ICC043/IEC582 – Paradigmas de Linguagens de Programação

Análise semântica

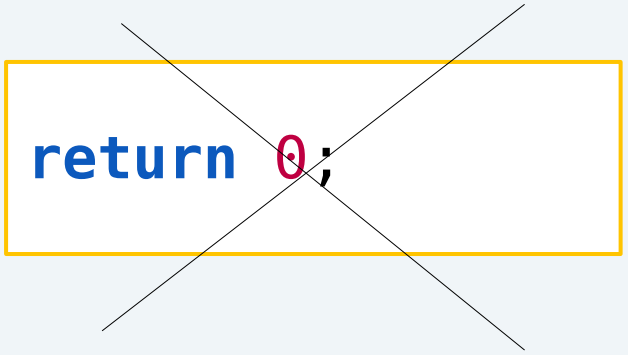
Prof. Dr. Rafael Giusti
rgiusti@icomp.ufam.edu.br

Leitura recomendada

- Sebesta. *Concepts of programming languages*
  - Capítulo 3: *Describin Syntax and Semantics*
    - Seção 3.4: *Attribute Grammars*
    - Seção 3.5:

Semântica

- Diz respeito ao significado dos programas
- Ocorre depois ou junto com a análise sintática
  - A estrutura hierárquica do programa já é ao menos parcialmente conhecida




```c
int main(void)
{
    return 0;
}
```

Qual é o significa de um comando "return" no contexto de uma função que retorna certo tipo de dados?

Semântica estática e semântica dinâmica

- Semântica estática está marginalmente relacionada com o significado dos programas
  - Ela questiona **se** uma certa forma sintática possui um significado válido
  - Cuida de validações que são difíceis ou impossíveis se fazer com GLC
- A semântica dinâmica trata do comportamento do programa
  - Ela question **qual é** o significa de uma forma sintática válida

Semântica estática

- Exemplo:
  - Declaração correta de nomes

```go
package main
import "fmt"

func main() {
    var a string
    var a string = "Hello, world!"
    fmt.Println(a)
}
```

./main.go:6:9: a redeclared in this block
previous declaration at ./prog.go:5:9

Semântica estática

- Exemplo:
  - Verificação de tipos

```
package main
import "fmt"

func main() {
    var f float32 = 3.14
    var d int = f
    fmt.Println(d)
}
```

./prog.go:7:7: cannot use f (type float32) as type int in assignment

Gramática de atributos

- Uma forma de especificar semântica estática
- As gramáticas de atributos estendem as gramáticas livres de contexto
  - Acrescentam atributos que dão significado aos símbolos das gramáticas
  - Acrescentam funções que permitem especificar relações entre os atributos
  - Acrescentam predicados que permitem verificar se as relações são corretas

Gramáticas de atributos

- Os atributos acrescentam significado aos símbolos terminais e não terminais
  - O conjunto de atributos de uma gramática é definido no projeto da linguagem
  - Atributos podem especificar, por exemplo, tipos de dados ou lexemas

```
<atrib> ::= <var> := <expr>
```

Podemos estipular que **<var>.string** é a sequência de caracteres que foi associado ao *token* **<var>** durante a análise sintática.

Gramáticas de atributos

- As funções de atributos permitem realizar computações com os atributos
  - A função lookup(·) pode verificar se uma variável existe na tabela de símbolos e/ou retornar seu tipo

```
<atrib> ::= <var> := <expr>
```

Podemos estipular que **lookup(<var>.string)** retorna o tipo de dados de uma variável cujo lexema foi associado ao *token* **<var>** durante a análise sintática

Gramáticas de atributos

- Os predicados permitem estabelecer condições para um programa ser considerado válido
  - Predicados são expressões lógicas envolvendo atributos e funções de atributos
  - Se um predicado for falso, então o programa contém um erro

```
<atrib> ::= <var> := <expr>
```

Podemos escrever o predicado
is_declared(<var>.string) == TRUE

Gramáticas de atributos

- Exemplo (1):
  - Na linguagem Ada, a declaração de um procedimento exige que o mesmo nome seja utilizado no protótipo e no end

```
procedure Hello is
begin
   Put_Line("Hello, world!");
end Hello;
```

