Centro de Informática U·F·P·E

# Sintaxe e Semântica

George Darmiton da Cunha Cavalcanti
(gdcc@cin.ufpe.br)

# Tópicos

- Introdução
- O problema de descrever a sintaxe
- Métodos formais para descrever a sintaxe
- Gramáticas de atributos
- Descrevendo o significado dos programas: semântica dinâmica

# Semântica

- Descreve o significado das expressões, das instruções e das unidades de programas
- Razões para descrever a semântica
  - Saber precisamente o que as instruções de uma linguagem fazem
  - As provas de exatidão do programa recorrem a descrição formal da semântica da linguagem
- Não existe uma notação ou um formalismo amplamente aceito para descrever semântica
  - Semântica Operacional
  - Semântica Axiomática
  - Semântica Denotacional

# Tipos de Semântica Formal


- **Semântica Operacional**
  - Como o programa é executado? Que operações são realizadas?
- **Semântica Denotacional**
  - O que o programa significa? Que objetos matemáticos ele denota?
- **Semântica Axiomática**
  - Quais proposições lógicas são válidas para um programa?

    ex.:{x = 4} x = x + 1 {x = 5}

# Semântica Operacional


- Descreve o significado de um programa pela execução de suas instruções em uma máquina, seja ela real ou simulada

- As mudanças no estado da máquina (memória, registradores, etc.) definem o significado da instrução

# Semântica Operacional

- Para usar a semântica operacional em linguagens de alto nível, uma máquina virtual é necessária
- Um interpretador puro (*hardware*) seria muito caro
- Um interpretador puro (*software*) possui problemas
  - Os detalhes de um computador em particular (do hardware e do sistema operacional) tornariam algumas ações de difícil entendimento
  - Tal definição semântica apenas estaria disponível àquelas pessoas com um computador identicamente configurado
- Esses problemas podem ser evitados substituindo um computador real por uma computador virtual

# Semântica Operacional



- Uma alternativa melhor
  - Uma simulação completa por computador
- O processo:
  - Construir um tradutor (traduzir o código-fonte para o código de máquina de um computador idealizado)
  - Construir um simulador para o computador idealizado

# Semântica Operacional

- Esse processo básico das semânticas operacionais não é incomum.
  - É usado nos livros didáticos de programação e nos manuais de consulta das linguagens
- Por exemplo, a semântica da construção do `for` em `C`

Instrução C

```
for (expr1; expr2; expr3) {
   ...
}
```

Semântica Operacional

```
expr1;
loop: if expr2 = 0 goto out
   ...
   expr3
   goto loop
out:...
```

# Semântica Operacional: Avaliação



- Bom se usado informalmente (manuais da linguagem, etc.)
- Extremamente complexo se usado formalmente

# Semântica Axiomática


- Baseada em lógica (cálculo de predicados)
- Propósito original
  - Verificação formal de programas
- Axiomas ou regras de inferência são definidas para cada tipo de instrução na linguagem (para permitir transformações de expressões para outras expressões)
- As expressões são chamadas de asserções (predicados)

# Semântica Axiomática

- Pré-condição
  - Uma asserção que precede imediatamente uma instrução de programa e descreve as variáveis dela nesse ponto
- Pós-condição
  - Uma asserção que segue imediatamente a uma instrução e descreve as novas restrições a essas variáveis (e possivelmente a outras) depois da execução da instrução
- Exemplo
  - `soma = 2*x + 1 {soma >1}`

Pós-condição

Uma **pré-condição** possível é {x > 10}

# Semântica Axiomática

- A **pré-condição mais fraca** é a menos restritiva que garantirá a validade da pós-condição associada.
- Exemplo
  - `soma = 2*x + 1 {soma >1}`
- Nesse exemplo existem várias pré-condições possíveis `{x>10}, {x>100}, {x>1000}`
- Entretanto, a mais fraca de todas as pré-condições, nesse caso, é `{x>0}`

# Forma da Semântica Axiomática


- Forma:
  - pré-condição {P} e pós-condição {Q}
  - `{P} statement {Q}`

- Um exemplo
  - `a = b + 1  {a > 1}`
  - Uma pré-condição possível
    - `{b > 10}`
  - pré-condição mais fraca
    - `{b > 0}`

