LALR Parser

This is the last (and often used) parser construction method, the LALR (Look Ahead - LR) parser.

Key reasons of it being most used are:

1) It is stronger than SLR and hence can construct parser tables for grammars that cause conflict in SLR technique.
2) It constructs parser table with less or equal numbers of states than CLR. (Optimized parser table).
3) ~~It is~~ yet most common syntactic constructs of programming languages can be expressed by LALR grammars.

Steps of LALR construction:

I) It constructs LR(1) set of items. [similar to CLR].

II) It looks for set of LR(1) items having same core. (ie same first component of set) and merges them into one common set (like they would have been in LR(0) set of items) provided they don't cause conflict. [S-R or R-R conflict].

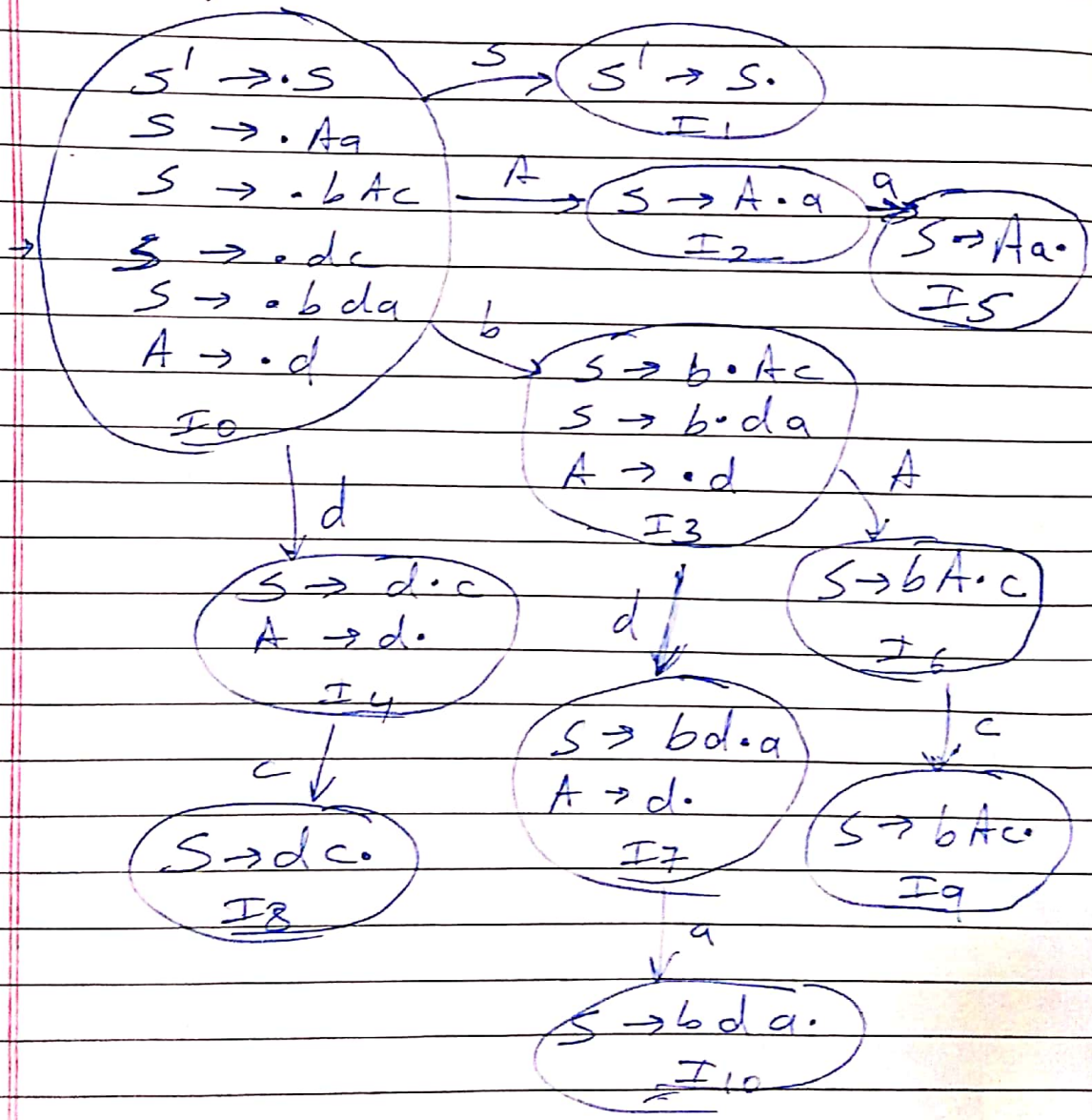
Example: Consider grammer.

$S \to Aa \mid bAc \mid dc \mid bda$

$A \to d$

Constructing by SLR technique:

LR(0) Set of Items.



