INSTITUTO MAUÁ DE TECNOLOGIA
MAUÁ

Curso de Engenharia de Computação
ECM253 – Linguagens Formais, Autômatos e Compiladores

Conceitos de Análise Semântica – I

Slides da disciplina ECM253 – Linguagens Formais, Autômatos e Compiladores
Curso de Engenharia de Computação
Instituto Mauá de Tecnologia – Escola de Engenharia Mauá
Prof. Marco Antonio Furlan de Souza
<marco.furlan@maua.br>

Análise semântica

- **Conceitos**
  - A análise semântica tem por objetivo **obter** o **significado** da **estrutura sintática reconhecida** (ou em reconhecimento) pelo analisador sintático;
  - A análise semântica realizada **antes da execução do código** é conhecida por **análise semântica estática** e basicamente cuida da **verificação** de **tipos**, criação da **tabela de símbolos** e **geração de código intermediário**;
  - A análise semântica realizada **durante a execução do código** é conhecida por **análise semântica dinâmica** e cuida de aspectos que não podem ser previstos durante a análise estática, tais como **divisão por zero** e **verificação dos limites de um vetor**;
  - **Diferente dos métodos de análise sintática**, **não há** um **padrão de algoritmo** e **nem geradores de análise semântica** – depende fortemente da linguagem em questão – em **linguagens dinâmicas** como **LISP** praticamente **não há análise semântica** enquanto que em **linguagens** fortemente **baseadas em tipos** como **C** e **Pascal** ela é muito **forte**.

Análise semântica

- **Atributos e gramáticas de atributos**
  - **Atributos** ou **propriedades** de **entidades** da linguagem são computados e então são definidas **regras semânticas** (ou **equações de atributos**) que indicam como a **computação** desses **atributos** estão **relacionados** às **regras gramaticais da linguagem**;
  - São úteis em linguagens que obedecem ao princípio da **semântica direcionada por sintaxe** – o conteúdo semântico do programa é muito próximo de sua sintaxe;
  - Uma ferramenta que simplifica o entendimento das computações semânticas são as **árvores de sintaxe abstrata**;
  - Infelizmente estes processos são **manuais** ou *ad-hoc*;
  - Normalmente as **computações semânticas** são **executadas** durante a **análise sintática** – consegue-se executá-la em um **único passo**;
  - Em linguagens onde isso não é possível realiza-se a **análise semântica atrasada**, que implica normalmente em mais de um passo na compilação.

Atributos

- **Conceito de atributo**
  - É qualquer **propriedade** de uma construção de uma linguagem de programação;
  - **Exemplos típicos**:
    - **Tipo de dados de uma variável**: é importante para linguagens estaticamente “tipadas” como C e Pascal. Um **verificador de tipos** é um analisador semântico que calcula o atributo de tipo de dados de todos os elementos da linguagem;
    - **Valor de uma expressão**: são dinâmicos e o compilador irá gerar código para eles. Algumas expressões são constantes e o analisador semântico pode avaliá-los;
    - **Localização de variáveis na memória**: pode ser estática ou dinâmica dependendo da linguagem (FORTRAN – estático; LISP – dinâmico; C e Pascal – ambos);
    - **Código objeto de um procedimento**: é um atributo estático;
    - **Número de dígitos significativos de um número**: não é tratado durante a compilação, mas o gerador de varredura pode limitá-lo.

Gramáticas de atributos

- **Atributos** são **associados** diretamente aos **símbolos gramaticais** (terminais e não terminais) em **semântica dirigida por sintaxe**;
- A semântica dirigida por sintaxe implica que para cada regra gramatical $X_0 \rightarrow X_1 X_2 \ldots X_n$ (X_0 é um não terminal) e uma **coleção de atributos** $a_1 \ldots a_k$, os valores de atributos $X_i.a_j$ de cada símbolo gramatical X_i são relacionados aos valores de atributos de outros símbolos da regra;
- Cada relacionamento é descrito por uma **equação de atributo** ou **regra semântica** e possui a forma:

$$X_i.a_j = f_{ij}(X_0.a_1, \ldots, X_0.a_k, X_1.a_1, \ldots, X_1.a_k \ldots X_n.a_1, \ldots, X_n.a_k,)$$

- Uma **gramática de atributos** para atributos $a_1 \ldots a_k$ é uma coleção dessas equações;
- Na prática essas equações são simples e não dependem de um grande número de outros atributos.

