

– INF01147 –
Compiladores

Análise Sintática
Gramáticas Livres de Contexto

Prof. Lucas M. Schnorr
– Universidade Federal do Rio Grande do Sul –



Revisão

- ▶ O que é análise léxica? Para que serve?
 - ▶ Quais os passos para se obter um analisador léxico?
-
- ▶ Quais são as seções de um código Flex?
 - ▶ O que faz o caractere . (ponto final)?
 - ▶ Qual a maneira de escrever um literal?
 - ▶ Como se representa a palavra vazia?

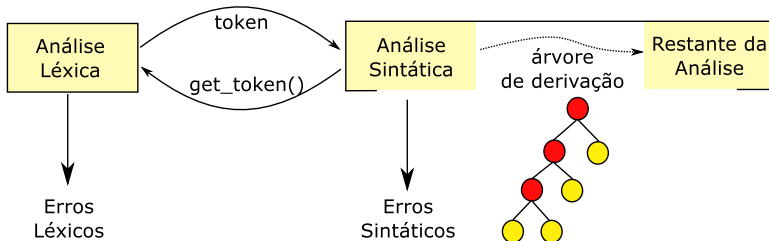
Plano da Aula de Hoje

- ▶ Introdução à Análise Sintática
- ▶ Noções sobre Gramáticas Livres de Contexto
 - ▶ Definições
 - ▶ Propriedades
 - ▶ Derivações (Árvore de Derivação)
 - ▶ Transformações de Gramáticas Livres de Contexto
 - ▶ Eliminação de produções vazias
 - ▶ Eliminação da recursividade à esquerda
 - ▶ Fatoração
- ▶ Lançamento da Etapa 2
- ▶ Yacc/Bison

Análise Sintática – Introdução

► Funções da Análise Sintática

- **Verificar a ordem de tokens** pertence à linguagem
- Emitir mensagens para **erros sintáticos**
 - Maior parte dos erros de análise são sintáticos
- Construir a **árvore de derivação** para a entrada
 - Na prática → Ações de síntese são feitas imediatamente



Análise Sintática – Tratamento de Erros

- ▶ Objetivos da recuperação de erros
 - ▶ Informar o erro de forma clara e precisa
 - ▶ Recuperar-se rapidamente (para detectar mais erros)
 - ▶ Custo baixo quando a entrada é correta
- ▶ Várias estratégias
 - ▶ **Modo Pânico**
 - Descarta tokens até um token de sincronismo
 - ▶ **Recuperação em nível de frase**
 - Realizar a correção local
 - ▶ Substituição de tokens
 - ▶ Exclusão
 - ▶ Inserção
 - ▶ **Produções de erro**
 - Prever erros com produções gramaticais
 - ▶ **Correção global**
 - Sequência mínima de mudanças
 - Não utilizado na prática (a semântica pode mudar)

Hierarquia de Gramáticas (Chomsky)

- ▶ Regular (Tipo 3)

$A \rightarrow a \mid aB \mid \epsilon$

- ▶ Livre de Contexto (Tipo 2)

$A \rightarrow v$ (um símbolo a esquerda)

- ▶ Sensível ao Contexto (Tipo 3)

$wAz \rightarrow wvz$ (onde w e z são o contexto)

- ▶ Recursivamente Enumeráveis (Tipo 4)

$u \rightarrow v$ (desde que u não seja vazio)

- ▶ Sintaxe de Linguagens de Programação

Gramáticas Livres de Contexto (GLC) são suficientes

→ Expressividade da recursão

GLC – Definição Formal

- ▶ **Terminais (T)**
são os símbolos básicos da linguagem
todos os tipos de tokens (palavras-chave, identificador, caracteres especiais, ...)
- ▶ **Não-terminais (NT)**
variáveis sintáticas, representam conjuntos de cadeiras
→ Impõem uma **estrutura hierárquica**
- ▶ **Símbolo inicial** (que é um não-terminal)
por convenção, listado primeiro
- ▶ **Produções**
 - ▶ Cabeça (lado esquerdo) – sempre é um NT
 - ▶ Um símbolo “→” (ou ::= utilizando a notação original BNF)
 - ▶ Corpo (lado direito) – sequência de T e NT

GLC – Exemplo para Expressões Aritméticas

- ▶ Símbolos terminais

id + - * / ()

