

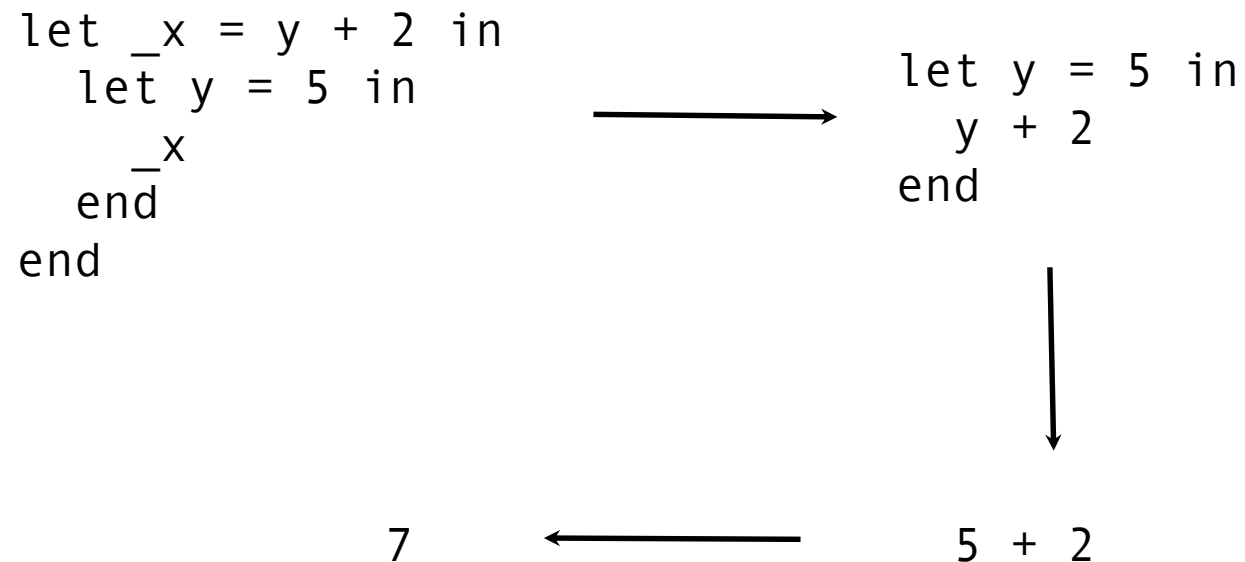
Linguagens de Programação

Fabio Mascarenhas - 2013.1

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

Substituição CBN

- O passo a passo:



- 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: Set[String] = this match {
  case Num(_) => Set()
  case True() => Set()
  case False() => Set()
  case Var(x) => Set(x)
  case Soma(e1, e2) => e1.fvs ++ e2.fvs
  case Mult(e1, e2) => e1.fvs ++ e2.fvs
  case Div(e1, e2) => e1.fvs ++ e2.fvs
  case Menor(e1, e2) => e1.fvs ++ e2.fvs
  case If(c, et, ee) => c.fvs ++ et.fvs ++ ee.fvs
  case Let(x, e, c) => e.fvs ++ (c.fvs - x)
  case Ap1(n, es) => es.foldRight(Set())(
    (e, s) => e.fvs ++ s)
}
```


Funções de primeira classe

- Vamos adicionar funções anônimas a *fun*, e uma forma de chamá-las:

```
exp : ...
    | FUN '(' params ')' exp END
    | FUN '(' ')' exp END
    | '(' exp ')' '(' ')'
    | '(' exp ')' '(' exps ')'
```

```
case class Fun(params: List[String], corpo: Exp) extends Exp
case class Ap(fun: Exp, args: List[Exp]) extends Exp
```

- Uma função agora pode ser o valor de uma expressão, então também precisamos de:

```
case class FunV(params: List[String], corpo: Exp) extends Valor
```

Funções de primeira classe

- O resultado de avaliar uma Fun é um FunV
 - Mas no momento da avaliação uma Fun não pode ter variáveis livres
 - Ou temos captura indevida na substituição mesmo para variáveis call by value

```
fun soma(a)
  fun (b)
    a + b
  end
end
```

```
(soma(2))(3)
```

```
fun soma()
  fun (b)
    a + b
  end
end
```

```
let a = 2 in (soma())(3)
```

Let é açúcar sintático

- Notem a semelhança no interpretador entre o código de let e o código de aplicar uma função
- Com funções de primeira classe, o let pode virar açúcar sintático:

`let nome = exp
in corpo` \longrightarrow `(fun (nome) corpo)(exp)`

- Mas já temos toda a infraestrutura do let no lugar, então vamos deixar como está!

Recursão

- Funções anônimas parecem não poder ser recursivas

```
let fat = fun (x)
    if x < 2 then 1
    else x * (fat)(x-1) end
end
in (fat)(5)
```

- Erro, fat está livre dentro da função anônima!
- Precisamos de um novo let:

```
letrec fat = fun (x)
    if x < 2 then 1
    else x * (fat)(x-1) end
end
in (fat)(5)
```


Letrec em duas partes

- A definição de letrec fica mais simples se ele for açúcar sintático para um let e outro termo, rec:

```
let fat = rec fat = fun (x)
    if x < 2 then 1
    else x * (fat)(x-1) end
end
in (fat)(5)
```

- O que rec faz? Ele resolve uma equação $x = T(x)$, onde $T(x)$ é um termo usando x (no caso acima, uma fun), e retorna essa solução
- Ou seja, rec acha o *ponto fixo* de $T(x)$!

Avaliando rec

- Não se assuste em aparecer o ponto fixo, não vamos ver as implicações matemáticas disso
- “Resolver” o ponto fixo, ou seja, dar uma implementação para *rec*, é fácil!
 - Basta substituir x por *rec* $x = T(x)$ em $T(x)$
 - Ou seja, *desenrolamos* $T(x)$
 - Depois avaliamos $T(x)$, ou não, se x é uma variável CBN
 - Isso quer dizer que *rec loop = loop* entra em loop infinito