COSE312: Compilers

Lecture 4 — Syntax Analysis (2): Top-Down Parsing

Hakjoo Oh
2026 Spring

Expression Grammar

Expression grammar:

$$E \to E + E \mid E * E \mid (E) \mid \mathbf{id}$$

Unambiguous version:

$$\begin{array}{l} E \to E + T \mid T \\ T \to T * F \mid F \\ F \to \mathbf{id} \mid (E) \end{array}$$

Non-left-recursive version:

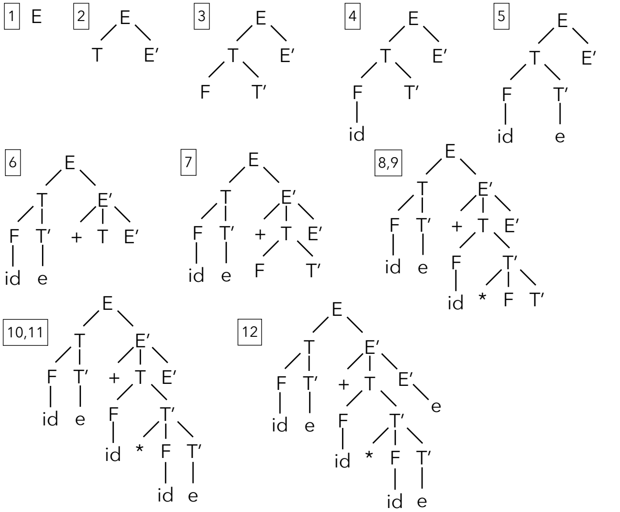
$$\begin{array}{lcl} E & \to & T\ E' \\ E' & \to & +\ T\ E' \mid \epsilon \\ T & \to & F\ T' \\ T' & \to & *\ F\ T' \mid \epsilon \\ F & \to & (E) \mid \mathbf{id} \end{array}$$

Top-Down Parsing

- Parsing is a process of constructing a parse tree of a given input string.
- Top-down parsing begins with the root of the parse tree and extends the tree downward until leaves match the input string.

Top-Down Parsing Example

Top-down parsing sequence for the input string $\mathbf{id} + \mathbf{id} * \mathbf{id}$:



The Key Problem in Top-Down Parsing

At each step of the derivation, top-down parsing replaces the leftmost derivation by the body of some production. How to determine which production to use?

- *Recursive-decent parsing* uses backtracking.
- *Predictive parsing* uses a parsing table without backtracking.

Parsing Table

The parsing table for the expression grammar:

	id	$+$	$*$	(	)	\$
E	$E \to T\ E'$			$E \to T\ E'$		
E'		$E' \to +\ T\ E'$			$E' \to \epsilon$	$E' \to \epsilon$
T	$T \to F\ T'$			$T \to F\ T'$		
T'		$T' \to \epsilon$	$T' \to *\ F\ T'$		$T' \to \epsilon$	$T' \to \epsilon$
F	$F \to \text{id}$			$F \to (E)$		

(**\$** is a special "endmarker" to indicate the end of file.)

Predictive Parsing

The sequence of predictive parsing for $\mathbf{id} + \mathbf{id} * \mathbf{id}$:

Stack	Input	Action
$E\$$	$\mathrm{id} + \mathrm{id} * \mathrm{id}\$$	
$TE'\$$	$\mathrm{id} + \mathrm{id} * \mathrm{id}\$$	
$FT'E'\$$	$\mathrm{id} + \mathrm{id} * \mathrm{id}\$$	
$\mathrm{id}T'E'\$$	$\mathrm{id} + \mathrm{id} * \mathrm{id}\$$	
$T'E'\$$	$+\mathrm{id} * \mathrm{id}\$$	match
$E'\$$	$+\mathrm{id} * \mathrm{id}\$$	
$+TE'\$$	$+\mathrm{id} * \mathrm{id}\$$	match
$TE'\$$	$\mathrm{id} * \mathrm{id}\$$	
$FT'E'\$$	$\mathrm{id} * \mathrm{id}\$$	
$\mathrm{id}T'E'\$$	$\mathrm{id} * \mathrm{id}\$$	match
$T'E'\$$	$*\mathrm{id}\$$	
$*FT'E'\$$	$*\mathrm{id}\$$	match
$FT'E'\$$	$\mathrm{id}\$$	
$\mathrm{id}T'E'\$$	$\mathrm{id}\$$	match
$T'E'\$$	$\$$	
$E'\$$	$\$$	
$\$$	$\$$	