- ▶ Não-terminais

S, expressão, termo, fator

- ▶ Regras de Produção (utilizando a notação BNF)

| | | |
|-----------|---|-------------------|
| S | → | expressão |
| expressão | → | expressão + termo |
| expressão | → | expressão - termo |
| expressão | → | termo |
| termo | → | termo * fator |
| termo | → | termo / fator |
| termo | → | fator |
| fator | → | (expressão) |
| fator | → | (id) |

GLC – Derivações

- ▶ Sequência de Derivações

- ▶ Começa no símbolo inicial
 - ▶ Passos

Substitui-se um NT pelo corpo de uma de suas produções

- ▶ Utiliza-se o símbolo “ \Rightarrow ” para representar um passo

- ▶ Exemplo: derivar **-(id*id)** com a gramática

$$E \rightarrow E + E \mid E * E \mid - E \mid (E) \mid id$$

- ▶ \Rightarrow^* deriva em zero ou mais passos

- ▶ \Rightarrow^+ deriva em um ou mais passos

- ▶ \Rightarrow^n deriva em n passos

- ▶ Forma sentencial *versus* sentença

- ▶ Definição de **Linguagem Gerada**

- ▶ Conjunto de todas as sentenças a partir do símbolo inicial

GLC – Derivações – Escolhas

- ▶ **Várias possibilidades** de derivação
- ▶ Exemplo: derivar **-(id*id)** com a gramática
$$E \rightarrow E + E \mid E * E \mid - E \mid (E) \mid id$$
- ▶ Cada aplicação de derivação exige **duas escolhas**
 - ▶ Qual não-terminal substituir
 - ▶ Qual produção do não-terminal escolher
- ▶ Qual não-terminal substituir?
 - ▶ Derivação **mais a esquerda**, com o símbolo \Rightarrow_{me}
 - ▶ Derivação **mais a direita**, com o símbolo \Rightarrow_{md}

GLC – Árvores de Derivação

- ▶ **Representação gráfica** do processo de derivação
 - Mostra a ordem na qual as produções são aplicadas
- ▶ Estrutura hierárquica que origina uma sentença
 - ▶ Raiz é o símbolo inicial da gramática
 - ▶ Vértices intermediários são não-terminais
 - ▶ Folhas são os terminais e palavras vazias (ϵ)
- ▶ Exemplo de **-(*id*+*id*)** com a gramática
$$E \rightarrow E + E \mid E * E \mid - E \mid (E) \mid id$$

GLC – Gramáticas Ambíguas

- ▶ Mais de uma árvore de derivação para a mesma sentença
→ Somente com um tipo de derivação
- ▶ Exemplo de **id + id * id** com a gramática
 $E \rightarrow E + E \mid E * E \mid - E \mid (E) \mid id$
- ▶ Regra geral: **ambiguidade é indesejada**
 - ▶ Não há um método estabelecido para removê-la
 - ▶ Algumas vezes a linguagem é inerentemente ambígua

GLC – Estratégias de Análise

- ▶ Existem três estratégias
 - ▶ Universal – pode analisar qualquer gramática
Algoritmos de Cocke-Younger-Kasami e Earley
 - ▶ **Descendente** *top-down*
 - ▶ **Ascendente** *bottom-up*
- ▶ Exemplo para a entrada **ccbca** e a gramática
$$S \rightarrow AB$$
$$A \rightarrow c|\epsilon$$
$$B \rightarrow cbB|ca$$
 - ▶ Descendente parte de S e tenta chegar a **ccbca**
 - ▶ Ascendente parte de **ccbca** e tenta chegar a S

GLC – Outras Definições

- ▶ Gramática **sem ciclos** – inexistência de produções tipo $A \Rightarrow^+ A$ para algum $A \in NT$
- ▶ Gramática **ϵ -Livre** – inexistência de produções tipo $A \rightarrow \epsilon$, salvo $S \rightarrow \epsilon$ onde S é o símbolo inicial
- ▶ Gramática **Fatorada à Esquerda** – sem produções tipo $A \rightarrow \alpha\beta_1 | \alpha\beta_2$, sendo α uma forma sentencial
- ▶ Gramática **Recursiva à Esquerda** – com a produção $A \Rightarrow^+ A\alpha$ para algum $A \in NT$
 - ▶ Recursão direta ou indireta
 - ▶ Impossibilita uma análise descendente (*top-down*)
 - ▶ Consumo do *token* é feito após a escolha da produção
 - ▶ Exemplo: $A \rightarrow Aa | b$ e a entrada **ba**

Transformações em GLC

- ▶ Eliminação de produções vazias
Transformando a gramática em ϵ -Livre
- ▶ Eliminação da recursividade à esquerda
Habilitando uma gramática para análise descendente
- ▶ Fatoração