Gramáticas de atributos

- Exemplo (1):
  - Regra sintática

```
<def_proc> ::= procedure <id>[1] is
                    begin <stmt_list> end <id>[2]
```

  - Regra semântica

```
<id>[1].string == <id>[2].string
```

Gramáticas de atributos

- Exemplo (2):
  - Em Pascal, variáveis são declaradas com a palavra reservada **var**

```
procedure Hello(Nome: string);
var
  Nome: string;
begin
  WriteLn("Hello " + Nome)
end
```

A variável local Nome foi redeclarada no mesmo escopo que o parâmetro Nome

Gramáticas de atributos

- Exemplo (2):
  - Regra sintática

```
<var_decl_full> ::= var <var_decl> {; <var_decl>}

<var_decl> ::= <var_list>: <type>

<var_list> ::= <id>
             | <id>, <var_list>
```

  - Regra semântica

```
lookup(<id>.string) == FALSE
```

Tipos de dados

- O tipo verdadeiro de uma variável ou expressão é o tipo verificado para ela
  - Obtido a partir da declaração da variável
  - Ou ao analisar uma expressão

- O tipo esperado de uma variável ou expressão é aquele que o compilador espera encontrar
  - Normalmente deriva do tipo verdadeiro de uma variável em uma atribuição ou declaração
  - Usado antes que o tipo verdadeiro de uma expressão seja conhecido

Tipos de dados

- Suponha a seguinte declaração em um programa em Pascal

```
var
  Idade: Integer = Constante;
```

  - O tipo verdadeiro de Idade é Integer
  - O tipo esperado de Constante é Integer
  - O compilador precisa verificar qual é o tipo verdadeiro de Constante para conhecer a semântica dessa declaração de variável

Tipos de dados

- Suponha a seguinte declaração em Go

```
var expressao float32 = 42.5;
var idade int = expressao;
```

  - O tipo verdadeiro de `Idade` é `int`
  - O tipo esperado de `expressao` é `int`
  - Precisamos verificar qual é o tipo verdadeiro de `expressao`
    - Quando descobrirmos que seu tipo verdadeiro é `float32`, teremos um erro de truncamento

Gramática de atributos com tipo

- As verificações de tipos de dados podem ser especificadas por meio de funções de atributos e predicados em gramáticas de atributos
  - É necessário considerar o momento em que cada regra está sendo analisada
    - Em determinados momentos da análise sintática, podemos conhecer tanto o tipo verdadeiro quanto o esperado
    - Em outros momentos, conheceremos apenas o tipo esperado

Gramática de atributos com tipo

- Suponha uma linguagem de expressões simples
  - Permite apenas três variáveis, cujos nomes são A, B ou C
  - As variáveis podem ser `int` ou `float`
  - O único tipo de expressão permitido em nossa linguagem é a soma de duas variáveis
  - Uma expressão tem tipo `float` se pelo menos uma das duas variáveis for `float`
  - Uma variável só pode receber uma expressão do seu próprio tipo

Gramática de atributos com tipo

- Exemplo:
  - `A = 1`
    - Declara uma variável de nome A e tipo `int`
  - `A = 3.14`
    - Declara uma variável de nome A e tipo `float`
  -
    ```
    A = 1
    B = 3.14
    C = A + B
    ```
    - Declara uma variável de nome C e tipo `float`

Gramática de atributos com tipo

- Exemplo:
  - `A = 1`
    `A = 10.10`
    - Declara uma variável de nome A e tipo `int`
    - Tenta atribuir um ponto flutuante a uma variável de tipo int → erro
  - `A = B`
    - Tenta declarar uma variável a partir de uma variável que ainda não foi declarada
    - Vamos tratar isso como "comportamento indefinido"

Gramática de atributos com tipo

- Sintaxe da linguagem

