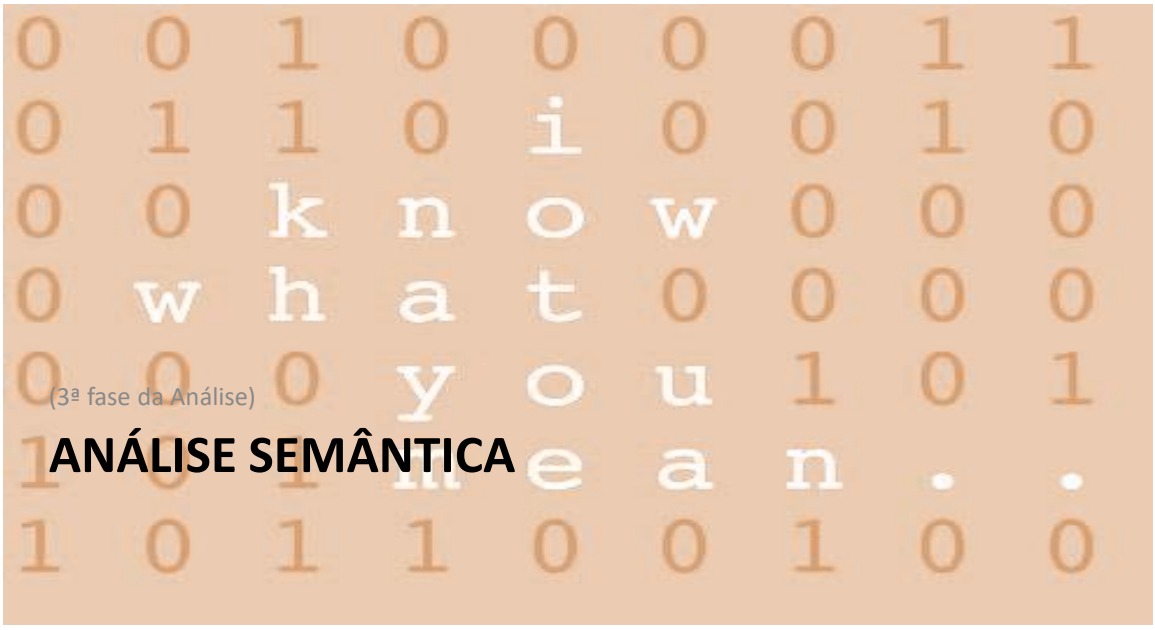
Compiladores e Computabilidade

Prof. Leandro C. Fernandes
UNIP – Universidade Paulista, 2018



(3ª fase da Análise)

ANÁLISE SEMÂNTICA

A análise semântica

Resultados da Análise Sintática ...

- As técnicas de análise sintática vistas descrevem basicamente reconhecedores, ou seja, máquinas que determinam se uma cadeia pertence ou não à linguagem reconhecida. Entretanto:
  - *No caso de aceitação*: a informação sobre a forma de derivação de *x* deve ser usada na geração do código para *x*; e
  - *No caso de não aceitação*: a informação disponível deve ser usada para tratamento de erros.
    - Sinalização, para que o programador possa corrigir o erro encontrado; e recuperação, para que o *parser* possa dar continuação à análise.

O componente semântico

- Durante a compilação o analisador deve verificar a utilização adequada dos identificadores
  - Análise Contextual: declarações prévias de variáveis, escopo de uso, etc.
  - Checagem de tipos e compatibilidade
- Note que estas tarefas estão além do domínio da sintaxe (Gram. Livres de Contexto - GLC)
  - Tais questões aumentam a GLC e completam a definição do que são programas válidos.

O componente semântico

A análise ocorre em dois momentos:

- Semântica Estática
  - Conjunto de restrições que determinam se os programas sintaticamente corretos são válidos.
- Semântica de Tempo de Execução
  - Usada para especificar o que o programa faz. Em outras palavras, aspectos ligados a execução.

