Compiler Design

Fatemeh Deldar

Isfahan University of Technology

1403-1404

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.

SS* => SS+S* => aS+S* => aa+S* => aa+a*

b) Give a rightmost derivation for the string.

SS* => Sa* => SS+a* => Sa+a* => aa+a*

c) Give a parse tree for the string.

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

Non-ambiguous

! e) Describe the language generated by this grammar.

Postfix expressions involving addition and multiplication

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 α
- Top-down parsing methods cannot handle left-recursive grammars, so a transformation is needed to eliminate left recursion
- **Example**

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

- **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**

$$
\begin{aligned}
S &\rightarrow A\ a \mid b \\
A &\rightarrow A\ c \mid S\ d \mid \epsilon
\end{aligned}
$$

$$\Downarrow$$

$$A \rightarrow A\ c \mid A\ a\ d \mid b\ d \mid \epsilon$$

$$\Downarrow$$

$$
\begin{aligned}
S &\rightarrow A\ a \mid b \\
A &\rightarrow b\ d\ A' \mid A' \\
A' &\rightarrow c\ A' \mid a\ d\ A' \mid \epsilon
\end{aligned}
$$

Left Factoring

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

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

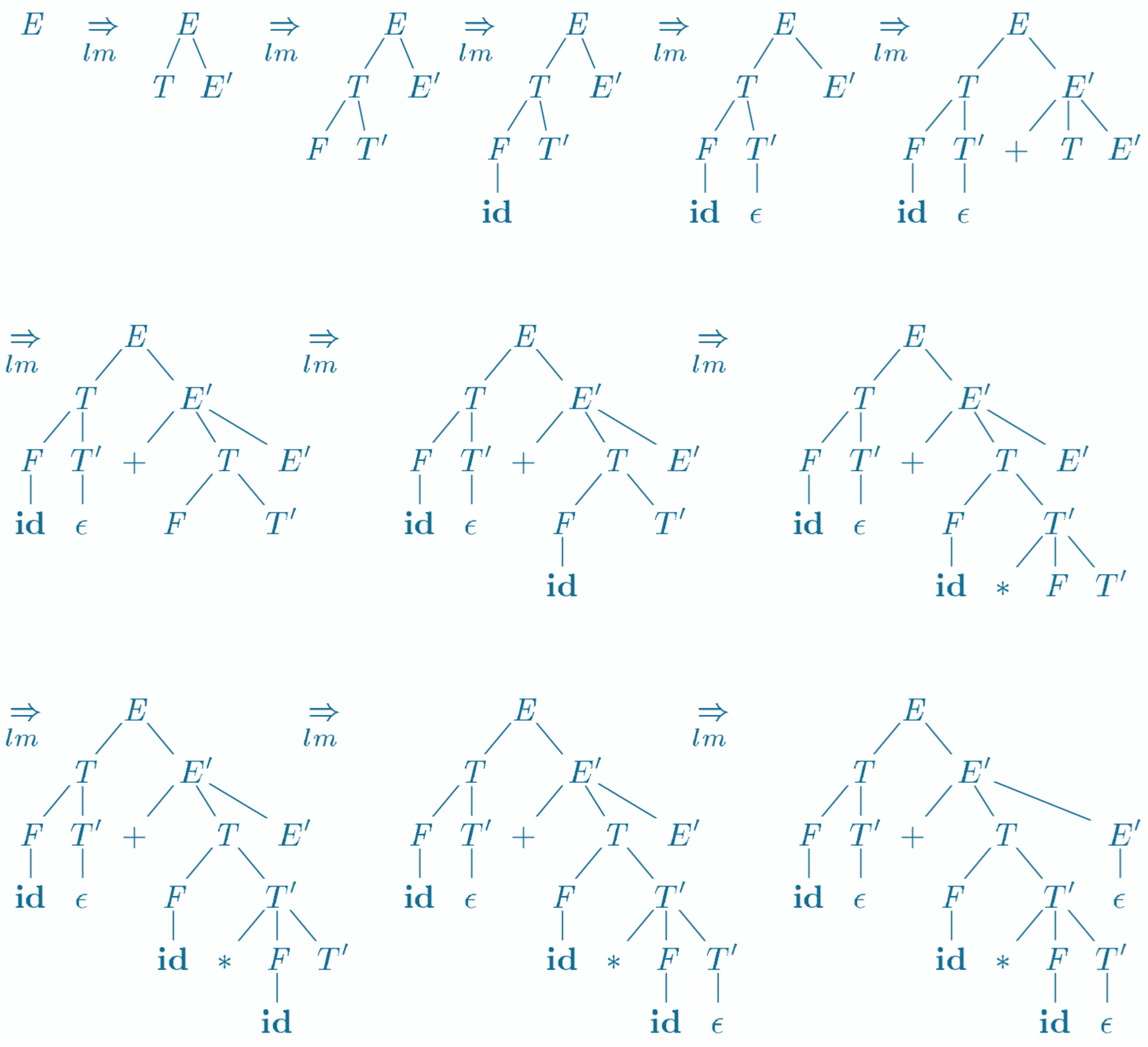
- **Example**

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

Top-Down Parsing

