Designing SLR for $S \to aAb \mid aBa \mid bAa \mid bBb$

$A \to c$

$B \to c$

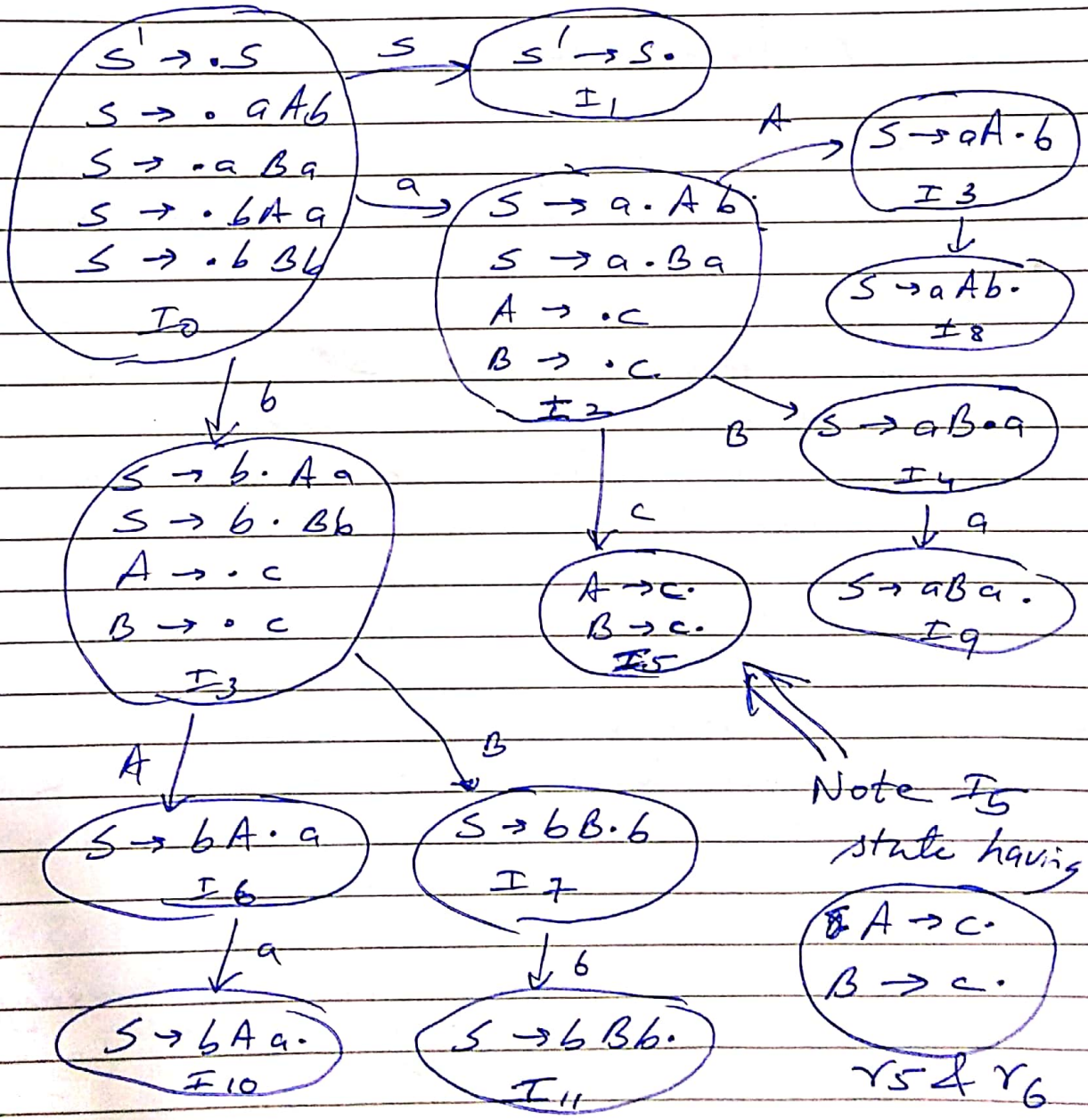
Production list : (0) $S' \to S$

(1) $S \to aAb$ (2) $S \to aBa$

(3) $S \to bAa$ (4) $S \to bBb$

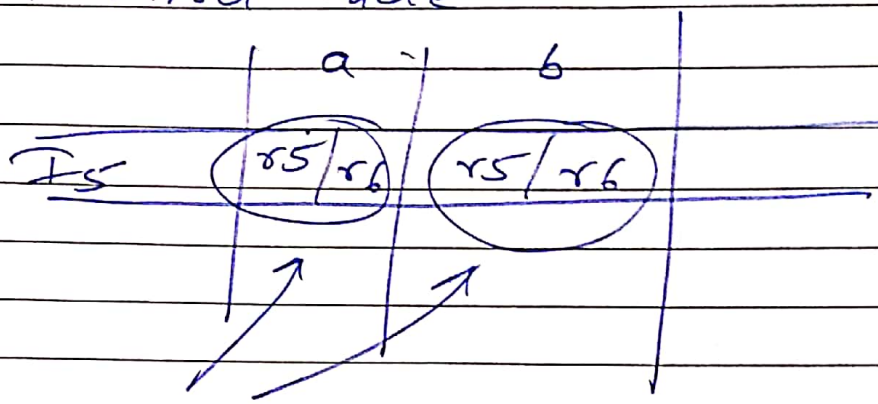
(5) $A \to c$ (6) $B \to c$

LR(0) Set of Items.



FOLLOW(A) = {a, b} FOLLOW(B) = {a, b}.

In Parser Table



Reduce - Reduce conflict.

Having multiple action entries for same ~~state~~ state of stack on same input is a conflict.

In above example SLR parser has reduce - reduce conflict.
It can also have shift - reduce conflict.

TASK:

Try designing SLR parser for grammar: $S \rightarrow L=R \mid R$
$L \rightarrow *R \mid id$
$R \rightarrow L$

[NOTE - It will have shift-reduce conflict in state I_2]

CLR Parser

It carries more information in a state that allow us to rule out some of the invalid reductions and hence reduce conflicts (S-R and R-R conflicts).

The extra information is a terminal symbol (or a group of terminals) as a second component.

In general written as $[A \to \alpha . \beta, a]$

where 'a' is a terminal or $, termed as the lookahead component.

