Q. Which one of the following is Top down parser?

(a). Recursive Descent parser
(b). Operator ~~pe~~ precedence parser
(c). LR(K)
(d). LALR(K).

BOTTOM - UP PARSER

In bottom up parsing we attempt to construct a parse tree starting from leaf nodes and we work towards root node.
The primary objective of bottom up parsing is reducing the given string to start symbol.
In each step of the parsing, we identify the substring in a w that matches with right side string of some production (called Handle) & whenever we encounter a handle, we replace it by left side non-terminal of the production, this process is called reduction.
A bottom up parser is also called as shift reduce parser (SRP) in which the parser peform 4 actions namely shift, reduce, accept & error.
A shift reduce parser can be viewed as Right most derivation in reverse order.

eg. $E \to E+E \mid E*E \mid id$.

$w = id + id * id$

Handle

$w = \underline{id} + id * id$

$= E + \underline{id} * id$

Handle

$\Rightarrow \underline{E + E} * id$

$\Rightarrow E * \underline{id}$

$\Rightarrow E * E$

$\Rightarrow E.$

order of handle

id, id, E+E, id, E*E.

Q. consider the following grammar G,

$S \to aABe$

$A \to Abc \mid b$

$B \to d$.

find the order of handle for string $w = abbcde$

$w = a\underline{b}bcde$

$= a\underline{Abc}de$

$= aA\underline{d}e$

$= \underline{aABe}$

$\Rightarrow S.$

Order :- b, Abc, d, aABe

Q. Consider the grammar

$E \to E+n \mid E*n \mid n$

for a sentence $(n+n*n)$, the handles in the right sentential form of the reduction are :-

(1). n, $E+n$ and $E+n*n$

(2). n, $E+n$, $E+E*n$

(3). n, $n+n$, $n \times n$

(4). n, $E+n$, $E \times n$

⇒. $w = \underline{n}+n*n$

$\underline{E+n}*n$

$\underline{E*n}$

E.

Q. Which of the following describes a handle appropriately :- ?
GATE 2008

(a). It is the position in a sentential form where next shift or reduced operation will occur.

(b). It is a non-terminal whose production will be used for reduction in the next step.

(c). It is production that may be used for reduction in a future step along with a position in a sentential form where next shift reduce operation will take place.

(d). It is the production p_1 that will be used for reduction in the next step along with a position in a sentential form where a right hand side of the production may be found.

OPERATOR PRECEDENCE PARSING.

An operator precedency parsing technique uses operator precedency matrix of the ~~conte~~ given context free grammar.

To construct a precedency matrix for a given G, Grammar should be operator precedency grammar

Every operator precedency grammar is basically operator grammar with some conditions :-

① operator Grammar :- A Grammar G is said to be operator grammar if every production in P do not contains 2 non-terminals side by side.

eg. $E \rightarrow EAE \mid a$

$A \rightarrow + \mid - \mid *$.

↳ Not operator grammar

↳ New grammar equivalent to above.

$E \rightarrow E+E \mid E-E \mid E*E \mid a \mid (E) \mid \varepsilon$ → ε also creates no problem

⇓

Operator Grammar.

Operator precedency Grammar :- In operator precedency grammar, the following condition holds-

cond! ① Between any 2 terminals it must hold, at most one precedency relation as shown in the table.

Relation	Meaning
$a <\cdot b$	a yields precedence to b
$a \doteq b$	a has same precedence as b.
$a \cdot> b$	a takes precedence over b.

cond2. A grammar G should be operator grammar and free from ε-production.

CONSTRUCTION OF PRECEDENCY MATRIX:

(1) Let θ_1 and θ_2 are 2 operators, and they have equal precedence make $\theta_1 \cdot> \theta_2$ & $\theta_2 \cdot> \theta_1$ if they are left associative otherwise make. $\theta_1 <\cdot \theta_2$ and $\theta_2 <\cdot \theta_1$ if they are Right associative.

(2) If θ_1 has higher precedence than θ_2 then make $\theta_1 \cdot> \theta_2$ & $\theta_2 <\cdot \theta_1$

(3) Also make $\theta \cdot> \$$ $\$ <\cdot \theta$ $id \cdot> \theta$ $\theta <\cdot id$ $id \cdot> \$$ $\$ <\cdot id$.

Q. Construct a operator precedency matrix for the following grammar -

$E \rightarrow E+E \mid E-E \mid E*E \mid id$

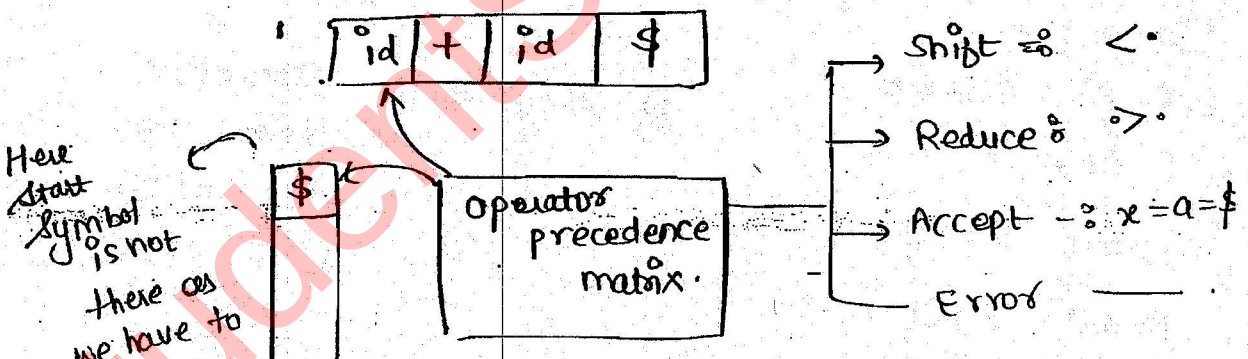
Q_1 \ Q_2	+	−	*	id	$
+	·>	·>	<·	<·	·>
−	·>	·>	<·	<·	·>
*	·>	·>	·>	<·	·>
id	·>	·>	·>	ERR	·>
$	<·	<·	<·	<·	Accept

