31.07.2023
Monday

Syntax Analysis → sentence is grammar valid or not

Parsing

- Top Down Parser
  - with backtracking
  - without backtracking
    - Brute force parser
    - Recursive parser
    - Predictive parser (LL(1) Parser)
- Bottom up Parser
  - SLR parser
  - LALR parser
  - LR Parser (CLR Parser) (Canonical LR Parser)
  - Operator Precedence Parser

Consider a grammar

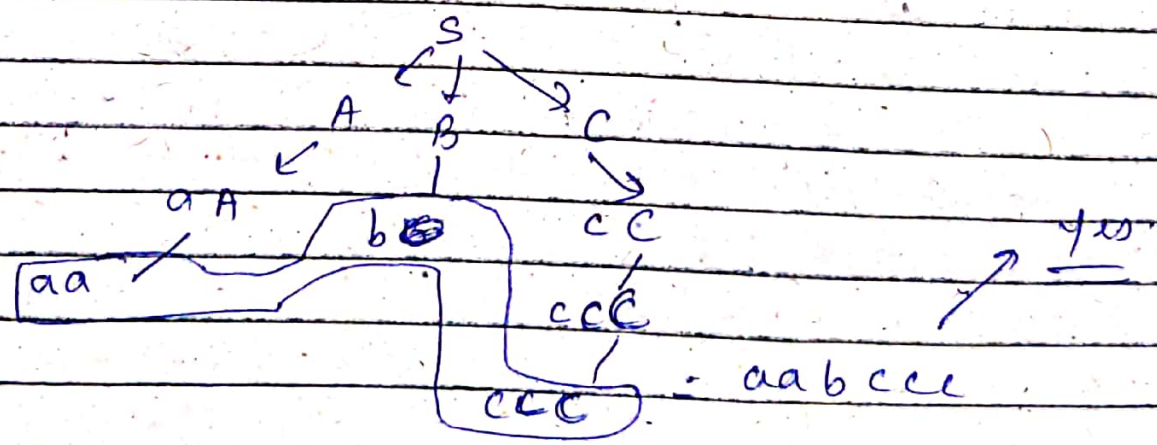