# Processo de Prova de Programa


- A pós-condição do programa inteiro é o resultado desejado
  - Volte no programa até a primeira instrução
  - Se a pré-condição da primeira instrução é a mesma da especificação do programa, então o programa é correto
  - Ou seja, a primeira pré-condição declara as condições sob as quais se computará os resultados desejados
- Axioma
  - É uma afirmação lógica que se presume verdadeira
- Regra de inferência
  - É um método de suposição da verdade de uma asserção, baseando-se nos valores de outras asserções

# Instruções de Atribuição


- `a = b/2-1 {a<10}`
- A pré-condição mais fraca será computada da seguinte maneira
- `b/2-1 < 10`
- `b < 22`

# Instruções de Atribuição



- Prova de exatidão de programas
- Um outro exemplo
- `{x>3} x = x - 3 {x>0}`
- Usando o axioma de atribuição em
- `x = x - 3 {x>0}`
- Produz `{x>3}`, que é a pré-condição dada
- Portanto, provamos a instrução lógica acima

# Instruções de Atribuição

- Entretanto, nesse caso
- `{x>5} x = x - 3 {x>0}`
- A pré-condição não é a mesma que a asserção produzida pelo axioma.
- Porém, $\{x>5\} \Rightarrow \{x>3\}$
- Para usarmos isso numa prova, precisamos de uma regra de inferência, chamada **regra de conseqüência**

$$\frac{\{P\}S\{Q\}, P' \Rightarrow P, Q \Rightarrow Q'}{\{P'\}S\{Q'\}}$$

# Instruções de Atribuição

- A regra de conseqüência é bastante útil em provas
- Se admitirmos que `P` é `{x>3}`, que `Q` e `Q'` são `{x>0}` e `P'` é `{x>5}`, teremos

$$\frac{\{P\}S\{Q\}, P' \Rightarrow P, Q \Rightarrow Q'}{\{P'\}S\{Q'\}}$$

$$\frac{\{x>3\}\ \ x=x-3\ \ \{x>0\}, \{x>5\} \Rightarrow \{x>3\},\ \ \{x>0\} \Rightarrow \{x>0\}}{\{x>5\}\ \ x=x-3\ \ \{x>0\}}$$

# Semântica Axiomática: Seqüências

- Seqüência de instruções: S1 e S2
- {P1} S1 {P2}
- {P2} S2 {P3}
- A regra de inferência para essa seqüência de duas instruções será

$$\frac{\{P1\}\ S1\ \{P2\},\ \{P2\}\ S2\ \{P3\}}{\{P1\}\ S1;\ S2\ \{P3\}}$$

# Semântica Axiomática: Seqüências

## Exemplo

```
y = 3 * x + 1
x = y + 3
{x < 10}
```

A pré-condição para a última instrução de atribuição

```
y < 7
```

A pré-condição para a primeira instrução de atribuição

```
3 * x + 1 < 7
x < 2
```

# Semântica Axiomática: Seleção


{B and P} S1 {Q}, {(not B) and P} S2 {Q}

{P} **if** B **then** S1 **else** S2 {Q}

Exemplo

```
if (x>0)
    y = y - 1
else y = y + 1
```

Supondo que a pós-condição {Q} é `{y > 0}`

Axioma de atribuição para o **then**

`y = y - 1 {y>0}`, produzirá `y>1`

Axioma de atribuição para o **else**

`y = y + 1 {y>0}`, produzirá `y>-1`

Sabendo que `{y>1}⇒{y>-1}`, a regra de conseqüência nos permite usar `{y>1}` para a pré-condição da instrução de seleção.

# Semântica Axiomática: Laços de Pré-teste Lógico


- Regra de inferência para laços de pré-teste lógico

{P} **while** B **do** S **end** {Q}

$$\frac{(\text{I and B})\ \text{S}\ \{\text{I}\}}{\{\text{I}\}\ \text{while B do S}\ \{\text{I and (not B)}\}}$$

I é a invariante do laço (hipótese indutiva)

I é uma asserção crucial para encontrar a pré-condição mais fraca

# Semântica Axiomática: Laços de Pré-teste Lógico


- Características do invariante do laço: I deve seguir as seguintes condições:
  - $P \Rightarrow I$ -- o invariante de laço deve ser verdadeiro inicialmente
  - {I} B {I} -- avaliação Boolean não deve mudar a validade de I
  - {I and B} S {I} -- I não é modificada pela execução do corpo do laço
  - (I and (not B)) $\Rightarrow$ Q -- se I é verdadeiro e B é falso, então o laço é finalizado

