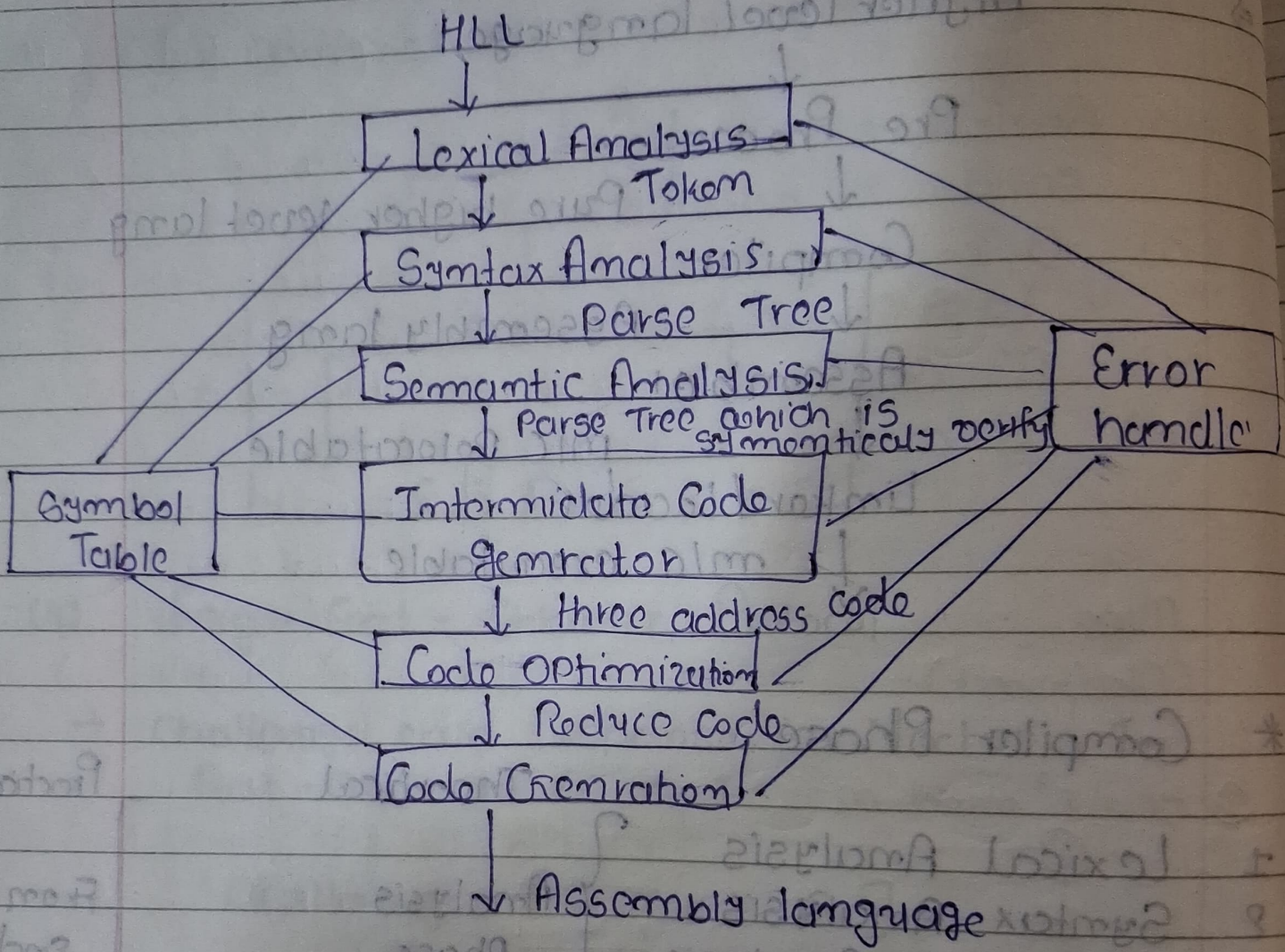
CD

⇒ Higher level language
↓
Pre Processor
↓ Pure higher level lang
Compiler
↓ Assembly lang
Assembler
↓ m|c Relocatable
Linker / Loader
↓ m|c executable