$S \to ABC$

$A \to aA \mid a$

$B \to bB \mid b$

$C \to cC \mid c$

String : aabccc



LMD $S \to$

RMD

aabbbcc

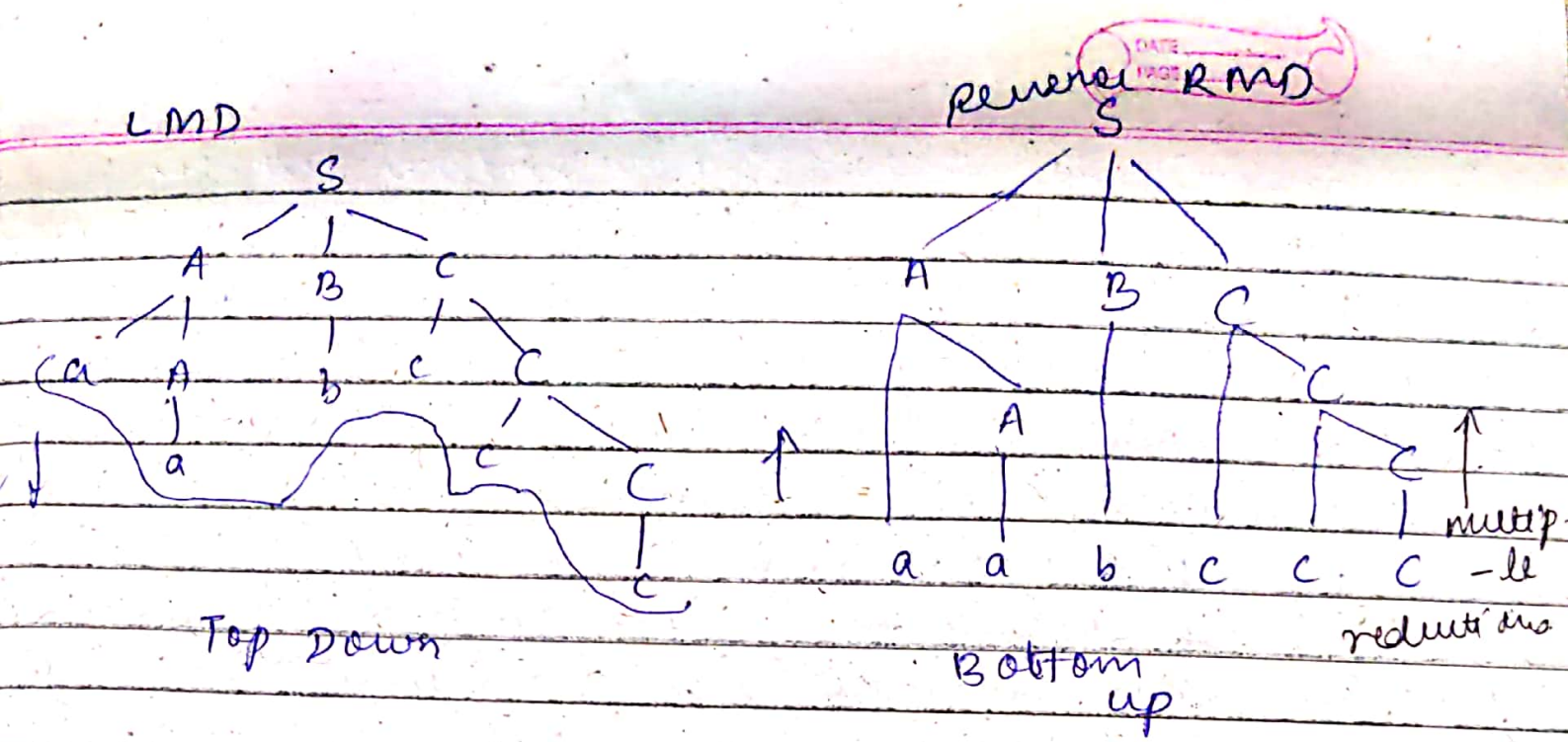
LMD : Left Most Derivation

Steps	Left Sential Form	Production
1.	S	$S \to ABC$
2.	ABC	$A \to aA$
3.	aABC	$A \to a$
4.	aaBC	$B \to bB$
5.	aabBC	$B \to bB$
6.	aabbBC	$B \to b$
7.	aabbbC	$C \to cC$
8.	aabbbcC	$C \to c$
9.	aabbbcc	

