Syntax Analysis- III

LR Parsers

LR(0) and SLR(1)

An LR parser is a parser that input from left to right and generates right most derivations. This is basically non-recursive shift reduce bottom up parser.

L→ Left to right scanning of i/p string

R→ Right most derivation in reverse order.

k→ Look Ahead symbol.

LR parsers have two parts:-

1) Driving Routine

2) Parsing Table

Advantage of LR:

1. Can run on practically any programming language which follows CFG.

2. Errors can be detected in quick.

3. Extremely efficient.

Disadvantage of LR:

1. Difficult to implement by hand as they are highly complex.

Structure of LR Parsing Table:

1) Action

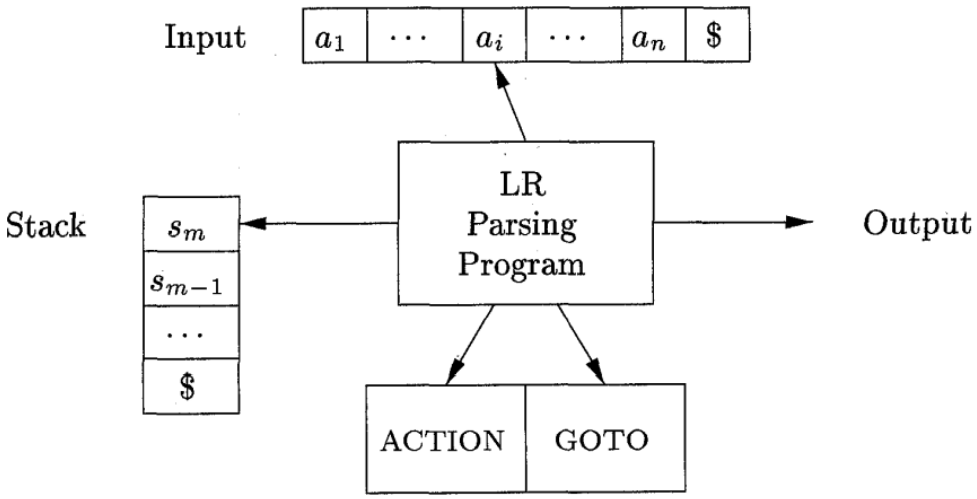
2) GOTO

Rows of Action-GOTO table :- The no of set of items.

Columns of Action table:- All are terminals.

Columns of GOTO table:- All are non-terminals.

A schematic of an LR parser is shown in Fig. 4.35. It consists of an input, an output, a stack, a driver program, and a parsing table that has two parts (ACTION and GOTO). The driver program is the same for all LR parsers; only the parsing table changes from one parser to another. The parsing program reads characters from an input buffer one at a time. Where a shift-reduce parser would shift a symbol, an LR parser shifts a *state*. Each state summarizes the information contained in the stack below it.



The parsing table has the following actions:-

1) Shift
2) Reduce
3) Accept
4) Error

1) Shift→The format is S_n where n is a number defines the state. Shift S_n means that current input symbol is to be shifted and current state will be given by 'n'.

Therefore, Control moves to row n.

2) Reduce→The format is R_n where n is a number represents the production rule used for reduction. Controls stays in same row.

It is followed by GOTO move. GOTO accepts current state and non-terminal of Reduce step to generate s state S_n. Control then shifts to row n.

3) Accept→ If the string is valid the parsing table reaches to the accept cell and the string is accepted.

4) Error→If the parser reaches an empty cell, an error has occurred and the error handler routine is called.

Items

An ***LR(0) item*** simply called an 'item' is defined as a production rule which has a dot(**.**) on its RHS. The dot represents the input symbols that have been read and input symbols waiting to be read.

Closure of Item Sets

If I is a set of items for a grammar G, then CLOSURE(I) is the set of items constructed from I by the two rules:

1. Initially, add every item in I to CLOSURE(I).
2. If $A \to \alpha \cdot B\beta$ is in CLOSURE(I) and $B \to \gamma$ is a production, then add the item $B \to \cdot\gamma$ to CLOSURE(I), if it is not already there. Apply this rule until no more new items can be added to CLOSURE(I).

