# Compiler Design

Fatemeh Deldar

Isfahan University of Technology

1402-1403

# Verifying the Language Generated by a Grammar

- A proof that a grammar $G$ generates a language $L$ has two parts:
  - Show that every string generated by $G$ is in $L$
  - Show that every string in $L$ can indeed be generated by $G$
- **Example**
  - **The following grammar generates all strings of balanced parentheses**

$$S \rightarrow (\ S\ )\ S \ \mid\ \epsilon$$

# Context-Free Grammars Versus Regular Expressions

- **Every regular language is a context-free language, but not vice-versa**
- **Example**
  - Construct a context-free grammar from an NFA

$$(a|b)^*abb$$

$$\begin{array}{lcl} A_0 & \rightarrow & aA_0 \mid bA_0 \mid aA_1 \\ A_1 & \rightarrow & bA_2 \\ A_2 & \rightarrow & bA_3 \\ A_3 & \rightarrow & \epsilon \end{array}$$

- The language $L = \{a^n b^n \mid n \geq 1\}$ is context-free but not regular
- ***Finite automata cannot count***

# Context-Free Grammars

- **Example**

Consider the context-free grammar:

$$S \rightarrow S\,S\,+ \mid S\,S\,* \mid a$$

and the string $aa + a*$.

a) Give a leftmost derivation for the string.

b) Give a rightmost derivation for the string.

c) Give a parse tree for the string.

! d) Is the grammar ambiguous or unambiguous? Justify your answer.

! e) Describe the language generated by this grammar.

# Elimination of Left Recursion

- A grammar is left recursive if it has a nonterminal $A$ such that there is a derivation $A \overset{+}{\Rightarrow} A\alpha$ for some string $\alpha$
- Top-down parsing methods cannot handle left-recursive grammars, so a transformation is needed to eliminate left recursion
- **Example**

$$\begin{array}{lcl} E & \rightarrow & E + T \mid T \\ T & \rightarrow & T * F \mid F \\ F & \rightarrow & ( E ) \mid \textbf{id} \end{array} \quad \Longrightarrow \quad \begin{array}{l} E \rightarrow T\ E' \\ E' \rightarrow +\ T\ E' \mid \epsilon \\ T \rightarrow F\ T' \\ T' \rightarrow *\ F\ T' \mid \epsilon \\ F \rightarrow ( E ) \mid \textbf{id} \end{array}$$

- **Example**

$$A \rightarrow A\alpha_1 \mid A\alpha_2 \mid \cdots \mid A\alpha_m \mid \beta_1 \mid \beta_2 \mid \cdots \mid \beta_n$$

$$\Downarrow$$

$$\begin{array}{l} A \rightarrow \beta_1 A' \mid \beta_2 A' \mid \cdots \mid \beta_n A' \\ A' \rightarrow \alpha_1 A' \mid \alpha_2 A' \mid \cdots \mid \alpha_m A' \mid \epsilon \end{array}$$

# Elimination of Left Recursion

- **Algorithm to eliminate left recursion from a grammar**

1) arrange the nonterminals in some order $A_1, A_2, \ldots, A_n$.
2) **for** ( each $i$ from 1 to $n$ ) {
3)     **for** ( each $j$ from 1 to $i - 1$ ) {
4)         replace each production of the form $A_i \to A_j\gamma$ by the productions $A_i \to \delta_1\gamma \mid \delta_2\gamma \mid \cdots \mid \delta_k\gamma$, where $A_j \to \delta_1 \mid \delta_2 \mid \cdots \mid \delta_k$ are all current $A_j$-productions
5)     }
6)     eliminate the immediate left recursion among the $A_i$-productions
7) }

# Elimination of Left Recursion

- **Example**

$$S \to A\,a \mid b$$
$$A \to A\,c \mid S\,d \mid \epsilon$$

$$\Downarrow$$

$$A \to A\,c \mid A\,a\,d \mid b\,d \mid \epsilon$$

$$\Downarrow$$

$$S \to A\,a \mid b$$
$$A \to b\,d\,A' \mid A'$$
$$A' \to c\,A' \mid a\,d\,A' \mid \epsilon$$

# Left Factoring

- Left factoring is a grammar transformation that is useful for producing a grammar suitable for predictive, or top-down, parsing

$$
\begin{array}{lcl}
stmt & \rightarrow & \textbf{if } expr \textbf{ then } stmt \textbf{ else } stmt \\
 & | & \textbf{if } expr \textbf{ then } stmt
\end{array}
$$

- **Example**

$$A \rightarrow \alpha\beta_1 \mid \alpha\beta_2 \quad \Rightarrow \quad \begin{array}{l} A \rightarrow \alpha A' \\ A' \rightarrow \beta_1 \mid \beta_2 \end{array}$$