Semântica estática

- Relaciona-se ao conjunto de restrições que vão além dos aspectos sintáticos, mas que determinam o que são programas válidos.
- As atividades compreendidas nesta etapa são:
  - Checagem de tipos,
  - Análise de escopo de declarações,
  - Verificação da quantidade e dos tipos dos parâmetros em chamadas à sub-rotinas.
- Pode ser especificada formalmente por uma *Gramática de Atributos*

Semântica de tempo de execução

- Aspectos relacionados ao comportamento do programa, isto é, a relação que há entre o programa-fonte e a sua execução dinâmica.

  Exemplos:

  `L: goto L;`
  - Implica em loop infinito, decorrente de um salto para a própria instrução. Alguns compiladores proíbem, outros não.

  `if (i<>0) && (K/I > 10) ...`
  - Haverá divisão por zero caso o compilador teste todas as cláusulas, entretanto não seria necessário proceder a segunda verificação caso a primeira já fosse

Semântica de tempo de execução (Cont)

- Esta etapa considera questões importantes para a fase de geração de código
  - Permite, por exemplo, uma organização diferenciada das instruções de máquina.
- Muitas vezes é o compilador quem serve de definição para a linguagem quando esta não está totalmente especificada.
- Geralmente esta componente é especificada de maneira informal, mas também pode ser feito através do uso de formalismos (tais com as Gramáticas de Atributos).

Gramática de Atributos

- É uma Gramática Livre de Contexto (GLC) estendida para fornecer sensitividade ao contexto através de conjunto de atributos e regras semânticas ligados aos terminais e não-terminais da gramática.
- Um atributo é qualquer propriedade de uma construção da linguagem de programação.
  - Tipo de dado de uma variável;
  - Valor de uma expressão;
  - Localização de uma variável na memória;
  - Endereço do início do código objeto de um procedimento, etc.

Gramática de Atributos (Cont)

- Aspectos importantes:
  - Nem todo símbolo gramatical tem atributos
  - Pode haver a manipulação de mais de um atributo em uma mesma regra e, inclusive, para um mesmo símbolo.
- Em geral, a gramática de atributos especifica:
  - O comportamento semântico das operações
  - A checagem de tipos
  - A manipulação de erros
  - A tradução do programa

Uma Gramática de Atributos

(1) $N \rightarrow I_1.I_2$
```
N.v := I1.v + I2.v
I1.p := 0
I2.p := -I2.l
```

(2) $N \rightarrow I$
```
N.v := I.v
I.p := 0
```

(3) $I_0 \rightarrow I_1B$
```
I0.v := I1.v + B.v
I0.l := I1.l + 1
I1.p := I0.p + 1
B.p := I0.p
```

(4) $I \rightarrow B$
```
I.v := B.v
I.l := 1
B.p := I.p
```

(5) $B \rightarrow 0$
```
B.v := 0
```

(6) $B \rightarrow 1$
```
B.v := 2^B.p
```

- A gramática ao lado apresenta três atributos:
  - v (valor),
  - p (posição) e
  - l (comprimento)
- Neste caso, em particular, as regras de cálculo foram especificadas de forma a permitir a conversão de um número binário em seu equivalente em decimal.

Como calcular os atributos?

- Com base na árvore sintática explícita
  - Grafos de dependência
  - Utilizada em compiladores de mais de um passo
- *Ad hoc*
  - Análise semântica "*comandada*" pela análise sintática (*Parser Driven*)
  - Utilizada em compilador de um único passo

Gramáticas *S-atribuídas*

- Na tradução dirigida pela sintaxe assume-se que os terminais tenham somente atributos sintetizados na medida em que as definições não providenciem quaisquer regras semânticas.
  - Os valores para os atributos dos terminais costumam ser fornecidos pelo léxico:
    (n) `F → dígito`
    `F.val := dígito.lexval`
- Atributos sintetizados são bastante usados na prática. Uma definição dirigida pela sintaxe somente com atributos sintetizados é chamada de definição *S-atribuída*.