id id ko compare nai kar sakte not operator grammar ther

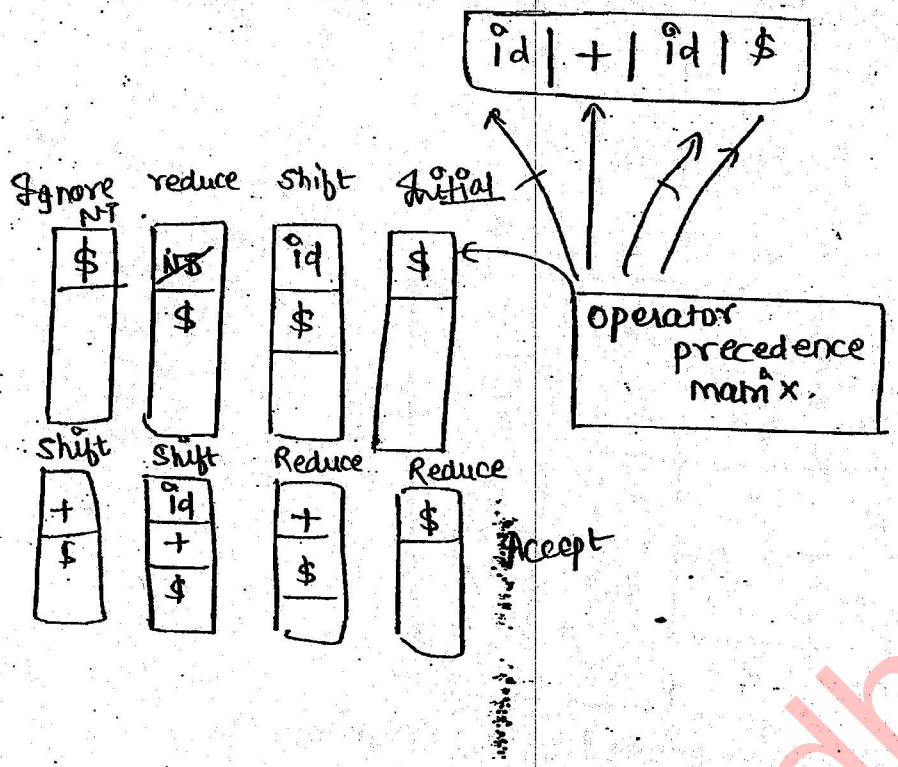
Operator precedence matrix.

parse the string $w = id + id$ string operator precedence parsing Techniques -

Perform 4 function -



Initial configuration of Bottom up parser



Q. Which of the following grammar rules violates the requirements of an operator grammar - ?

(i). $P \to QR$

(ii). $Q \to QSR$

(iii) $P \to \varepsilon$

(iv). $P \to QtRt$

(a). 1 only

(b). 1 and 3 only

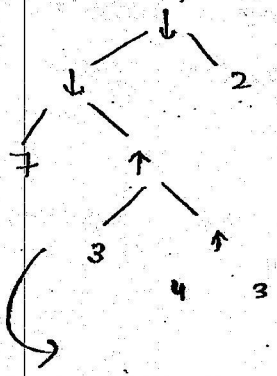
(c). 2 and 3 only

(d). 3 and 4 only.

If they ask operator precedency, then (1) and (3) as ε creates problem there.

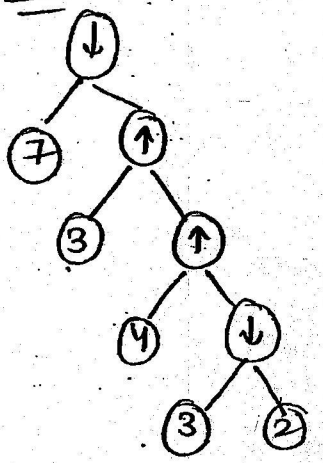
Q. Consider the 2 binary operators $\uparrow$ & $\downarrow$ with the precedency operator $\downarrow$ being lower then that of $\uparrow$. [$\downarrow < \uparrow$]. operator $\uparrow$ is right associative while $\downarrow$ is left associative. Which of the following parse tree represents the

Expression $((7 \downarrow (3 \uparrow (4) \uparrow 3)) \downarrow 2)$.

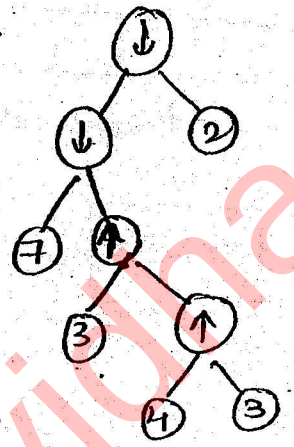


Options.

(a).



(b).



