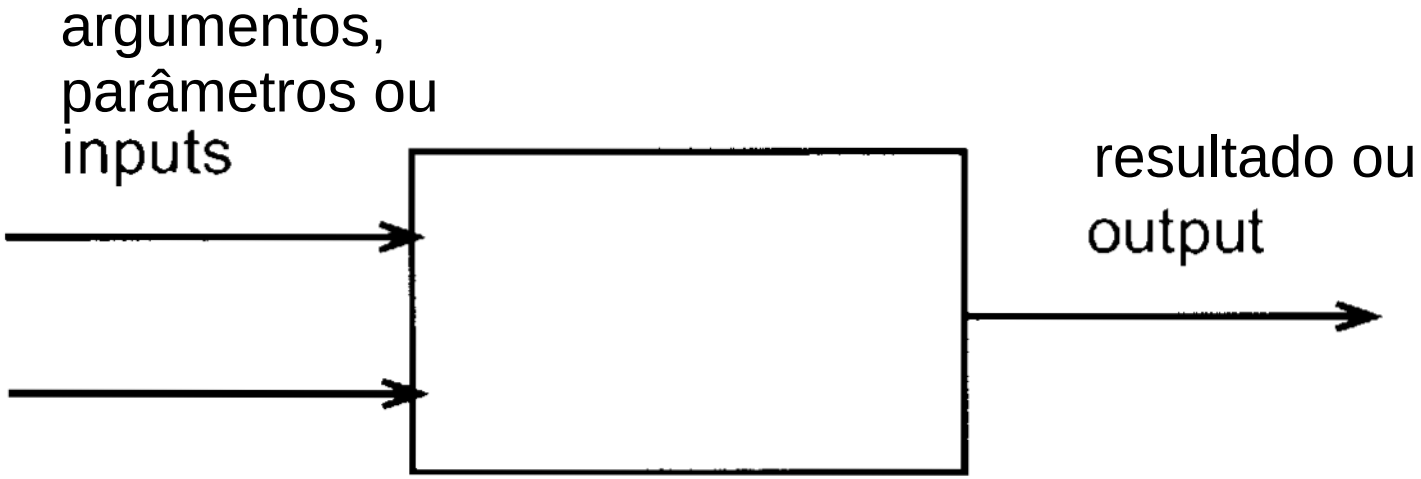
Tipos Básicos e Definição de Funções

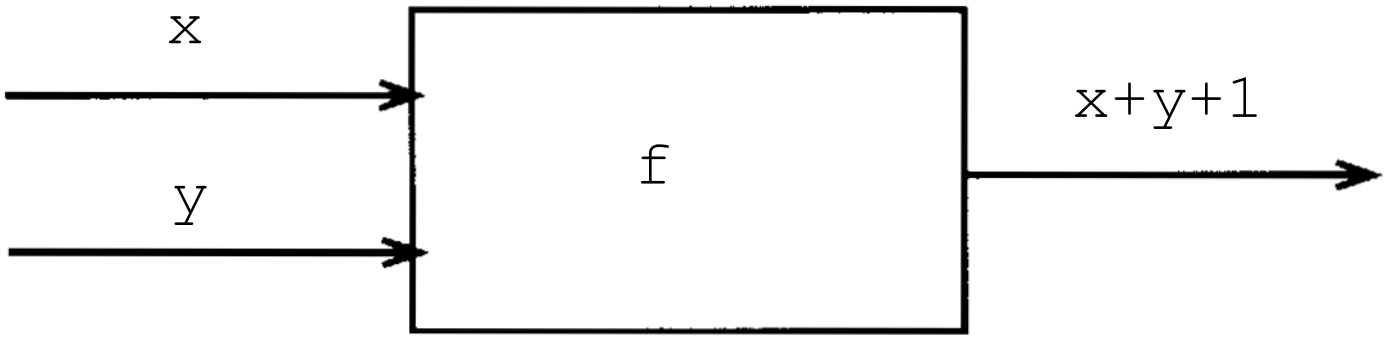
Programação Funcional
DCOMP-UFS

Função

- Relação binária tal que ….
- “Caixas pretas” que recebem valores de entrada e produzem um valor que depende dos valores da entrada



