

Compiladores

Análise sintática (1)
Noções sobre
Gramáticas Livres de contexto

Compilação típica com FLEX

- Edição do texto de especificação
 - No arquivo 'minhas_regras.l'
 - 3 partes:
 - Declarações
 - Regras (ERs -> Ação)
 - Procedimentos anexos.
- **flex minha_regras. l**
 - O programa C resultado se encontra em 'lex.yy.c'
 - Contém a rotina 'yylex()' que efetua a análise lexical.
- **gcc -o analisador lex.yy.c -lfl**
 - Compila e amarra o analisador com a biblioteca libfl.a
 - O executável se encontra em 'analisador'

Especificação (F)lex

Definições

%%

Regras

%%

Subrotinas opcionais(código C)

- Lex define várias variáveis globais:
 - yytext e yyleng
 - Lexema corrente e seu tamanho
 - yyval
 - De tipo YYSTYPE, a partir de #defines
 - Depende do parser, você escolhe

A seção das regras

- Consiste de linhas do tipo:
ER { ação }
- A ER é especificada conforma a sintaxe de LEX, ou pode ser uma variável.
- '{ }' é a ação nula – não faz nada.
 - Retorna para o yylex() que analisará o próximo token!
 - Usado para detectar os "brancos".
- '{ printf("é isso aí\n");}' é a "ação" que consiste em imprimir uma mensagem na tela.

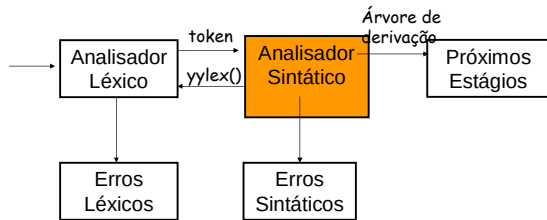
Escolha das regras

- Quando há ambigüidade entre regras, o LEX escolhe a que provê o lexema maior.
- A ação default é copiar para a saída o texto de entrada.
- A regra 'r1/r2' reconhece r1 somente se r2 aparece após (aceitação condicional).
 - Quando se encontra o lexema r1r2, o r2 é descartado e colocado de volta no buffer de entrada. r1 é reconhecido.

Plano da aula

- Introdução: porque a análise sintática?
- Noções sobre Gramáticas Livres de Contexto:
 - Definição
 - Propriedades
 - Derivações
 - Árvore de derivação
 - Transformações de GLC
 - Eliminação de ε
 - Eliminação da recursividade à esquerda
 - Fatoração

Análise Sintática



Os erros típicos são sintáticos

- Ripley e Druseikis [78]
 - Erros em uma amostra de programas Pascal de estudantes
 - 60% semântica e sintaticamente corretos
 - Erros:
 - 60% erros de pontuação (uso incorreto do ;)
 - 20% operadores
 - 15% palavras-chave
 - 5% outros
1. program prmax(input, output) ;
 2. var
 3. x, y : integer;
 4. function max (i: integer, j: integer) : integer
 5. { return maximum of integers i and j }
 6. begin
 7. if i > j then max = i;
 8. else max := j
 9. end;
 10. begin
 11. readln (x, y);
 12. writeln (max (x, y))
 13. end.

Usando Gramáticas

- Muitas construções de linguagens de programação são recursivas
- se S_1 e S_2 são enunciados e E é uma expressão, então:
 - “if E then S_1 else S_2 ” é um enunciado
- Exemplo de definição graças a gramática:


```

cmd → if expr then cmd else cmd
expr → ( bool ) | expr && expr | expr || expr
bool → true | false
      
```

Notação e Definições

- Gramática Formal
 - $G := \{S, P, N, T\}$
- S : Símbolo Inicial ($S \in N$)
- P : Conjunto de produções
- T : conjunto de terminais
 - Palavras ou tokens da linguagem
- N : conjunto de não terminais
 - Símbolos que podem ser substituídos
- Vocabulário:
 - V : alfabeto
 - $V = N \cup T$ ($N \cap T = \emptyset$)
 - $\underline{v}$: string pertencente a V
 - Símbolo Vazio ϵ
 - BNF: Backus-Naur form
 - $\langle u \rangle ::= \langle v \rangle | \langle w \rangle$
 - Igual a $\{ \underline{u} \rightarrow \underline{v}, \underline{u} \rightarrow \underline{w} \}$ em P

Derivações

- Seja $A \in N$, $p = \{ A \rightarrow \underline{v} \} \in P$, e $\underline{w}, \underline{z}$ quaisquer. Então
 - “derivação em um passo usando p ”
$$\underline{w}A\underline{z} \Rightarrow \underline{w}\underline{v}\underline{z}$$
- Se $\underline{w}_1 \Rightarrow \underline{w}_2 \Rightarrow \underline{w}_3 \dots \Rightarrow \underline{w}_n$ então podemos dizer
 - “derivação em múltiplos (0 ou mais) passos”; também $\Rightarrow^*$ (um ou mais)
$$\underline{w}_1 \Rightarrow^* \underline{w}_n$$