Such objects are an LR(1) item. Hence the set of states are LR(1) set of items.

The terminal 'a' in above representation is a subset of FOLLOW(A)

In a derivation: ~~$S \overset{*}{\Rightarrow} \alpha A$~~

$$S \overset{*}{\Rightarrow} \gamma A \delta \Rightarrow \gamma \alpha \beta \delta$$

for production $[A \to \alpha . \beta, a]$ → a is FIRST(δ) or $ if $\delta = \epsilon$

we will call this as local follow.

The initial set I0 starts with

$[S' \to .S, \$]$ ⟹ since follow of entire string is $

Consider the grammar $S \to aAb / aBa / bAa /$
$A \to c$
$B \to c$

The initial set I_0 will be

I_0 : $S' \to .S, \$$ // fixed
$S \to .aAb, \$$
$S \to .aBa, \$$
$S \to .bAa, \$$
$S \to .bBb, \$$
} Since follow of S is $

After transition on 'a' from I_0 we get

I_2 : $S \to a.Ab, \$$
$S \to a.Ba, \$$
} same follow as before transition
$A \to .c, b$ // Follow A above is
$B \to .c, a$ // Follow of B above is 'a'

Note that in I_2 :
production $A \to .c$ is included due to $S \to a.Ab$ where the Follow of A is 'b'
and production $B \to .c$ is included due to $S \to a.Ba$ where the FOLLOW of B is 'a'