Definição de função




```
f x y = x+y+1
```

nome da função

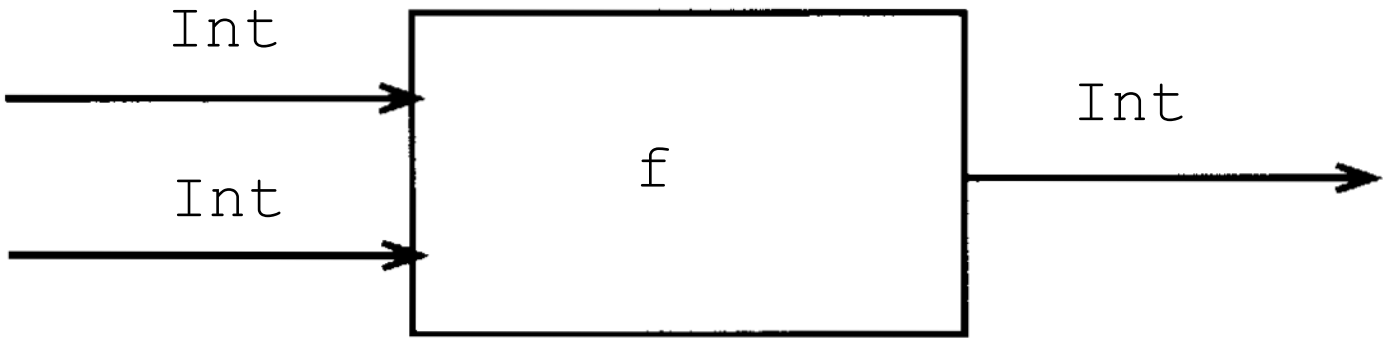
parâmetros formais

resultado
(definido em termos dos parâmetros formais)

Tipo

- Funções recebem e devolvem valores/dados
- Dados são classificados em tipos
- Tipo
  - Conjunto de valores
  - Funções que operam uniformemente sobre os valores

Declaração de tipo




```
f :: Int -> Int -> Int
```

Em Haskell toda expressão têm tipo

- Restringe a usos coerentes
- Checagem estática (antes da execução)
  - Programas mais confiáveis (erros detectados cedo)
  - Programas mais legíveis
  - Programas mais eficientes
- Tipos podem ser explicitados pelo programador
  - Se não for explicitado, Haskell infere o tipo

Definição junto com a declaração de tipo

```
f :: Int -> Int -> Int
f x y = x+y+1
```

Expressões

- O interpretador de Haskell (GHCi) permite avaliar expressões como

```
(7-2)*3
```

- Expressões podem ser construidas chamando funções

- Notação de chamada por justaposição

```
f (2*3) 5
```

Regras de nomeação

- Haskell é "*case sensitive*"
  - Nomes de tipos começam com maiúsculas
    - Int, Integer, Char, ...
  - Nomes de variáveis e funções, com minúsculas
    - div, mod, ...

```
F:: int -> int -- erro na compilação
F X = X + 1    -- erro na compilação
```

Classificação de Tipos

- Primitivos
  - valores atômicos
  - inteiros, caracteres, reais, booleanos, enumerados, ...
- Tipos compostos (ou estruturados)
  - tuplas, listas, ...

O tipo dos booleanos

- **`Bool`**
  - True, False
  - Operações: &&, ||, ==
  - Funções: not

- Tabelas de verdade

Tipos Inteiros

- **`Int`** (64 bits)
  - Literais: 0, 4, -345, 2147483647, maxBound::Int, ...
  - Operações: +, *, ^, -, <, <=, ==, /=, >=, >
  - Funções: succ, div, mod, abs, negate

- **`Integer`** (ilimitado)

Sobrecarga

- Considere

```
a::Int
b::Int
c::Integer
d::Integer
   … a+b… c+d …
```

- As operações efetuadas ao avaliar `a+b` e `c+d` são diferentes
- **Sobrecarga**: um mesmo nome para diferentes operações ou funções
- Similarmente, os nomes das funções succ, div, mod, abs, negate são sobrecarregados

Caracteres

- **`Char`**

```
'a','b','\t','\n','\\','\34'
```

```
fromEnum :: Int -> Char
```

```
toEnum :: Char -> Int
```

Convertem de caractere para código numérico e vice-versa

Reais em ponto flutuante

- **Float, Double**
  - 0, 4, -345, 0.31416, -23.12, 2.45e+2
  - +, -, *, /, ^, **, <, <=, ...
  - abs, acos, asin, atan, cos, sin, tan, exp, fromIntegral, abs, negate

Literais numéricos são sobrecarregados, por ex.

`12`

representa tanto o número `Int`, `Integer`, `Float` e `Double`

Haskell descobre qual é analisando o contexto, por exemplo, em

```
12 + a
```

se `a` é de tipo `Int`, o `12` também será de tipo `Int`

Sequências de caracteres

- **`String`**

  Literais:

  ```
  "Maria"
  "carro azul",
  ""
  "gorila\thipopótamo\tgarça\n"
  ```

  Operações: `++` (concatenação)

  Funções: `putStr, show, read`

- Não confunda 'a' com "a"

Funções

- Se recomenda explicitar o tipo
  - Exceto para pequenas definições auxiliares

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

Como é feita a avaliação

- Usando substituição, de maneira similar como na aritmética clássica, porém só realizando uma substituição a cada passo.
- A ordem de avaliação não interfere no resultado (transparência referencial).

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

square (2+5) ↗ (2+5) * (2+5) ↗ 7 * (2+5) ↗ 7*7 ↗ 49

square (2+5) ↗ square 7 ↗ 7 * 7