# Semântica Axiomática: Laços de Pré-teste Lógico


- Exemplo: é útil tratarmos o processo de produção de uma condição mais fraca como uma função, wp

- wp(instrução, pós-condição) = pré-condição

- **`while`** `y<>x` **`do`** `y=y+1` **`end`** `{y=x}`

# Semântica Axiomática: Laços de Pré-teste Lógico


```
while y<>x do y=y+1 end {y=x}
```

Para zero iterações, a pré-condição mais fraca é

```
{y=x}
```

Para uma iteração:

```
wp(y=y+1, {y=x}) = {y+1=x}, ou {y=x-1}
```

Para duas iterações:

```
wp(y=y+1, {y=x-1}) = {y+1=x-1}, ou {y=x-2}
```

Para três iterações:

```
wp(y=y+1, {y=x-2}) = {y+1=x-2}, ou {y=x-3}
```

# Semântica Axiomática: Laços de Pré-teste Lógico


- Pode-se ver que `{y<x}` para casos de uma ou mais iterações
- Combinando isso com `{y=x}`, zero iterações
- Obtemos `{y<=x}`
- O qual pode ser usado como a invariante de laço

# Semântica Axiomática: Laços de Pré-teste Lógico

- Uma pré-condição para a instrução **while** pode ser determinada a partir da invariante de laço

- Nesse caso P = I, pode ser usado

- Mas devemos assegurar a exigência dos critérios anteriores

# Semântica Axiomática: Laços de Pré-teste Lógico

Avaliação dos critérios

1. P = I, P $\Rightarrow$ I (I = {y<=x})
2. I não é afetada pela avaliação da expressão booleana (y <> x)
3. {I and B} S {I}
   - {y<=x and y<>x} y = y + 1 {y<=x}
   - Aplicando o axioma da atribuição y=y+1 {y<=x},
   - obtemos {y+1<=x}, que é equivalente a {y<x}
   - O que é conseqüência de {y<=x and y<>x}
4. (I and (not B)) $\Rightarrow$ Q
   - {(y<=x) and not(y<>x)} => {y=x}
   - {(y<=x) and y=x} => {y=x}
   - {y=x} => {y=x}
5. O laço é finalizado?
   - `{y<=x} while y<>x do y=y+1 end {y=x}`

# Avaliação da Semântica Axiomática

- Desenvolver axiomas ou regras de inferência para todas as instruções de uma linguagem é difícil
- É uma boa ferramenta para executar provas de exatidão de programas
- É um excelente *framework* para ponderar acerca de programas, mas não é *fácil de ser usada* por usuários das linguagens e nem por codificadores de compiladores
- É bastante útil para descrever o significado das linguagens de programação

# Semântica Denotacional

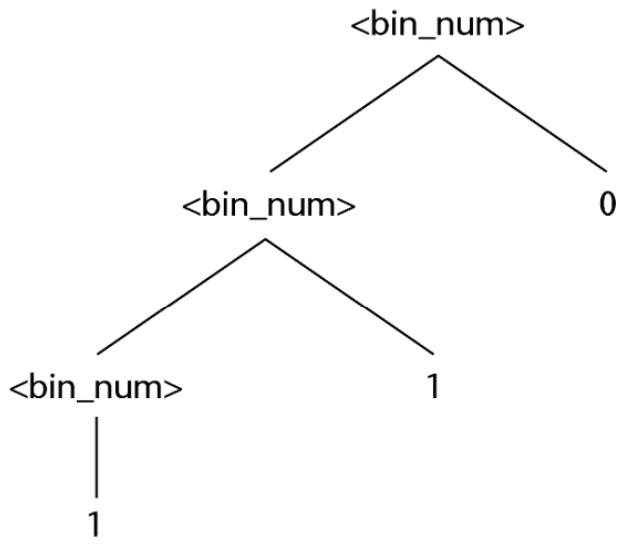
- Baseado na teoria de funções recursivas
- O método mais abstrato de descrição da semântica
- Originalmente desenvolvido Scott e Strachey (1970)

# Semântica Denotacional

- O processo de construção da especificação denotacional para uma linguagem define um objeto matemático para cada entidade da linguagem
  - Define uma função que mapeia instâncias das entidades da linguagem em instâncias dos correspondentes objetos matemáticos
- A idéia baseia-se no fato de que há maneiras rigorosas de manipular objetos matemáticos, mas não construções de linguagens de programação

