Semântica de Linguagens de Programação

Fabio Mascarenhas - 2011.2

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

Voltando a *Eager* vs *Lazy*

```
{with {x {+ 2 y}}
  {with {y 5} {+ x y}}}
```

O que acontece nesse exemplo?

Eager

```
{with {x {+ 2 y}}
  {with {y 5} {+ x y}}}
  = “unbound identifier”
```

Lazy

```
{with {x {+ 2 y}}
   {with {y 5} {+ x y}}}
   = {with {y 5} {+ {+ 2 y} y}}
   = {+ {+ 2 5} 5}
   = {+ 7 5}
   = 12
```

Oops!

Substituição de Termos

- Nossa definição de substituição assume que o que vamos substituir é um *valor*, não um *termo*
- Por consequência, a implementação só funciona para valores
- Na verdade, funciona para qualquer termo que *não tenha variáveis livres*
- Várias soluções...
- Problema parecido com o que vamos ter com F1WAE daqui a pouco

Voltando a F1WAE

```
F1WAE ::= <num>
        | {+ <F1WAE> <F1WAE>}
        | {- <F1WAE> <F1WAE>}
        | {with {<id> <F1WAE>} <F1WAE>}
        | <id>
        | {<id> <F1WAE>}
```

Ambientes

- Nosso interpretador não é muito eficiente, já que ele percorre todo o resto do programa a cada substituição que tem que fazer
- Podemos consertar isso fazendo usando um *ambiente*
- A ideia é não fazer a substituição, mas associar o identificador ao valor que ele precisa ter nesse ambiente; quando interpretarmos um identificador procuramos ele no ambiente e retornamos o valor dele, ou um erro de identificador livre
- Parecido com o que fizemos com funções

Ambientes

```
(define-type Env
  [env-empty]
  [env-entry (name symbol?)
             (val number?)
             (next Env?)])
```

Escopo Estático e Dinâmico

- Qual deve ser o ambiente em que avaliamos uma função?

Escopo Estático e Dinâmico

- Qual deve ser o ambiente em que avaliamos uma função?
- Estamos usando o ambiente atual extendido com uma associação entre o parâmetro e o argumento
- Qual o valor de `{with {n 5} {f 10}}` com `f` tendo parâmetro `p` e corpo `{+ n p}`?

Escopo Estático e Dinâmico

- Qual deve ser o ambiente em que avaliamos uma função?
- Estamos usando o ambiente atual extendido com uma associação entre o parâmetro e o argumento
- Qual o valor de `{with {n 5} {f 10}}` com `f` tendo parâmetro `p` e corpo `{+ n p}`?
- Nosso interpretador de ambientes mudou as regras de escopo de F1WAE de escopo *estático* para escopo *dinâmico*

Escopo Estático e Dinâmico

- Vamos comparar com o interpretador de substituição

```
f(x) = {+ x y}

{with {x 5}
  {with {y 2} {f {+ x y}}}}
  = {with {y 2} {f {+ 5 y}}}
  = {f {+ 5 2}}
  = {f 7}
  = {+ 7 y}
  = “unbound identifier”
```

Escopo Estático e Dinâmico

- O corpo do segundo `with` é reescrito duas vezes (uma pra `x`, uma pra `y`)

```
f(x) = {+ x y}

{with {x 5}
  {with {y 2} {f {+ x y}}}}
  = {with {y 2} {f {+ 5 y}}}
  = {f {+ 5 2}}
  = {f 7}
  = {+ 7 y}
  = “unbound identifier”
```

Escopo Estático e Dinâmico

- O corpo de `f` só é reescrito uma vez, por causa do parâmetro de `f`!

```
f(x) = {+ x y}

{with {x 5}
  {with {y 2} {f {+ x y}}}}
  = {with {y 2} {f {+ 5 y}}}
  = {f {+ 5 2}}
  = {f 7}
  = {+ 7 y}
  = “unbound identifier”
```

Escopo Estático e Dinâmico

- Solução: o ambiente em que avaliamos o corpo de uma função deve ter *apenas* a substituição do seu parâmetro

- Avaliar uma expressão `e` em um ambiente `env` é como aplicar `(subst name val e)` para *cada par* de `env`, do mais antigo pro mais novo

Intermezzo - Compilação

- A posição onde vamos encontrar um identificador F1WAE no ambiente é sempre a mesma!
- Escopo léxico = Escopo *estático*
- Se é estático, é fácil de compilar -> CF1WAE
- Vamos trocar identificadores pela posição deles no ambiente, relativa ao topo

Intermezzo - Compilação

```
(define-type CF1WAE
  [cnum (n number?)]
  [cadd (left CF1WAE?)
        (right CF1WAE?)]
  [csub (left CF1WAE?)
        (right CF1WAE?)]
  [cwith (expr CF1WAE?)
         (body CF1WAE?)]
  [cid (pos number?)]
  [capp (func symbol?)
        (arg CF1WAE?)])
```

Funções de Primeira Classe

- F1WAE é uma linguagem com funções de *primeira ordem*
    - Funções não são valores que podem ser passados/retornados
- Em linguagens funcionais como Scheme as funções são valores como quaisquer outros (números, strings, listas...)
    - Dizemos que essas linguagens têm *funções de primeira classe*
- Vamos adicionar funções de primeira classe a WAE -> FWAE

FWAE - Exemplos

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

FWAE - Exemplos

```
{with {double {fun {x} {+ x x}}}
      {+ {double 5} {double 5}}}
  = {+ {{fun {x} {+ x x}} 5}
       {{fun {x} {+ x x}} 5}}
  = {+ {+ 5 5}
       {{fun {x} {+ x x}} 5}}
  = {+ 10
       {{fun {x} {+ x x}} 5}}
  = {+ 10 {+ 5 5}}
  = {+ 10 10}
  = 20
```

FAE

```
FWAE ::= <num>
       | {+ <FWAE> <FWAE>}
       | {- <FWAE> <FWAE>}
       | {with {<id> <FWAE>} <FWAE>}
       | <id>
       | {<FWAE> <FWAE>}
       | {fun {<id>} <FWAE>}
```

Valores de FWAE

- Qual o valor de `{fun {x} x}`?

Valores de FWAE

- Qual o valor de `{fun {x} x}`?
- Vamos usar a AST como valor -> `(fun 'x (id 'x))`
- O interpretador agora não vai retornar apenas números
  - Por higiene, vamos por uma tag nos números também, e usar `(num 5)` ao invés de `5`, por exemplo

Implementando *subst* para FWAE

- Escopo léxico tem que ser preservado
  - `{with {x 3} {{fun {y} {+ x y}} 2} => 5`
- Solução óbvia: tratar `fun` que nem `with`, substituindo variáveis livres no corpo da função

Identificadores Livres, de novo

```
{with {x {fun {z} {+ 2 y}}}
  {with {y 5} {+ {x 0} y}}}
  = {with {y 5} {+
                 {{fun {z} {+ 2 y}} 0} y}}
  = {+ {{fun {z} {+ 2 5}} 0} 5}
  = {+ {+ 2 5} 5}
  = {+ 7 5}
  = 12
```

Oops!

Dispensando *with*

- As implementações de `with` e `app` são muito parecidas
- Lembre que podemos pensar em `with` como construindo uma função a usando na mesma hora
- Logo, `{with {<var> <expr>} <body>}` pode ser reescrito como a expressão `{{fun {<var>} <body>} <expr>}`
- Podemos tirar `with` e obter FAE, mas mantendo `with` no parser como *açúcar sintático*