LR parsing :-

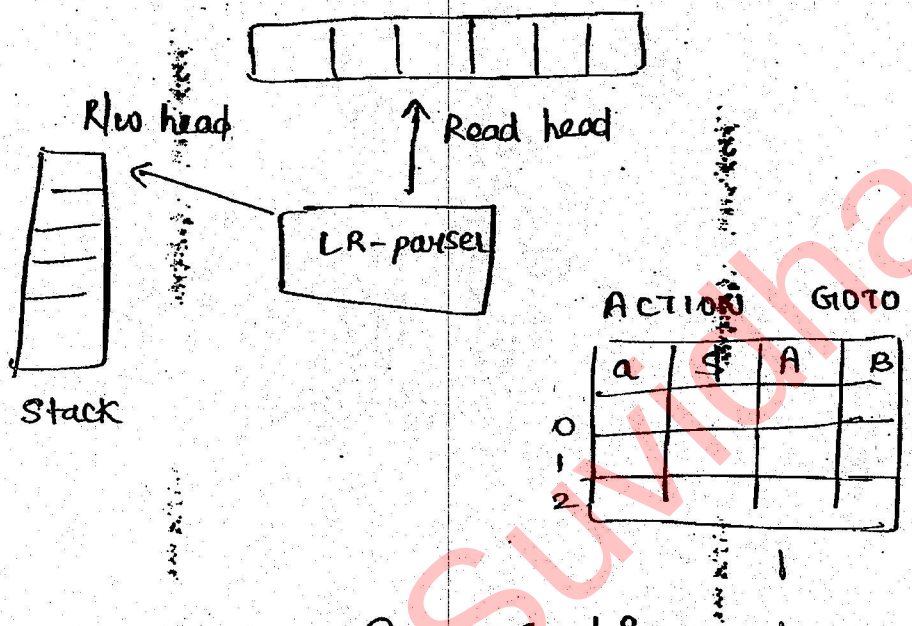
LR stands for left to Right scan of the input and will produce a derivation in right most in reverse order.

Among all parsing methods LR is powerful techniques because it can be used to parse any arbitrary context free grammar. There are 3 types of LR techniques.

1. Simple LR (SLR)
2. Canonical LR (CLR)
3. Lookahead LR (LALR)

for all the LR parsing techniques parsing program is same. But the construction of LR parsing table procedures different from 1 technique to another technique.

The general model of LR parser as shown in the figure - :



SIMPLE LR

SIMPLE LR parsing Table construction - :

The construction of SLR parsing table depends on Canonical Sets of LR(0) items

LR(0) item - : LR(0) item is a context free grammar production with the dot operator anywhere on the rightside string.

for eg. $A \rightarrow XYZ$

induces 4 items

$A \rightarrow .XYZ$

(OR) $A \rightarrow X.YZ$

(OR) $A \rightarrow XY.Z$

(OR) $A \rightarrow XYZ.$

The production $A \to \varepsilon$ use only one LR(0) item i.e $A \to .$

A item is said to be complete one if it is of the form $A \to \alpha .$ or $A \to .$

Construction of canonical sets of LR(0) items depends on 3 functions :-

1. Augmented grammar
2. closure function.
3. Goto function.

* Augmented Grammar :- The augmented grammar G_1 for the given G_1 consists of all production in the P of the G_1 with additional productio $S' \to S$ where S' is a new variable in G' of V^n then S is a start symbol of G_1.

Eg. $G_1 \Rightarrow$ $S \to AB$
$A \to a$
$B \to b$

then $G' = S' \to S$
$S \to AB$
$A \to a$
$B \to b$.

* closure function :- If $A \to \alpha . B\beta$ is in I_i and $B \to \gamma$ is in B production then add $B \to .\gamma$ to I_i.

Repeat the same if $B \to .\gamma$ follows same condition i.e . followed by immediate symb is variable.

* Goto – If $A \to \alpha \cdot X\beta$ is in I_i then GOTO of $GOTO(I_i, X) = A \to \alpha X \cdot \beta$ will be in I_j.

It may also calls closure function if ~~dotted~~ followed by immediate symbol is variable in $A \to \alpha X \cdot \beta$

* Closure is applicable only for dot followed by variable.
* goto is applicable for dot followed by variable, terminal etc.
* Complete items never further have a goto operations.

The construction of LR(0) sets of items begins with in general,

$S' \to \cdot S$ with set name I_0.