* Compiler Phase

		Theortical	Practical
1	Lexical Analysis	Analysis Phase	Front End
2	Symtax Analysis		
3	Semantic Analysis		
4	Intermidate Code genration	Sythensis Phase	Back End
5	Code Optimization		
6	Code genration		

1 Lexical Analysis :
↳ Find Tokens
↳ Recognize the tokens
↳ Remove white Space
↳ Remove # statment



2 Symtax Analysis :
Indemtify proper order

3 Semantic Analysis
Variable Type check
Scop of variable check

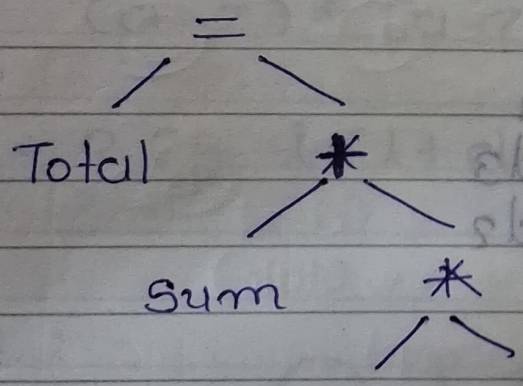
⇒ Three address code : maximum three oprand used

Ex. Total = Sum + rate * 60;

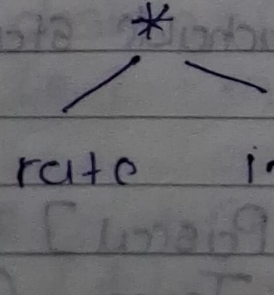
1. Lexical Tokan:

Tokan	Identifier
Total	Identifier
=	assign operator
Sum	identifier
+	pluse operator
rate	id
*	mul operator
60	Constant
;	Special Symbol

2 Syntax Analysis



3 Symantic Analysis: Type Castings



4 Intermidate Code Genrator

```
temp1 = int ToReal (60)
temp2 = rate * temp1
temp3 = sum + temp2
total = temp3
```

5 Code Optimization

```
temp1 = rate * int To Real (60)
Total = sum + temp1
```

→ Unessary Symbol reduce into loop.

6 Code Genration

```
id1 = id2 + id3 * 60
move  R1, 60
MUL   id3, R1
ADD   id2, id3
MOV   id1, id2
```

* Error handling

→ Lexical Analysis: Invalid Variable
Eg: 1abc, @;, $a (Invalid Tokens)

→ Code Optimization: Unreachable statment

* Type of Compiler

1 OnePass Compiler [Eg: Pascal]
2 MultiPass Compiler [Eg: Java, C, C++..]

OnePass	Multipass
→ Faster	→ Slower
→ knowledge all phase	→ distribute work
→ Can't recognize Error easily	→ Find Error Easily

* Cross Compiler: It genrate executable file which run in another platform.
Exc: Java (Platform Independant)

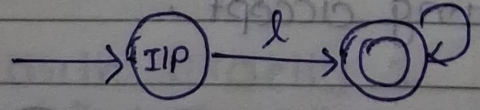
x: Write the R.E. that accept the striing, which is identifier, the design F.A

(a-z) * [a-z]

OR

R.E. $l(l+d)^*$ $\quad l = a \dots z$, $d = 0 \dots 9$

F.A.:

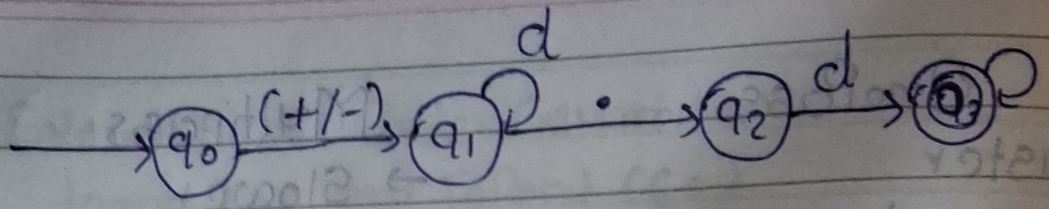


Exc. Write R.E. That accept the string all real numbe and design F.A

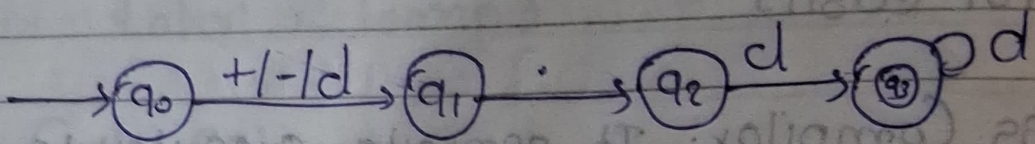
r: {0, 1, ... m} or {-1, -2, ... m} OR {0.5, 0.2 .. }

R.E $r(r)^*$

$(+/-)d^*.d^+$

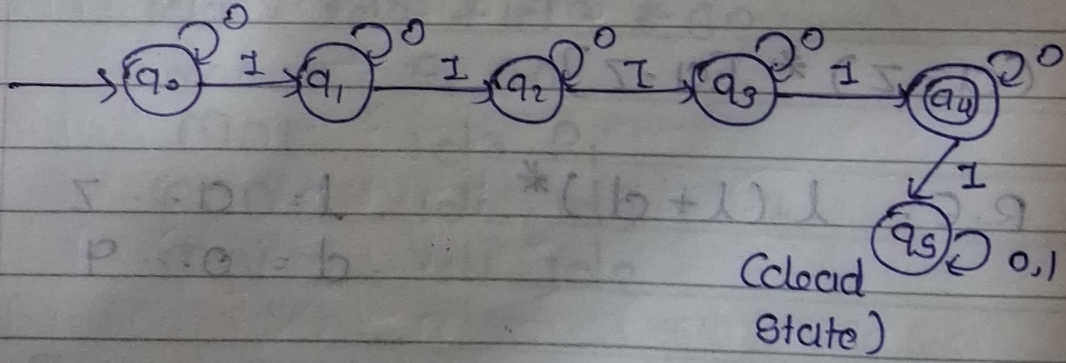


OR

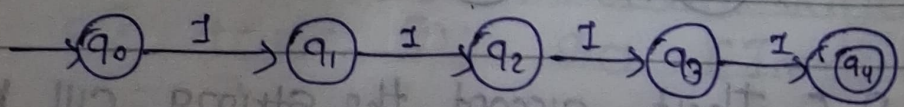


Ex: R.E that accept the string that Contain IIII

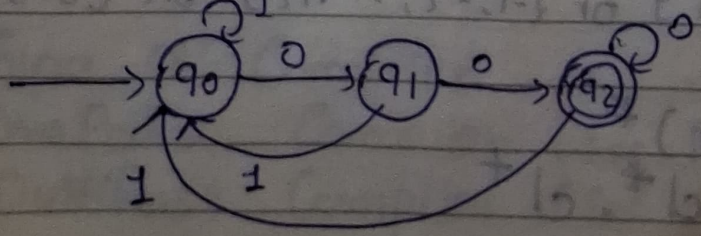
R.E: (0)* I (0)* I (0)* I (0)* I (0)*



→ Only IIII String accept.