Gramáticas de atributos

- **Exemplo 1**
  - Seja a gramática:

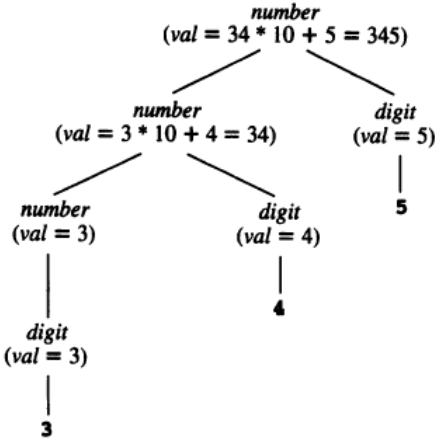
$$number \rightarrow number\ digit \mid digit$$
$$digit \rightarrow 0 \mid 1 \mid 2 \mid 3 \mid 4 \mid 5 \mid 6 \mid 7 \mid 8 \mid 9$$

  - Sua **gramática de atributos** poderia ser assim definida:

Grammar Rule	Semantic Rules
$number_1 \rightarrow number_2\ digit$	$number_1.val = number_2.val * 10 + digit.val$
$number \rightarrow digit$	$number.val = digit.val$
$digit \rightarrow \mathbf{0}$	$digit.val = 0$
$digit \rightarrow \mathbf{1}$	$digit.val = 1$
$digit \rightarrow \mathbf{2}$	$digit.val = 2$
$digit \rightarrow \mathbf{3}$	$digit.val = 3$
$digit \rightarrow \mathbf{4}$	$digit.val = 4$
$digit \rightarrow \mathbf{5}$	$digit.val = 5$
$digit \rightarrow \mathbf{6}$	$digit.val = 6$
$digit \rightarrow \mathbf{7}$	$digit.val = 7$
$digit \rightarrow \mathbf{8}$	$digit.val = 8$
$digit \rightarrow \mathbf{9}$	$digit.val = 9$

Gramáticas de atributos

- **Exemplo 1**
  - **Computação dos atributos na árvore de análise sintática**



Gramáticas de atributos

- **Exemplo 2**
  - Seja a gramática:

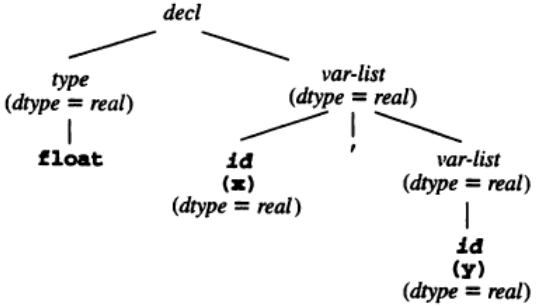
$$decl \rightarrow type\ var\text{-}list$$
$$type \rightarrow \texttt{int} \mid \texttt{float}$$
$$var\text{-}list \rightarrow \texttt{id}, var\text{-}list \mid \texttt{id}$$

  - Sua **gramática de atributos** poderia ser assim definida:

Grammar Rule	Semantic Rules
decl → *type var-list*	*var-list.dtype* = *type.dtype*
type → **int**	*type.dtype* = *integer*
type → **float**	*type.dtype* = *real*
$var\text{-}list_1$ → **id**, $var\text{-}list_2$	**id**.*dtype* = $var\text{-}list_1.dtype$
	$var\text{-}list_2.dtype = var\text{-}list_1.dtype$
var-list → **id**	**id**.*dtype* = *var-list.dtype*

Gramáticas de atributos

- **Exemplo 2**
  - **Computação dos atributos na árvore de análise sintática**



Algoritmos para a computação de atributos

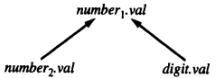
- **Grafos de dependência**
  - Dada uma gramática de atributo, cada escolha de regra gramatical possui um **grafo de dependência associado**;
  - Este grafo possui um **nó rotulado para cada atributo** $X_i.a_j$ de cada símbolo na regra da gramática e **para cada equação de atributo** na forma $X_i.a_j = f_{ij}(\ldots, X_m.a_k, \ldots$ associado **há uma aresta** de cada nó $X_m.a_k$ do lado direito **direcionada para** o nó $X_i.a_j$ ($X_i.a_j$ depende de $X_m.a_k$);
  - Para uma **cadeia de símbolos** legal de uma linguagem gerada por uma gramática livre de contexto o **grafo de dependência** desta cadeia é a **união** dos grafos de dependência das escolhas de regras gramaticais representando cada nó (não folha) da árvore de análise da cadeia.

