

Notas de aula – Curso de Compiladores – Prof. Isidro

Aulas: www.youtube.com/watch?v=9cOGaBPtfck&list=PLjcmNukBom6--0we1zrpoUE2GuRD-Me6W

Aula #1 - Introdução

- Compilador é uma das principais ferramentas de um desenvolvedor;
- Compiladores envolvem muitas áreas: é multidisciplinar;
- Definição: Lê uma linguagem A e traduz para uma linguagem B;
 - Pode ser para linguagem de máquina;
 - Pode verificar erros de programação (semântica e sintaxe);
- Etapas:
 - Análise: representação intermediária em memória (objeto);
 - Síntese: gera o programa alvo.
- Exigências:
 - Gerar códigos corretamente;
 - Tratar programas com qualquer tamanho;
 - Velocidade de compilação não é a característica principal;
 - Tamanho do compilador já não é mais um problema;
 - User-Friendliness --> qualidade das mensagens de erro;
 - Velocidade e tamanho do código gerado dependem do propósito do compilador (velocidade vem em primeiro lugar).
- Usaremos linguagens imperativas
 - Não vamos trabalhar com linguagens funcionais (como LISP) nem lógicas (como Prolog).
- Estrutura geral do compilador:



- Existem tabelas de palavras reservadas;
- Definição de escopo de variáveis;
- Programas relacionados a compiladores:
 - Interpretadores: Executam instrução por instrução do código-fonte;

- Montadores (assemblers): traduzem a linguagem de montagem para linguagem de máquina;
- Editores: Gerador de arquivos de texto ASCII;
- Depuradores: Determinam erros de execução em um programa compilado.

Aula #2 - Revisão de Gramáticas

- Autômatos são usados para reconhecimento de caracteres ou símbolos;
- Nem toda combinação de letras forma palavras válidas --> Temos um dicionário;
- Nem toda combinação de palavras forma uma sentença válida;
- Gramática é a especificação de uma linguagem --> Regras de formação de cadeias;
- Se usa uma linguagem livre de contexto ou regulares (não pode haver ambiguidades);
 - Livres de contexto: Análise sintática;
 - Regulares: Análise léxica.
- Ideias de uso das ideias de compiladores: arquivos de configuração em programas.

Aula #3 - Análise léxica

- Primeira etapa de um compilador;
- Nesta aula, faremos um analisador léxico;
- Funções:
 - Abre o arquivo;
 - Lê o programa;
 - Identifica os tokens correspondentes;
 - Relata erros.
- O que são tokens:
 - Identificadores;
 - Palavras reservadas e símbolos especiais;
 - Números.
- Exemplo:

• x:=y*2;

Cadeia	Token
x	id
:=	simb_atrib
y	id
*	simb_mult
2	num
;	simb_pv

• x:=y*2;

Token	Código
id	1
num	2
simb_mult	3
simb_atrib	4
simb_pv	5

Cadeia	Token
x	1
:=	4
y	1
*	3
2	2
;	5

- Se esses símbolos fazem parte de um comando, o analisador sintático é quem vai verificar;
- Temos que nos atentar:
 - O que delimita um token? Espaços? caracteres especiais?
 - Case sensitive?
 - Conjunto de palavras reservadas;
 - Regras de formação de identificadores;
 - Operadores aceitos;

- Delimitadores aceitos: (. ; :) [] { }
 - Regra para formação de números;
 - Regras para comentários.
- A análise léxica é baseada em rodar autômatos não determinísticos.

Aula #4 - Implementação do analisador léxico

- Aula prática (testes no Eclipse).