Predictive Parsing Algorithm

Input: a string w and a parsing table M for grammar G
Output: a leftmost derivation of w or an error indication

```
let $a$ be the first symbol of $w$
let $X$ be the top stack symbol
while ($X \neq \$$) {
    if ($X = a$) pop the stack and let $a$ be the next symbol of $w$
    else if ($X$ is a terminal) error
    else if ($M[X, a]$ is empty) error
    else if ($M[X, a] = X \to Y_1 Y_2 \cdots Y_k$) {
        output the production $X \to Y_1 Y_2 \cdots Y_k$
        pop the stack
        push $Y_k, Y_{k-1}, \ldots, Y_1$ onto the stack, with $Y_1$ on top
    }
```

Constructing Parsing Table

1. Compute $\boldsymbol{FIRST}$ and $\boldsymbol{FOLLOW}$ sets of the grammar.
2. Construct the parsing table using these sets.

$FIRST$ and $FOLLOW$

Definition

Given a string α of terminal and non-terminal symbols, $FIRST(\alpha)$ is the set of all terminal symbols that can begin any string derived from α.

- If $\alpha \Rightarrow^* c\beta$, then $c \in FIRST(\alpha)$.
- If $\alpha \Rightarrow^* \epsilon$, $\epsilon \in FIRST(\alpha)$.

Definition

For a non-terminal X, $FOLLOW(X)$ is the set of terminals a that can appear immediately to the right of X in some sentential form.

- If $S \Rightarrow^* \alpha X a \beta$, then $a \in FOLLOW(X)$.
- If $S \Rightarrow^* \alpha X$, $\$ \in FOLLOW(X)$

Example

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

- $FIRST(F)$
- $FIRST(T)$
- $FIRST(E)$
- $FIRST(E')$
- $FIRST(T')$
- $FOLLOW(E)$
- $FOLLOW(E')$
- $FOLLOW(T)$
- $FOLLOW(T')$
- $FOLLOW(F)$

Algorithm for computing $FIRST$

To compute $FIRST(X)$ for all grammar symbol X, apply the following rules until no more terminals or ϵ can be added to any $FIRST$ set:

- If X is a terminal, then $FIRST(X) = \{X\}$.
- When X is a nonterminal and $X \rightarrow Y_1Y_2\cdots Y_k$ is a production for some $k \geq 1$,
  - place a in $FIRST(X)$ if for some i, a is in $FIRST(Y_i)$ and ϵ is in all of $FIRST(Y_1), \ldots, FIRST(Y_{i-1})$ (where a means a terminal symbol).
  - If ϵ is in $FIRST(Y_j)$ for all $j = 1, 2, \ldots, k$, then add ϵ to $FIRST(X)$.
- If $X \rightarrow \epsilon$ is a production, then add ϵ to $FIRST(X)$.

To compute $FIRST$ for any string $X_1X_2\cdots X_n$: Add to $FIRST(X_1X_2\cdots X_n)$

- all non-ϵ symbols of $FIRST(X_1)$
- all non-ϵ symbols of $FIRST(X_2)$, if $\epsilon \in FIRST(X_1)$
- all non-ϵ symbols of $FIRST(X_3)$, if $\epsilon \in FIRST(X_1)$ and $\epsilon \in FIRST(X_2)$
- $\cdots$
- ϵ if, for all i, $\epsilon \in FIRST(X_i)$