Definição da Linguagem Gerada

- A linguagem gerada G ($L(G)$):
 - Conjunto formado por todos os strings de símbolos terminais deriváveis a partir do símbolo inicial S
 - $L(G) = \{ s \mid s \text{ é um elemento de } T^* \text{ e } S \Rightarrow^+ s \}$
- Equivalência:
 - G_1 e G_2 são equivalentes se $L(G_1) = L(G_2)$
 - $\Rightarrow$ Todos strings gerados estão em L
 - $\Leftarrow$ Todos strings $\underline{w} \in L$ podem ser gerados
- Convenções
 - Símbolos que representam terminais em minúsculos:
 - $u, v, x, y, \dots$
 - Símbolos que representam não-terminais em maiúsculos:
 - $X, Y, \text{TERM}, S, \dots$
 - Símbolos que representam formas sentenciais (seqüências de terminais e não-terminais): letras gregas ou sublinhadas:
 - $\alpha, \beta, \omega, \underline{w}, \underline{z}$

Definição de Gramática Regular

- **Gramática regular:**

- produções exclusivamente da forma:
 - $A \rightarrow wB$
 - $A \rightarrow w,$
 - onde $w \in T^*$ e $A, B \in N$ (gramática linear à direita)
- A linguagem gerada por uma gramática regular é regular.

Exemplo $G = \{N, T, P, S\}$

$N = \{S, B, C, D\}$

$T = \{0, 1\}$

$S \rightarrow 0B \mid 1C$
 $B \rightarrow 1S$
 $C \rightarrow 0C \mid 0D \mid 1$
 $D \rightarrow 0$

S

Fornecer expressão regular para a gramática acima ?
 $(01)^*10^*(00 \mid 1)$

Hierarquia de Chomsky

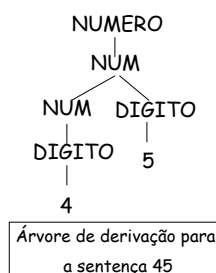
Tipo 0	Sem restrição recur. enum.	Qualquer $\underline{u} \rightarrow \underline{v}$ desde que $\underline{u}$ seja não vazio
Tipo 1	Sensível ao contexto	$\underline{w}A\underline{z} \rightarrow \underline{w}v\underline{z}$ ($\underline{w}, \underline{z}$ são o contexto)
Tipo 2	Livre do contexto	$A \rightarrow \underline{v}$ (1 símbolo a esquerda)
Tipo 3	regular	$A \rightarrow a \mid aB \mid \epsilon$ (derivações a direita)

Árvore de derivação (Parse Tree)

- **Árvore de derivação:**
 - representação gráfica de uma derivação de sentença
 - estrutura hierárquica que originou a sentença
 - A raiz da árvore representa o símbolo inicial
 - Os vértices interiores são não-terminais
 - Os símbolos terminais e a palavra vazia são folhas

Árvore de derivação – exemplo 1

- Gramática $G = (\{ \text{NUMERO, NUM, DIGITO} \}, \{0, 1, 2, \dots, 9\}, P, \text{NUMERO})$
- P (regras) =
 - $\text{NUMERO} \rightarrow \text{NUM}$
 - $\text{NUM} \rightarrow \text{NUM DIGITO} \mid \text{DIGITO}$
 - $\text{DIGITO} \rightarrow 0 \mid 1 \mid \dots \mid 9$



Derivação mais à esquerda e mais à direita

- **Derivação mais à esquerda de uma sentença:**
 - sequência de formas sentenciais que se obtém derivando sempre o símbolo não-terminal mais à esquerda.
- **Derivação mais à direita de uma sentença:**
 - sequência de formas sentenciais que se obtém derivando sempre o símbolo não-terminal mais à direita.

Árvore de derivação – exemplo 2

Gramática $G = (\{E\}, \{+, -, *, /, (,), x\}, P, E)$ sendo

$P = \{E \rightarrow E + E \mid E - E \mid E * E \mid E / E \mid (E) \mid x\}$

A mesma árvore pode representar mais de uma derivação para uma mesma sentença

Possíveis derivações para a árvore:

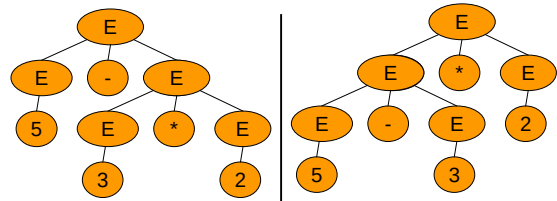
$E \Rightarrow E + E \Rightarrow x + E \Rightarrow x + E * E \Rightarrow x + x * E \Rightarrow x + x * x$
 $E \Rightarrow E + E \Rightarrow E + E * E \Rightarrow E + E * x \Rightarrow E + x * x \Rightarrow x + x * x$
 $E \Rightarrow E + E \Rightarrow E + E * E \Rightarrow x + E * E \Rightarrow x + x * E \Rightarrow x + x * x$

