INTRODUCTION TO LALR PARSER

- The last parser method LALR (LookAhead-LR) technique is often used in practice because the tables obtained by it are considerably smaller than the canonical LR tables.
- The SLR and LALR tables for a grammar always have the same number of states whereas CLR would typically have several number of states.
- For example, for a language like Pascal SLR and LALR would have hundreds of states but CLR would have several thousands of states.

Algorithm: LALR table construction.

INPUT: An augmented grammar G'.

OUTPUT: The LALR parsing-table functions ACTION and GOT0 for G'.

METHOD:

1. Construct C = (I0 I1, . . , In), the collection of sets of LR(1) items.

2. For each core present among the set of LR(1) items, find all sets having that core, and replace these sets by their union.

3. Let C' = {J0J1, . . . ,Jm) be the resulting sets of LR(1) items. If there is a parsing action conflict, the algorithm fails to produce a parser, and the grammar is said not to be LALR(1).

4. The GOTO table is constructed as follows. If J is the union of one or more sets of LR(1) items, that is, J = I1 U I2 U . . .U Ik , then the cores of GOTO(I1, X) , GOTO(I2, X) , . . . , GOTO(Ik, X) are the same, since I1,I2, . . ,Ik all have the same core. Let K be the union of all sets of items having the same core as GOTO(I1, X). Then GOTO(J,X) = K.

Let us consider grammar

S->CC

C->cC

C->d whose sets of LR(1) items are

I_0 : S'->.S , $ S->.CC, $ C->.cC , c/d C->.d , c/d Goto(I_0,S) I_1 : S'->S., $ Goto(I_0, C) I_2 : S->C.C,$ C-> .cC,$ C->.d ,$	Goto(I_0, c) I_3: C->c.C, c/d C->.cC, c/d C-> .d , c/d Goto(I_0,d) I_4: C->d., c/d Goto(I_2, C) I_5: S->CC., $ Goto(I_2,c) I_6: C->c.C, $ C-> .cC, $ C->.d, $	Goto(I_2,d) I_7: C ->d., $ Goto(I_3, C) I_8: C->cC. , c/d Goto(I_3,c) = I_3 Goto(I_3, d) = I_4 Goto (I_6, C) I_9: C->cC., $ Goto(I_6, c) = I_6 Goto(I_6, d) = I_7

In the above LR(1) items, there are three pairs of sets of items that can be merged. I_3 and I_6 are replaced by their union:

I_{36}: C->c.C, c/d/$
C->.cC, c/d /$
C-> .d , c/d/$

I_4 and I_7 are replaced by their union:

I_{47}: C->d., c/d/$

and I_8 and I_9 are replaced by their union

I_8: C->cC. , c/d/$

The LALR action and goto functions for the condensed sets of items are

STATE	ACTION			GOTO	
	c	*d*	$	*S*	*C*
0	s36	s47		1	2
1			acc		
2	s36	s47			5
36	s36	s47			89
47	r3	r3	r3		
5			r1		
89	r2	r2	r2		

Error recovery in LR parsing:

- An LR parser will detect an error when it consults the parsing action table and finds an error entry. An LR parser will announce an error as soon as there is no valid continuation for the portion of the input thus far scanned.
- A canonical LR parser will not make even a single reduction before announcing an error. SLR and LALR parsers may make several reductions before announcing an error, but they will never shift an erroneous input symbol onto the stack.

Panic-mode error recovery:

- We scan down the stack until a state s with a goto on a particular nonterminal A is found. Zero or more input symbols are then discarded until a symbol a is found that can legitimately follow A.
- The parser then stacks the state GOTO (s, A) and resumes normal parsing.

Phrase-level recovery:

- It is implemented by examining each error entry in the LR parsing table and deciding on the basis of language usage the most likely programmer error that would give rise to that error.
- An appropriate recovery procedure can then be constructed; presumably the top of the stack and/or first input symbols would be modified in a way deemed appropriate for each error entry.

ERROR HANDLING AND RECOVERY IN SYNTAX ANALYZER

The different strategies that a parse uses to recover from a syntactic error are:

1. Panic mode
2. Phrase level
3. Error productions
4. Global correction

Panic mode recovery:

- On discovering an error, the parser discards input symbols one at a time until a synchronizing token is found. The synchronizing tokens are usually delimiters, such as semicolon or end.
- It has the advantage of simplicity and does not go into an infinite loop. When multiple errors in the same statement are rare, this method is quite useful.

Phrase level recovery:

- On discovering an error, the parser performs local correction on the remaining input that allows it to continue.
- Example: Insert a missing semicolon or delete an extraneous semicolon etc.

Error productions:

- The parser is constructed using augmented grammar with error productions. If an error production is used by the parser, appropriate error diagnostics can be generated to indicate the erroneous constructs recognized by the input.

Global correction:

- Given an incorrect input string x and grammar G, certain algorithms can be used to find a parse tree for a string y, such that the number of insertions, deletions and changes of tokens is as small as possible. However, these methods are in general too costly in terms of time and space.