Linguagens de Programação

Fabio Mascarenhas - 2013.1

http://www.dcc.ufrj.br/~fabiom/lp

Funções de primeira ordem

- Agora podemos adicionar definições de funções à nossa linguagem
- Para começar, as definições de funções não serão expressões, e nem funções serão valores
- Um programa *fun* de primeira ordem é uma sequência de definições de funções, seguida por uma expressão
- Cada definição de função tem uma lista de parâmetros formais e o corpo da função (também uma expressão!)
- Finalmente, uma *aplicação* é uma expressão que chama uma função com uma lista de expressões (os argumentos)

Sintaxe

- Sintaxe concreta e abstrata para funções de primeira ordem:

```
prog : {fun} exp
fun  : FUN ID '(' params ')' exp END
exp  : ...
     | ID '(' exps ')'
```

```
case class Fun1(nome: String, params: List[String], corpo: Exp)
case class Prog(defs: Set[Fun1], corpo: Exp)
case class Var(nome: String) extends Exp
case class Ap1(fun: String, args: List[Exp]) extends Exp
```

Interpretador

- O interpretador de *fun* agora precisa do conjunto de funções que estão definidas

- Podemos interpretar uma aplicação de função usando o modelo de *substituição*

  - Primeiro avaliamos cada argumento da aplicação

  - Depois substituímos cada parâmetro pelo respectivo valor no corpo da função

  - Então avaliamos o corpo

Aridade

- O que deve acontecer se o número de parâmetros não bate com o número de argumentos?
- O número de parâmetros de uma função é a sua *aridade*
- Geralmente, número de argumentos incompatível com a aridade é um erro, mas outras linguagens podem se comportar de outras maneiras
  - Ignorar argumentos extras, substituir parâmetros que faltam por algum valor *default*...
  - Podem existir também funções com *aridade variável*, onde os argumentos são empacotados em uma lista

Implementando a substituição

- A substituição é uma transformação de Exp para Exp
- Para simplificar, vamos usar uma estrutura de dados de Scala que não vimos ainda: `Map[K,V]`, ou *mapeamento*
- Um Map é um mapeamento entre valores do tipo K (as *chaves*) para valores do tipo V (os *valores*)
- Existem duas operações básicas em Maps: busca e adição

```
scala> val m = Map(2 -> "Foo", 3 -> "Bar")
m: Map[Int,String] = Map(2 -> "Foo", 3 -> "Bar")
scala> m.get(3)
res4: Option[String] = Some("Bar")
scala> m + (5 -> "Bar")
res6:[Int,String] = Map(2 -> "Foo", 3 -> "Bar", 5 -> "Bar")
scala> m.get(5)
res7: Option[String] = None
```

O que substituir?

- A substituição vai afetar os nomes no corpo da função, mas não todos!
- Nomes que identificam funções nas aplicações não devem ser afetados
    - Espaços de nomes separados para funções e variáveis, comum em linguagens de primeira ordem
- Nomes que não têm nenhum argumento no mapa de substituição são erros!
- Não existe escopo global em *fun*
- Aliás, não existe escopo nenhum!

Chamada por valor vs chamada por nome

- O interpretador de *fun* está fazendo chamada por valor
- Mudá-lo para fazer chamada por nome é simples no entanto!
  - Apenas precisamos mudar a avaliação das chamadas de função para passar *expressões* ao invés de valores para a substituição
  - A substituição fica até mais simples! Não é preciso mais converter os valores primitivos em expressões para plugá-los no corpo de função
- Para não complicar o parser vamos adotar uma convenção léxica: parâmetros que começam com _ serão por nome, e os outros por valor

Nomes locais: let

- Vamos introduzir uma nova expressão em *fun*, para dar nomes para expressões

```
exp  : ...
     | LET ID '=' exp IN exp END
```

```
case class Let(nome: String, exp: Exp, corpo: Exp) extends Exp
```

- O let é parecido com o *val* de Scala; dentro do *corpo* do let o *nome* é associado ao valor de *exp*

- Podemos dar a semântica de *fun* com let via substituição também, mas a substiuição fica mais complicada

Substituição com *let*

- Para substituir um identificador *x* em uma expressão *e* por um valor *v*, troque todas as **instâncias livres** de *x* em *e* por *v*

- Ou seja, a substituição do identificador x não “entra” em um termo `let x = ...`

- Essa definição funciona muito bem para substituição de valores (call-by-value), mas o que acontece com substituição de termos?