```
<assign> ::= <var> = <expr>

<expr> ::= <var> + <var>
         | <var>

<var> ::= A | B | C
```

- Para cada regra, criaremos as regras semânticas e os predicados necessários

Gramática de atributos com tipo

1. Regra sintática: <assign> ::= <var> = <expr>
   Regra semântica: <expr>.*tipo_esperado* = <var>.*tipo_real*

2. Sintaxe: <expr> ::= <var>[2] + <var>[3]
   Semântica: <expr>.*tipo_real* = **if** (<var>[2].*tipo_real* == int)
   and (<var>[3].*tipo_real* == int)
   then int
   else float
   end if
   Predicado: <expr>.*tipo_real* == <expr>.*tipo_esperado*

3. Sintaxe: <expr> ::= <var>
   Semântica: <expr>.*tipo_real* = <var>.*tipo_real*
   Predicado: <expr>.*tipo_real* == <expr>.*tipo_esperado*

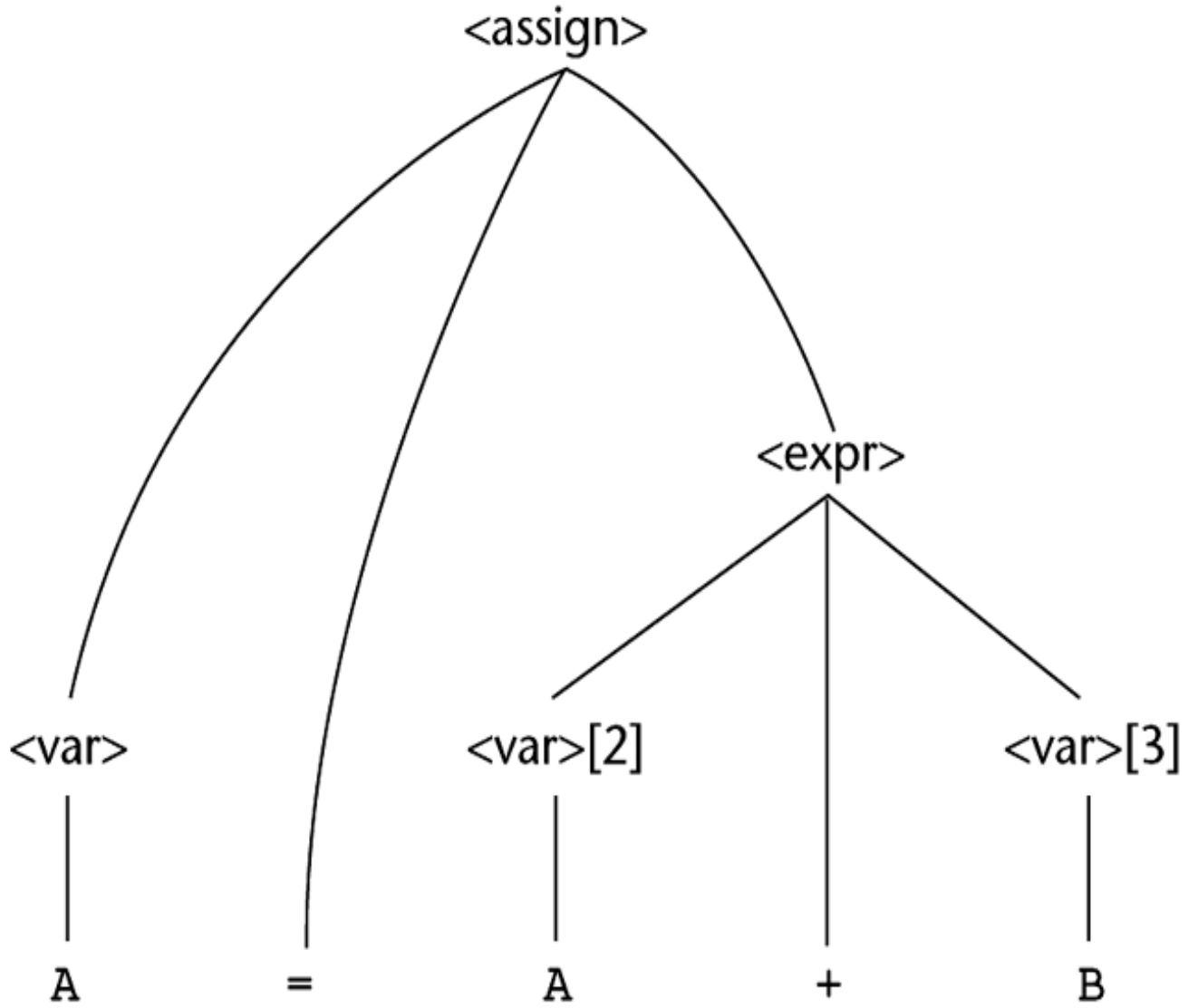
4. Sintaxe: <var> ::= A | B | C
   Semântica: <var>.tipo_real = lookup_type(<var>.string)

Gramática de atributos com tipo

Figure 3.6

A parse tree for

`A = A + B`



Fonte: SEBESTA, R. W.

1. <var>.tipo_real = lookup_type(A) //Regra 4

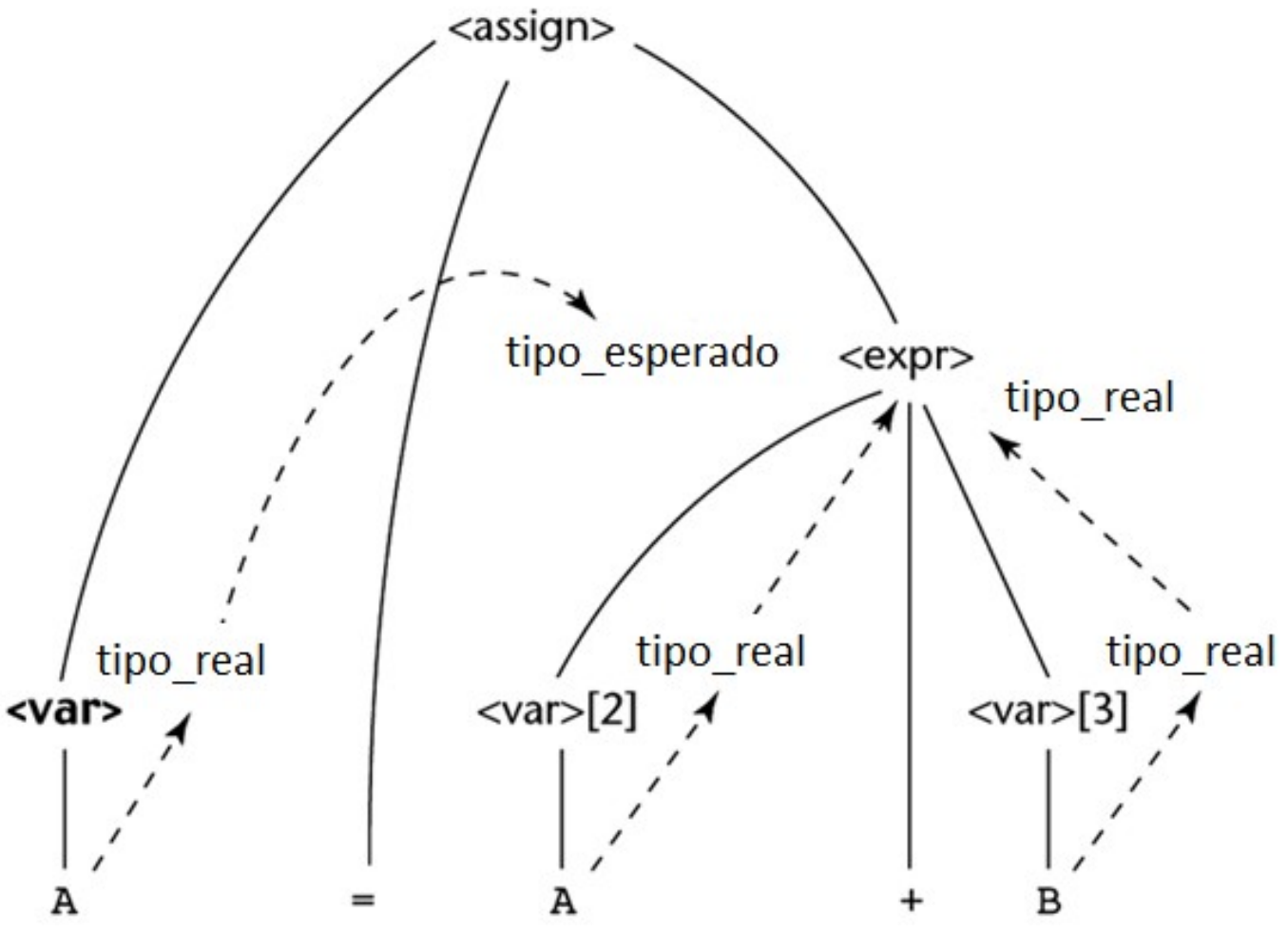
2. <expr>.tipo_esperado = var.tipo_real //Regra 1

3. <var>[2].tipo_real = lookup_type (A) //Regra 4

<var>[3].tipo_real = lookup_type (B) //Regra 4

4. <expr>.tipo_real = int ou float //Regra 2

5. <expr>.tipo_esperado == <expr>.tipo_real é TRUE ou FALSE //Regra 2



Categorias de atributos

- Atributos podem ser sintetizados ou herdados
  - Atributos sintetizados são aqueles cujos valores derivam de nós descendentes na árvore de análise sintática
    - Os atributos sintetizados transmitem informação "de baixo para cima"
    - O tipo verdadeiro de uma expressão é um atributo normalmente sintetizado
      - Ele precisa ser calculado após a derivaçãodo não terminal <expr>

Categorias de atributos

- Atributos podem ser sintetizados ou herdados
  - Atributos herdados são aqueles cujos valores dependem dos ancestrais ou dos irmãos dos nós na árvore
    - Os atributos herdados transmitem informação "de cima para baixo" e "transversalmente" na árvore de derivação
    - O tipo esperado de uma expressão é um atributo normalmente herdado
      - Ele é definido em uma regra e influencia a derivação das regras subsequentes

Semântica dinâmica

- A semântica dinâmica se ocupa com o significado de programas válidos
  - Sabendo-se que uma forma sintática é válida, qual é o seu significado?

