

# Semântica de Linguagens de Programação

---

Fabio Mascarenhas - 2011.2

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

# F AE com Ambientes

---

- Em qual ambiente devemos interpretar o corpo de uma função?

# F AE com Ambientes

---

- Em qual ambiente devemos interpretar o corpo de uma função?
- Vamos tentar usar o ambiente que está à mão primeiro

# F AE com Ambientes

---

- Em qual ambiente devemos interpretar o corpo de uma função?
- Vamos tentar usar o ambiente que está à mão primeiro

```
{with {x 3}
  {with {f {fun {y} {+ x y}}}}
  {with {x 5} {f 4}}}}
= {with {f {fun {y} {+ x y}}}}
  {with {x 5} {f 4}}
= {with {x 5} {f 4}}
= {f 4} = {+ x y} = {+ 5 y}
= {+ 5 4} = 9
```

# F AE com Ambientes

---

- Em qual ambiente devemos interpretar o corpo de uma função?
- Vamos tentar usar o ambiente que está à mão primeiro

```
{with {x 3}
  {with {f {fun {y} {+ x y}}}
    {with {x 5} {f 4}}}}
= {with {f {fun {y} {+ x y}}}
  {with {x 5} {f 4}}}
= {with {x 5} {f 4}}
= {f 4} = {+ x y} = {+ 5 y}
= {+ 5 4} = 9
```



x = 3

# F AE com Ambientes

---

- Em qual ambiente devemos interpretar o corpo de uma função?
- Vamos tentar usar o ambiente que está à mão primeiro

```
{with {x 3}
  {with {f {fun {y} {+ x y}}}
    {with {x 5} {f 4}}}}
{f {fun {y} {+ x y}}}
```

$f = \{\text{fun } \{y\} \{+ x y\}\}$   
 $x = 3$

$x = 3$

```
= {with {x 5} {f 4}}
= {f 4} = {+ x y} = {+ 5 y}
= {+ 5 4} = 9
```

# F AE com Ambientes

- Em qual ambiente devemos interpretar o corpo de uma função?
- Vamos tentar usar o ambiente que está à mão primeiro

```
{with {x 3}
```

```
  {with {f {fun {y} {+ x y}}}}
```

```
    {with {x 5} {f 4}}}}
```

```
f = {fun {y} {+ x y}}
```

```
  x = 5  
f = {fun {y} {+ x y}}  
  x = 3
```

```
    {fun {y} {+ x y}}}}
```

```
  {x 5} {f 4}}}}
```

```
= {with {x 5} {f 4}}
```

```
= {f 4} = {+ x y} = {+ 5 y}
```

```
= {+ 5 4} = 9
```

x = 3

# FAC com Ambientes

- Em qual ambiente devemos interpretar o corpo de uma função?
- Vamos tentar usar o ambiente que está à mão primeiro

```
{with {x 3}
```

```
  {with {f {fun {y} {+ x y}}}}
```

```
    {with {x
```

```
      f = {fun {y} {+ x y}}
```

```
        x = 5  
        f = {fun {y} {+ x y}}  
        x = 3
```

```
          y = 4  
          x = 5  
          f = {fun {y} {+ x y}}  
          x = 3
```

```
x = 3
```

```
    = {with {x 5} f 4}}
```

```
    = {f 4} = {+ x y} = {+ 5 y}
```

```
    = {+ 5 4} = 9
```

# F AE com Ambientes

---

- O que acontece com FAE usando subst?

```
{with {x 3}
  {with {f {fun {y} {+ x y}}}}
  {with {x 5} {f 4}}}}
= {with {f {fun {y} {+ 3 y}}}}
  {with {x 5} {f 4}}}}
= {with {x 5}
  {{fun {y} {+ 3 y}} 4}}
= {{fun {y} {+ 3 y}} 4}
= {+ 3 4} = 7
```

# Closures

---

- Corpo da função tem que ser interpretado no ambiente em que a função foi *criada* (passou de *termo* para *valor*)
- O interpretador de substituição faz isso naturalmente (modulo o problema das variáveis livres)
- Uma solução é empacotar o ambiente junto com a função quando ela vira um valor
- Indiretamente, isso é o que fazíamos com `with`!