GOTO (I,X) → GOTO(I,X) is function which inputs set of items I and an input symbol X for every rule A→α.Xβ, GOTO(I,X) produces the closure of set I_n.,

Where I_n = αX.β

Algorithm for generation of canonical collection of set of items:

Begin

C= { Closure of S'→.S }*

for all items in closure C and each grammar symbol X present,

find GOTO(I,X)

add it to C

until no more sets can be added to C

End

*S'→S is a part of the augmented grammar.

Behaviour of the LR Parser:

1. If ACTION$[s_m, a_i]$ = shift s, the parser executes a shift move; it shifts the next state s onto the stack, entering the configuration

$$(s_0 s_1 \cdots s_m s,\ a_{i+1} \cdots a_n \$)$$

   The symbol a_i need not be held on the stack, since it can be recovered from s, if needed (which in practice it never is). The current input symbol is now a_{i+1}.

2. If ACTION$[s_m, a_i]$ = reduce $A \to \beta$, then the parser executes a reduce move, entering the configuration

$$(s_0 s_1 \cdots s_{m-r} s,\ a_i a_{i+1} \cdots a_n \$)$$

3. If ACTION$[s_m, a_i]$ = accept, parsing is completed.

4. If ACTION$[s_m, a_i]$ = error, the parser has discovered an error and calls an error recovery routine.

The LR-parsing algorithm is summarized below. All LR parsers behave in this fashion; the only difference between one LR parser and another is the information in the ACTION and GOTO fields of the parsing table.

Q. Construct the ***LR(0)*** parser table for the following grammar and check the validity of the given string, **W= a a b b $**

Ex:-

$S \rightarrow A\ A$

$A \rightarrow a\ A \mid b$

Augmented Grammar :-

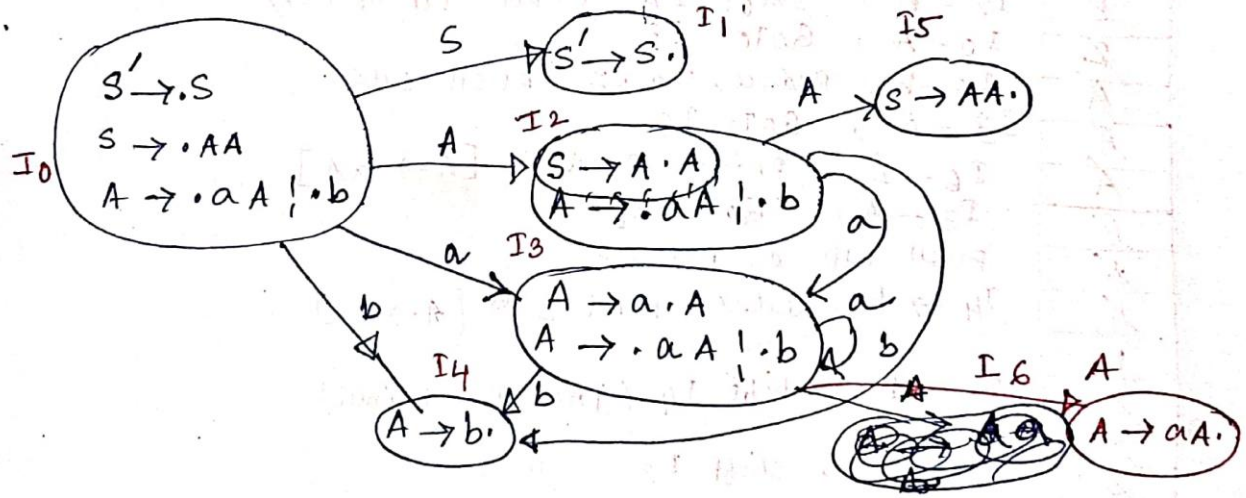
$S' \rightarrow S$

$S \rightarrow AA$

$A \rightarrow aA \mid b$

step-1

- RHS would be reduced by LHS of production.
- Now find the closure for each item.