Algorithm for computing $FOLLOW$

To compute $FOLLOW(A)$ for all nonterminals A, apply the following rules until nothing can be added to any $FOLLOW$ set:

1. Place \$ in $FOLLOW(S)$, where S is the start symbol.
2. If there is a production $A \rightarrow \alpha B \beta$, then everything in $FIRST(\beta)$ except for ϵ is in $FOLLOW(B)$.
3. If there is a production $A \rightarrow \alpha B$, then everything in $FOLLOW(A)$ is in $FOLLOW(B)$.
4. If there is a production $A \rightarrow \alpha B \beta$, where $FIRST(\beta)$ contains ϵ, then everything in $FOLLOW(A)$ is in $FOLLOW(B)$.

Exercise

$$\begin{array}{rcl} X & \rightarrow & Y \mid a \\ Y & \rightarrow & c \mid \epsilon \\ Z & \rightarrow & d \mid X\ Y\ Z \end{array}$$

- $FIRST(X)$
- $FIRST(Y)$
- $FIRST(Z)$
- $FOLLOW(X)$
- $FOLLOW(Y)$
- $FOLLOW(Z)$

Construction of Parsing Table

Predictive parsing uses $FIRST$ to choose a production:

- For $A \to \alpha \mid \beta$, where $FIRST(\alpha) \cap FIRST(\beta) = \emptyset$, choose $A \to \alpha$ if the next symbol $a \in FIRST(\alpha)$.
- If $FIRST(\alpha) \cap FIRST(\beta) \neq \emptyset$, the grammar cannot be parsed using predictive parsing.

$LL(1)$: Grammars that can be parsed by predictive parsing (Left-to-right parse, Leftmost derivation, 1-symbol lookahead).

Construction of Parsing Table

- Goal: Collect the information from $FIRST$ and $FOLLOW$ sets into a predictive parsing table $M[A, a]$, where A is a nonterminal and a is a terminal or $\$$.
- Idea:
  - Choose $A \to \alpha$, if the next input symbol a is in $FIRST(\alpha)$.
  - If $\alpha \Rightarrow^* \epsilon$, choose $A \to \alpha$ if $a \in FOLLOW(A)$.

Construction of Parsing Table

Algorithm:

- Input: grammar G
- Output: parsing table M.
- Algorithm: For each production $A \to \alpha$ of the grammar, do the following:
  1. For each terminal a in $FIRST(\alpha)$, add $A \to \alpha$ to $M[A, a]$.
  2. If $\epsilon \in FIRST(\alpha)$, then for each terminal b in $FOLLOW(A)$, add $A \to \alpha$ to $M[A, b]$. If $\epsilon \in FIRST(\alpha)$ and $\$ \in FOLLOW(A)$, add $A \to \alpha$ to $M[A, \$]$ as well.

Example

	id	$+$	$*$	(	)	\$
E	$E \to T\ E'$			$E \to T\ E'$		
E'		$E' \to +\ T\ E'$			$E' \to \epsilon$	$E' \to \epsilon$
T	$T \to F\ T'$			$T \to F\ T'$		
T'		$T' \to \epsilon$	$T' \to *\ F\ T'$		$T' \to \epsilon$	$T' \to \epsilon$
F	$F \to \text{id}$			$F \to (E)$		

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

Non $LL(1)$ Grammars

Non $LL(1)$ grammars generate parsing tables with multiple entries.
Example:

$$\begin{array}{rcl} S & \to & i\ E\ t\ S\ S' \mid a \\ S' & \to & e\ S \mid \epsilon \\ E & \to & b \end{array}$$

Parsing table:

	a	b	e	i	t	$\$$
S	$S \to a$			$S \to i\ E\ t\ S\ S'$		
S'			$S' \to \epsilon, S' \to e\ S$			$S' \to \epsilon$
E		$E \to b$				

Summary

- Some grammars can be parsed in top-down by just looking at the next input symbol.
- Predictive parsing algorithm: $\boldsymbol{FIRST}$, $\boldsymbol{FOLLOW}$, parsing table