- Top-down parsing can be viewed as the problem of constructing a parse tree for the input string, starting from the root and creating the nodes of the parse tree in *preorder* (depth-first)
- Top-down parsing can be viewed as finding a *leftmost derivation* for an input string
- ***LL*(*k*) grammars**
  - The class of grammars for which we can construct predictive parsers looking k symbols ahead in the input

- **Example**



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

Recursive-Descent Parsing

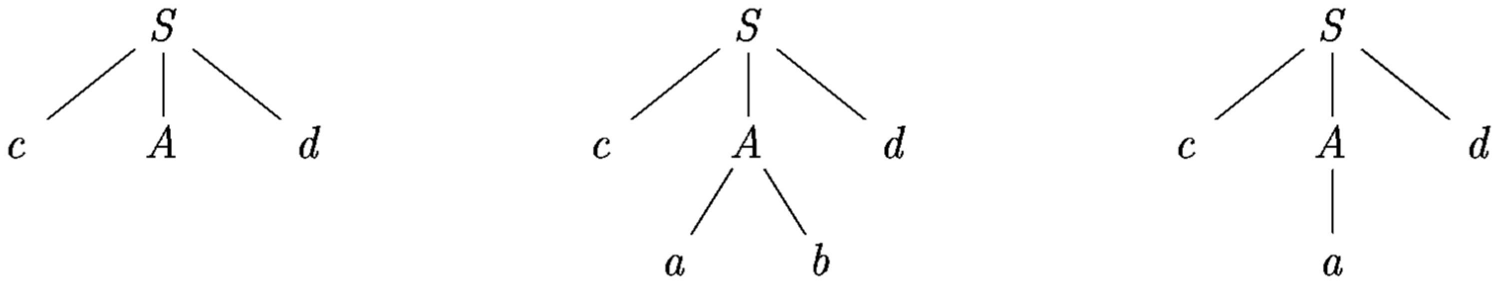
- A recursive-descent parsing program consists of a set of procedures, one for each nonterminal
- Execution begins with the procedure for the start symbol

```
void A() {
        Choose an A-production, A → X1 X2 ··· Xk;
        for ( i = 1 to k ) {
                if ( Xi is a nonterminal )
                        call procedure Xi();
                else if ( Xi equals the current input symbol a )
                        advance the input to the next symbol;
                else /* an error has occurred */;
        }
}
```

Recursive-Descent Parsing

- General recursive-descent may require backtracking; that is, it may require repeated scans over the input
- **Example**
  - Parse tree for $w = cad$

$$
\begin{array}{lcl}
S & \rightarrow & c\ A\ d \\
A & \rightarrow & a\ b \mid a
\end{array}
$$



(a) (b) (c)

  - In going back to A, we must reset the input pointer to position 2