Compiler Design

Unit-3

Bottom-up Parsing :-

$E \rightarrow E + T \mid T$
$T \rightarrow T * F \mid F$
$F \rightarrow (E) \mid id$

Generate Parse tree for id * id

⇒ Step 1 → F * id
|
id

Step 2 → T * id
|
F
|
id

Step 3 → T * F
| |
F id
|
id

Step 4 → T * F
| |
F id
|
id

Step 5 →
E
|
T
/ | \
T * F
| |
F id
|
id

⇒ Reductions :- id * id, F * id, T * id, T * F, F, E

Handle Pruning :-

The handle is the substring that matches the body of a production whose reduction represents one step along with the reverse of the rightmost derivation.

→ Removing the children of left-hand side non-terminal from parse tree is called Handle pruning.

A rightmost derivation in reverse can be obtained by handle pruning.

For the same example we saw in last page, lets see how it works :-

Input String	Handle	Reductions
$id_1 * id_2$	id_1	$F \rightarrow id_1$
$F * id_2$	F	$T \rightarrow F$
$T * id_2$	id_2	$F \rightarrow id_2$
$T * F$	$T * F$	$T \rightarrow T * F$
T	T	$E \rightarrow T$
E (Start)		

↑ accept

Q. $S \rightarrow aABe$
$A \rightarrow Abc \mid b$
$B \rightarrow d$
Input String abbcde

Input String	Handle	Reduction
abbcde	b	A → b
aAbcde	Abc	A → Abc
aAde	d	B → d
aABe	aABe	S → aABe
S (Start)		

↑
accept

Shift Reduce Parsing :-

Q
$E \rightarrow E+T / T$
$T \rightarrow T*F / F$
$F \rightarrow (E) / id$

Input String → $id_1 * id_2$.

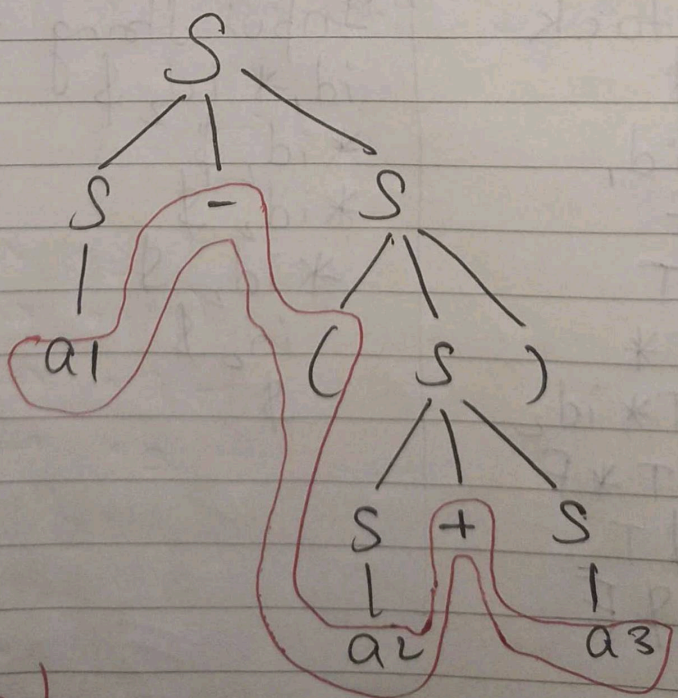
⇒

Stack	Input String	Action
$	$id_1 * id_2$ $	Shift id_1
$ id_1	$* id_2$ $	Replace by $F \rightarrow id$
$ F	$* id_2$ $	Replace by $T \rightarrow F$
$ T	$* id_2$ $	Shift *
$ T *	id_2 $	Shift id_2
$ T * id_2	$ $	Replace by $F \rightarrow id$
$ T * F	$ $	Replace $T \rightarrow T*F$
$ T	$ $	Replace $E \rightarrow T$
$ E	$ $	Accept String.

Q. $S \to S+S / S-S$
$S \to (S) / a$ String $\to a_1 - (a_2 + a_3)$

⇒

Stack	Input String	Action
$	$a_1-(a_2+a_3)$$	Shift a_1
$a_1	$-(a_2+a_3)$$	Replace $S \to a_1$
$S	$-(a_2+a_3)$$	Shift -
$S-	$(a_2+a_3)$$	Shift (
$S-(	$a_2+a_3)$$	Shift a_2
$S-(a_2	$+a_3)$$	Replace $S \to a_2$
$S-(S	$+a_3)$$	Shift +
$S-(S+	$a_3)$$	Shift a_3
$S-(S+a_3	$)$$	Replace $S \to a_3$
$S-(S+S	$)$$	Replace $S \to S+S$
$S-(S	$)$$	Shift)
$S-(S)	$$	Replace $S \to (S)$
$S-S	$$	Replace $S \to S-S$
$S	$$	Accept string.



$a_1-(a_2+a_3)$

LR(0) Parsing

Left to right — Rightmost Symbol — 0 lookahead

Q Create LR(0) Parsing table for

$E \rightarrow T + E / T$

$T \rightarrow id$

⇒ Step 1 → Augment starting Symbol.