- Exemplo:

(1) $E_0 \rightarrow E_1 + T$
$E_0.val := E_1.val + T.val$

(2) $E \rightarrow T$
$E.val := T.val$

(3) $T_0 \rightarrow T_1 * F$
$T_0.val := T_1.val * F.val$

(4) $T \rightarrow F$
$T.val := F.val$

(5) $F \rightarrow (E)$
$F.val := E.val$

(6) $F \rightarrow id$
$F.val := id.lexval$

Gramáticas *L-atribuídas*

- Atributos herdados são convenientes para expressar construções de Linguagem de Programação em relação ao contexto em que aparecem.
  - Bastante útil na verificação de tipos; ou
  - Controlar se um identificador aparece do lado esquerdo (endereço) ou direito (valor) de uma atribuição

- Exemplo:

(1) $D \rightarrow T\ L$

L.*in*:= T.*tipo*

(2) $T \rightarrow$ *int*

T.*tipo*:= `inteiro`

(3) $T \rightarrow$ *float*

T.*tipo*:= `real`

(4) $L_0 \rightarrow L_1$,*id*

L_1.*in*:= L_0.*in*

incluir_tipo (
    id.*token*, L_0.*in*
)

(5) $L \rightarrow$ *id*

incluir_tipo (
    id.*token*, L.*in*
)

Armazenando informações sobre os identificadores ...

TABELA DE SÍMBOLOS

Tabela de Símbolos

- Armazena as informações sobre todos os identificadores do código fonte
  - Captura a sensitividade ao contexto e as ações executadas no decorrer do programa
- Está atrelada a todas as etapas da compilação, sendo a estrutura principal do processo.
- E por sua vez, é fundamental para:
  - Realizar a análise semântica
  - A geração de código

Operações com a Tabela de Símbolos

Podem ser implementadas como:

- Chamadas na **Gramática de Atributos**

$L_0 \rightarrow$ **id,** L_1

```
if (buscaTS(id) == false)
  incluirTS(id, L0.tipo)
else
  ERRO("Identificador já declarado")
```

- Diretamente na **Análise Sintática**
  - Inserção: quando analisa declarações de variáveis, sub-rotinas, parâmetros ...
  - Busca: em atribuições, expressões, chamadas de sub-rotinas ou qualquer outro uso de um identificador em um bloco de comandos.

Tarefas da Análise Semântica

Responsável fundamentalmente por três tarefas:

1. Construir a descrição interna dos tipos e estruturas de dados definidos no programa do usuário;
2. Armazenar na **Tabela Símbolos** as informações sobre os identificadores (de constante, tipos, variáveis, procedimentos, parâmetros e funções) que são usados no programa;
3. Verificar o programa quanto a erros semânticos (erros dependentes de contexto) e checagens de tipos com base nas informações contidas na Tabela de Símbolos.

1. Representação de tipos e estruturas de dados definidos pelo programador

- As linguagens modernas oferecem um grande repertório de tipos e também permitem que o programador especifique seus próprios tipos de dados.
- Ao compilador cabe:
  - representar as especificações dos tipos
  - e usar tais informações para a previsão do uso de memória em tempo de execução, pelos objetos que forem declarados como sendo de um tipo especificado.

2. Armazenar informações sobre os identificadores na Tabela de Símbolos

- Uma Tabela de Símbolos reflete a estrutura do programa, pois guarda informações sobre o escopo de identificadores.
  - À medida que o compilador processa o programa fonte encontra os identificadores definidos e também os utilizados pelo programa.
    - Por exemplo, em situações como declarações de variáveis ou de procedimentos com seus parâmetros.
  - Para cada referência, o compilador terá necessidade de conhecer os atributos correspondentes (que sejam de interesse para a geração de código e verificação de erros semânticos).
    - Por exemplo, no caso de variáveis, poderiam ser a categoria, tipo e endereço na área de dados.

3. Verificar quanto a erros semânticos dependentes de contexto e de tipos

- Identificador já declarado no escopo (nível) atual
- Tipo não definido
- Limite inferior > limite superior na declaração de vetores/matrizes
- Função não declarada
- Função, variável, parâmetro, ou constante não definidas (checagem no lado direito de atribuições)
- Incompatibilidade no número de parâmetros
- Procedimento não declarado
- Função, variável, procedimento ou parâmetros não definidos (lado esquerdo de atribuições/lado esquerdo)
- Identificador de tipo esperado