# Semântica Denotacional: Exemplo

- Um exemplo simples usando números binários




<num_bin> → 0
| 1
| <num_bin> 0
| <num_bin> 1

# Semântica Denotacional: Exemplo

- Nesse caso, o significado de um número binário será seu número decimal equivalente

- A função $M_{bin}$ relaciona os objetos sintáticos com valores decimais

$$M_{bin}(\text{`0'}) = 0$$

$$M_{bin}(\text{`1'}) = 1$$

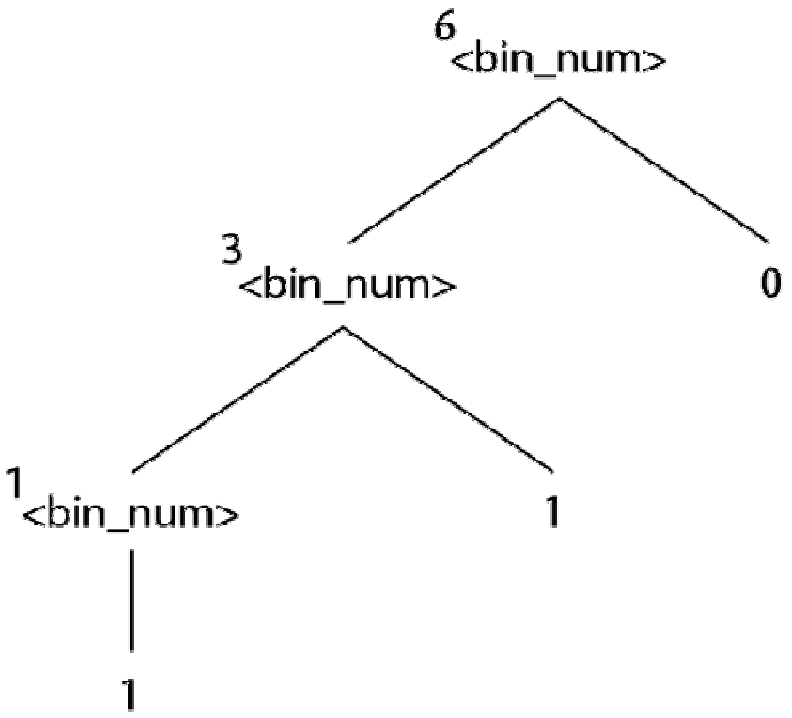
$$M_{bin}(\langle num\_bin \rangle\ \text{`0'}) = 2 * M_{bin}(\langle num\_bin \rangle)$$

$$M_{bin}(\langle num\_bin \rangle\ \text{`1'}) = 2 * M_{bin}(\langle num\_bin \rangle) + 1$$

# Semântica Denotacional: Exemplo

**Figure 3.10**

A parse tree with denoted objects for 110




- Essa é a semântica dirigida para a sintaxe
- As entidades sintáticas são associadas a objetos matemáticos com significado concreto

# Semântica Denotacional: Exemplo

- Um exemplo semelhante para descrever o significado sintático dos literais decimais

```
<dec_num> →  0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9
           | <dec_num> (0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9)
```

As relações denotacionais para essas regras de sintaxe são:

$M_{dec}$('0') = 0,  $M_{dec}$ ('1') = 1, …,  $M_{dec}$ ('9') = 9

$M_{dec}$ (<dec_num> '0') = 10 * $M_{dec}$ (<dec_num>)

$M_{dec}$ (<dec_num> '1') = 10 * $M_{dec}$ (<dec_num>) + 1

…

$M_{dec}$ (<dec_num> '9') = 10 * $M_{dec}$ (<dec_num>) + 9

# Semântica
# Denotacional vs Operacional


- Na semântica operacional, as mudanças de estados são definidas por **algoritmos codificados**

- Na semântica denotacional, as mudanças de estados são definidas por **funções matemáticas**

# Semântica Denotacional: o estado de um programa


- O estado de um programa é denotado pelos valores de todas as variáveis correntes

$$s = \{\langle i_1, v_1\rangle, \langle i_2, v_2\rangle, \ldots, \langle i_n, v_n\rangle\}$$

cada $i$ é o nome de uma variável e $v$ o seu valor

- Seja **VARMAP** uma função que, quando é dado o nome da variável e um estado, retorna o valor corrente da variável

$$\mathrm{VARMAP}(i_j, s) = v_j$$

# Números Decimais


<dec_num> → 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9
| <dec_num> (0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9)