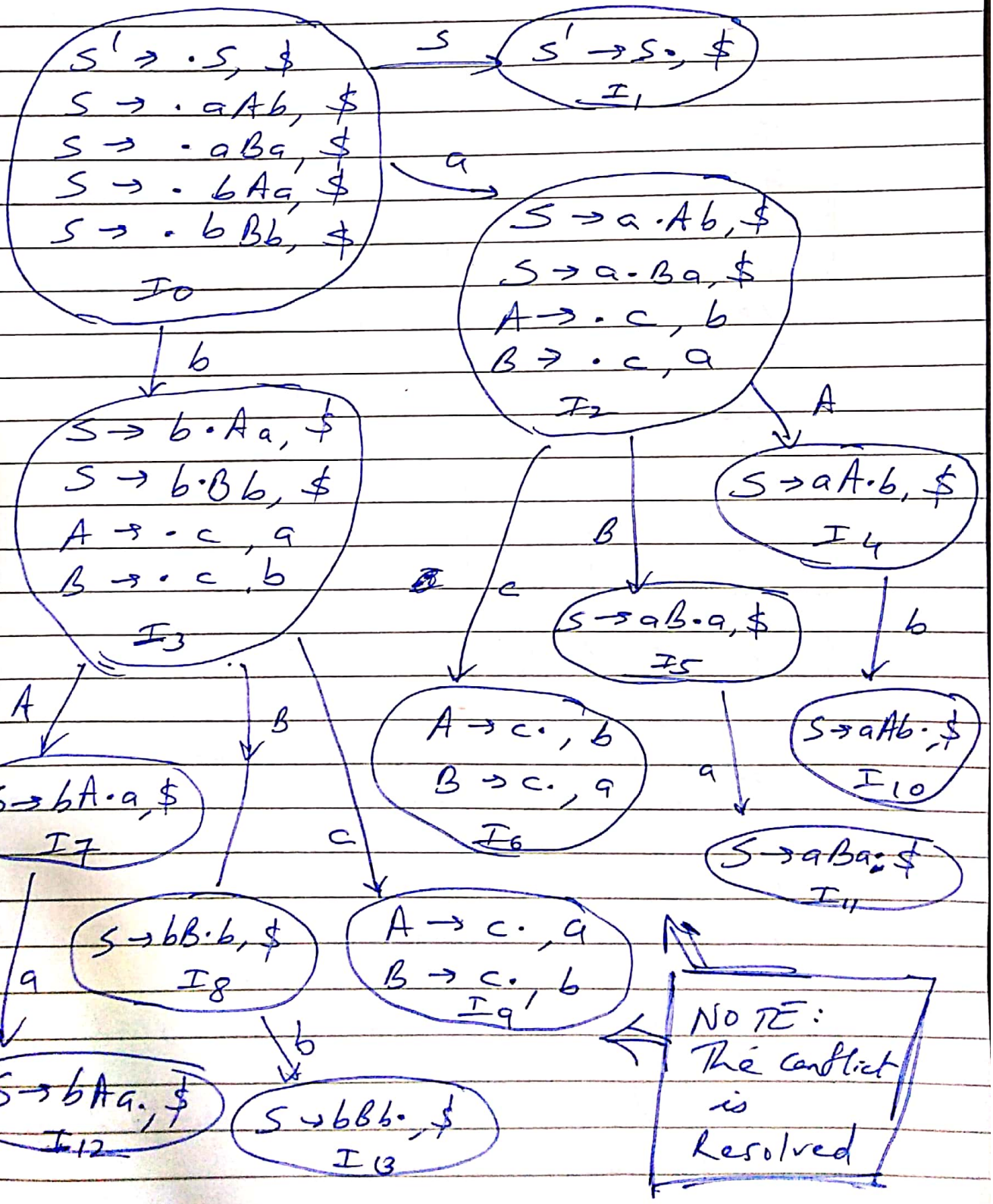
Whereas the FOLLOW(A) = {a, b}
and FOLLOW(B) = {a, b}
} in entire grammar

This helps us in removing invalid reductions.

CLR design for the prev grammar.

0) $S' \to S$ (1) $S \to aAb$
2) $S \to aBa$ (3) $S \to bAa$
4) $S \to bBb$ (5) $A \to c$ (6) $B \to c$

LR(1) set of items.



Parser Table

State	Action (Terminal)				goto (Variable)		
	a	b	c	$	S	A	B
I_0	S2	S3			1		
I_1				Accept			
I_2			S6			4	5
I_3			S9			7	8
I_4		S10					
I_5	S11						
I_6	r6	r5					
I_7	S12						
I_8		S13					
I_9	r5	r6					
I_{10}			r1				
I_{11}			r2				
I_{12}			r3				
I_{13}			r4				

CLR (Canonical LR) is a superset of SLR (Simple LR).
ie. Every SLR grammar is CLR but not every CLR grammar is SLR!

Hence
CLR is more powerful than SLR.

Drawback: It has more states.

Unwanted states can be minimized by LALR [core matching].

TASK:

1) Design CLR for the previous grammar (which was also not SLR)

$S \rightarrow L=R \mid R$

$L \rightarrow *R \mid id$

$R \rightarrow L$

and check if the shift-reduce conflict is resolved in CLR.

2) Design SLR(1) and CLR(1) for the following grammars.

a) $S \rightarrow Aa \mid bAc \mid dc \mid bda$

$A \rightarrow d$

b) $S \rightarrow CC$

$C \rightarrow aC \mid b$

c) $S \rightarrow Aa \mid bAc \mid Bc \mid bBa$

$A \rightarrow d$

$B \rightarrow d.$