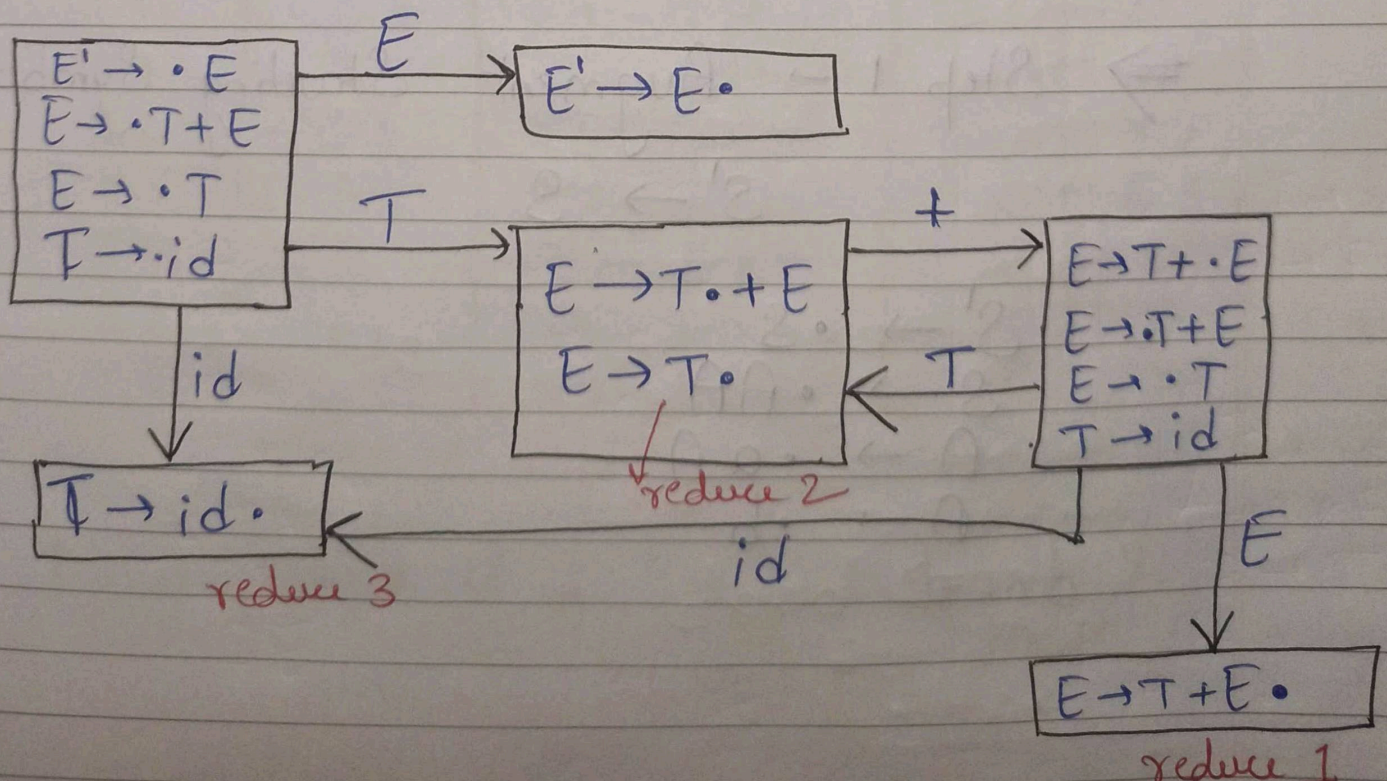
$S \rightsquigarrow E' \rightarrow E$

$E' \rightarrow \cdot E$

$E \rightarrow \cdot T + E$

$E \rightarrow \cdot T$

$T \rightarrow \cdot id$



States	Actions			Go to	
	+	id	$	E	T
I0		S3		1	2
I1			accept		
I2	S4/r2	r2	r2		
I3	r3	r3	r3		
I4		S3		5	2
I5	r1	r1	r1		

