

# Table-Driven Top-Down Parsing

Prof. James L. Frankel  
Harvard University

Version of 5:43 PM 3-Oct-2023  
Copyright © 2023, 2022, 2018, 2015 James L. Frankel. All rights reserved.

# LL(1) Grammars

- Predictive parsers, or recursive descent parsers, which need no backtracking can be constructed from LL(1) grammars
- First “L” means that input is scanned from left to right
- Second “L” means that a leftmost derivation is applied
- The “1” means that one symbol of lookahead is possibly required

# FIRST Function

- $\text{FIRST}(\alpha)$  is the set of terminals that begin strings derived from  $\alpha$ , where  $\alpha$  is any string of grammar symbols
- If  $\alpha \Rightarrow^* \varepsilon$ , then  $\varepsilon$  is also in  $\text{FIRST}(\alpha)$

# FOLLOW Function

- FOLLOW( $A$ ), for non-terminal  $A$ , is the set of terminals  $a$  that can appear immediately to the right of  $A$  in some sentential form
- That is, the set of terminals  $a$  such that
$$S \Rightarrow^* \alpha A a \beta$$
 for some  $\alpha$  and  $\beta$ 
  - Symbols between  $A$  and  $a$  can exist so long as they derived  $\epsilon$  and disappeared
  - If  $A$  can be the rightmost symbol in some sentential form, then  $\$$  is in FOLLOW( $A$ )
  - $\$$  is a special endmarker symbol that is not a symbol of any grammar

# Algorithm for FIRST Function

- Compute  $\text{FIRST}(X)$  for all grammar symbols  $X$
  - Apply these rules until no more terminals or  $\epsilon$  can be added to  $\text{FIRST}(X)$
1. If  $X$  is a terminal, then  $\text{FIRST}(X) = \{ X \}$
  2. If  $X$  is a non-terminal and  $X \rightarrow Y_1 Y_2 \dots Y_k$  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  $\epsilon$  is in all of  $\text{FIRST}(Y_1), \dots, \text{FIRST}(Y_{i-1})$ ; that is,  $Y_1 \dots Y_{i-1} \Rightarrow^* \epsilon$ . If  $\epsilon$  is in  $\text{FIRST}(Y_j)$  for all  $j = 1, 2, \dots, k$ , then add  $\epsilon$  to  $\text{FIRST}(X)$ . For example, everything in  $\text{FIRST}(Y_1)$  is surely in  $\text{FIRST}(X)$ . If  $Y_1$  does not derive  $\epsilon$ , then we add nothing more to  $\text{FIRST}(X)$ , but if  $Y_1 \Rightarrow^* \epsilon$ , then we add  $\text{FIRST}(Y_2)$ , and so on.
  3. If  $X \rightarrow \epsilon$  is a production, then add  $\epsilon$  to  $\text{FIRST}(X)$

# Algorithm for FOLLOW Function

- Compute FOLLOW( $A$ ) for all non-terminals  $A$
- Apply these rules until nothing can be added to FOLLOW( $A$ )
  - 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  $\epsilon$  is in FOLLOW( $B$ )
  - 3. If there is a production  $A \rightarrow \alpha B$ , or a production  $A \rightarrow \alpha B \beta$ , where FIRST( $\beta$ ) contains  $\epsilon$ , then everything in FOLLOW( $A$ ) is in FOLLOW( $B$ )

# Construct Predictive Parsing Table M

- Algorithm 4.31 on pp. 224-225
- Rows are labelled with non-terminals
- Columns are labelled with terminal symbols
- Deal with each production separately, including those A-productions with a common A head that have been combined and rewritten as  $A \rightarrow \alpha_1 \mid \alpha_2 \mid \dots \mid \alpha_k$
- For each production  $A \rightarrow \alpha$  of the grammar, do the following:
  - 1. For each terminal  $a$  in  $\text{FIRST}(\alpha)$ , add  $A \rightarrow \alpha$  to  $M[A, a]$
  - 2. If  $\epsilon$  is in  $\text{FIRST}(\alpha)$ , then for each terminal  $b$  in  $\text{FOLLOW}(A)$ , add  $A \rightarrow \alpha$  to  $M[A, b]$ . If  $\epsilon$  is in  $\text{FIRST}(\alpha)$  and  $\$$  is in  $\text{FOLLOW}(A)$ , add  $A \rightarrow \alpha$  to  $M[A, \$]$  as well

# Table-Driven Predictive Parsing Algorithm

- Algorithm 4.34 on pp. 226-228
- Input is string  $w$  and parsing table  $M$  for grammar  $G$ ; Output is a leftmost derivation of  $w$  or an error indication
- The input buffer is initialized with  $w\$$ ; the stack is initialized with the start symbol  $S$  on top of stack, above  $\$$ ;  $a$  denotes the current input symbol
- set  $ip$  to point to the first symbol of  $w$ ;  
set  $X$  to the top stack symbol;  
while (  $X \neq \$$  ) { /\* stack is not empty \*/  
  if (  $X$  is  $a$  ) pop the stack and advance  $ip$ ;  
  else if (  $X$  is a terminal ) error();  
  else if (  $M[X, a]$  is an error entry ) error();  
  else if (  $M[X, a] = X \rightarrow Y_1 Y_2 \dots Y_k$  ) {  
    output the production  $X \rightarrow Y_1 Y_2 \dots Y_k$ ;  
    pop the stack;  
    push  $Y_k, Y_{k-1}, \dots, Y_1$  onto the stack, with  $Y_1$  on top;  
  }  
  set  $X$  to the top stack symbol;  
}