Substituição CBN

- Vamos avaliar essa expressão:

```
let _x = y + 2 in
  let y = 5 in
    _x
  end
end
```

Substituição CBN

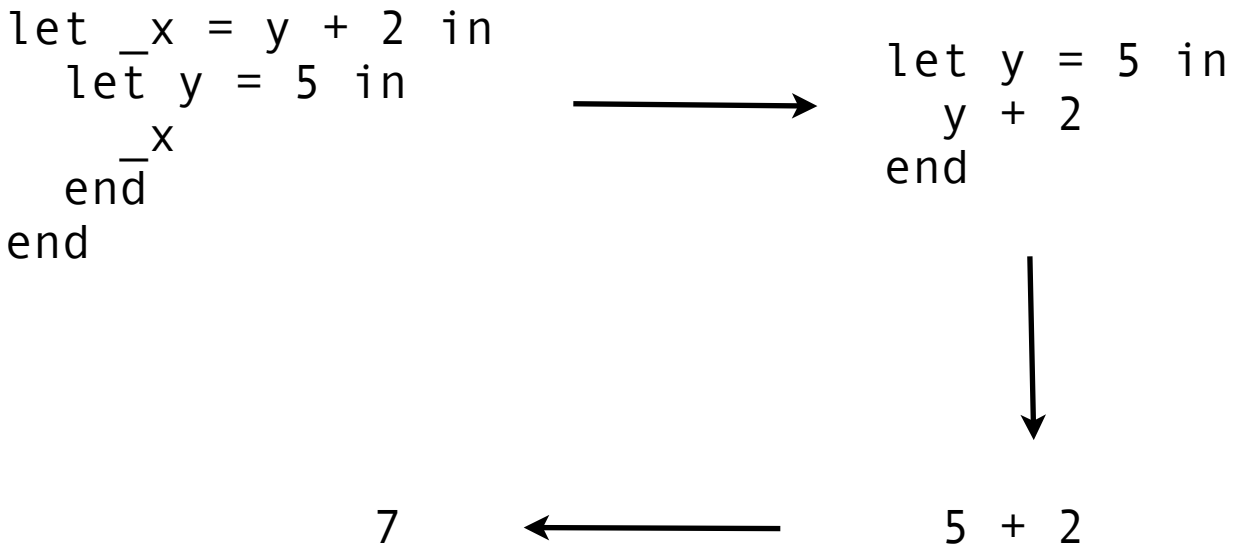
- Vamos avaliar essa expressão:

```
let _x = y + 2 in
  let y = 5 in
    _x
  end
end
```

- O resultado é 7!

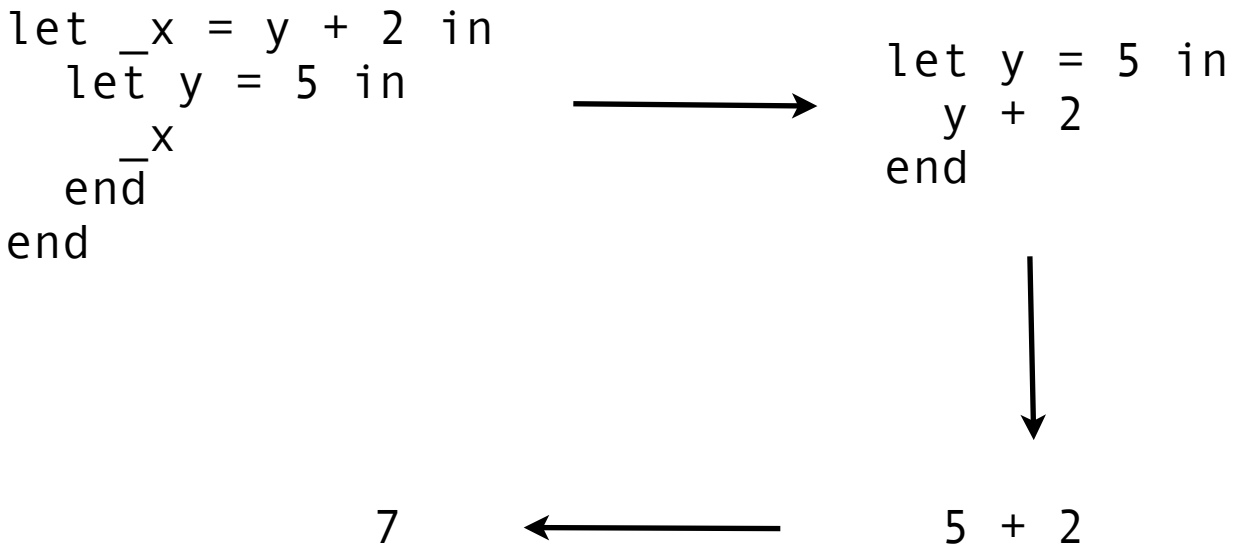
Substituição CBN

- O passo a passo:



Substituição CBN

- O passo a passo:



- O termo pelo qual estamos substituindo não pode ter variáveis livres!