DFA with canonical clasoure of items.

Final items: $S \rightarrow AA.$ (I_5), $A \rightarrow aA.$ (I_6), $A \rightarrow b.$ (I_4)

step-2

Number the productions:

$S \rightarrow AA$ — (1)

$A \rightarrow aA$ — (2)

$A \rightarrow b$ — (3)

Sl-3 Creation of LR Parsing Table

No. of states (n) = 7 , (I_0 - - - I_6)

	Action			GOTO	
	a	b	$	A	S
0	s_3	s_4		2	1
1			Accept		
2	s_3	s_4		5	
3	s_3	s_4		6	
4	r_3	r_3	r_3		
5	r_1	r_1	r_1		
6	r_2	r_2	r_2		

(*) For a state having final item, we will write the Reduce move to the complete row / entire row.

Now, I/P string w = a a b b $

1	Accept
S	$S \to I_0$ GOTO I_1 & $I_1 - \$$. Accept [$S \to AA$]
S	$I_5 - \$$, Reduce, Prod - 1.
A	$I_2 - A$, GOTO I_5
A	$I_4 - \$$, Reduce prod 3
b	$I_2 - b$, Shift I_4 (Incr. I/P pointer)
2	$I_0 - A$, GOTO - I_2
A	$I_6 - b$, Reduce, prod 2, PUSH LHS.
b	$I_3 - A$, GOTO I_6
A	$I_6 - b$, Reduce prod 2. [$A \to aA$]
b	$I_3 - A$, GOTO I_6.
A	PUSH LHS of Rule - 3.
4	$I_4 \to b$, Reduce prod 3. (*) [$A \to b$]
b	
3	$I_3 \to b$, Shift I_4 (Incr. I/P Pointer)
a	
3	$I_3 \to a$, Shift I_3 (")
a	$I_0 \to a$, Shift I_3 (")
0	Accepting state.

(*) Here we are reducing the previous symbol of current look ahead. Not the current symbol.

Now, (⊙) POP from TOP of the stack (2x) items.

x = length of RHS. on that rule.

(⊙) PUSH LHS on the stack

(*) LR(0) means here we are not aware of next symbol. Whatever the next symbol is, we can reduce always. (Entire Row)

Parsing Algorithm for SLR(1) or LR(1)

METHOD:

1. Construct $C = \{I_0, I_1, \ldots, I_n\}$, the collection of sets of LR(0) items for G'.

2. State i is constructed from I_i. The parsing actions for state i are determined as follows:

   (a) If $[A \to \alpha \cdot a\beta]$ is in I_i and $\text{GOTO}(I_i, a) = I_j$, then set $\text{ACTION}[i, a]$ to "shift j." Here a must be a terminal.

   (b) If $[A \to \alpha \cdot]$ is in I_i, then set $\text{ACTION}[i, a]$ to "reduce $A \to \alpha$" for all a in $\text{FOLLOW}(A)$; here A may not be S'.

   (c) If $[S' \to S \cdot]$ is in I_i, then set $\text{ACTION}[i, \$]$ to "accept."

3. The goto transitions for state i are constructed for all nonterminals A using the rule: If $\text{GOTO}(I_i, A) = I_j$, then $\text{GOTO}[i, A] = j$.

4. All entries not defined by rules (2) and (3) are made "error."

5. The initial state of the parser is the one constructed from the set of items containing $[S' \to \cdot S]$.

Do the following:

$$
\begin{aligned}
&E' \to \cdot E \\
&E \to \cdot E + T \\
&E \to \cdot T \\
&T \to \cdot T * F \\
&T \to \cdot F \\
&F \to \cdot (E) \\
&F \to \cdot \mathbf{id}
\end{aligned}
$$