$S \to ABC$
$A \to aA / a$
$B \to bB / b$
$C \to cC / c$

Steps	Right Sential Form	Production
1.	S	$S \to A$

Scanned with CamScanner

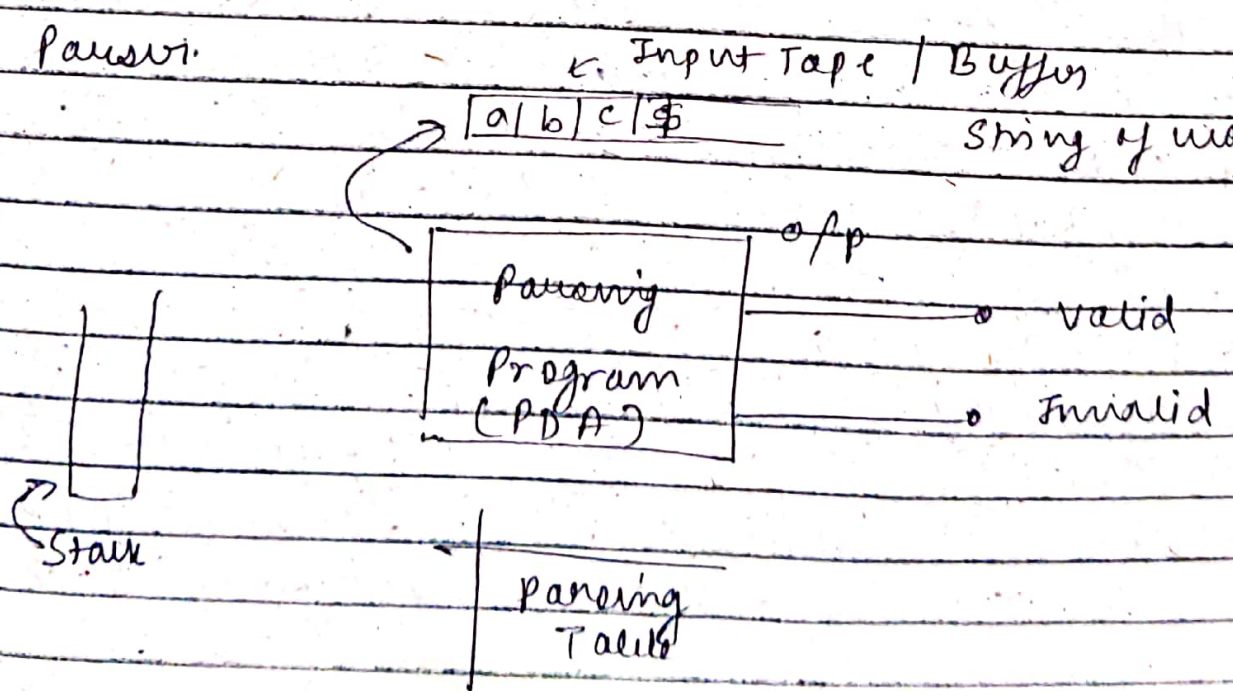


① All Top Down parsers uses LMD to construct logical parse trees of given string.

② All Bottom up parsers uses Reverse RMD to construct logical parse trees for the given input string.

In Reverse RMD, 'a' is reduced to 'A' & similarly other productions

Parser.



01.08.2023
Tuesday.

Input Tape contains tape which needs to test for acceptability.

Predictive Parser / Top Down Parser / LL(1) Parser / Non Recursive descent Parser

construct parsing table from given grammar

1. Remove left Recursion if is present in grammar.
2. Remove left factoring if needed.
3. Calculate first & follow sets of each grammar variable.
4. Construct parsing table.

① Removal of left Recursion

General Rule:

$A \rightarrow A\alpha_1 / A\alpha_2 / A\alpha_3 / \ldots / A\alpha_n$

$A \rightarrow \beta_1 / \beta_2 / \beta_3 / \ldots / \beta_m$

$$\boxed{\begin{aligned} A &\rightarrow \beta_1 A' / \beta_2 A' / \ldots / \beta_m A' \\ A' &\rightarrow \alpha_1 A' / \alpha_2 A' / \ldots / \alpha_n A' / \epsilon \end{aligned}}$$

Q) $A \rightarrow Abc / Aa / b / ac$

Sol: $A \rightarrow Abc / Aa$

$A \rightarrow b / ac$

Sol: $A \rightarrow bA' / acA'$

$A' \rightarrow bcA' / aA' / \epsilon$

Starting symbol same है तो Compiler कन्फ्यूज होता है select करने में Problem

$A \rightarrow aA / a$

Scanned with CamScanner

Whenever RHS of the production of the same variable have common prefix if 2 or more production of the same variable have their prefix common then we have to do left factoring.

$A \to aA / a \Rightarrow A \to aA'$

$A' \to A / \epsilon$

Q) $A \to abA / abB / ab$

$B \to bB / b$

Do left factoring

Sol: $A \to abA'$

$A' \to A / B / \epsilon$

$B \to bB'$

$B' \to B / \epsilon$

Q) $A \to abA / abB / a$

$B \to bB / b$

Do left factoring

Sol: $A \to aA'$

~~$A' \to A / B /$~~

$\times [A' \to bA / bB / \epsilon] \times$

$A' \to bA'' / \epsilon$

$A'' \to A / B$

$B \to bB'$

$B' \to B / \epsilon$

Q) $A \to Ab / ABa / Ba / a$

$B \to Ab / b$

Remove left Recursion & Do left factoring.