```
$a = 42;
$b = ;
print $a + $b;
```

Forma inválida

```
$a = 42;
$b = "13";
print $a + $b;
```

Imprima 42 + 13 (55)

Semântica dinâmica

- Não existe consenso quanto a notação ou formalismo para descrever semântica

- Algumas formas incluem
  - Semântica operacional
  - Semântica denotacional
  - Semântica axiomática
  - Semântica pragmática

Semântica operacional

- A semântica operacional descreve o significado de uma sentença especificando os efeitos de sua execução em uma máquina hipotética
  - Transcreve uma sentença em um programa
    - A metalinguagem deve ser mais simples e intuitiva do que a linguagem cuja semântica queremos descrever
    - A linguagem da semântica operacional é mais próxima de um Assembly do que uma linguagem de alto nível

Semântica operacional: exemplos

- Sintaxe:

```
<for_stmt> ::= for ( <expr>[1] ; <expr>[2] ;
                                  <expr>[3] ) <stmt>
```

- Semântica:

```
      evaluate( <expr>[1] )
loop: evaluate( <expr>[2] )
      if <expr>[2] == 0 goto out
      parse( <stmt> )
      evaluate( <expr>[3] )
      goto loop
out:  ...
```

Semântica operacional: exemplos

- Sintaxe:

```
<if_stmt> ::= if <expr> then <stmt>
```

- Semântica:

```
        evaluate( <expr> )
        if <expr> == false goto out
        parse( <stmt> )
out:    ...
```

Semântica operacional: exemplos

- Sintaxe:

```
<if_stmt> ::= if <expr> then <stmt>[1]
              else <stmt>[2]
```

- Semântica:

```
      evaluate( <expr> )
      if <expr> == false goto else
      parse( <stmt>[1] )
      goto out
else: parse( <stmt>[2] )
out:  ...
```