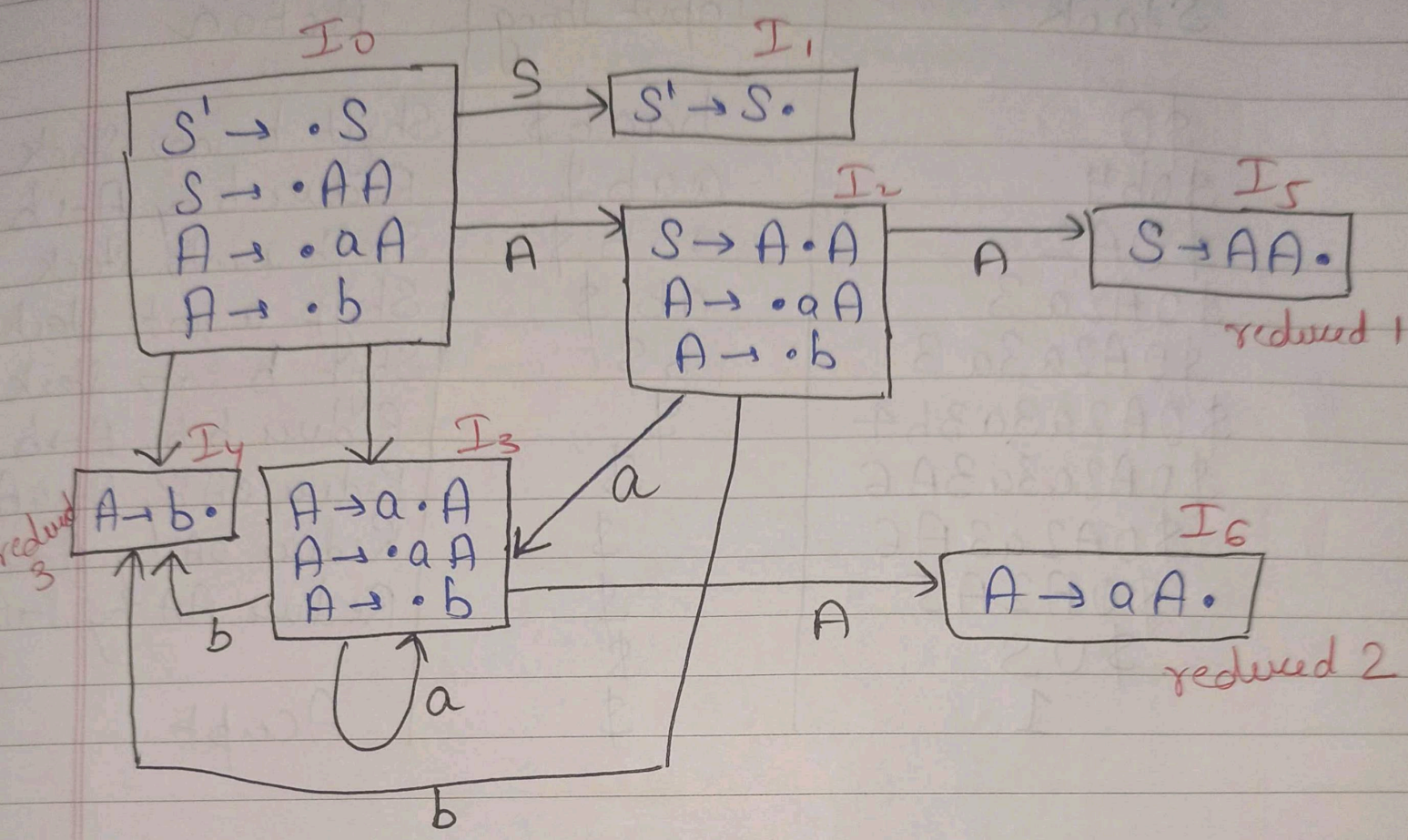
For terminals only, we use
1) Shift (S)
2) Reduce (r)

Q. $S \rightarrow AA$
$A \rightarrow aA \mid b$

⇒ Step 1 → Augment Starting Symbol

$S' \rightarrow S$

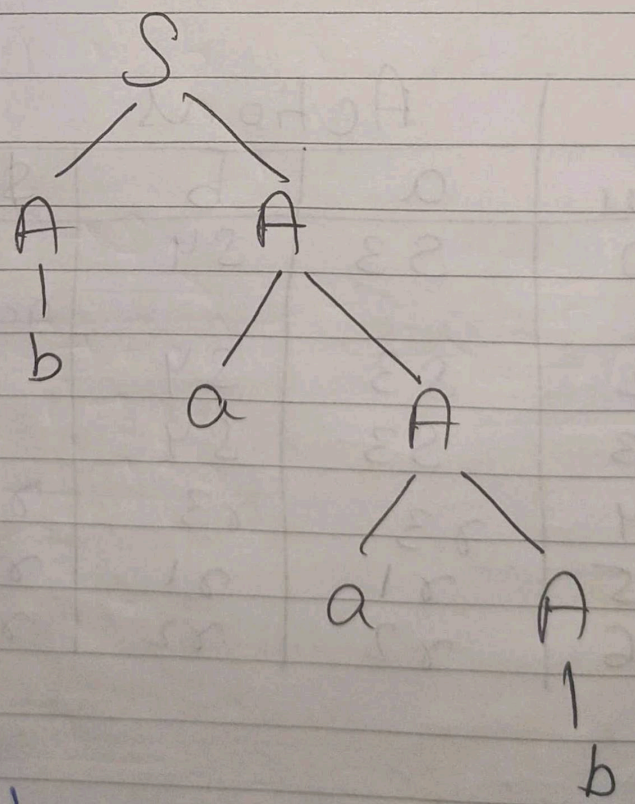
$S' \rightarrow \cdot S$
$S \rightarrow \cdot AA$
$A \rightarrow \cdot aA$
$A \rightarrow \cdot b$



States	Actions			S	A
	a	b	$		
I0	S3	S4		1	2
I1			accept		
I2	S3	S4			5
I3	S3	S4			6
I4	r3	r3	r3		
I5	r1	r1	r1		
I6	r2	r2	r2		

Now, check if it accept baab ?

