Parse a string using LALR parser

→ Consider the string aadd & parse it using LALR parsing table.

Stack	Input buffer	ACTION Table	GOTO Table	Parsing action
$0	aadd$	action[0,a]=s36		shift
$0a36	add$	action[36,a]=s36		shift
$0a36a36	add$	action[36,d]=s47		shift
$0a36a36d47	d$	action[47,d]=r3	[36,C]=89	Reduce by C→d
$0a36a36C89	d$	action[89,d]=r2	[36,C]=89	Reduce by C→aC
$0a36C89	d$	action[89,d]=r2	[0,C]=2	Reduce by C→aC
$0C2	d$	action[2,d]=s47		shift
$0C2d47	$	action[47,$]=r3	[2,C]=5	Reduce by C→d
$0C2C5	$	action[5,$]=r1	[0,S]=1	Reduce by S→CC
$0S1	$	action[1,$]= Accept		

→ The table shows the successful parsing of string.

→ The execution is same as in CLR parsing only the merged state in LALR is treated as a single state like 36, 47, 89.

Algorithm for LALR parsing table

Step-1 Construct the LR(1) set of items

Step-2 Merge the two states I_i and I_j (if productions are exactly same but lookahead differ) then create a new state

$$I_{ij} = I_i \cup I_j$$

Step-3 The parsing actions are based on each item I_i. The actions are

(a) If $[A \rightarrow \alpha \cdot X\beta, a]$ is in I_i & $goto(I_i, X) = I_j$ then create an entry in action table $action[I, x] = shift\,j$.

(b) If there is a production $[A \rightarrow \alpha \cdot, a]$ in I_i then in the action table $action[I_i, x] =$ reduce by $A \rightarrow \alpha$

(c) If there is a production $S' \rightarrow S \cdot, \$$ in I_i then $action[I, \$] = accept$.

The goto part of the LR table can be filled as

Step-4 If $goto(I_i, A) = I_j$ then $goto[I_i, A] = j$

Step-5 If the parsing action conflict then the algorithm fails to produce LALR parsing & grammar is not LALR(1).