$A \to \underset{\alpha_1}{Ab} / \underset{\alpha_2}{ABa} / \underset{\alpha_3}{Ba}$

$A \to \underset{\beta_1}{a}$

~~$B \to A$~~

~~$A \to Ab / AA$~~

$A \to aA'$

$A' \to bA' / BaA' / Ba A'$

$A' \to bA' / AbaA' / baA' /$

Left Recursion का ~~है~~ $A \to A\alpha$ type.

$A \to Ab \mid AAb \mid Aba \mid Aba \mid ba \mid a$

$A \to Ab \mid AAb \mid Aba$

$A \to ba \mid a$

} Left Recursion

$A \to baA' \mid aA'$

$A' \to bA' \mid AbA' \mid baA' \mid \epsilon$

} No Left factoring.

$A \to \overset{\alpha_1}{Ab} \mid \overset{\alpha_2}{Ba} \mid \overset{\beta_1}{Ba} \mid \overset{\beta_2}{a}$

$B \to Ab \mid b$

$A \to BaA' \mid aA'$

$A' \to bA' \mid BaA' \mid \epsilon$

$B \to Ab \mid b$

Indirect Left Recursion

$\Rightarrow A \to \overset{\alpha_1}{Aba}A' \mid \overset{\beta_1}{baA'} \mid \overset{\beta_2}{aA'}$

~~$A' \to bA' \mid AbaA' \mid baA' \mid \epsilon$~~ $A' \to bA' \mid BaA' \mid \epsilon$

$B \to Ab \mid b$

$\Rightarrow A \to baA'A'' \mid aA'A''$

$A'' \to baA'A'' \mid \epsilon$

$A' \to bA' \mid BaA' \mid \epsilon$

$B \to Ab \mid b$

Scanned with CamScanner

07.08.2023
monday

FIRST & FOLLOW SETS

Q) $A \to aAb / cb$ — first(A) = {a, c}
$B \to bB / a$ — first(B) = {b, a}

Q) $A \to Bc / ba$ — first(B) = {c, a}
$B \to cC / aB$ — first(A) = {c, a, b}
$C \to a$ — first(C) = {a}

put here & get first of A.

Q) $A \to baA'A'' / aA' / A''$ — first(A) = {b, a}

first of grammar variable is all the terminals which appear as a left most symbol obtained from all possible derivations of that variable.

Q) $A \to baA'A'' / aA'A''$ → {b, a} or {a, b} (equal)
$A'' \to baA'A'' / \epsilon$ → {b, ε}
$A' \to bA' / BaA' / \epsilon$ → {b, a, ε}
$B \to Ab / b$ → {b, a}

first(A) = {b, a}

first(A') = {b, ε} ∪ first(B)
~~{b, ε}~~ ~~{b, a, ε}~~
= {a, b, ε}

first(A'') = {b, ε}

first(B) = {b, a}

ε comes in first → Null string.

$\alpha, \beta \rightarrow$ string of variables

$\{A, \epsilon, a, aABcd\}$

→ Look in RHS

A के Just बाद Terminal

FOLLOW

(1) Include "$" in the follow of start variable of the grammar

(2) For all the productions of type

$B \rightarrow \alpha A \beta$ $\qquad$ α, β : string of variable or terminal

$\text{Follow}(A) = \text{First}(\beta)$

(3) If the productions are of type

$B \rightarrow \alpha A$

or $B \rightarrow \alpha A \beta$ where $\beta \rightarrow \epsilon$ (direct / or after multiple productions)

(Last symbol is variable)

$\text{Follow}(A) = \text{Follow}(B)$

NOTE: ϵ is never included in the follow of any variable

Q) $A \rightarrow baA'A'' \mid aA'A''$

$A'' \rightarrow baA'A'' \mid \epsilon$

$A' \rightarrow bA' \mid BaA' \mid \epsilon$ $\qquad$ $A'A''$ 1st rule

$B \rightarrow Ab \mid b$

Sol: follow (A) = ~~{$, b, }~~ $\{\$, b\}$, 3rd rule $\beta = \epsilon$ Fol(A) = Z

follow (A'') = ~~{$, b}~~ $\{\$, b\}$

follow (A') = ~~{$, b, a}~~ $\{b, \$\}$

follow (B) = ~~{$}~~ $\{a\}$

अब A' वाले RHS देखो

first (A) = {b, a}

first (A'') = {b, ε}

first (A') = {b, a, ε}

first (B) = {b, a}

$A'' = \{b, \epsilon\}$

put

$baA'A'' : baA''$ 3rd rule

A : the answer is what it is

Q) $S \to \cancel{ABC}\ ABc \mid BAd$

$A \to Bba \mid Bb \mid \epsilon$

$B \to Ae \mid \epsilon$

Sol:

$A \to \underbrace{Aeba}_{\alpha_1} \mid \underbrace{Aeb}_{\alpha_2} \mid \underbrace{ba}_{\beta_1} \mid \underbrace{b}_{\beta_2} \mid \underbrace{\epsilon}_{\beta_3}$ [left Recursion]

$A \to baA' \mid bA' \mid A'$

$A' \to eba A' \mid eb A' \mid \epsilon$

$B \to Ae \mid \epsilon$

$S \to ABc \mid BAd$

left factoring

$A \to bA'''$

$A''' \to aA' \mid A'$

$A'' \to eba A'' \mid \epsilon$

$B \to Ae \mid \epsilon$

$S \to ABc \mid BAd$

$A''' \to aA'''$

~~first(A) = {b}~~

~~first(A''') = {a}~~

first

A {

A'

A''

B

S {

left factoring

$A \to baA' \mid bA' \mid A'$

$A' \to eb A'' \mid \epsilon$

$A'' \to aA' \mid A'$

$B \to Ae \mid \epsilon$

$S \to ABc \mid BAd$

Scanned with CamScanner

ε नहीं आएगा। c A d terminal

	first c,d	follow
$S \to ABc / BAd$	$\{b, e\}$	$\{\$\}$
$A \to bA''' / A'$	$\{b, e, \epsilon\}$	$\{b, e, d, c\}$
$A''' \to aA' / A'$	$\{a, e, \epsilon\}$	$\{b, e, d, c\}$
$A' \to ebA'' / \epsilon$	$\{e, \epsilon\}$	$\{b, e, d, c\}$
$A'' \to aA' / A'$	$\{a, e, \epsilon\}$	$\{b, e, d, c\}$
$B \to Ac / \epsilon$	$\{b, c\}$ ϵ	$\{b, e, d, c\}$

these 2 are same $A'' = A'''$

first(A''') = first(A'') ∪ {partial}

first(A'') = fn.

Q) $E \to \underbrace{E+T}_{\alpha_1} / \underbrace{T}_{\beta_1}$

$T \to T*F / F$

$F \to (E) / id$

Sol: construct prediction parser for given grammar

$S \to ABc / BAd$

$A \to bA'' / A'$

$A'' \to aA' / A'$

$A' \to ebA'' / \epsilon$

$B \to Ac / \epsilon$

Sol: Remove left Recursion

E : Expression

T :

id : identifier

$E \to TE'$

$E' \to +TE' / \epsilon$

$T \to FT'$

$T' \to *FT' / \epsilon$

$F \to (E) / id$

No left factoring

	first	follow
E	$\{(, id\}$	$\{\$,)\}$
E'	$\{+, \epsilon\}$	$\{\$,)\}$
T	$\{(, id\}$	$\{+, \$,)\}$
T'	$\{*, \epsilon\}$	$\{+, \$,)\}$
F	$\{(, id\}$	$\{+, *, \$,)\}$

follow

Step. 4 : Parsing Table :-

Terminals

	+	*	(	)	id	$
E			$E \to TE'$		$E \to TE'$	
E'	$E' \to +TE'$			$E' \to \varepsilon$		$E' \to \varepsilon$
T			$T \to FT'$		$T \to FT'$	
T'	$T' \to \varepsilon$	$T' \to *FT'$	~~T' → FT'~~	$T' \to \varepsilon$		$T' \to \varepsilon$
F			$F \to (E)$		$F \to id$	