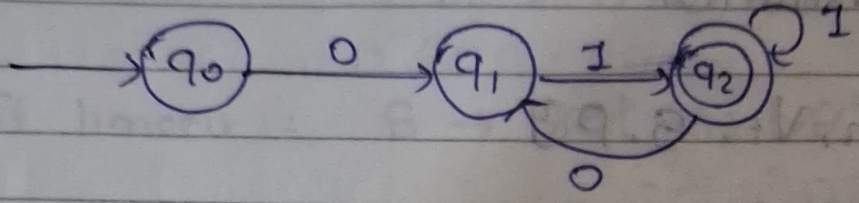


Ex Ending with 00



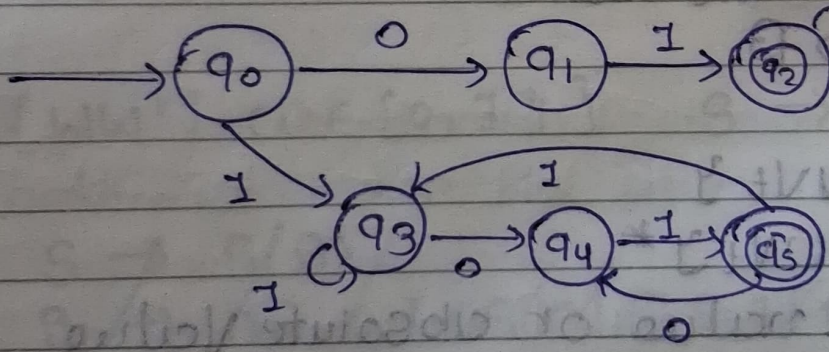
Ex: desig FA and RA the string Start and End with 01

R.E: 01 (0+1)*



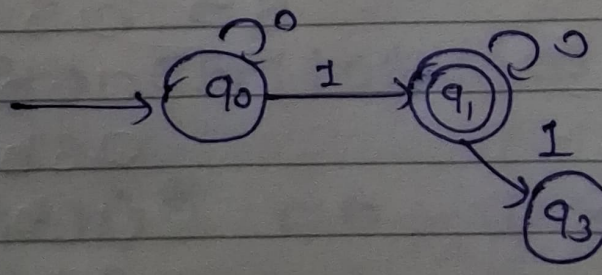
Ex: R.E and F.A that Starting OR Ending with 01

R.E: 01 (0+1)* + (0+1)*01



Ex: R.E with exactly 1

(0)* 1 (0)*



Grammar:

It is combination of terminal and non terminal

$G = \{Vn, Vt, S, P\}$

Vt: Set of Terminal
Vn = Set of Non terminal
S = String Symbol
P = Production Rule

Production Rule is

$$\alpha \rightarrow \beta$$

$\alpha \in \{Vn \cup Vt\}$
$\beta \in \{Vn \cup Vt\}^*$
Null value or absolute Value

$S \rightarrow a$
$S \rightarrow aS \mid \epsilon$
$S \rightarrow bS$

Exc: Make CFG $0(01)^*$

Right linear : $S \to 0S_1$
$S_1 \to 01S_1 \mid \epsilon$

Left linear : $S \to S01 \mid 0$

Exc: $0^* 1 (0+1)^*$

$S \to S_1 1 S_2$
$S_1 \to 0S_1 \mid \epsilon$
$S_2 \to 0S_2 \mid 1S_2 \mid \epsilon$

Exc: $\{WW^r \mid w \in \{0,1\}\}$ $S \to 1S_1 1 \mid 0S_1 0 \mid \epsilon$

$S \to S/S_2$
$S_1 \to 0S_1 1 \mid 1S_1 0 \mid \epsilon$ $S_1 1 / 0S_1 0 / \epsilon$
$S_2 \to$

Exc: Palindrom string

$S \to 1S_1 1 \mid 0S_1 0 \mid 0 \mid 1 \mid \epsilon$

Exc: $(a+b)^* \; ab \; (a+b)^*$

$S \to S_1 ab S_1$
$S_1 \to aS_1 \mid bS_1 \mid \epsilon$

Ex: Equal Number of parenthesis

$S \to (S)S \mid \epsilon$

Ex: $a^{en} b^{m}$. $n \geq 0$. $m \geq 0$

$a^e b^*$

$S \to aaS_1 S_2$.