Q. Construct canonical sets of LR(0) items –

$S \to AB$

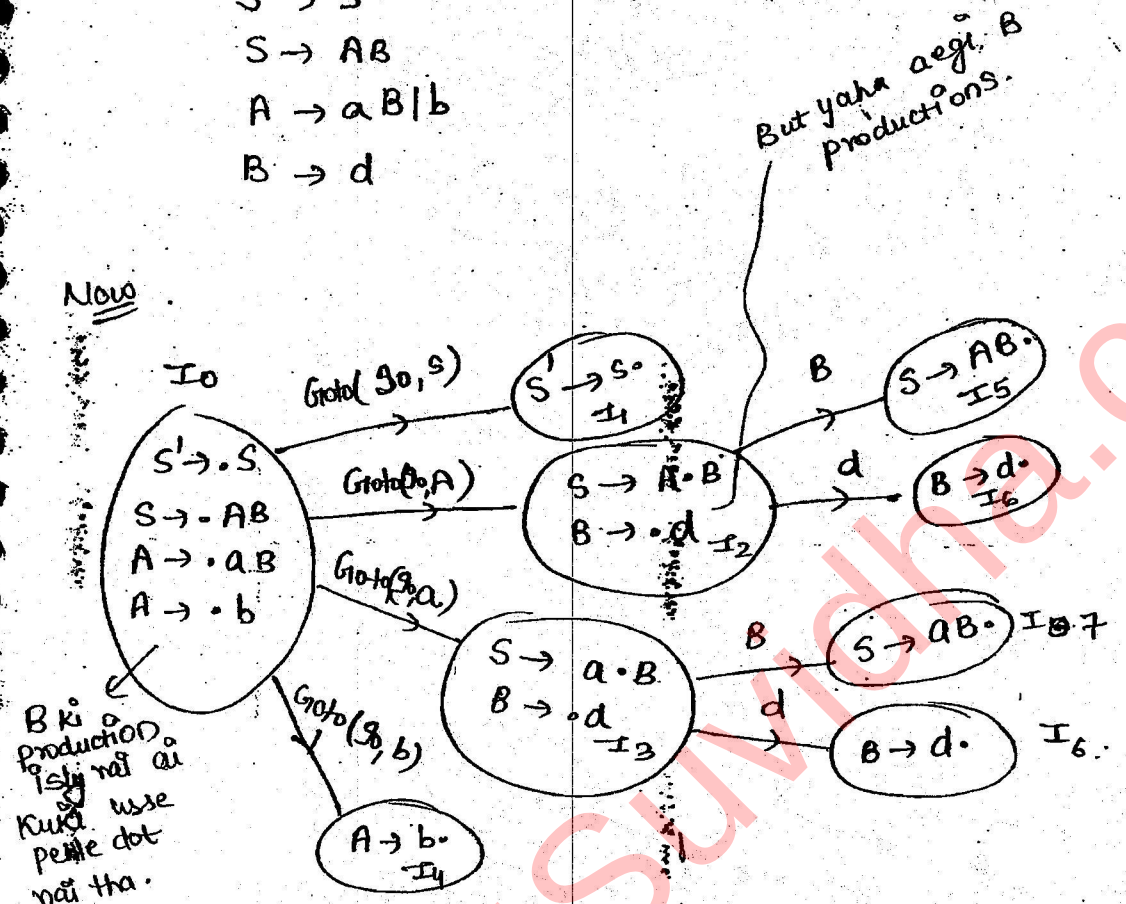
$A \to aB \mid b$

$B \to d$.

Augmented Grammar :-

$S' \rightarrow S$
$S \rightarrow AB$
$A \rightarrow aB \mid b$
$B \rightarrow d$