Grammar variables

Predictive Parser

id + id * id

Draw LMD

~~→ id~~

LMD

$E \to TE'$

$\to FT'E'$

$\to id\ T'E'$

$\to id\ E'$ $(T' \to \varepsilon)$

$\to id + TE'$

$\to id + FT'E'$

$\to id + id * FT'E'$

$\to id + id * id$ Yes it is possible

See Production

First में जो आ रहे है उसमे Production लिखो।

$E' \to +TE' / \varepsilon$

$\Rightarrow E' \to \varepsilon$

इस नाम के Production में

RHS $= \varepsilon$

Follow वाले में से Prod^n लिखो।

①

Stack	input tape	Action
$E	id + id * id $	E → TE'

② In opp order

$, E', T	id + id * id $	T → FT'
$, E', T', F	id + id * id $	F → id
$, E', T', id	id + id * id $	pop id
$, E', T'	+ id * id $	T' → ε
$, E', ε	+ id * id $	E' → +TE'
$, E', T, +	+ id * id $	pop +
$, E', T	id * id $	T → FT'
$, E', T', F	id * id $	F → id
$, E', T', id	id * id $	pop id
$, E', T'	* id $	T' → *FT'
$, E', T', F, *	* id $	pop *
$, E', T', F	id $	F → id
$, E', T', id	id $	pop id
$, E', T'	$	T' → ε
$ E'	$	E' → ε
$	$	Accepted

Q) S → iEtS / iEtSeS
E → b

Sol: No left Recursion

Scanned with CamScanner

Left Factoring

		first	Follow
$S \rightarrow iEtSS'$		$\{i\}$	$\{\$, e\}$
$S' \rightarrow \epsilon / eS$		$\{e, \epsilon\}$	$\{\$, e\}$
$E \rightarrow b$		$\{b\}$	$\{t\}$

Parsing table

	i	t	e	b	$
S	$S \rightarrow iEtSS'$				
S'			$S' \rightarrow eS$ $S' \rightarrow \epsilon$		$S' \rightarrow \epsilon$
E		$E \rightarrow b$		$E \rightarrow b$	

This is not LL(1) grammar
(as there are multiple entries in same column)

NOTE: (1) For the given grammar it is not possible to construct predictive parser.
(2) The given grammar is not LL(1) parser.
(3) What is LL(1) grammar?
→ Grammar for which it is possible to construct LL(1) parser without any multiple entries is called LL(1) ~~parser~~ grammar.

(1) LL(1) Parser

L → Left to Right Scanning
L → LMD
1 → Scanning only one symbol at a time.

id + id + id
↑ ↑ ↑
If checking 3 at a time LL(3) parser

NOTE: Bottom up parsers are more powerful than Top Down Parsers

How to check grammar is LL(1) or not?

Let $A \to \alpha / \beta$ are the productions of A (variable)

The given grammar is LL(1) if

(1) $first(\alpha) \cap first(\beta) = \phi$
(intersection)

(2) Both α & β should not simultaneously derive to ϵ

(3) If $\beta \xrightarrow{*} \epsilon$ (β after certain no. of production produces ϵ)

then $first(\alpha) \cap follow(A) = \phi$

First remove left recursion & left factoring then it is $A \to \alpha / \beta$

Bottom Up Parser

SLR(1) parser

- S → Simple
- L → Left to Right Scanning
- R → RMD in reverse order
- 1 → one symbol at a time

Steps for constructing SLR(1) parser:-

1. Calculate first & follow sets of each grammar variable.

(2) Assign no. to different production of grammar.

(3) Argument the grammar by adding a grammar production $S' \rightarrow S$ where S: start symbol of grammar.

(4) Calculate canonical collection of LR(0) item sets.

(5) With the use of canonical collection construct parsing table.

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

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

Sol: Left Recur (previous example)

	first	follow
E	{ (, id }	{ $,) }
E'	{ +, ε }	~~{ +, $ }~~ { $,) }
T	{ (, id }	{ +, $,) }
T'	{ *, ε }	{ +, $,) }
F	{ (, id }	{ +, *, $,) }

Scanned with CamScanner

18.08.2023
Friday

3.