$S_1 \to aaS_1 \mid \epsilon$

$S_2 \to bS_2 \mid \epsilon$

Ex Equal No of a and b

$S_1 \to aSbS \mid bSaS \mid \epsilon$

→ LMD: Left most Derivative
(start with left most terminal)

RMD: Right most Derivative

String: ababab

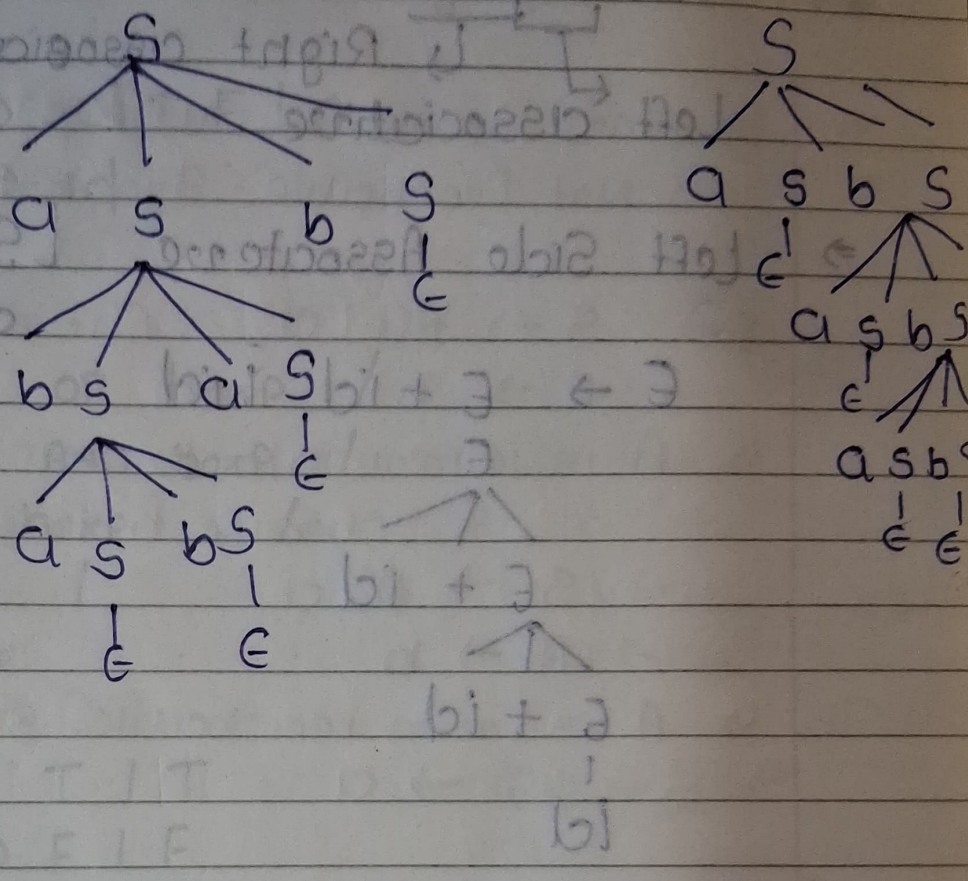
LMD:

$S \to aSbS$
$\to abSaSbS$
$\to abaSbSbS$
$\to ababab$

RMD

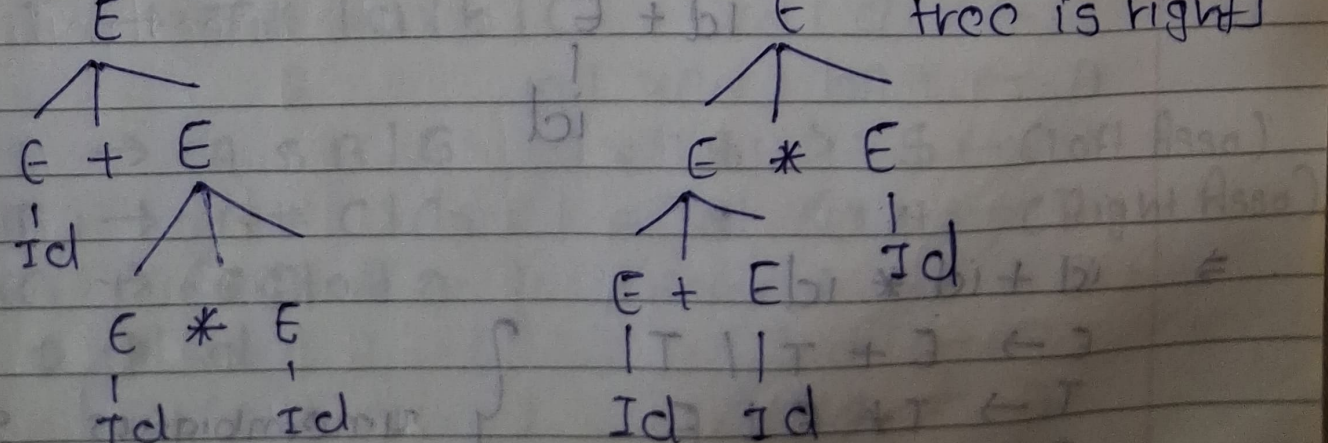
$S \to aSbS$
$aSbaSb$
$aSbaSbab$
$aSbabab$
$ababab$

Parse Tree



If any grammar contains two parse tree / LMD / RMD, It's calle Ambiguos Grammar

E → E + E / E * E / id [higher level precedance is lower level then tree is right]



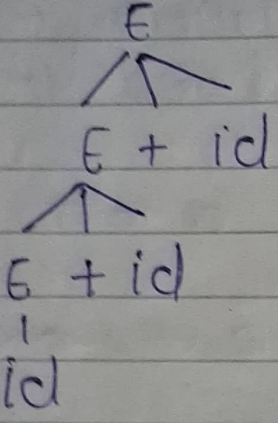
id + id * id

↳ Right associative

Left associative

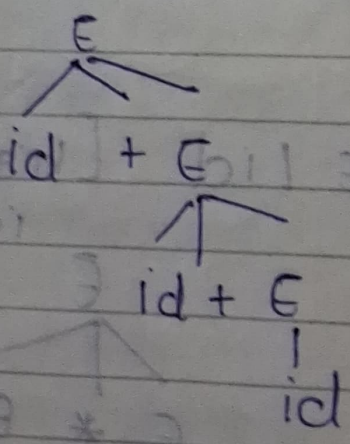
⇒ Left Side Associative. [Same precedence then always used left associative]

$E \rightarrow E + id \mid id$



⇒ Right side Associative

$E \rightarrow id + E \mid id$



⇒ id + id * id

$E \rightarrow E + T \mid T$

$T \rightarrow T * F \mid F$

$F \rightarrow id$

} unambiguous grammar

Page
Date

Ex 2^{3^2} → (2 ↑ (3 ↑ 2)) (Right Associative)

Rule: E → G ↑ F | G
G → id

Ex: bexp → bexp OR bexp /
bexp AND bexp /
Not bexp /
True /
False

E → E OR T / T
T → T AND F / F
F → Not F / true / False

Ex: R → R+R / RR / R* / a / b / C

E → E + F / F
F → FT / T
T → T* / a / b / C

⇒ A → A $ B / B
B → B # C / C
C → C @ D / D
D → d

$ > $ (Left Asso)
$ < # (Right Asso

* Restriction on CFG

⇒ Remove unnessary (useless) Symbol
↳ Not Reachable Symbol remove

Ex: S → AB | a (B - non genrating)
A → b

↳ B is elimincate because it's not useful
(because B has not expand)

S → a
A → b

↳ A is non reachable. So,

S → a

Ex: S → ABC | d
B → bC

⇓ Remove useless symbol

S → d

Ex S → aB | bX
A → BAd | bSx | a
B → aSB | bBx
X → SBD | aBx | ad

1. B is Non-genrating

S → bX
A → bSX | a
X → ad

2. A is not reachable

S → bX
X → ad

Ex: $A \rightarrow xyz \mid xyzz$
$x \rightarrow xz \mid xyz$
$y \rightarrow yyy \mid xz$
$Z \rightarrow Zy \mid Z$