Stack	Input String	Action
$0	baab$	Shift b into Stack
$0b4	aab$	Reduce b by A→b
$0A2	aab$	Shift a into Stack
$0A2a3	ab$	Shift a into stack
$0A2a3a3	b$	Shift b into Stack
$0A2a3a3b4	$	Reduce b by A→b
$0A2a3a3A6	$	Reduce aA by A→aA
$0A2a3A6	$	Reduce aA by A→aA
$0A2A5	$	Reduce AA by S→AA
$0S	$	
1	$	Acept



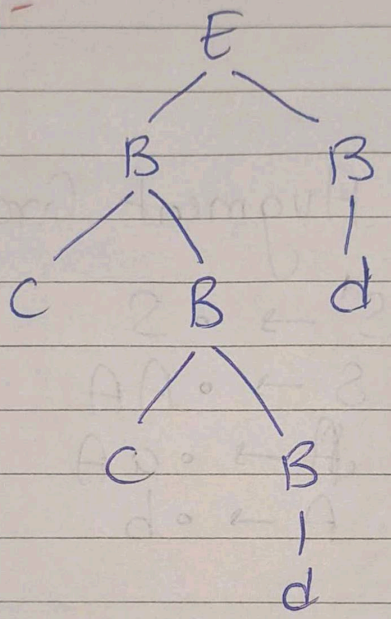
⇒ baab

Question to Practise :-

1) $E \rightarrow BB$
$B \rightarrow cB/d$
Generate string ccdd.

↳ It is similar to last quest.

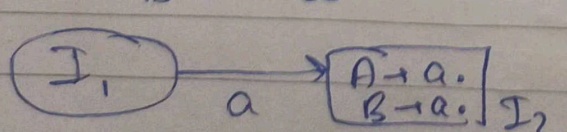
Ans to match :-



Note → While using LR(0), there are 2 types of Conflict that may arise.

1) RR Conflict

$A \rightarrow .a$
$B \rightarrow .a$

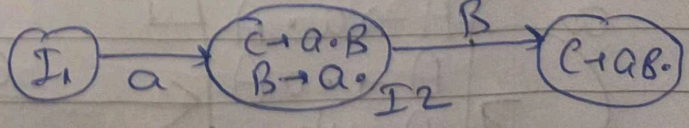


	a	A	B
I_1	S2		
I_2	r1/r2		

↑ 2 reduce (RR)

2) SR Conflict

$C \rightarrow .aB$
$B \rightarrow .a$



	a	B
I_1	S2	
I_2	S3/r2	S3

↳ Shift Reduce (SR)

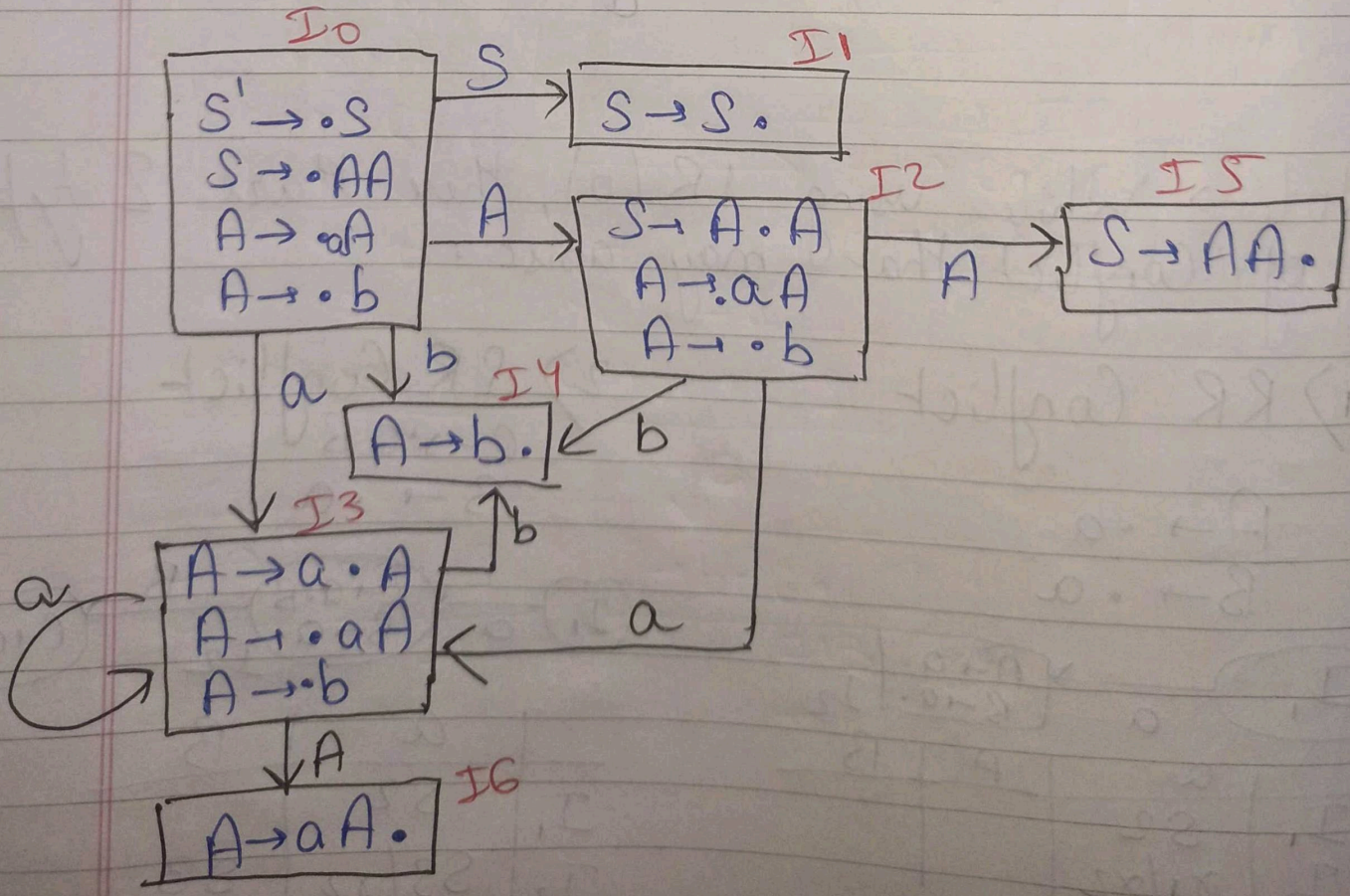
To overcome these conflicts, we use SLR parser (Simple LR Parser).