Exemplo

$E \rightarrow E \text{ OP } E \mid "(E)" \mid x$

$\text{OP} \rightarrow "*" \mid "/" \mid "+" \mid "-"$

- Considere $5 - 3 * 2$



Análise Top-Down vs. Bottom Up

Gramática: $S \rightarrow AB$ String: $ccbca$
 $A \rightarrow c \mid \epsilon$
 $B \rightarrow cbB \mid ca$

Top-Down		Bottom-Up	
$S \Rightarrow AB$	$S \rightarrow AB$	$ccbca \leftarrow Acbca$	$A \rightarrow c$
$\Rightarrow cB$	$A \rightarrow c$	$\leftarrow AcbB$	$B \rightarrow ca$
$\Rightarrow ccbB$	$B \rightarrow cbB$	$\leftarrow AB$	$B \rightarrow cbB$
$\Rightarrow ccbca$	$B \rightarrow ca$	$\leftarrow S$	$S \rightarrow AB$

Outras Classificações de Gramáticas

- Gramática sem ciclos:**
 - Gramática sem ciclos é uma GLC que não possui derivações da forma $A \Rightarrow^+ A$ para algum $A \in N$
- Gramática ϵ -livre :**
 - GLC que não possui produções do tipo $A \rightarrow \epsilon$ **Exceto** a produção $S \rightarrow \epsilon$ (S é o símbolo inicial).

Outras Classificações de Gramáticas

- Gramática fatorada à esquerda:**
 - GLC que **não** possui produções do tipo $A \rightarrow \alpha\beta_1 \alpha\beta_2$ para alguma forma sentencial α .
- Gramática recursiva à esquerda:**
 - GLC que permite a derivação

OBS: RECONHECEDOR TOP-DOWN não aceita gramáticas recursivas à esquerda indireta, como símbolo mais à esquerda de uma subpalavra gerada.

Transformações de GLCs

- Eliminação de produções vazias
- Eliminação de recursividade à esquerda:
 - recursão direta
 - recursão indireta
- Fatoração de uma gramática

Eliminação de produções vazias (1)

- **Objetivo:**
 - eliminar produções da forma $A \rightarrow \epsilon$.
- **Algoritmo:** seja $G = (N, T, P, S)$ uma GLC
 - **Etapa 1:**
 - construir N_ϵ , o conjunto de não-terminais que geram a palavra vazia:

$$N_\epsilon = \{A \mid A \rightarrow \epsilon\};$$

Repita

$$N_\epsilon = N_\epsilon \cup \{X \mid X \rightarrow X_1 \dots X_n \in P \text{ tq } X_1, \dots, X_n \in N_\epsilon\}$$

Até que o cardinal de N_ϵ não aumente.

Eliminação de produções vazias (2)

- **Etapa 2:**
 - construir o conjunto de produções sem produções vazias:
- gera $G_1 = (N, T, P_1, S)$, onde P_1 é construído como segue:

$$P_1 = \{A \rightarrow \alpha \mid \alpha \neq \epsilon\};$$

Repita

Para toda $A \rightarrow \alpha \in P_1$ e $X \in N_\epsilon$ tal que

$$\alpha = \alpha_1 X \alpha_2 \text{ e } \alpha_1 \alpha_2 \neq \epsilon$$

Faça $P_1 = P_1 \cup \{A \rightarrow \alpha_1 \alpha_2\}$

Até que o cardinal de P_1 não aumente

Eliminação de produções vazias (3)

- **Etapa 3:**
 - incluir geração da palavra vazia, se necessário.
- Se a palavra vazia pertence à linguagem, então a gramática resultante é
- $G_2 = (N, T, P_2, S)$, onde $P_2 = P_1 \cup \{S \rightarrow \epsilon\}$

Eliminação de recursividade à esquerda

- Exemplo

$$A \rightarrow Aa \mid b$$

Com a palavra vazia

$$A \rightarrow bX$$

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

Sem a palavra vazia

$$A \rightarrow b \mid bX$$

$$X \rightarrow a \mid aX$$

Obs: pode ainda haver recursão indireta!

Exemplo

- $E \rightarrow E+T \mid T$
- $T \rightarrow T^*F \mid F$
- $F \rightarrow (E) \mid \text{Id}$

Fatoração de uma gramática

- Elimina indecisão de qual produção aplicar quando duas ou mais produções iniciam com a mesma forma sentencial

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

Se torna:

$$A \rightarrow \alpha X$$

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

Exemplo de Fatoração a Esquerda

Cmd $\rightarrow$ if Expr then Cmd else Cmd

Cmd $\rightarrow$ if Expr then Cmd

Cmd $\rightarrow$ Outro

- Fatorando a esquerda:

Cmd $\rightarrow$ if Expr then Cmd ElseOpc

Cmd $\rightarrow$ Outro

ElseOpc $\rightarrow$ else Cmd | ϵ

Próxima aula

- Análise sintática...
 - Como reconhecer uma sentença de uma GLC?