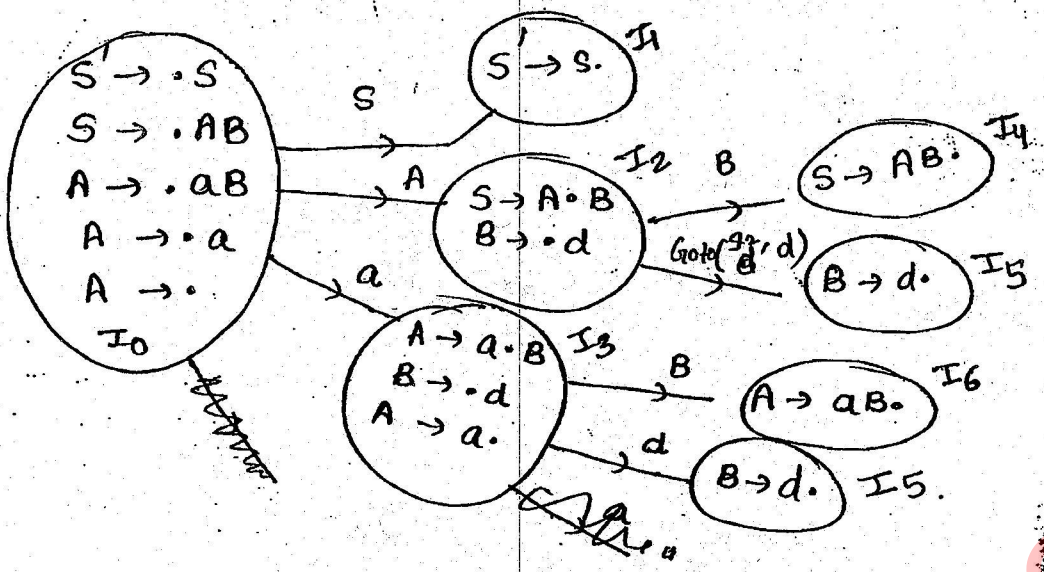
Now



Canonical ~~data~~ LR(0) sets of items = 1

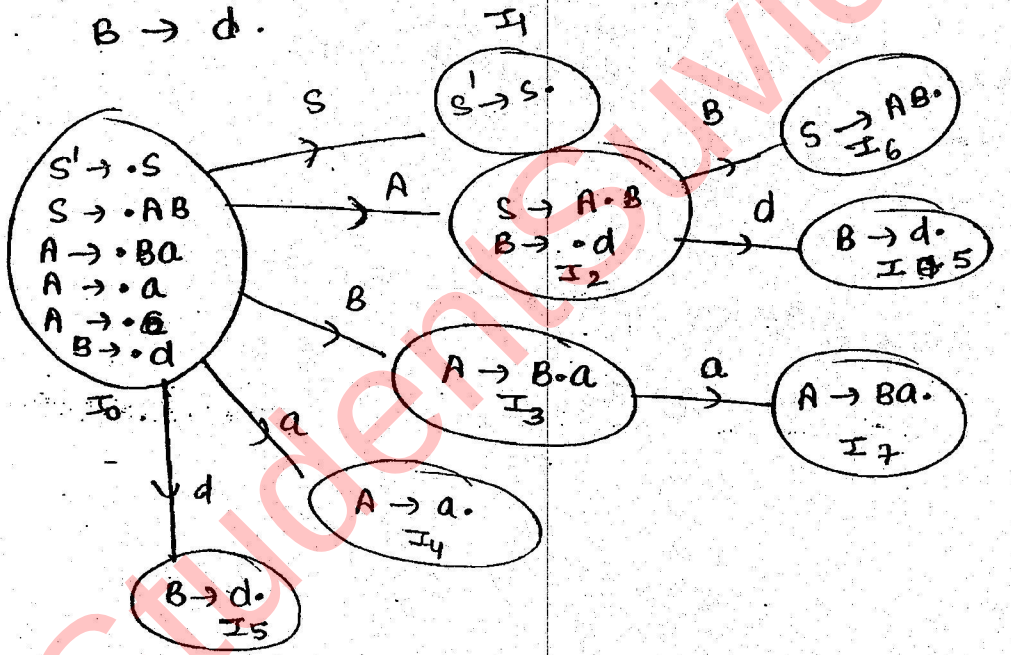
$\{ I_0, I_1 \cdots I_7 \}$

(2). $S \rightarrow AB$
$A \rightarrow aB \mid a \mid \epsilon$
$B \rightarrow d$.



$C = \{ I_0, I_1 - - - I_6 \}$

(3) $S \rightarrow AB$
$A \rightarrow Ba \mid a \mid \varepsilon$
$B \rightarrow d.$

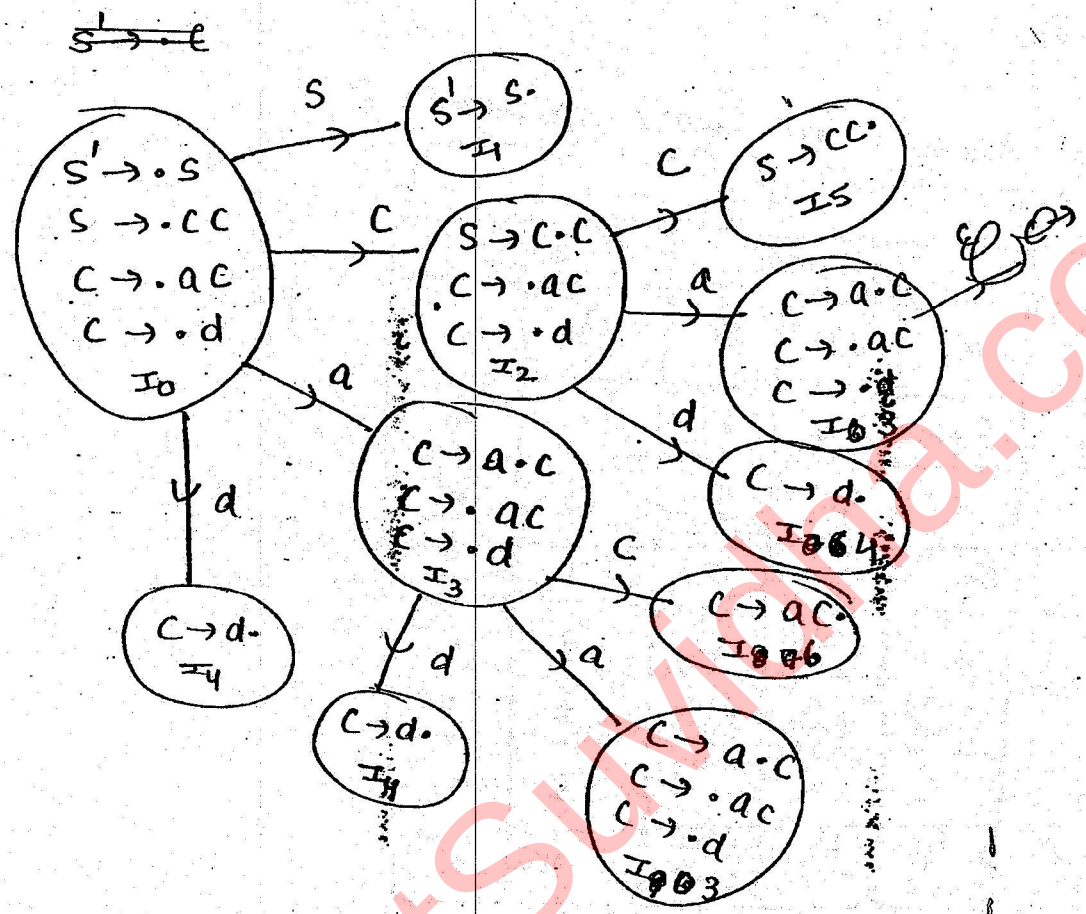


$C = \{ I_0, I_1 - - - - I_7 \}$.

④. $S \to CC$

$C \to aC \mid d$.

Goto graph.



$C = \{I_0 \text{ ---- } I_6\}$

* GOTO graphs are always DFA.

Construction of SLR parser Table :-

The SLR parsing table consists of 2 parts

① action part

② Goto part.

Rows of the SLR table corresponding to the no. of sets of GOTO graph of LR(0), i.e each set I_i represent the state i.

Action part columns are corresponding to the terminals of G1 and $ sign.