SLR(1) Parser

Q. $S \to AA$
$A \to aA / b$

⇒ Step 1 → Augment Grammar.

$S' \to \cdot S$
$S \to \cdot AA$
$A \to \cdot aA$
$A \to \cdot b$



Step 2 → Calculate follow of S and A.

FOLLOW(S) = {$}
FOLLOW(A) = {a, b, $}

	Action			Goto	
Stage	a	b	$	S	A
I0	S3	S4		1	2
I1			acept		
I2	S3	S4			5
I3	S3	S4			6
I4	r3	r3	r3		
I5			r1		
I6	r2	r2	r2		

Q. E → T + E / T
T → id

⇒ Follow(E) = {$}
Follow(T) = {+, $}

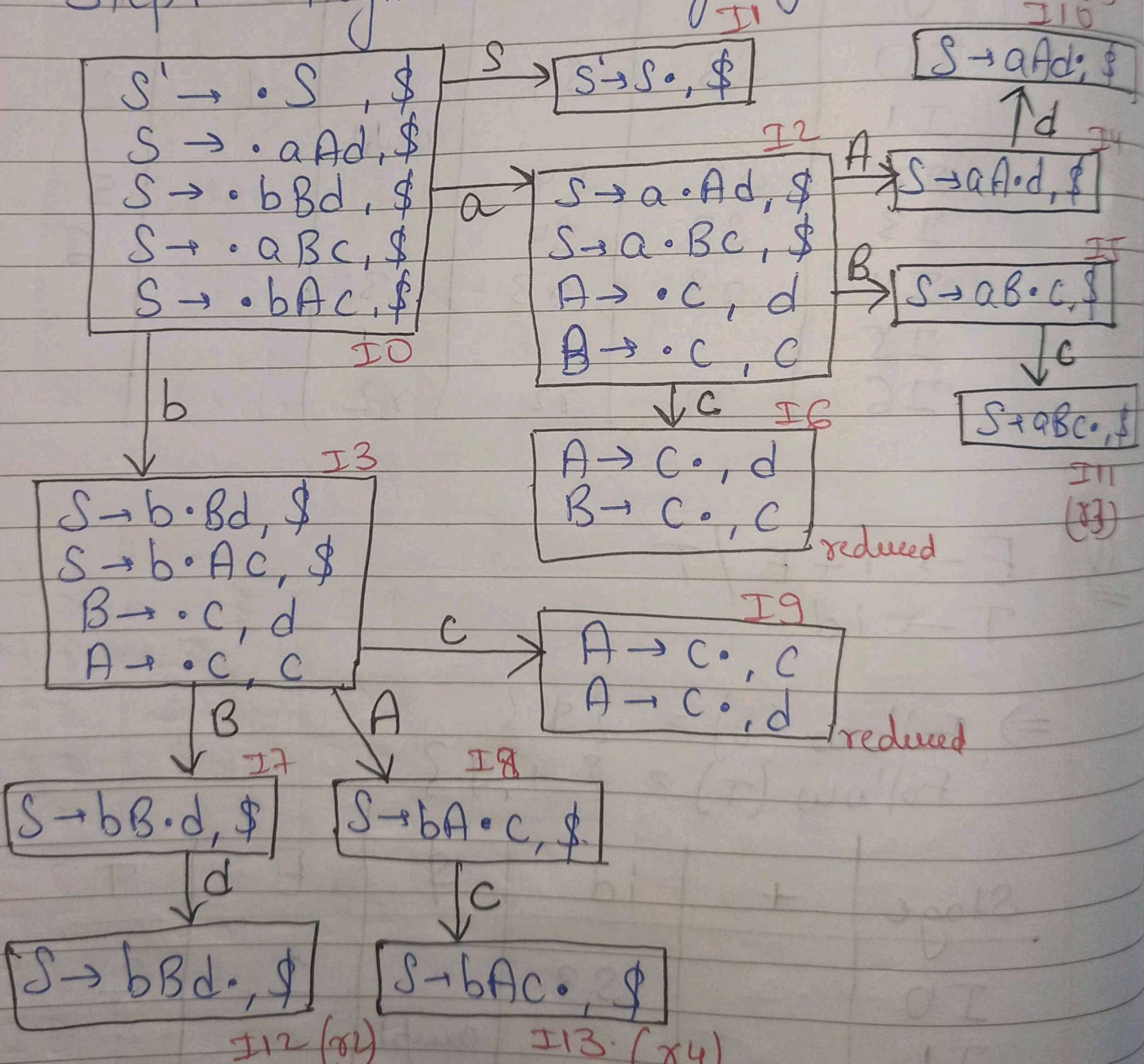
Stage	+	id	$	E	T
I0		S3		1	2
I1			acept		
I2	S4		r2		
I3	r3		r3		
I4		S3		5	2
I5			r1		