		First	Follow
0	$E' \to E$		
1	$E \to E+T$	$\{(, id\}$	$\{\$, +,)\}$
2	$E \to T$		
3	$T \to T*F$	$\{id, (\}$	$\{*, \$, +,)\}$
4	$T \to F$		
5	$F \to (E)$	$\{id, (\}$	$\{*, \$, +,)\}$
6	$F \to id$		

Canonical form :- Collection of LR(0) :-

Production : $A \to aBC$ → Production

$$\left.\begin{array}{l} A \to .aBC \\ A \to a.BC \\ A \to aB.C \\ A \to aBC. \checkmark \end{array}\right\} \text{item sets}$$

$A \to aB$ → Production

$$\left.\begin{array}{l} A \to .aB \\ A \to a.B \\ A \to aB. \checkmark \end{array}\right\} \text{item sets}$$

$A \to a$

$$\left.\begin{array}{l} A \to .a \\ A \to a. \checkmark \end{array}\right\}$$

$A \to \epsilon$

$$\left.\begin{array}{l} A \to . \checkmark \end{array}\right\}$$

$$\boxed{N(\text{item sets}) = \text{String length on RHS} + 1}$$

$\checkmark$ = Complete item

Item Set I_0

Now apply closure property. Make a set I_0 & make the argument set. $E' \to .E$. Now apply closure property if immediately after '.' we have variable then make all the productions in the item set.

Scanned with CamScanner

8006882870
Higher Education Dept.

apply goto property

Step. 1

I_0 : $E' \to .E$
$E \to .E+T$
$E \to .T$
$T \to .T*F$
$T \to .F$
$F \to .(E)$
$F \to .id$

'.' के बाद जो Variable उसका Production भी लिखो।
→ Rule ①

Step. 2

I_1 : goto (I_0, E)

जिस भी Production में E (RHS) उसमें '.' को एक आगे ले जाओ। in set I_0

$E' \to E.$
$E \to E.+T$

Step. 3

I_2 : goto (I_0, T)
$E \to T.$
$T \to T.*F$

Step. 4

I_3 : goto (I_0, F)
$T \to F.$

Step. 5

I_4 : goto (I_0, ( )
$F \to (.E)$
$E \to .E+T$ (Apply Rule ①)
$E \to .T$
$T \to .T*F$
$T \to .F$
$F \to .(E)$
$F \to .id$

Step. 6

I_5 : goto (I_0, id)
$F \to id.$

Step 7

I_6 : goto (I_1, +)
$E \to E+.T$
$T \to .T*F$ (Rule ①)
$T \to .F$ (Rule ①)
$F \to .(E)$
$F \to .id$

Step 8

I_7 : goto (I_2, *)
$T \to T*.F$ (Rule ①)
$F \to .(E)$
$F \to .id$

Step. 9

~~I_9~~ : goto (I_4, T) = I_2
$E \to T.$
$T \to T.*F$

Scanned with CamScanner

Step. 10 कोई set Equivalent हो तो उसे उसका Name दे दो।

I_{10}: goto $(I_4, F) = I_3$
$T \to F.$

I_{10} : goto (I_7, F)
$T \to T * F.$

Step. 11

I_{11} : goto $(I_4, ()) = I_4$
$F \to (.E)$
$E \to .(E+T)$
$E \to .T$
$T \to .T * F$
$T \to .F$
$F \to .(E)$
$F \to .id$

I_4 : goto $(I_7, ()$

I_5 : goto (I_7, id)

I_{11} : goto $(I_8,))$
$F \to (E).$

I_6 : goto $(I_8, +)$

I_7 : goto $(I_9, *)$

Step. 12

I_5 : goto (I_4, id)

Step. 13

I_9 : goto (I_6, T)
$E \to E+T.$
$T \to T.*F$

:

I_3 : goto (I_6, F)
I_4 : goto $(I_6, ()$
I_5 : goto (I_6, id)

There are 12 item sets
(0 to 11)

Rows : I_0 to I_{11}
Columns : All terminals + ~~columns~~ variables
also $, E, T, F
+, *, (,), id

