Compiler Design

Fatemeh Deldar

Isfahan University of Technology

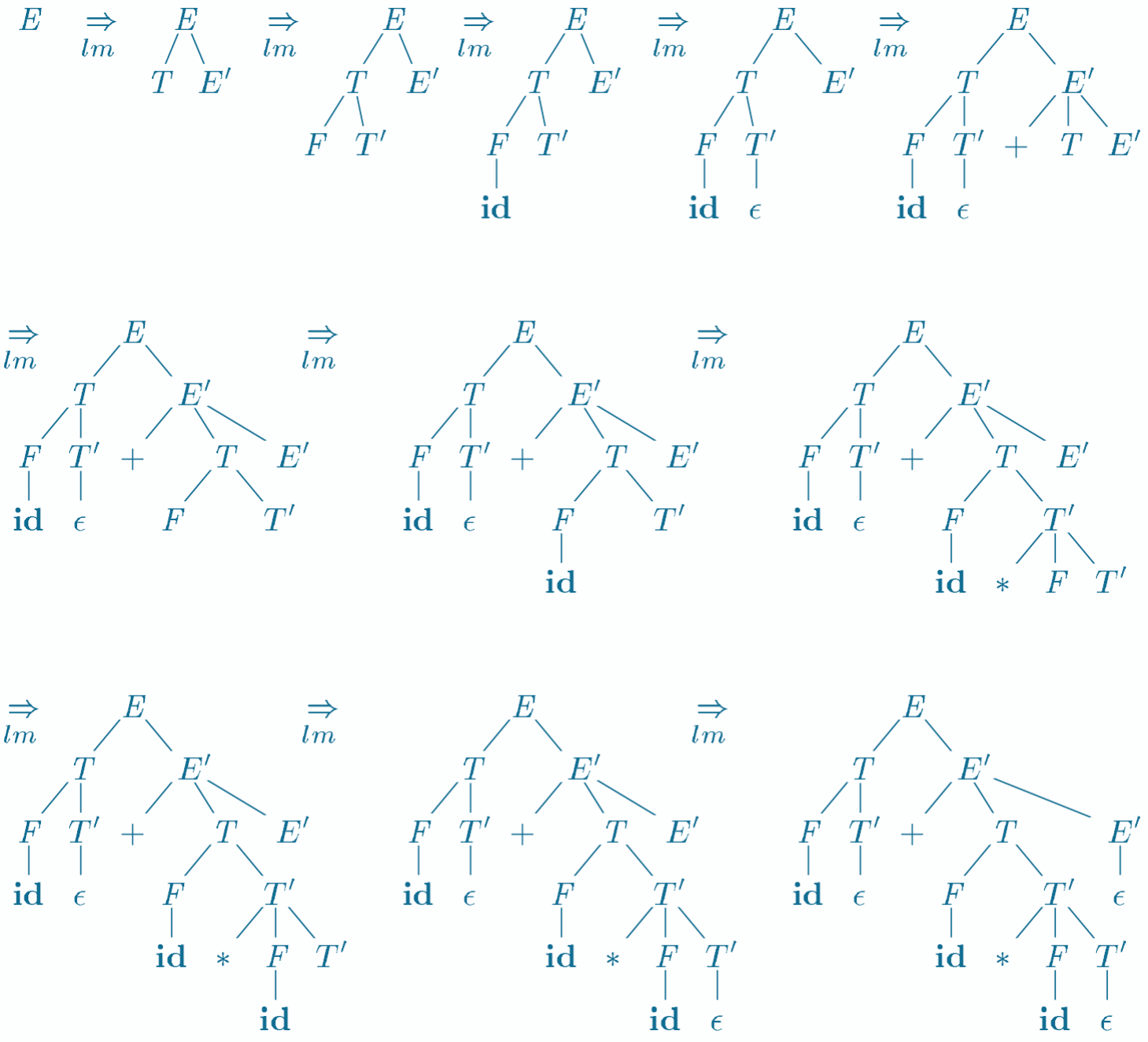
1402-1403

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**

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



Recursive-Descent Parsing

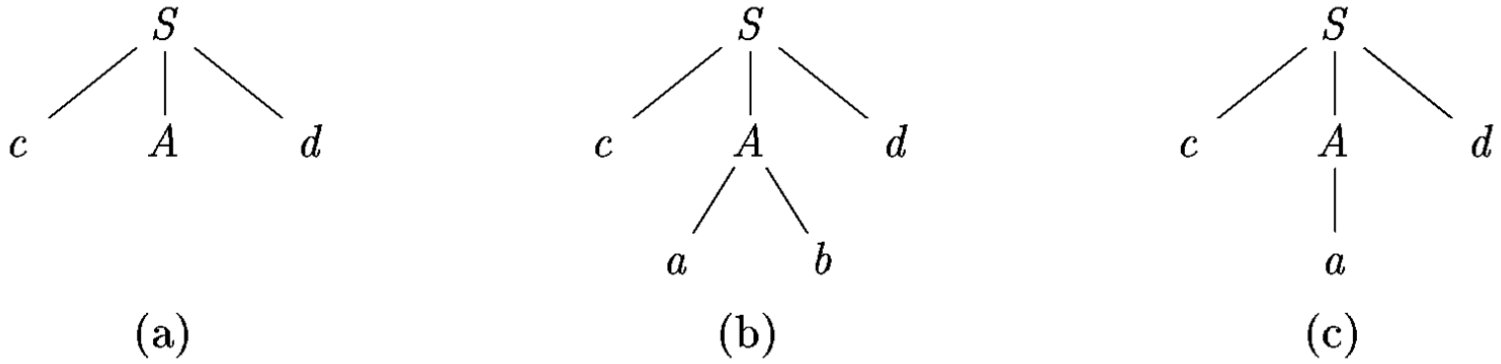
- A recursive-descent parsing program consists of a set of procedures, one for each nonterminal
- Execution begins with the procedure for the start symbol

```
     void A() {
1)       Choose an A-production, A → X1 X2 ··· Xk;
2)       for ( i = 1 to k ) {
3)           if ( Xi is a nonterminal )
4)               call procedure Xi();
5)           else if ( Xi equals the current input symbol a )
6)               advance the input to the next symbol;
7)           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 $w = cad$

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



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

FIRST and FOLLOW

- The construction of both top-down and bottom-up parsers is aided by two functions, **FIRST** and **FOLLOW**, associated with a grammar G
- ***FIRST*(*X*) for all grammar symbols *X***

1. If X is a terminal, then $\text{FIRST}(X) = \{X\}$.
