Semântica de Linguagens de Programação

Fabio Mascarenhas - 2011.2

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

Introdução

- Quais das expressões abaixo têm o mesmo significado?
  - a[42]
  - (vector-ref a 42)
  - a[42]
  - a[42]

Introdução

- Quais das expressões abaixo têm o mesmo significado?
  - a[42]
  - (vector-ref a 42)
  - a[42]
  - a[42]

Java

Introdução

- Quais das expressões abaixo têm o mesmo significado?
  - a[42] Java
  - (vector-ref a 42) Scheme
  - a[42]
  - a[42]

Introdução

- Quais das expressões abaixo têm o mesmo significado?
  - a[42] Java
  - (vector-ref a 42) Scheme
  - a[42] C
  - a[42]

Introdução

- Quais das expressões abaixo têm o mesmo significado?
  - a[42] Java
  - (vector-ref a 42) Scheme
  - a[42] C
  - a[42] Haskell

Introdução

- Quais das expressões abaixo têm o mesmo significado?
  - a[42] — Java ←
  - (vector-ref a 42) — Scheme ←
  - a[42] — C
  - a[42] — Haskell

Introdução

- Quais das expressões abaixo têm o mesmo significado?
  - a[42] — Java ←
  - (vector-ref a 42) — Scheme ←
  - a[42] — C
  - a[42] — Haskell
- Nesse curso vamos estudar o *significado* dos programas

Como estudar semântica?

- Precisamos de uma linguagem pra descrever semântica
- Técnicas matemáticas?

Como estudar semântica?

- Precisamos de uma linguagem pra descrever semântica
- Técnicas matemáticas?
  - Denotacional
  - Operacional
  - Axiomática

Como estudar semântica?

- Precisamos de uma linguagem pra descrever semântica
- Técnicas matemáticas?
  - Denotacional
  - Operacional
  - Axiomática
- Não, vamos usar *interpretadores*

Linguagem de Implementação

- Vamos implementar nossos interpretadores em *Scheme*
  - Sintaxe se parece com uma árvore sintática abstrata, que é o que vamos interpretar
  - Semântica simples, que reflete a semântica as primeiras linguagens que vamos definir
  - Possui as ferramentas para implementar facilmente linguagens com outras semânticas

Linguagens Implementadas

- Vamos usar uma sintaxe a la Scheme (*s-expressions*)
  - { } ao invés de ( ), [ ] — evita confusão
  - Operações prefixadas, símbolos e números
  - Vamos converter a sintaxe em uma AST
- Sintaxe descrita com BNF

AE - Expressões Aritméticas

```
AE ::= <num>
     | {+ <AE> <AE>}
     | {- <AE> <AE>}
```

AE - Expressões Aritméticas

```
AE ::= <num>
     | {+ <AE> <AE>}
     | {- <AE> <AE>}
```

```
5

{+ 3 4}

{+ {- 3 4} 7}
```

Interpretando AE

WAE - Expressões + Identificadores

- Identificadores nomeiam espressões

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

WAE - Expressões + Identificadores

- Identificadores nomeiam espressões

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

```
{with {x {+ 5 5}} {+ x x}}
```

Substituição

- Uma maneira de interpretar *with*

```
{with {x {+ 5 5}} {+ x x}}
```

Substituição

- Uma maneira de interpretar *with*

```
{with {x {+ 5 5}} {+ x x}}
  = {with {x 10} {+ x x}}
```

Substituição

- Uma maneira de interpretar *with*

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

Substituição

- Uma maneira de interpretar *with*

```
{with {x {+ 5 5}}
  {with {y {- x 3}} {+ y y}}
```

Substituição

- Uma maneira de interpretar *with*

```
{with {x {+ 5 5}}
  {with {y {- x 3}} {+ y y}}
  = {with {x 10} {y {- x 3}} {+ y y}}
```

Substituição

- Uma maneira de interpretar *with*

```
{with {x {+ 5 5}}
  {with {y {- x 3}} {+ y y}}}
  = {with {x 10}
      {with {y {- x 3}} {+ y y}}}
  = {with {y {- 10 3}} {+ y y}}
```

Substituição

- Uma maneira de interpretar *with*

```
{with {x {+ 5 5}}
  {with {y {- x 3}} {+ y y}}}
  = {with {x 10}
      {with {y {- x 3}} {+ y y}}}
  = {with {y {- 10 3}} {+ y y}}
  = {with {y 7} {+ y y}}
```

Substituição

- Uma maneira de interpretar *with*

```
{with {x {+ 5 5}}
  {with {y {- x 3}} {+ y y}}}
  = {with {x 10}
      {with {y {- x 3}} {+ y y}}}
  = {with {y {- 10 3}} {+ y y}}
  = {with {y 7} {+ y y}}
  = {+ 7 7}
  = 14
```

O que é substituição?

- **Substituição:** para *substituir* um identificador *i* em uma expressão *e* por um valor *v*, troque todos os identificadores com nome *i* em *e* pelo valor *v*.
    - Está correto?

O que é substituição?

- **Substituição:** para *substituir* um identificador *i* em uma expressão *e* por um valor *v*, troque todos os identificadores com nome *i* em *e* pelo valor *v*.

    - Está correto? **Não!**

Definições, Escopo e Amarração

- **Definição:** uma *definição* de um identificador é a instância do identificador que dá o seu valor. Em WAE, a posição `<id>` de um `with` é a única definição.

- **Escopo:** o *escopo* de uma definição é a região do texto do programa em que instâncias do identificador se referem ao valor amarrado pela definição.