no conflict this time using SLR.

Canonical Left Right Parser (CLR(1)) :-

Q $S \to aAd / bBd / aBc / bAc$
$A \to c$
$B \to c$

⇒ Step 1 → Augment starting Symbol.



Stage	a	b	c	d	$	S	A	B
I0	S2	S3				1		
I1					accept			
I2			S6				4	5
I3			S9				8	7
I4				S10				
I5			S11					
I6			r6	r5				
I7				S12				
I8			S13					
I9			r5	r6				
I10				r1				
I11				r3				
I12				r2				
I13				r4				

Look-ahead left-Right (LALR)

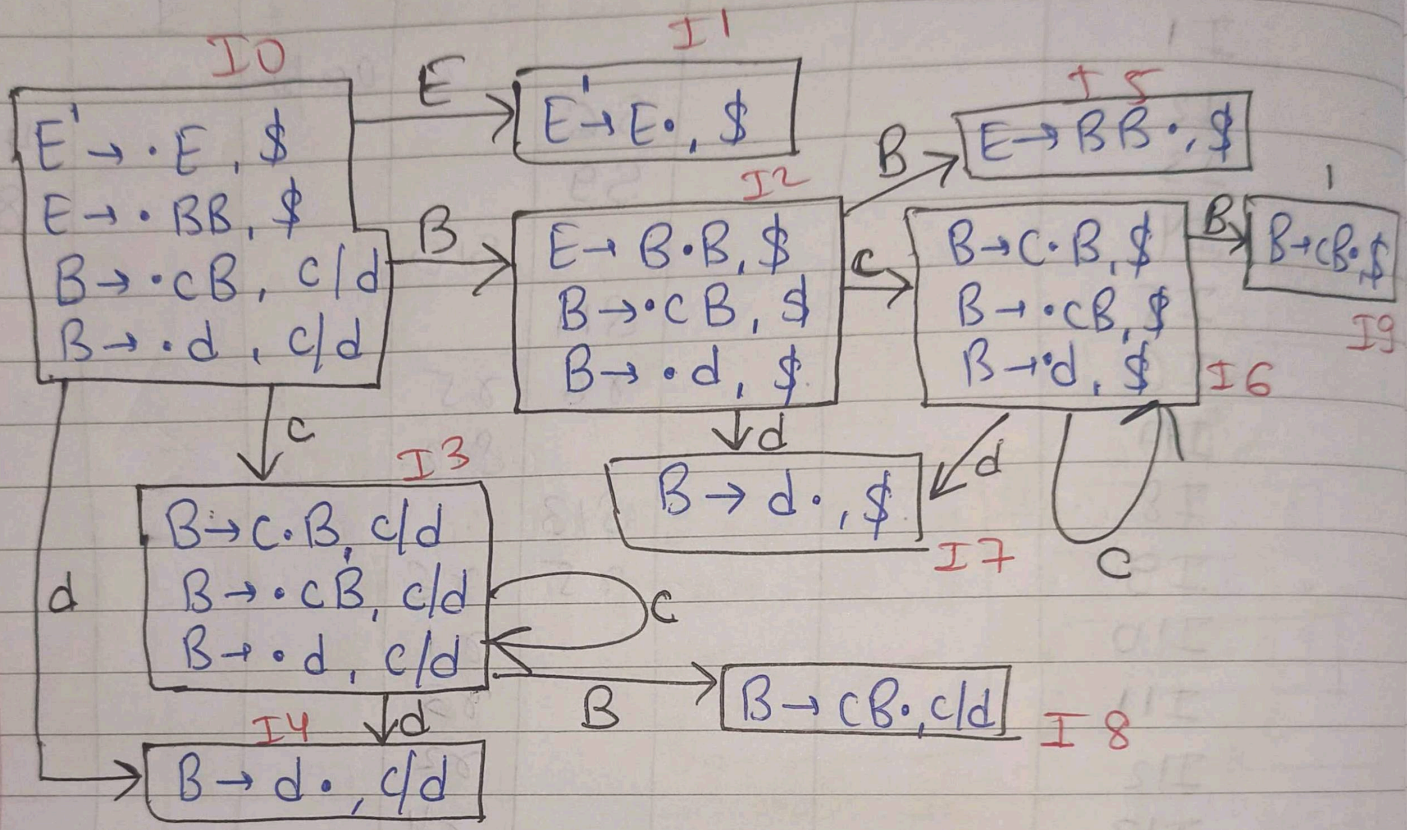
⇒ Similar to CLR But, we merge the production on same reduction. Like in this question, I6 and I9 is reducing same production. So, we can merge it.