Shift : I_4 : goto(I_0, F)
Same to

Row column
↑ ↑
I_1 : goto(I_0, E)
put this number

SLR(1) Parsing Table

~~Item + * ( ) id $~~

	Action						Goto		
Item	+	*	(	)	id	$	E	T	F
I_0			S_4		S_5		1	2	3
I_1	S_6					accept			
I_2	r_2	S_7		r_2		r_2			
I_3	r_4	r_4		r_4		r_4			
I_4			S_4		S_5		8	2	3
I_5	r_6	r_6		r_6		r_6			
I_6			S_4		S_5			9	3
I_7			S_4		S_5				10
I_8	S_6			S_{11}					
I_9	r_1	S_7		r_1		r_1			
I_{10}	r_3	r_3		r_3		r_3			
I_{11}	r_5	r_5		r_5		r_5			

Scanned with CamScanner

s : shift
r : reduced

Complete set

I_1 : $E' \to E.$
I_2 : $E \to T.$
I_3 : $T \to F.$
I_5 : $F \to id.$
I_9 : $E \to E+T.$
I_{10} : $T \to T*F.$
I_{11} : $F \to (E).$

$E' \to E$	0
$E \to E+T$	1
$E \to T$	2
$T \to T*F$	3
$T \to F$	4
$F \to (E)$	5
$F \to id$	6

यहाँ पर Augmented Production Complete हो रही है
I_i us $. → accepted

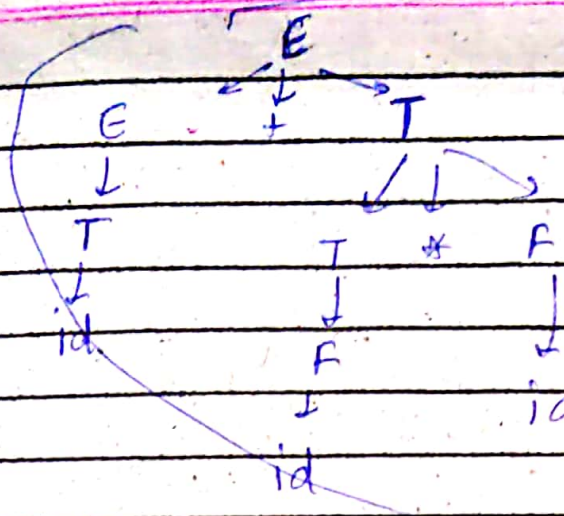
This no. only r_i

Complete state	Production	Entry	Follow
I_2	$E \to T$	r_2	$, + ,)$
I_3	$T \to F$	r_4	$, + , * ,)$
I_5	$F \to id$	r_6	$, + ,), *$
I_9	$E \to E+T$	r_1	$, + ,)$
I_{10}	$T \to T*F$	r_3	$, + , * ,)$
I_{11}	$F \to (E)$	r_5	$, + , * ,)$
↓ row		↓ Entry	↓ column

How to fill

Q) Check string $ id + id * id $

Sol: Draw Reverse RMD



LMD

E	$
E + T	E → E + T
T + T	E → T
F + T	T → F
id + T	F → id
id + T * F	T → T * F
id + F * F	T → F
id + id * F	F → id
id + id * id	F → id

RMD

E → E + T

→ E + T * F (T → T * F)

→ E + T * id (F → id)

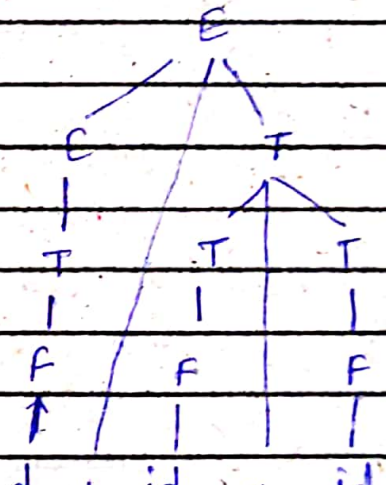
→ E + F * id (T → F)

→ E + id * id (F → id)

→ T + id * id (E → T)

→ F + id * id (T → F)

→ id + id * id (F → id)



~~Au~~ How to check

(1) Stack top will always be the state Number (0 to 11) 12 states

(2) Grammar symbols & state no. will always appear alternatively fashion in stack.