Aula #5 – First e Follow

- Início da análise sintática. Até então, não houve preocupação com o contexto, apenas com os símbolos e suas regras de formação;
- Gramática livre de contexto --> Um símbolo não terminal deriva qualquer sequência de símbolos (terminais ou não terminais);
- Queremos saber para um não terminal A:

- Cabeça
 - Primeiro símbolo que consegue identificar.
 - Pode ser símbolo terminal ou não terminal.
- Último
 - Último símbolo da regra.
 - Pode ser símbolo terminal ou não terminal.
- Se gera ou não cadeia vazia
 - Marcação com “sim” ou “não” (seguir algoritmo):

• Algoritmo: Identificação dos não terminais que derivam a cadeia vazia λ :

- (0) Inicialmente a lista L contém todas as regras de G, **exceto** aquelas que contém um símbolo terminal;
 - (1) Se existe um não terminal A sem regras marque A com **não**;
 - (2) Se um não terminal A tem uma regra $A \rightarrow \epsilon$, retire de L todas as regras com A do lado esquerdo, e retire todas as ocorrências de A do lado direito das regras de L. Marque A com **sim**;
 - (3) Se uma regra tem do lado direito um não terminal marcado **não**, retire a regra;
 - (4) Repita os passos (1), (2), (3) até que nenhuma nova informação seja adicionada.
- Iniciadores (First): Quais símbolos terminais iniciadores das cadeias geradas.
 - Todos os conjuntos first devem ser disjuntos.
 - Contém somente terminais.

- Seguir algoritmo:
 - O algoritmo se baseia nos seguintes pontos:
 - Regra $A \rightarrow a\alpha$, então $a \in \text{First}(A)$:
 - Derivação correspondente é $A \Rightarrow a\alpha$
 - Regra $A \rightarrow B_1 \dots B_m a\alpha$, e para todo $i = 1, \dots, m$, $B_i \Rightarrow^* \varepsilon$, então também temos $a \in \text{First}(A)$:
 - Não é o primeiro símbolo;
 - Passa a ser quando todos os B_i forem substituídos por ε ;
 - Derivação correspondente é $A \Rightarrow B_1 \dots B_m a\alpha \Rightarrow^* a\alpha$.
 - Regra $A \rightarrow B\alpha$, e se $a \in \text{First}(B)$, temos também $a \in \text{First}(A)$:
 - Se $a \in \text{First}(B)$, temos $B \Rightarrow^* a\beta$
 - Derivação correspondente é $A \Rightarrow B\alpha \Rightarrow^* a\beta\alpha$
 - Regra $A \rightarrow B_1 \dots B_m B\alpha$, e para todo $i = 1, \dots, m$, $B_i \Rightarrow^* \varepsilon$, então se $a \in \text{First}(B)$, temos também $a \in \text{First}(A)$, de forma semelhante, já que os B_i "desaparecem":
 - Se tivermos $B \Rightarrow^* a\beta$ a derivação correspondente é $A \Rightarrow B_1 \dots B_m B\alpha \Rightarrow B\alpha \Rightarrow^* a\beta\alpha$
 - **Algoritmo:** Cálculo de $\text{First}(A)$, para todos os **não terminais** A de uma gramática G :
 - (0) Inicialmente, para todos os não terminais A da gramática G , todos os conjuntos $\text{First}(A)$ estão vazios;
 - (1) Para cada regra $A \rightarrow B_1 \dots B_m a\alpha$, tal que para todo $i = 1, \dots, m$, $B_i \Rightarrow^* \varepsilon$, acrescente **a** a $\text{First}(A)$;
 - (2) Para cada regra $A \rightarrow B_1 \dots B_m B\alpha$, tal que, para todo $i = 1, \dots, m$, $B_i \Rightarrow^* \varepsilon$, acrescente $\text{First}(B)$ a $\text{First}(A)$;
 - (3) Repita o passo (2) enquanto houver alteração no valor de algum dos conjuntos First .
- Seguidores (Follow): Símbolos terminais seguidores de A .
 - Símbolo $\$$ indica EOF;