- **Identificador Amarrado:** um identificador está *amarrado* se ele está no escopo de uma definição de seu nome.

- **Identificador Livre:** um identificador está *livre* se não está no escopo de nenhuma definição se seu nome.

Substituição, tomada 2

- **Substituição:** *para substituir* um identificador *i* em uma expressão *e* por um valor *v*, troque todos os identificadores com nome *i* em *e* que não são definições pelo valor *v*.
    - Está correto?

Substituição, tomada 2

- **Substituição:** *para substituir* um identificador *i* em uma expressão *e* por um valor *v*, troque todos os identificadores com nome *i* em *e* que não são definições pelo valor *v*.

    - Está correto? **Não!**

Substituição, tomada 3

- **Substituição:** para *substituir* um identificador *i* em uma expressão *e* por um valor *v*, troque todos os identificadores em *e* com o nome *i* que não são definições pelo valor *v*, a não ser que o identificador esteja em um escopo diferente do introduzido por *i.*

    - Está correto?

Substituição, tomada 3

- **Substituição:** para *substituir* um identificador *i* em uma expressão *e* por um valor *v*, troque todos os identificadores em *e* com o nome *i* que não são definições pelo valor *v*, a não ser que o identificador esteja em um escopo diferente do introduzido por *i.*

    - Está correto? **Não!**

Substituição, tomada 4

- **Substituição:** para *substituir* um identificador *i* em uma expressão *e* por um valor *v*, troque todos os identificadores em *e* com o nome *i* que não são definições pelo valor *v*, exceto em escopos aninhados de *i*.
    - Está correto?

Substituição, tomada 4

- **Substituição:** para *substituir* um identificador *i* em uma expressão *e* por um valor *v*, troque todos os identificadores em *e* com o nome *i* que não são definições pelo valor *v*, exceto em escopos aninhados de *i*.

    - Está correto? **Sim!**

Uma definição mais sucinta

- **Substituição:** para *substituir* um identificador *i* em uma expressão *e* por um valor *v*, troque todas as instâncias livres de *i* em *e* por *v*.

- Vamos implementar WAE!

Outra semântica pra `with`

- Lembre esse exemplo:

```
{with {x {+ 5 5}}
  {with {y {- x 3}} {+ y y}}}
  = {with {x 10}
      {with {y {- x 3}} {+ y y}}}
  = {with {y {- 10 3}} {+ y y}}
  = {with {y 7} {+ y y}}
  = {+ 7 7}
  = 14
```

Outra semântica pra `with`

- Ele poderia ser escrito desse jeito?

```
{with {x {+ 5 5}}
  {with {y {- x 3}} {+ y y}}}
  = {with {y {- {+ 5 5} 3}} {+ y y}}
  = {+ {- {+ 5 5} 3} {- {+ 5 5} 3}}
  = {+ {-10 3} {- {+ 5 5} 3}}
  = {+ 7 {- {+ 5 5} 3}}
  = {+ 7 {- 10 3}}
  = {+ 7 7}
  = 14
```

Eager vs. *Lazy*

- Calcular e depois substituir → regime de avaliação *eager*
  - Regime da maioria das linguagens de programação
  - Vamos continuar usando ele por ora
- Substituição textual → regime de avaliação *lazy* (quase...)
  - Haskell
  - Mesmo resultado para essa WAE, mas logo isso vai mudar
  - Voltaremos a isso mais tarde!

Funções

- O `with` de WAE parece muito com uma função com argumento constante
- Vamos introduzir funções na nossa linguagem
  - De início, vamos separar a linguagem em *declarações* e *expressões*
  - O interpretador irá interpretar uma expressão assumindo um conjunto de *declarações de funções*
  - A expressão é como a função `main` de um programa C

F1WAE - Expressões

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

F1WAE - Declarações

- Vamos usar um tipo pra declarações, mas sem uma sintaxe específica
  - O *corpo* de uma declaração é uma expressão F1WAE

F1WAE - Declarações

- Vamos usar um tipo pra declarações, mas sem uma sintaxe específica
  - O *corpo* de uma declaração é uma expressão F1WAE

```
(interp (parse `{double {double 5}})
        (list (fundef `double
                      `n
                      (parse
                           `{+ n n}))))
```

Avaliando

```
{double {double 5}}
  = {double {+ 5 5}}
  = {double 10}
  = {+ 10 10}
  = 20
```

1-Lisp vs *2-Lisp*

- O que acontece se avaliarmos `{f 10}` se `f` é declarada com parâmetro `n` e corpo `{n n}`?

1-Lisp vs *2-Lisp*

- O que acontece se avaliarmos `{f 10}` se `f` é declarada com parâmetro `n` e corpo `{n n}`?
  - Nosso interpretador tem espaços separados pra funções e pra outros identificadores, mesmo ambos sendo `<id>`!
  - Dizemos que F1WAE é um 2-Lisp
  - Scheme é um 1-Lisp

Recursão

- O que acontece se declararmos `f` com parâmetro `x` e corpo `{g {+ x 5}}` e `g` com parâmetro `x` e corpo `{- x 1}`?

Recursão

- O que acontece se declararmos `f` com parâmetro `x` e corpo `{g {+ x 5}}` e `g` com parâmetro `x` e corpo `{- x 1}`?

- As funções de F1WAE enxergam-se umas às outras, e a ordem de declaração não importa

- Isso implica que F1WAE tem recursão! Mas falta uma maneira de terminar a recursão...

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

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*