$M_{dec}$('0') = 0, $M_{dec}$ ('1') = 1, …, $M_{dec}$ ('9') = 9
$M_{dec}$ (<dec_num> '0') = 10 * $M_{dec}$ (<dec_num>)
$M_{dec}$ (<dec_num> '1') = 10 * $M_{dec}$ (<dec_num>) + 1
…
$M_{dec}$ (<dec_num> '9') = 10 * $M_{dec}$ (<dec_num>) + 9

# Semântica Denotacional: Expressões


```
<expr> → <num_dec> | <var> | <expr_binária>

<expr_binária>
   → <expr_esquerda> <operador> <expr_direita>

<operador> → + | *
```

Mapeia as expressões no conjunto Z ∪ {error}

Assumindo que expressões podem ser: números decimais, variáveis ou expressões booleanas com um operador aritmético e dois operandos

# Semântica Denotacional: Expressões


```
Me(<expr>, s) Δ=
    case <expr> of
      <dec_num> => Mdec(<dec_num>, s)

          <var> =>  if VARMAP(<var>, s) == undef
                    then error
                    else VARMAP(<var>, s)
  <binary_expr> =>
            if (Me(<binary_expr>.<left_expr>, s) == undef OR
                Me(<binary_expr>.<right_expr>, s) = undef)
            then error
            else if (<binary_expr>.<operator> == '+' then
                              Me(<binary_expr>.<left_expr>, s) +
                              Me(<binary_expr>.<right_expr>, s)
                         else Me(<binary_expr>.<left_expr>, s) *
                              Me(<binary_expr>.<right_expr>, s)
        ...
```

# Semântica Denotacional: Instrução de Atribuição

Mapeamento de um conjunto de estados em outro conjunto de estados

$M_a$(x := E, s) Δ=
    if $M_e$(E, s) == **error**
    then **error**
    else s′ = {<$i_1$′,$v_1$′>,<$i_2$′,$v_2$′>,...,<$i_n$′,$v_n$′>},
        where for j = 1, 2, ..., n,
            $v_j$′ = VARMAP($i_j$, s), if $i_j$ <> x;
                = $M_e$(E, s), if $i_j$ == x

As duas comparações nas duas últimas linhas, $i_j$<>x e $i_j$==x, são de nomes, não de valores

# Semântica Denotacional: Laços de Pré-teste Lógico

- Mapeamento de conjunto de estado em conjunto de estado
- Para simplificar, duas novas funções são usadas
  - $M_{s1}$ que relaciona listas de instruções com estados
  - $M_b$ que relaciona expressões booleanas com valores booleanos (ou **error**)

```
M1(while B do L, s) Δ=
    if Mb(B, s) == undef
        then error
        else if Mb(B, s) == false
            then s
            else if Ms1(L, s) == error
                 then error
                 else M1(while B do L, Ms1(L, s))
```

# Significado do Laço

- O significado do laço é o valor das variáveis do programa após os comandos do laço terem sido executados um número pré-determinado de vezes, assumindo que não ocorram erros

- Na essência, o laço foi convertido de iteração para recursão, sendo o controle recursivo definido matematicamente por outras funções de mapeamento de estados recursivos

- Recursão, quando comparada com iteração, é mais fácil de descrever com rigor matemático

# Avaliação da Semântica Denotacional


- Pode ser usada para provar a correção de um programa
- Fornece um meio rigoroso para se pensar sobre o programa
- Pode ser útil no projeto da linguagem
  - Instruções com descrições complexas podem ser reavaliadas
- Pode ser usada na geração automática de compiladores
  - Somente para linguagens bastante simples
- Devido à complexidade pouco usada por usuários da linguagem

# Resumo

- BNF e gramática livre de contexto são equivalentes a meta-linguagens
  - Adequadas para descrever a sintaxe de linguagens de programação
- Gramática de atributos é um formalismo que permite descrever a sintaxe e a semântica estática de uma linguagem
  - A semântica estática é aquela que pode ser verificada em tempo de compilação
- Três métodos para a descrição da semântica dinâmica são
  - Operacional, Axiomática, Denotacional