Algoritmos para a computação de atributos

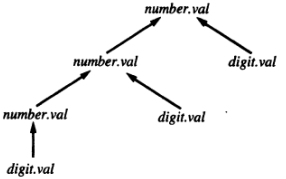
- **Grafos de dependência**
  - No Exemplo 1 do slide 6, a regra:

$$number_1.val = number_2.val * 10 + digit.val$$

Possui o seguinte grafo de dependência:

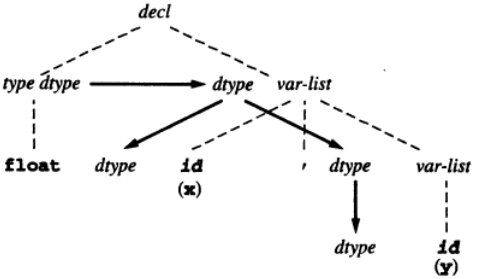


Omitindo os índices dos símbolos, já que as diversas ocorrências na árvore são distintas, tem-se o seguinte grafo de dependência para a cadeia 345:



Algoritmos para a computação de atributos

- **Grafos de dependência**
  - Em um grafo de dependência pode-se indicar também a árvore de análise, desenhada com tracejado para não ofuscar as dependências;
  - O grafo de dependência do Exemplo 2 do slide 8 para a cadeia `float x,y` é assim desenhado:



Algoritmos para a computação de atributos

- **Discussão sobre algoritmos**
  - Para grafos de dependência direcionados e acíclicos, o próprio **grafo de dependência indica as restrições** que o algoritmo deve seguir para computar o valor dos atributos;
  - Para isso, pode-se utilizar algoritmos de **ordenação topológica** que, partindo das folhas do grafo (cujos valores podem ter sido obtidos pelo analisador sintático ou léxico) seguem as arestas computando os atributos (podem existir diversas ordenações topológicas para o mesmo grafo);
  - Métodos que utilizam ordenação topológica sobre uma árvore criada durante a análise sintática são denominados de **métodos de árvore de análise**, mas possuem dois problemas principais:
    - Adicionam complexidade durante a compilação;
    - Atrasam a compilação para determinar se a gramática possui circularidade.
  - Uma alternativa adotada **na prática** é **fixar uma ordem** para avaliar os atributos de acordo com a gramática em questão. Embora ainda sejam baseados na árvore de análise, esses métodos são denominados de **métodos baseados em regras**.

Algoritmos para a computação de atributos

- **Atributos sintetizados e herdados**
  - Um atributo é **sintetizado** se todas suas dependências apontam de nós filho para um nó pai, isto é, dada uma regra $A \rightarrow X_1 X_2 \ldots X_n$ um atibuto a é sintetizado se sua equação de atributo é do tipo:

$$A.a = f(X_1.a_1, \ldots X_1.a_k, \ldots, X_n.a_1, \ldots X_n.a_k)$$

  - Essa gramática de atributos também é denominada de **gramática-S** de atributos;
  - A partir de uma árvore de análise ou sintática os valores de atributos de uma gramática-S podem se calculados a partir de um **percorrimento pós-ordem** da árvore:

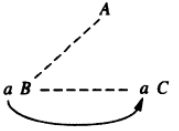
```
procedure PostEval ( T: treenode );
begin
   for each child C of T do
      PostEval ( C );
   compute all synthesized attributes of T;
end;
```

Algoritmos para a computação de atributos

- **Atributos sintetizados e herdados**
  - Um atributo que **não é sintetizado** é dito **herdado**.
  - **Tipos básicos** de atributos herdados:

(a) Inheritance from parent to siblings

(b) Inheritance from sibling to sibling



(c) Sibling inheritance via sibling pointers

Algoritmos para a computação de atributos

- **Atributos sintetizados e herdados**
  - Algoritmos para calcular atributos herdados podem utilizar **percorrimento** de árvore do tipo **pré-ordem** ou ainda **combinações** de **pré-ordem** e **em-ordem** da árvore de análise ou de análise sintática;
  - **Esquematicamente** são algoritmos da forma:

```
procedure PreEval ( T: treenode );
begin
   for each child C of T do
      compute all inherited attributes of C;
      PreEval ( C );
end;
```

Referências bibliográficas

AHO, A. V.; SETHI, R.; LAM, M. S. **Compiladores: princípios, técnicas e ferramentas**. 2. ed. [s.l.] Pearson, 2007.

COOPER, K.; TORCZON, L. **Construindo compiladores**. 2. ed. Rio de Janeiro: Elsevier, 2014.

LOUDEN, K. C. **Compiladores: princípios e práticas**. [s.l.] Pioneira Thomson Learning, 2004.