2. If X is a nonterminal and $X \to Y_1 Y_2 \cdots Y_k$ is a production for some $k \geq 1$, then place a in $\text{FIRST}(X)$ if for some i, a is in $\text{FIRST}(Y_i)$, and ϵ is in all of $\text{FIRST}(Y_1), \ldots, \text{FIRST}(Y_{i-1})$; that is, $Y_1 \cdots Y_{i-1} \stackrel{*}{\Rightarrow} \epsilon$. If ϵ is in $\text{FIRST}(Y_j)$ for all $j = 1, 2, \ldots, k$, then add ϵ to $\text{FIRST}(X)$. For example, everything in $\text{FIRST}(Y_1)$ is surely in $\text{FIRST}(X)$. If Y_1 does not derive ϵ, then we add nothing more to $\text{FIRST}(X)$, but if $Y_1 \stackrel{*}{\Rightarrow} \epsilon$, then we add $\text{FIRST}(Y_2)$, and so on.
3. If $X \to \epsilon$ is a production, then add ϵ to $\text{FIRST}(X)$.

FIRST and FOLLOW

- ***FOLLOW*(*A*) for all nonterminals *A***

1. Place \$ in FOLLOW(S), where S is the start symbol, and \$ is the input right endmarker.
2. If there is a production $A \to \alpha B \beta$, then everything in FIRST(β) except ϵ is in FOLLOW(B).
3. If there is a production $A \to \alpha B$, or a production $A \to \alpha B \beta$, where FIRST(β) contains ϵ, then everything in FOLLOW(A) is in FOLLOW(B).

FIRST and FOLLOW

- **Example**

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

- $FIRST(F) = FIRST(T) = FIRST(E) = \{(, id\}$
- $FIRST(E') = \{+, \epsilon\}$
- $FIRST(T') = \{*, \epsilon\}$
- $FOLLOW(E) = FOLLOW(E') = \{), \$\}$
- $FOLLOW(T) = FOLLOW(T') = \{+,), \$\}$
- $FOLLOW(F) = \{+, *,), \$\}$