Linguagens de Programação

Fabio Mascarenhas - 2015.2

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

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:

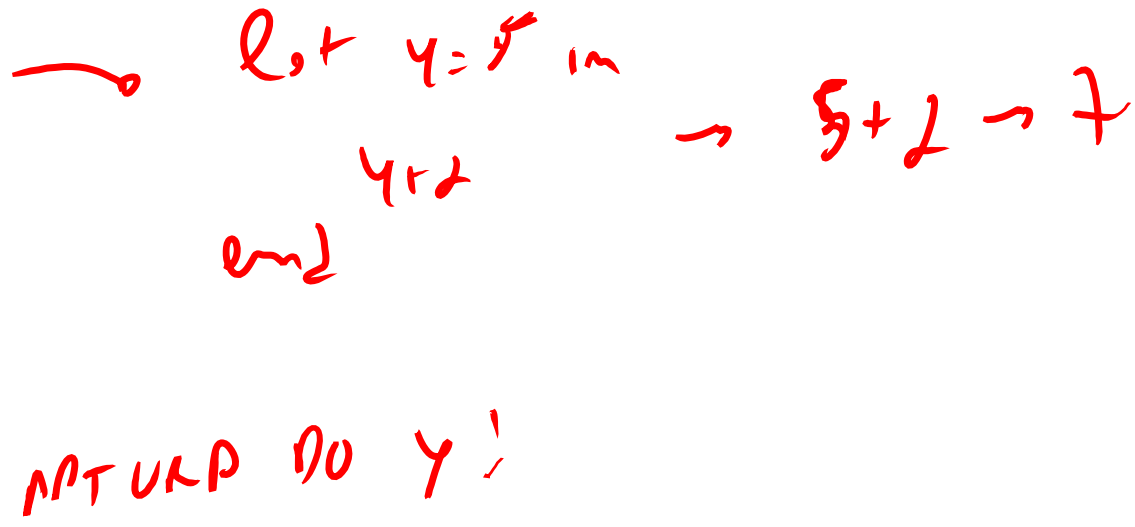
```
let _x = y + 2 in
  let y = 5 in
    _x
  end
end
```

- O resultado é 7!

Substituição CBN

- O passo a passo:

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



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

Variáveis livres

- Uma variável é *livre* se ela não é variável de nenhum let que a envolve nem parâmetro da função em que ela aparece

```
def fvs(e: Exp): Set[String] = e match {
   case Soma(e1, e2) => fvs(e1) ++ fvs(e2)
   case Mult(e1, e2) => fvs(e1) ++ fvs(e2)
   case Menor(e1, e2) => fvs(e1) ++ fvs(e2)
   case If(ec, et, ee) => fvs(ec) ++ fvs(et) ++ fvs(ee)
   case Var(x) => Set(x)
   case Ap1(nome, args) => args.map(arg => fvs(arg)).reduce((a, b) => a ++ b)
   case Let(nome, exp, corpo) => fvs(exp) ++ (fvs(corpo) - nome)
   case _ => Set()
 }
```

- Toda variável livre no termo sendo substituído é uma variável que não foi definida, portanto é um erro

pode ser!

Açúcar sintático: let múltiplo

- Um let com várias expressões, transformado em um let em cascata:

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

```
exp  : ...
     | LET ID '=' exp {, LET ID '=' exp} IN exp END
```

- Transformação fácil usando um fold no parser