prim ...  linking ...  done.
Loading package integer ...  linking ...  done.
Loading package base ...  linking ...  done.
[1 of 1] Compiling Main ( lect02b.hs, interpreted )
Ok, modules loaded:  Main.
*Main> quickCheck prop_square
Loading package old-locale-1.0.0.0 ...  linking ...  done.
Loading package old-time-1.0.0.0 ...  linking ...  done.
Loading package random-1.0.0.0 ...  linking ...  done.
Loading package mtl-1.1.0.1 ...  linking ...  done.
Loading package QuickCheck-2.1 ...  linking ...  done.
+++ OK, passed 100 tests.
*Main> quickCheck prop_squares
+++ OK, passed 100 tests.
*Main> quickCheck prop_pyth
+++ OK, passed 100 tests.
```

Parte IV

Regra de Leibniz (reprise)

Gottfried Wilhelm Leibniz (1646–1716)

Antecipou a lógica simbólica e descobriu o cálculo diferencial e integral (independentemente de Newton).

"The only way to rectify our reasonings is to make them as tangible as those of the Mathematicians, so that we can find our error at a glance, and when there are disputes among persons, we can simply say: Let us calculate, without further ado, to see who is right. "

"In symbols one observes an advantage in discovery which is greatest when they express the exact nature of a thing briefly and, as it were, picture it; then indeed the labor of thought is wonderfully diminished."