	a	b	c	d	$	S	A	B
S1 → I6+I9			r6,r5	r5,r6				
I6 →			r6	r5				
I9 →			r5	r6				

2 production is one block

So, it is not LALR Parser. It is only CLR Parser

Q. $E \rightarrow BB$
$B \rightarrow cB \mid d$



	Action			Goto	
Stages	c	d	$	E	B
I0	S3	S4		1	2
I1					
I2	S6	S7			5
I3	S3	S4			8
I4	r3	r3			
I5			r1		
I6	S6	S7			9
I7			r3		
I8	r2	r2			
I9			r2		

For LALR :-

I3 & I6 → I36
I4 & I7 → I47
I8 & I9 → I89

	c	d	$	E	B
I0	S36	S47		1	2
I1			accpt		
I2	S36	S47			5
I3	S36	S47			89
I4	r3	r3	r3		
I5			r1		
I6	S36	S47			89
I7	r3	r3	r3		
I8	r2	r2	r2		
I9	r2	r2	r2		

Copy I3 → I6
Copy I4 → I7
Copy I8 → I9

Now, I3&I6 having same values so we can ignore 1 of them. Same we can do with (I4&I7) & (I8&I9).

	c	d	$	E	B
I0	S36	S47		1	2
I1			accept		
I2	S36	S47			5
I36	S36	S47			89
I47	r3	r3	r3		
I5			r1		
I89	r2	r2	r2		

⇒ Both CLR(1) & LALR

Question to Practise :-

Q. Check if the following production is
(a) CLR or not
(b) LALR or not.

Ans to verify — (a) Yes,
(b) Yes

⇒ Isme dono table me difference v nhi ayega because, there is no 2 state which is proving same reduction.

Operator Precedence Parser :-

↳ Based on operator precedence grammar.
Rules :-

1. It should not have ϵ in R.H.S of production
$S \rightarrow \epsilon$ ✗
2. Two Non-terminal should not be adjacent
$E \rightarrow AC$ ✗
$E \rightarrow aC$ ✓
$E \rightarrow Ca$ ✓

Q. $E \rightarrow EAE \mid (E) \mid id$
$A \rightarrow + \mid - \mid * \mid /$

⇓

$E \rightarrow E+E \mid E-E \mid E*E \mid E/E \mid (E) \mid id$

Precedence table Rules :-

1. id, a, b, c = highest precedence
   (→ terminals)

2. (+, +)
   (*, *)
   (/, /)
   (-, -)
   (same operator) → Left side has more precedence.

3. $ → lowest precedence

4. $$ → accept

→ Right

↓ left	+	*	(	)	id	$
+	>	<	<	>	<	>
*	>	>	<	>	<	>
(	>	>	<	=	<	X
)	<	<	X	>	X	>
id	>	>	X	>	X	>
$	<	<	<	X	<	accept

Q. E → EAE / id
   A → + / *

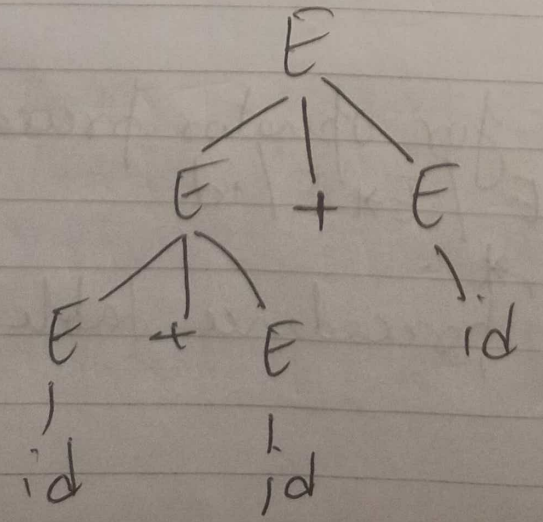
⇒ Step 1 → Check for operator precedence grammar
   E → E+E / E*E / id
   A → +, *.
Step 2 → Draw precedence table

	id	+	*	$
id	≠	>	>	>
+	<	>	<	>
*	<	>	>	>
$	<	<	<	Accept

Input String → id + id + id.
⇒ $id + id + id $.