FOLLOW(S) = {$}

FOLLOW(A) = {a, c}

Parser Table (as constructed by SLR(1))

State	Actions (terminals)					goto (Nonterminals)	
	a	b	c	d	$	S	A
I_0		s3		s4		1	2
I_1					Acc		
I_2	s5						
I_3				s7			6
I_4	r5		r5 s8				
I_5					r1		
I_6			s9				
I_7	r5 s10		r5				
I_8					r3		
I_9					r2		
I_{10}					r4		

Production list

1) $S \to Aa$
2) $S \to bAc$
3) $S \to dc$
4) $S \to bda$
5) $A \to d$

NOTE: $[I_4, c]$ & $[I_7, a]$ have multiple action entries.
Hence grammar is not SLR(1) (as it causes shift - reduce conflict)

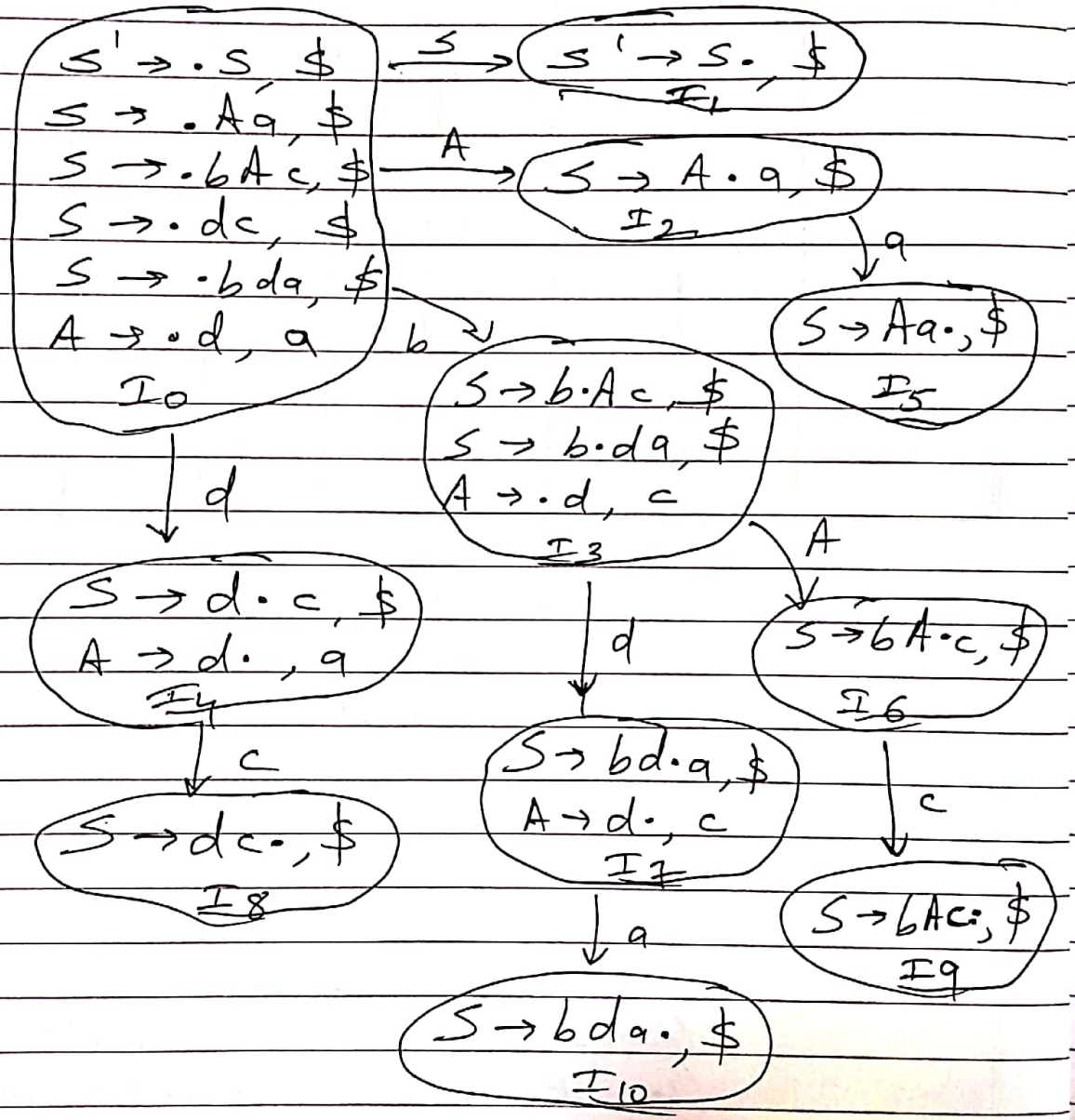
⇒ Next we construct LALR(1) parser for the same grammar.

LALR parser construction for

$S \to Aa \mid bAc \mid dc \mid bda$

$A \to d$

I) Constructing LR(1) set of items (like CLR)



No cores are matching. Hence no minimization / merging of states.

Parser Table (as constructed by LALR(1))

State	Actions (terminals)					goto (Non T.)	
	a	b	c	d	$	S	A
I_0		s3		s4		1	2
I_1					Acc		
I_2	s5						
I_3				s7			6
I_4	r5		s8				
I_5				r1			
I_6			s9				
I_7	s10		r5				
I_8				r3			
I_9				r2			
I_{10}				r4			

Production list
1) $S \rightarrow Aa$
2) $S \rightarrow bAc$
3) $S \rightarrow dc$
4) $S \rightarrow bda$
5) $A \rightarrow d$

Note that there are no conflicts in the above parser table.
Hence, the given grammar is LALR(1) but not SLR(1).

TASK: Check whether the previously given grammars (after CLR) are LALR(1) or not.

NOTE 1:

Number of states in SLR ≤ Number of states in LALR ≤ Number of states in CLR.

Try designing LALR parser for

$S \to CC$

$C \to aC \mid b$

You will find that the number of states are more than what you would have obtained by SLR (similar cores). Try merging these cores and you find no conflict and hence LALR will be seen to have lesser states than CLR.

NOTE 2:

SLR is less powerful than LALR which in turn is less powerful than CLR.

Try designing CLR and LALR for

$S \to Aa \mid bAc \mid Bc \mid bBa$

$A \to d$

$B \to d$.

It is CLR(1) but not LALR(1).