Table-Driven Top-Down Parsing

Prof. James L. Frankel
Harvard University

Version of 1:30 PM 26-Feb-2018

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

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

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 α and β
  - Symbols between *A* and *a* can exist so long as they derived ε 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 FIRST(X) for all grammar symbols X
- Apply these rules until no more terminals or ε can be added to FIRST(X)

- 1. If X is a terminal, then FIRST(X) = { X }
- 2. If X is a non-terminal and $X \rightarrow Y_1 Y_2 \ldots Y_k$ is a production for some $k \geq 1$, then place a in FIRST(X) if for some i, a is in FIRST(Y_i), and ε is in all of FIRST(Y_1), …, FIRST(Y_{i-1}); that is, $Y_1 \ldots Y_{i-1} \Rightarrow^* \varepsilon$. If ε is in FIRST($Y_j$) for all j = 1, 2, …, k, then add ε to FIRST(X). For example, everything in FIRST(Y_1) is surely in FIRST(X). If Y_1 does not derive ε, then we add nothing more to FIRST(X), but if $Y_1 \Rightarrow^* \varepsilon$, then we add FIRST($Y_2$), and so on.
- 3. If $X \rightarrow \varepsilon$ is a production, then add ε to 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(β) except ε is in FOLLOW(*B*)
- 3. If there is a production $A \rightarrow \alpha B$, or a production $A \rightarrow \alpha B \beta$, where FIRST(β) contains ε, then everything in FOLLOW(*A*) is in FOLLOW(*B*)

Construct Predictive Parsing Table M

- Algorithm 4.31 on pp. 224-225

- For each production A → α of the grammar, do the following:

- 1. For each terminal *a* in FIRST(α), add *A* → α to M[*A*, *a*]
- 2. If ε is in FIRST(α), then for each terminal *b* in FOLLOW(*A*), add *A* → α to M[*A*, *b*]. If ε is in FIRST(α) and $ is in FOLLOW(A), add *A* → α 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 ($M[X, a] = X \rightarrow Y_1 Y_2 \ldots Y_k$) {
      output the production $X \rightarrow Y_1 Y_2 \ldots Y_k$;
      pop the stack;
      push $Y_k, Y_{k-1}, \ldots, Y_1$ onto the stack, with Y_1 on top;
    }
    set *X* to the top stack symbol;
  }

Example of Table-Driven Parser

- Start with grammar:
  E → T E'
  E' → + T E' | ε
  T → F T'
  T' → * F T'| ε
  F → (E) | **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

- FIRST(+) = { + }
- FIRST(*) = { * }
- FIRST(( ) = { (}
- FIRST( )) = {) }
- FIRST(**id**) = { **id** }
- FIRST(F) = { (, **id** }
- FIRST(T') = { *, ε }
- FIRST(T) = FIRST(F) = { (, **id** }
- FIRST(E') = { +, ε }
- FIRST(E) = FIRST(T) = FIRST(F) = { (, **id** }

Computing the FOLLOW Sets

- FOLLOW(E) = { $ } U FIRST( )) = { $,) }
- FOLLOW(E') = FOLLOW(E) = { $,) }
- FOLLOW(T) = FIRST(E') … =
  { +, ε } – { ε } U FOLLOW(E') U FOLLOW(E) =
  { +, ε } – { ε } U { $,) } = { +,), $ }
- FOLLOW(T') = FOLLOW(T) = { +,), $ }
- FOLLOW(F) = FIRST(T') … =
  { *, ε } – { ε } U FOLLOW(T') U FOLLOW(T) =
  { *, ε } – { ε } U { +,), $ } = { +, *,), $ }

Parsing Table M

Non-Terminals	Input Terminals					
	id	+	*	(	)	$
E	E → T E'			E → T E'		
E'		E' → + T E'			E' → ε	E' → ε
T	T → F T'			T → F T'		
T'		T' → ε	T' → * F T'		T' → ε	T' → ε
F	F → **id**			F → (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
			DONE

Resulting Parse Tree

"(**id**+**id**)***id**+**id**"