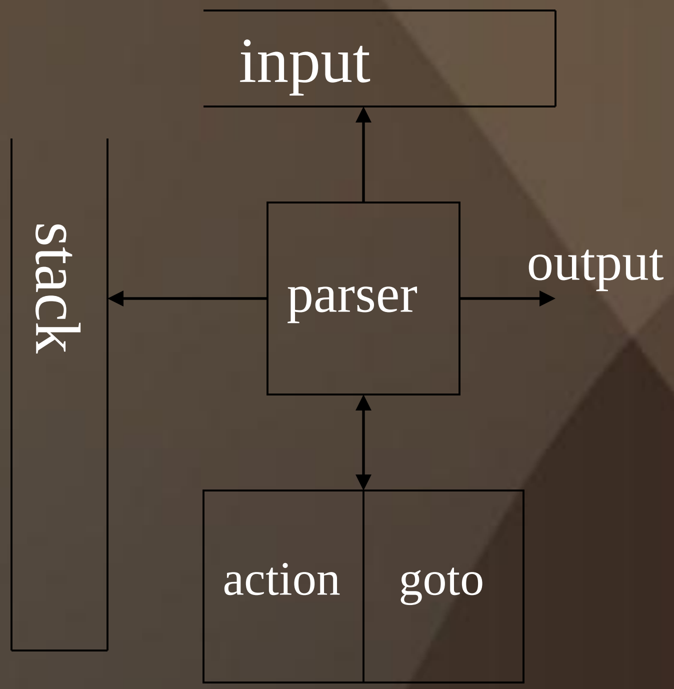
Syntax Analysis Slides – Part - II

LR Parsing – Obtaining the parsing table



Parse table

- Input contains the input string.
- Stack contents are of the form $S_0X_1, S_1X_2 \ldots X_nS_n$
    - Each X_i is a grammar symbol and each S_i is a state.
- Tables contain *action* and *goto* parts.
- Action table is indexed by state and terminal symbols.
- Goto table is indexed by state and parsing symbols (DFA Transition Table).

Building Action Table

For each state s_i and terminal a

- If s_i has item $X \rightarrow \alpha.a\beta$ and $goto[i,a] = j$ then $action[i,a] = shift\ j$
- If s_i has item $X \rightarrow \alpha.$ and $a \in Follow(X)$ and $X \neq S'$ then $action[i,a] = reduce\ X \rightarrow \alpha$
- If s_i has item $S' \rightarrow S.$ then $action[i,\$] = accept$
- Otherwise, $action[i,a] = error$

SLR(1) Parsing Algorithm

1. Let I[n] = w$ be the initial input; n = length of input
2. Let j = 0
3. Let DFA state 1 have item **S' → S.**
4. Let stack = < dummy, 1 >
5. Repeat
   a) Switch (Action [top_state(stack), I[j]])
      a) Case Shift k: Push < I[j++], k >
      b) Case Reduce **X** → **A**:
         a) Pop |**A**| pairs
         b) Push < Goto [top_state(stack), X], X >
      c) Case Accept: Halt normally
      d) Case Error: Halt and report error

SLR Parser Tracing

- Start with initial state S_0 on stack. The next input token is **a** and current state is S_t. The action of the parser is as follows:

1. If **Action[S_t, a]** is shift, we push the specified state onto the stack. We then call **yylex()** to get the next token **a** from the input.
2. If **Action[S_t, a]** is reduce $X \rightarrow Y_1 \ldots Y_k$, then we pop **k** states off the stack (one for each <symbol, state> pair) leaving state S_u on top. **Goto[S_u,X]** gives the new state S_v to be pushed onto the stack along with the symbol **X**. Input token is still **a** (i.e., the input remains unchanged).
3. If **Action[S_t, a]** is accept, then parse is successful and we are done.
4. If **Action[S_t, a]** is error (the table location is blank), then we have a syntax error.
   - *With the current top of stack and next input we can never arrive at a sentential form with a handle to reduce.*

LR (0) Parsing Table

Create the Action and goto tables for the DFA generated

STATE	ACTION						GOTO	
	int	+	*	(	)	$	E	T
1	Sh3			Sh8			2	5
2						Accept		
3	R5	R5	R5	R5	R5	R5		
4	R4	R4	R4	R4	R4	R4		
5	R3	R3	R3	R3	R3	R3		
6	Sh3			Sh8			7	5
7	R2	R2	R2	R2	R2	R2		
8	Sh3			Sh8			9	5
9					Sh10			
10	R6	R6	R6	R6	R6	R6		
11	Sh3			Sh8				4

1. **S’ → E**
2. **E → T + E**
3. **E → T**
4. **T → int * T**
5. **T → int**
6. **T → (E)**

Points to Note

- The SLR(1) parsing table is similar to the table in the previous slide, but not identical. Apply the algorithm in slide-3 to find the differences.

- The GOTO table has only been defined for non-terminals. This is because, this part of the GOTO table is sufficient for the execution of the SLR(1) parser (slide-4).

- Both the LR(0) and SLR(1) parsers use LR(0) items, because the items as such do not have any look-ahead associated with them.

- Do you require to change anything within the SLR(1) parsing algorithm to make it a LR(0) parser?