# Recursão

---

- Podemos escrever uma função sum em CFAE + if0?

```
{with {sum
      {fun {x}
        {if0 x
            0
            {+ x {sum {- x 1}}}}}}
      {sum 5}}
```

# Recursão

---

- Podemos escrever uma função fatorial em CFAE + if0?

```
{with {sum
      {fun {x}
        {if0 x
          0
          {+ x {sum {- x 1}}}}}}}
```

```
{sum 5}}
```

```
= {sum 5}
```

```
= {{sum {x}
```

```
  {if0 x
```

```
    0
```

```
    {+ x {sum {- x 1}}}}} 5}
```

```
= ... = "unbound identifier"
```

sum = {fun {x} ...}

vazio! (no closure)

# Recursão

---

- Escopo dinâmico resolveria o problema

```
{with {sum
      {fun {x}
        {if0 x
          0
          {+ x {sum {- x 1}}}}}}}
```

```
{sum 5}}
```

```
= {sum 5}
```

```
= {{fun {x}
```

```
  {if0 x
```

```
    0
```

```
    {+ x {sum {- x 1}}}}} 5}
```

```
= ... = 15
```

sum = {fun {x} ...}

sum = {fun {x} ...}

# Escopo Dinâmico

---

- Escopo dinâmico pode ser útil em pequenas doses
- As fundefs de F1WAE são um exemplo de escopo dinâmico restrito
- F1WAE tem escopo dinâmico para funções, mas escopo léxico para outros identificadores
- Poderíamos adicionar closures a F1WAE, e uma operação `apply` para aplicação de closures
- Common Lisp (um 2-Lisp) funciona mais ou menos assim

# Recursão

---

- Podemos escrever uma função `sum` em CFAE + `if 0`?

```
{with {sum
      {fun {x}
        {if 0 x
            0
            {+ x {sum {- x 1}}}}}}
  {sum 5}}
```

Uma outra estratégia...

# Recursão - fix

---

- Vamos definir sum como um parâmetro da função F abaixo

```
{fun {sum}
  {fun {x}
    {if0 x
      0
      {+ {sum {- x 1}} x}}}}
```

# Recursão - fix

---

- Vamos definir `sum` como um parâmetro da função `F` abaixo

```
{fun {sum}
  {fun {x}
    {if0 x
      0
      {+ {sum {- x 1}} x}}}}
```

- Se passarmos a função `sum` para a `F` vamos ter a função `sum` novamente
- A função `sum` é um *ponto fixo* da função acima! Um ponto fixo é uma solução para a equação `sum = F(sum)`.

# Recursão - fix

---

- Vamos definir uma primitiva `fix` que nos dá o ponto fixo de uma função

```
{fix {fun {<name>} <func>}}
```

```
{{fix {fun {sum}
      {fun {x}
        {if0 x
            0
            {+ {sum {- x 1}} x}}}}}} 5}
= 15
```

- Se o resultado de `fix` é uma função  $F$  tal que  $F = \text{fix}(F)$ , então  $\text{fix}(F) = F(\text{fix}(F))$ ! Ou seja,  $\text{fix}(F)$  é  $F$  substituindo o parâmetro de  $F$  por  $\text{fix}(F)$ .

# Recursão - fix

---

- E no interpretador de ambientes e closures?
- A ideia é `fix` criar uma closure cujo ambiente associa `<name>` a ela mesma

```
[fix (func)
  (local [(define newenv
            (env-rec (fun-param func) ; <name>
                    (lambda () closure)
                    env))
          (define closure ; <func>
            (interp (fun-body func) newenv))]
  closure)]
```

- A cláusula local de Scheme define um ambiente recursivo!
- Uma solução alternativa usa ponteiros: `box`, `set-box!`, `unbox`

# Recursão - rec

---

- Um pouco de açúcar sintático para facilitar

```
{rec {<name> <func>} <expr>} =>  
  {{fun {<name>} <expr>}  
   {fix {fun {<name>} <func>}}}}
```