Eliminação de produções vazias ($A \rightarrow \epsilon$)

► Seguindo a intuição

| | | |
|----------------------|---|--------------------------------|
| chamada_de_função | → | ident (argumentos_opcionais) |
| argumentos_opcionais | → | lista_argumentos ϵ |
| lista_argumentos | → | arg arg , lista_argumentos |

Eliminação de produções vazias ($A \rightarrow \epsilon$)

- ▶ Considerando a gramática $G = (N, T, P, S)$
 - ▶ Reunir em N_ϵ todos os não-terminais que geram ϵ
 $N_\epsilon = \{A \mid A \rightarrow \epsilon\}$
Repita
 $N_\epsilon = N_\epsilon \cup \{X \mid X \rightarrow X_1 \dots X_n \in P \text{ tal que } X_1 \dots X_n \in N_\epsilon\}$
até que o conjunto N_ϵ se estabilize (não aumente)
 - ▶ Construir um conjunto de produções sem produções vazias
Para obter $G_1 = (N, T, P_1, S)$, onde P_1 é:
 $P_1 = \{A \rightarrow \alpha \mid \alpha \neq \epsilon\}$
Repita
Para Para $A \rightarrow \alpha \in P_1$ e $X \in N_\epsilon$ tal que $\alpha = \alpha_1 X \alpha_2$ e $\alpha_1 \alpha_2 \neq \epsilon$
Faça $P_1 = P_1 \cup \{A \rightarrow \alpha_1 \alpha_2\}$ até que P_1 se estabilize
 - ▶ Incluir a produção vazia se necessário
 $G_2 = (N, T, P_2, S)$ onde $P_2 = P_1 \cup \{S \rightarrow \epsilon\}$

▶ Exemplo

$S \rightarrow B z B$

$B \rightarrow AAA$

$A \rightarrow a \mid \epsilon$

Eliminação da recursividade à esquerda

- ▶ Eliminando **recursão direta** (manualmente)

$$A \rightarrow Aa \mid b$$

- ▶ Gramáticas equivalentes

- ▶ Com a produção vazia (ϵ)

$$A \rightarrow bX$$

$$X \rightarrow aX \mid \epsilon$$

- ▶ Sem a produção vazia

$$A \rightarrow b \mid bX$$

$$X \rightarrow a \mid aX$$

- ▶ Pode ainda haver **recursão indireta**, descrita em
Dragão seção 4.3.3
Série Didática seção 3.1.2

Eliminação da recursividade à esquerda

Um segundo exemplo

- Considerando a gramática de operações aritméticas

$$E \rightarrow E + T \mid T$$

$$T \rightarrow T * F \mid F$$

$$F \rightarrow (E) \mid id$$

- A produção $E \rightarrow E + T \mid T$ se torna

$$E \rightarrow TE'$$

$$E' \rightarrow +TE' \mid \epsilon$$

- A produção $T \rightarrow T * F \mid F$ se torna

$$T \rightarrow FT'$$

$$T' \rightarrow *FT' \mid \epsilon$$

Fatoração gramatical

- Duas produções se iniciam com a mesma forma sentencial

$$A \rightarrow \alpha\beta_1 \mid \alpha\beta_2$$

- Fatorando

$$A \rightarrow \alpha X$$

$$X \rightarrow \beta_1 \mid \beta_2$$

- Exemplo: reconhecimento de condições aninhadas
comando \rightarrow **if** expressão **then** comando **else** comando
comando \rightarrow **if** expressão **then** comando

Especificação da Etapa 2

Yacc / Bison

Yacc – Introdução

- ▶ yacc -Yet Another Compiler Compiler
- ▶ Produz um analisador ascendente para uma gramática
- ▶ Usado para produzir compiladores para várias linguagens
 - ▶ Pascal, C, C++, ...
- ▶ Outros usos
 - ▶ bc (calculadora)
 - ▶ eqn & pic (formatadores para troff)
 - ▶ Verificar a sintaxe SQL
 - ▶ Lex
- ▶ **bison** – versão livre da GNU

Yacc – Especificação de Entrada

- ▶ Contém três seções
 - ▶ Definições (em C, incluído no início da saída)
 - ▶ Regras (Especificação da Gramática)
 - ▶ Código (em C, incluído no fim da saída)

- ▶ Sintaxe

Definições

%%

Regras

%%

Código Suplementar

Yacc – Seção de Regras (seção principal)