- Algoritmo:
 - O algoritmo se baseia nos seguintes pontos:
 - Como $S\$ \Rightarrow^* S\$$, $\$ \in \text{Follow}(S)$
 - Regra $A \rightarrow \alpha B \gamma \beta$, então $a \in \text{Follow}(B)$:
 - Derivação é: $S\$ \Rightarrow^* \delta A \varphi \Rightarrow^* \delta \alpha B \gamma \beta \varphi \Rightarrow^* \delta \alpha B a \beta \varphi$
 - Regra $A \rightarrow \alpha B \gamma C \beta$, então se $a \in \text{First}(C)$, temos também $a \in \text{Follow}(B)$:
 - Se $a \in \text{First}(C)$, temos $C \Rightarrow^* a \mu$.
 - Derivação é:
 - $S\$ \Rightarrow^* \delta A \varphi \Rightarrow^* \delta \alpha B \gamma C \beta \varphi \Rightarrow^* \delta \alpha B C \beta \varphi \Rightarrow^* \delta \alpha B a \mu \beta \varphi$
 - Regra $A \rightarrow \alpha B \gamma$, então se $a \in \text{Follow}(A)$, temos também $a \in \text{Follow}(B)$:
 - Se $a \in \text{Follow}(A)$, temos $S\$ \Rightarrow^* \delta A a \varphi$
 - Derivação é: $S\$ \Rightarrow^* \delta A a \varphi \Rightarrow^* \delta \alpha B \gamma a \varphi \Rightarrow^* \delta \alpha B a \varphi$
 - **Algoritmo: Cálculo de $\text{Follow}(A)$, para todos os não terminais A de uma gramática G :**
 - (0) Inicialmente, para todos os não terminais A da gramática G , todos os conjuntos $\text{Follow}(A)$ estão vazios, excetuando-se $\text{Follow}(S) = \{ \$ \}$
 - (1) Se há uma regra $A \rightarrow \alpha B \gamma \beta$, e $\gamma = B_1 \dots B_m \Rightarrow^* \epsilon$, então acrescente a a $\text{Follow}(B)$
 - (2) Se há uma regra $A \rightarrow \alpha B \gamma C \beta$, e $\gamma = B_1 \dots B_m \Rightarrow^* \epsilon$, então acrescente $\text{First}(C)$ a $\text{Follow}(B)$
 - (3) Se há uma regra $A \rightarrow \alpha B \gamma$, e $\gamma = B_1 \dots B_m \Rightarrow^* \epsilon$, então acrescente $\text{Follow}(A)$ a $\text{Follow}(B)$
 - (4) Repita o passo (3) enquanto houver modificação em algum dos conjuntos
- Estes conjuntos são fundamentais para a análise sintática descendente recursiva (próximas aulas).

Aula #6 - Análise sintática preditiva

- Podemos automatizar a análise sintática --> ANTLR;
- Podemos transformar a gramática em gramáticas equivalentes para que não sejam gerados loops infinitos ou outros tipos de inconsistências;
- Na análise sintática estamos preocupados com a sequência de símbolos;
- Estruturas recursivas e árvores;
- Noções de OOP serão fundamentais para o uso dos recursos do compilador;
- A Linguagem não pode ficar ambígua --> não podem existir duas ou mais árvores de derivação para uma mesma palavra;
- Implementação da análise sintática:
 - Sistema de varredura: Há apenas um método algorítmico (autômatos finitos);
 - Duas categorias:
 - Ascendente: Bottom-Up --> das folhas para a raiz;

- Descendente: Top-Down --> da raiz para as folhas (provavelmente usaremos esta!!).
- ASD --> Análise Sintática Descendente
 - Parte do símbolo inicial e tenta chegar às folhas;
 - Tipos:
 - ASD com retrocesso
 - Tentativa e erro
 - Consome recursos excessivos
 - Fácil implementação
 -
 - ASD preditiva (recursiva ou não)

Aula #7 – Implementando Análise Sintática

- OK

Aula #8 – Implementando com ANTLR

- Para gerar os arquivos do antlr (Parser, lexer, etc):

```
java -cp antlr-4.7.1-complete.jar org.antlr.v4.Tool -package
br.com.professorisidro.isilanguage.parser -o
src\br\com\professorisidro\isilanguage\parser IsiLang.g4
```

- Atenção, pois no Windows algumas coisas são diferentes! O comando acima funcionou corretamente! 😊

Aula #9 - Análise Semântica

- Última etapa da análise;
- OK

Aula #10 - Implementação da análise semântica

- Links importantes:
 - <https://github.com/professorisidro/AnalizadorLexico>
 - <https://github.com/coquizin/compiladores>
 - <https://tomassetti.me/antlr-mega-tutorial/>
 - <https://github.com/professorisidro/IsiLanguageEmbriao.git>