Stack	Relation	String	Action
		$id + id + id $	Shift $
$.	<	id + id + id $	Shift id
$ id	>	id + id + id $	Reduce by E→id
$ E	<	+ id + id $	Shift +
$ E +	<	id + id $	Shift id
$ E + id	>	+ id $	Reduce E→id
$ E + E	>	+ id $	Reduce E→E+E
$ E	<	+ id $	Shift +
$ E +	<	id $	Shift id
$ E + id	>	$.	Reduce E→id
$ E + E	>	$	Reduce E→E+E
$ E		$.	Accept

⇒



Q for Practise :-

$E \rightarrow EAE / id$

$A \rightarrow * / -$

Generate Parse tree for id - id * id.

Computation of Leading & Trailing :-

Leading :-

Rule 1 :- $A = \alpha a \gamma \rightarrow$ anything

(α : ϵ or single non-terminal; a : terminal)

$Leading(A) = a$

Rule 2 :- $A = B\alpha \rightarrow$ anything

(B : single Non-terminal)

$Leading(A) = Leading(B)$

Trailing :-

Rule 1 :- $A = \alpha a \gamma \rightarrow$ anything

(a : terminal; α : ϵ or single N-T)

$Trailing(A) = \{a\}$

Rule 2 :- $A = \alpha B a \rightarrow$ anything

(B : single N.T)

$trailing(A) = trailing\{B\}$

Q. $E \to E+T$ ① | T ②
$T \to T*F$ ③ | F ④
$F \to (E)$ ⑤ | id ⑥

⇒ 1. $E \to E+T$

Leading (E) = {+}, Leading (E)

Trailing (E) = {+}, trailing (T).

How?

Leading :-

Rule 1 → $E \to \underbrace{E}_{\alpha} \underbrace{+}_{a} \underbrace{T}_{\gamma}$

Leading (E) = {+}

Rule 2 → $E \to \underbrace{E}_{B} \underbrace{+ T}_{\alpha}$

Leading (E) = Leading {E}

Trailing :-

Rule 1 → $E \to \underbrace{E}_{\alpha} \underbrace{+}_{a} \underbrace{T}_{\gamma}$

trailing (E) = {+}

Rule 2 → $E \to \underbrace{E +}_{\alpha} \underbrace{T}_{B}$

trailing (E) = trailing (T)

2. $E \to T$

Leading (E) = Leading (T)

Trailing (E) = Trailing (T).

Rule 1 not applicable as $E \to T$ does not in form $\alpha a \gamma$

↑ Terminal compulsory

Similarly, $T \rightarrow T*F$

$\Rightarrow$ Leading $(T) = \{*\}$, Leading (F)

$\Rightarrow$ Trailing $(T) = \{*\}$, Trailing (F)

$T \rightarrow F$

$\Rightarrow$ Leading (T) = Leading (F)

Trailing (T) = Trailing (F).

$F \rightarrow (E)$

$\Rightarrow$ Leading $(F) = \{(\}$

Trailing $(F) = \{)\}$.

$F \rightarrow id$

$\Rightarrow$ Leading $(F) = \{id\}$

Trailing $(F) = \{id\}$

Operator Precedence Relation Table Algo :-

1. Calculate Leading & Trailing for each N.T.
2. (a) $S \rightarrow ab$

$a = b$

(b) $S \rightarrow aAb$

(i) $a = b$

(ii) $a <$ Leading (A)

(iii) Trailing $(A) > b$

(c) If S = Starting N.T., $\$ \rightarrow$ end terminal

(i) $\$ <$ Leading (S)

(ii) Trailing $(S) > \$$

By using that let's find Table of above question

N.T	Leading	Trailing
E	{+, *, id, (}	{+, *, id,)}
T	{*, id, (}	{*, id,)}
F	{id, (}	{id,)}

	+	*	(	)	id	$
+	>	<	<	>	<	>
*	>	>	<	>	<	>
(	<	<	<	=	<	X
)	>	>	X	>	X	>
id	>	>	X	>	X	>
$	<	<	<	X	<	accept

$E \to E+T$

↳ $E \to E+$ & $E \to +T$

form of $S \to Ab$ | form of $S \to aA$

Trailing $(E) > +$ | $+ < $ Leading (T)

$\{+, *, id,)\} > +$ | $+ < \{*, id, (\}$

(i) | (ii)

$E \to T \Rightarrow$ ignored

3. $T \rightarrow T * F$

↳ $T \rightarrow T *$

$\Downarrow$

Trailing $(T) > *$

$\{*, id,)\} > *$ — (iii)

$T \rightarrow * F$

$* <$ Leading (F)

$* < \{ id, (\}$ — (iv)

4. $T \rightarrow F \Rightarrow$ ignored

5. $F \rightarrow (E)$.

↳ $(=)$ — (v)

↳ $(<$ Leading (E)

$\Rightarrow (< \{+, *, id, (\}$ — (vi)

↳ Trailing $(E) >)$

$\Rightarrow \{+, *, id,)\} >)$ — (vii)

6. $F \rightarrow id \Rightarrow$ ignored

$\$ < \{+, *, id, (\}$ — (viii)

$\{+, id, id,)\} > \$$ — (ix)

Now, just fill table using these 9 equation.