- ▶ Contém a gramática
- ▶ Exemplo

```
expr : expr '+' term | term;  
term : term '*' factor | factor;  
factor : '(' expr ')' | ID | NUM;
```

Yacc – Seção de Definições (seção auxiliar)

- ▶ Contém a definição de tokens, símbolo inicial
- ▶ Exemplo

```
%{  
    #include <stdio.h>  
    #include <stdlib.h>  
%}  
%token ID NUM /* notar a declaração dos tokens */  
%start expr
```

Yacc – Interação com o analisador léxico

- ▶ Flex produz uma função `yylex()`
- ▶ Bison produz uma função `yyparse()`

- ▶ Flex e Bison: concebidos para interagirem
 - ▶ `yyparse()` chama `yylex()` para obter um token

- ▶ Duas opções
 - ▶ Ou implementamos manualmente `yylex()`
 - ▶ Ou **utilizamos Flex diretamente**

Yacc – Sequência básica operacional

- ▶ Supondo os arquivos
 - ▶ **scanner.l** com as especificações de tokens em lex
 - ▶ **parser.y** com a gramática em yacc
- ▶ Ordem de passos possível para construir o analisador

```
bison -d parser.y
```

```
flex scanner.l
```

```
gcc -c lex.yy.c y.tab.c
```

```
gcc -o parser lex.yy.o y.tab.o -lfl
```

- ▶ **scanner.l** deve incluir na seção de definições

```
#include "y.tab.h"
```

Yacc – Miscelânea

- ▶ As regras da gramática
 - ▶ Podem ser recursivas tanto a esquerda quanto a direita
 - ▶ Não podem ser ambíguas
- ▶ Usa um parser ascendente LALR(1)
 - ▶ Solicita um token
 - ▶ Empilha
 - ▶ Redução?
 - ▶ Sim \leadsto reduz usando a regra correspondente
 - ▶ Não \leadsto lê outro token na entrada
- ▶ Não pode olhar mais que um token de *lookahead*
- ▶ `bison -v gramatica.y` gera a tabela de estados

Exemplo (Lex) – arquivo scanner.l

```
%{  
#include <stdio.h>  
#include "y.tab.h"  
%}  
id [_a-zA-Z][_a-zA-Z0-9]*  
wspc [ \t\n]+  
semi [;]  
comma [,]  
%%  
int { return INT; }  
char { return CHAR; }  
float { return FLOAT; }  
{comma} { return COMMA; }  
{semi} { return SEMI; }  
{id} { return ID; }  
{wspc} {;
```

Exemplo (Yacc) – arquivo parser.y

```
%{  
#include <stdio.h>  
#include <stdlib.h>  
%}  
%start line  
%token CHAR, COMMA, FLOAT, ID, INT, SEMI  
%%  
decl : type ID list { printf("Success!\n"); } ;  
list : COMMA ID list | SEMI;  
type : INT | CHAR | FLOAT;  
%%
```

- Notem uma ação para a regra decl

Yacc – Ações e Atributos

- ▶ Cada regra pode ter **ações** (semânticas)
- ▶ Exemplo

```
E: E '+' E      { $$ = $1 + $3; }  
  | INT_LIT      { $$ = INT_VAL; };
```

- ▶ \$n é o atributo do n-ésimo símbolo na regra
- ▶ O default é que os atributos sejam do tipo inteiro
- ▶ Pode-se mudar o tipo através da diretiva

```
%token<...> /* com o tipo do token */  
%type<...>  /* tipo do não-terminal, com %union */
```

Yacc – Ações e Atributos (Exemplo)

```
%union {  
    char* nome;  
    int inteiro;  
    node* no;  
}  
%token<nome> IDF /* IDF terá atributo de tipo char* */  
%type<no> E      /* E terá atributo de tipo node* */  
%%  
E: E + E { $$ = create_node($1, $3, "plus"); }  
  | IDF { $$ = create_leaf($1); };
```

Conclusão

- ▶ Leituras Recomendadas
 - ▶ Livro do Dragão
 - ▶ Seção 2.4.5
 - ▶ Seções do Capítulo 4 até 4.3.4 inclusive
 - ▶ Série Didática
 - ▶ Seções do Capítulo 3 até 3.1.2 inclusive
- ▶ Próxima Aula
 - ▶ Apresentação das Etapas 0 e 1
 - ▶ Sala 104 do Prédio 67 (Turma A)
 - ▶ Sala 103 do Prédio 67 (Turma B)