# Example of Table-Driven Parser

- Start with grammar:

$$E \rightarrow T E'$$

$$E' \rightarrow + T E' \mid \varepsilon$$

$$T \rightarrow F T'$$

$$T' \rightarrow * F T' \mid \varepsilon$$

$$F \rightarrow ( E ) \mid \mathbf{id}$$

- Terminals are +, \*, (, ), and **id**
- Go over construction of FIRST and FOLLOW sets and the predictive parsing table M
- Execute the table-driven predictive parsing algorithm on input **( id + id ) \* id + id \$**
- Apply the actions in the order generated by the predictive parsing algorithm to build the parse tree

# Computing the FIRST Sets

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

# Computing the FOLLOW Sets

- $\text{FOLLOW}( E ) = \{ \$ \} \cup \text{FIRST}( ) = \{ \$, ) \}$
- $\text{FOLLOW}( E' ) = \text{FOLLOW}( E ) = \{ \$, ) \}$
- $\text{FOLLOW}( T ) = \text{FIRST}( E' ) \dots =$   
 $\{ +, \varepsilon \} - \{ \varepsilon \} \cup \text{FOLLOW}( E' ) \cup \text{FOLLOW}( E ) =$   
 $\{ +, \varepsilon \} - \{ \varepsilon \} \cup \{ \$, ) \} = \{ +, ), \$ \}$
- $\text{FOLLOW}( T' ) = \text{FOLLOW}( T ) = \{ +, ), \$ \}$
- $\text{FOLLOW}( F ) = \text{FIRST}( T' ) \dots =$   
 $\{ *, \varepsilon \} - \{ \varepsilon \} \cup \text{FOLLOW}( T' ) \cup \text{FOLLOW}( T ) =$   
 $\{ *, \varepsilon \} - \{ \varepsilon \} \cup \{ +, ), \$ \} = \{ +, *, ), \$ \}$

# Parsing Table M

Non-Terminals	Input Terminals					
	id	+	*	(	)	\$
E	$E \rightarrow T E'$			$E \rightarrow T E'$		
E'		$E' \rightarrow + T E'$			$E' \rightarrow \epsilon$	$E' \rightarrow \epsilon$
T	$T \rightarrow F T'$			$T \rightarrow F T'$		
T'		$T' \rightarrow \epsilon$	$T' \rightarrow * F T'$		$T' \rightarrow \epsilon$	$T' \rightarrow \epsilon$
F	$F \rightarrow \text{id}$			$F \rightarrow ( E )$		

# Predictive Parser Moves for (id+id)\*id+id\$

(1 of 3)

Matched	Stack	Input	Action
	E \$	(id+id)*id+id\$	
	T E' \$	(id+id)*id+id\$	E → T E'
	F T' E' \$	(id+id)*id+id\$	T → F T'
	( E ) T' E' \$	(id+id)*id+id\$	F → ( E )
(	E ) T' E' \$	id+id)*id+id\$	Match (
	T E' ) T' E' \$	id+id)*id+id\$	E → T E'
	F T' E' ) T' E' \$	id+id)*id+id\$	T → F T'
	id T' E' ) T' E' \$	id+id)*id+id\$	F → id
(id	T' E' ) T' E' \$	+id)*id+id\$	Match id
	E' ) T' E' \$	+id)*id+id\$	T' → ε
	+ T E' ) T' E' \$	+id)*id+id\$	E' → + T E'

# Predictive Parser Moves for (id+id)\*id+id\$

(2 of 3)

Matched	Stack	Input	Action
	+ T E' ) T' E' \$	+id)*id+id\$	E' → + T E' [line above]
(id+	T E' ) T' E' \$	id)*id+id\$	Match +
	F T' E' ) T' E' \$	id)*id+id\$	T → F T'
	id T' E' ) T' E' \$	id)*id+id\$	F → id
(id+id	T' E' ) T' E' \$	)*id+id\$	Match id
	E' ) T' E' \$	)*id+id\$	T' → ε
	) T' E' \$	)*id+id\$	E' → ε
(id+id)	T' E' \$	*id+id\$	Match )
	* F T' E' \$	*id+id\$	T' → * F T'
(id+id)*	F T' E' \$	id+id\$	Match *
	id T' E' \$	id+id\$	F → id

# Predictive Parser Moves for (id+id)\*id+id\$

(3 of 3)

Matched	Stack	Input	Action
	id T' E' \$	id+id\$	F → id [line above]
(id+id)*id	T' E' \$	+id\$	Match id
	E' \$	+id\$	T' → ε
	+ T E' \$	+id\$	E' → + T E'
(id+id)*id+	T E' \$	id\$	Match +
	F T' E' \$	id\$	T → F T'
	id T' E' \$	id\$	F → id
(id+id)*id+id	T' E' \$	\$	Match id
	E' \$	\$	T' → ε
	\$	\$	E' → ε
			<b>DONE</b>

# Resulting Parse Tree