GOTO part columns are corresponding to non terminals of the G1.

eg. Draw table for above grammar :-

	ACTION			GOTO	
	c	d	$	S	C
0	S_3	S_4	E	1	2
1	E	E	$ ACCEPT	E	E
2	S_3	S_4	E	E	5
3	S_3	S_4	E	E	6
4	r_3 by C→d	r_3 by C→d	r_3 by C→d	E	E
5	E	E	r_1 by S→CC	E	E
6	r_2 by C→aC	r_2 by C→aC	r_2 by C→aC	E	E

To fill entries of action part of the Table apply following :-

Shift :- If $A \to \alpha . a\beta$ is in I_i then ACTION $[i, a]$ = Shift J.

where J is GOTO $(I_i, a) = I_j$.

Reduce :- If $A \to \alpha .$ is in I_i then ACTION $[i, b]$ = reduced by $A \to \alpha$ $\forall$ $b \in$ FOLLOW(A,

ACCEPT – If $S' \to S.$ is in I_i, then ACTION $[i, \$] =$ accept;

Error – Then make all left over entries as error.

To fill entries of GOTO part of the Table apply following –

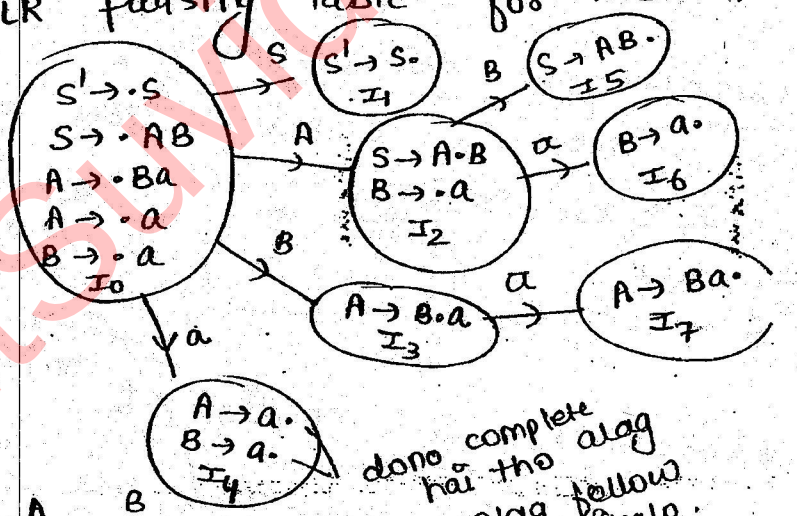
If $A \to \alpha . B\beta$ is I_i and GOTO $[I_i, B] = I_j$ then make GOTO $[i, B] = j$.

Q Construct a neat SLR parsing table for the grammar –

$S \to AB$

$A \to Ba \mid a$

$B \to a$



done complete alag hai the alag follow nikalo.

follow(A) = a

follow(B) = {a, $}

	a	$	S	A	B
I_0	S_4	ε	1	2	3
I_1	ε	Accept	ε	ε	ε
I_2	S_6	ε	ε	ε	5
I_3	S_4	ε	ε	ε	ε
I_4	r by A→a or B→a	r by B→a	ε	ε	ε
I_5	ε	ε	r by S→AB	ε	ε
I_6	r by B→a	r by B→a	ε	ε	ε
I_7	r by A→Ba	ε	ε	ε	ε

Since it contains multiple entries so not a SLR(1) grammar

* SLR(1) Grammar :-
A grammar G is said to be SLR(1) grammar if SLR parsing table do not contain multiple defined entries.

* LR(0) Grammar :-
A grammar G is said to be LR(0) if goto graph of the LR(0) sets of items do not contains the states with either R-R conflict or S-R conflict.

> Every LR(0) grammar is SLR(1).
> But vice versa is not true.

* Conditions for R-R conflict :-
When a set I_i contains items of the form.

$A \to \alpha .$
$B \to \beta .$

if $FL(A) \cap FL(B) = \phi$ then SLR(1)
O.w not SLR(1).

ye LR(0) mai ho tho bhi jruri nai hai ki SLR(1) mai hoge multiple entry hogi hi, it depends on follow.

* Conditions for S-R conflict :-
When a set I_i may have the items of the form

$A \to \alpha . a \beta$ → Shift
$B \to \beta .$ → Reduce

if $FO\{a\} \cap FL\{B\} = \phi$ then SLR(1)
O.w not SLR(1).

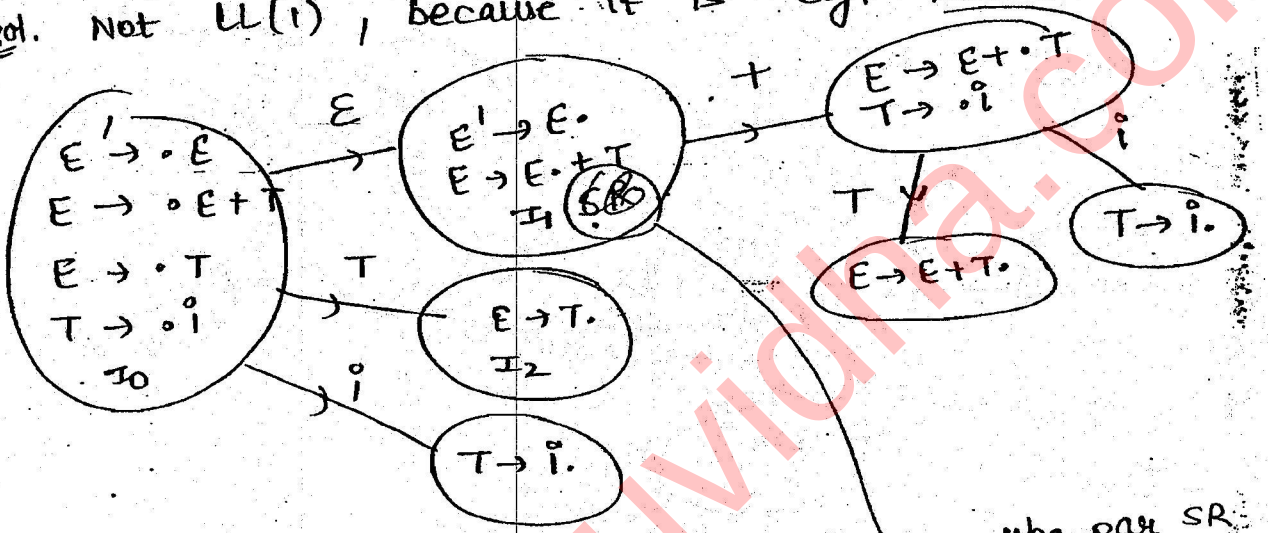
↳ But due to this not LR(0).

Shift-Shift conflict will never come. (as we always shift on one).

Q. Check whether the following grammar is LL(1) ? LR(0) ? SLR(1) ?

(1) $E \to E+T \mid T$
$T \to i$.

Sol. Not LL(1), because it is Left Recursive.



~~So not LR(0).~~

~~Now,~~ ~~of~~ ~~E(R0)~~ ~~F(G)~~ ~~#~~ ~~F(R5)~~.

yha par SR conflict nai hai kuki E' hmari given gramma ka part nai hai.

Apart from the ~~R0~~ E', or kisi K beech aisa hoga tho definitely SR conflict is there.

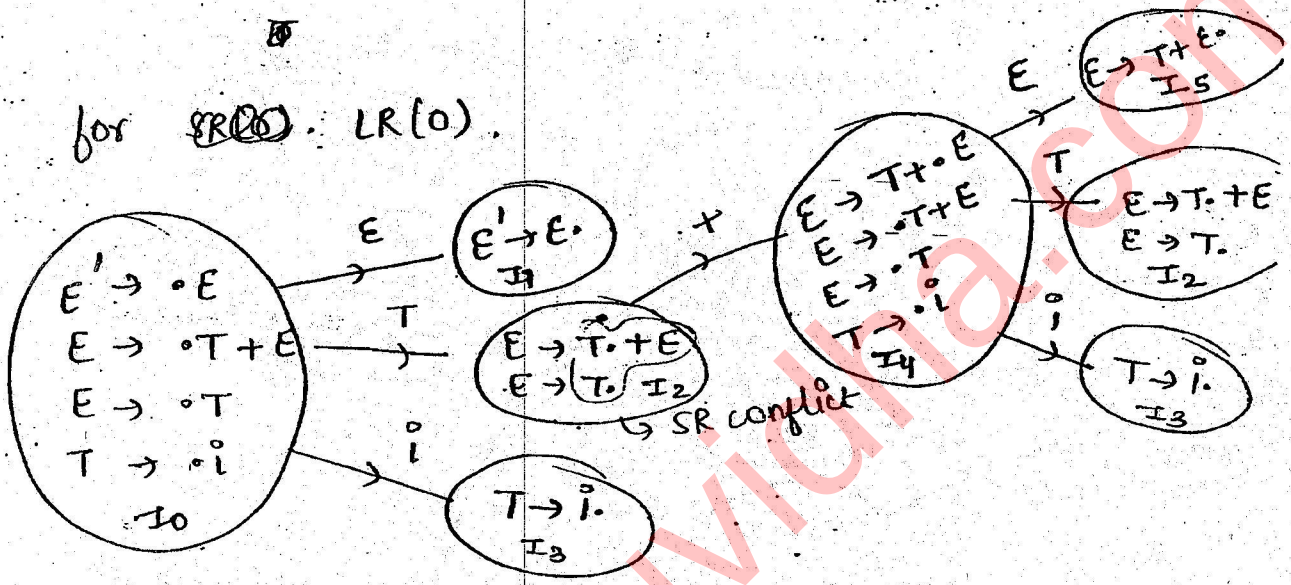
So ~~LR(0)~~ LR(0)
↳ then definitely SLR(1).

② $E \to T+E \mid T$

$T \to i$.

Sol not LL(1) as there is common prefix. (Left factored).

for SLR(1) : LR(0).



So, not LR(0)

but because,

follow(E) = $

FIRST(+) ∩ FOLLOW(E) = ϕ

+ ∩ $ = ϕ

So, SLR(1).

③ $S \to AB$

$A \to Ba \mid a$
$\alpha_1 \quad \alpha_2$

$B \to a$

⇒ LL(1).

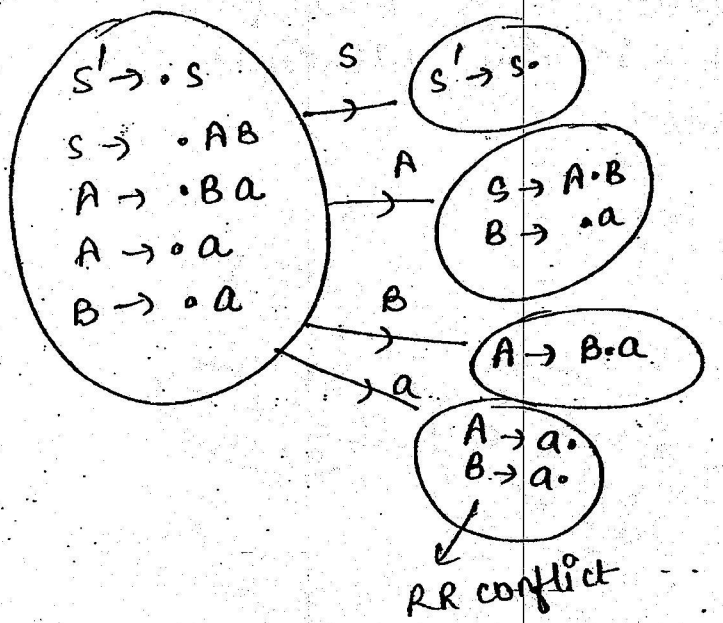
for LL(1)

↳ FT(Ba) ∩ FT(a) = ϕ.

a ∩ a ≠ ϕ

so not LL(1)

X	first(X)	follow(X)
S	a.	—
A	a	—
B	a	—



So not LR(0).

Check for SLR(1).

follow(A) ∩ follow(B) = {a} ∩ {$, a}

$\neq \phi$

So not SLR(1) also

So, not LL(1)
not LR(0)
not SLR(1).

Q 2006

(4). Consider the following LR(0) items for the gramm above.

$S \rightarrow S \times E$
$S \rightarrow E$
$E \rightarrow F + E$
$E \rightarrow F$
$F \rightarrow id$

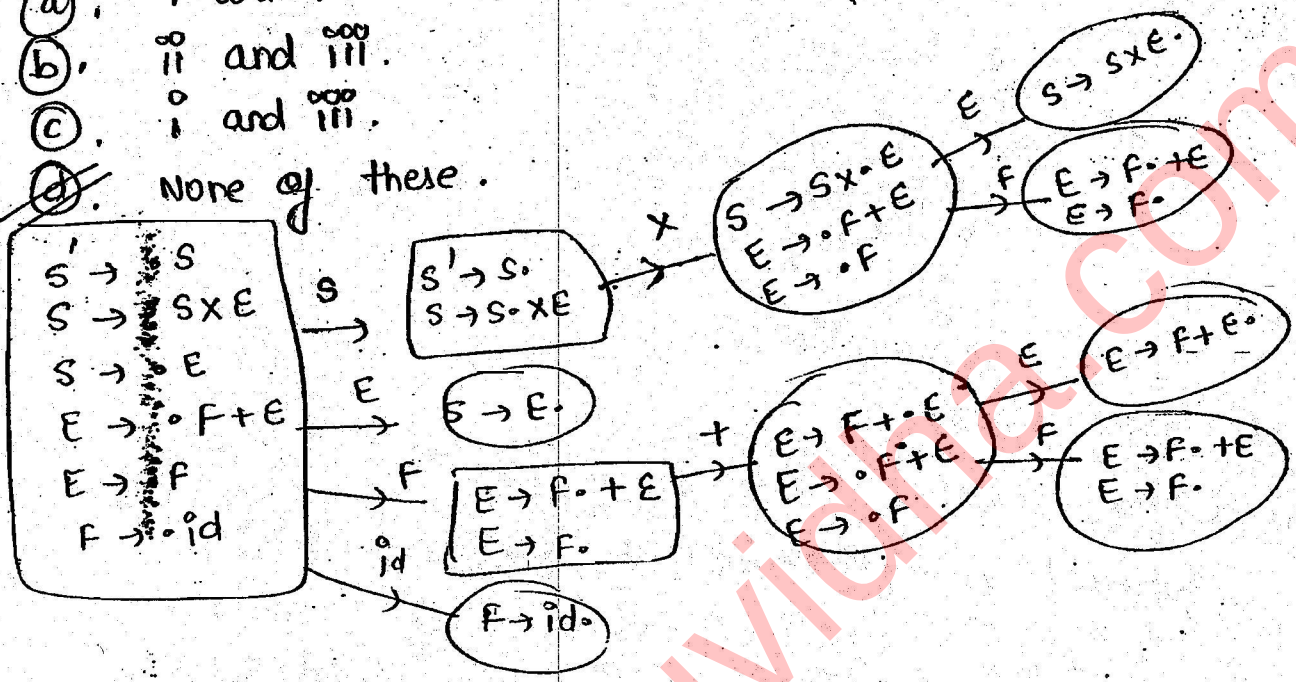
(i). $S \rightarrow S \times \cdot E$

(ii). $E \rightarrow F \cdot + E$

(iii). $E \rightarrow F + \cdot E$

Given the items above which 2 of them will appear in the same set in the canonical sets of items for the grammar.

(a). i and ii.
(b). ii and iii.
(c). i and iii.
(d). None of these.



(d).
No, none of the above.

Canonical LR' (CLR)

CLR is the most powerful technique among all parsing techniques. It can parse any arbitrary context free grammar.

The principle drawback of this method is it required lot of hardwork.

CLR parsing table construction depends on canonical sets of LR(1) items.

A LR(1) item is basically LR(0) item with lookahead symbols. The general form of LR(1) item is.

$$\underbrace{A \to X \cdot YZ}_{LR(0)}, \underbrace{\{a, b\}}_{LookAhead}$$