→ Non-Generating (x, y)

$A \rightarrow xyz \mid xyzz$
$x \rightarrow$
$Z \rightarrow Zy \mid z$

→ Not Reachable (Z)

$A \rightarrow xyz$

Ex: $S \rightarrow aC \mid SB$
$A \rightarrow bSCa$
$B \rightarrow aSB \mid bBC$
$C \rightarrow aBC \mid ad$

Non-generating

$S \rightarrow aC$
$C \rightarrow ad$
$A \rightarrow bSCa$

Not Reachable: (A)

$S \rightarrow aC$
$C \rightarrow ad$

⇒ Unit Production

non terminal → one non terminal

$A \to B$ (unique production)

Exc: $A \to B$

$B \to \alpha$

$\Downarrow$

$A \to \alpha$

Exc. $S \to A$

$A \to a$

↳ $S \to a$

Exc: $S \to AB$

$A \to a$

$B \to C \mid b$

$C \to D$

$D \to E$

$E \to a$

$S \to AB$

$A \to a$

$B \to a \mid b$

$S' \to aB$

~~$B \to a \mid b$~~

Ex: $I \to a \mid b \mid Ia \mid Ib \mid Io \mid I$

$F \to I \mid (E)$

$T \to F \mid T * F$

$E \to T \mid E + T$

$F \to I$

$T \to F \Rightarrow$

$E \to T$

$F \to I$

$E \to F \Rightarrow$

$I \to a \mid b \mid Ia \mid Ib \mid Io \mid I$

$F \to I \mid (E)$

$T \to I \mid T * I$

$E \to I \mid E + I$

(I ना वरील value असेल)

I → a|b| Ia | Ib| Io| I,
F → a|b|Ia| Ib| Io|I,|(E)
T → a|b| Ia| Ib | Io| I1 |(E)| T*F
E → a|b| Ia|Ib| Io| I,| (E)| T*F| E+T

⇒ Eliminate Null Value

Ex: S → aA ⇒ S → aA | a
A → b|ϵ A → b

Exc S → AaB
A → a|ϵ
B → b|ϵ

S → AaB | Aa | aB | a
A → a
B → b

Ex: S → ABAC
A → aA|ϵ
B → bB|ϵ
C → c

S → ABC| BAC | BC| C |AAC | AC | ABAC
A → aA|a
B → bB|b
C → c

* Issue in CFG

1. Ambiguity
2. Left Recursion
3. Indirect Left Recursion
4. Left Factoring

$A \rightarrow A\alpha \mid \beta$ (Accept string like $\beta\alpha^*$)

$A \rightarrow \alpha A / \beta$ (Right Recursion)
↳ Accept like $\alpha^*\beta$

↳ Problem In Left Recursion

```
A ( )
{
   A ( ) / α ;
}
```

⇒ It's going to infinite loop.

↳ Righ Recursion

```
A ( )
{
   α A ( )
}
```

⇒ Here problem of infinite not occured.

↳ Solution: Remove Left Recursion

$A \rightarrow BA'$

$A' \rightarrow \alpha A' / \epsilon$

A A α β

Ex: $E \to E+T \mid T$

$E \to TE'$

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

Ex: $A \to A\alpha_1 \mid A\alpha_2 \mid A\alpha_3 \mid \beta_1 \mid \beta_2 \mid \beta_3$

$A \to \beta_1 A' \mid \beta_2 A' \mid \beta_3 A'$

~~$A' \to \epsilon \mid A'\alpha_1 \mid A'\alpha_2 \mid A'\alpha_3$~~

$A' \to \epsilon \mid \alpha_1 A' \mid \alpha_2 A' \mid \alpha_3 A'$

Ex: $T \to T+F \mid F$

$F \to F * G \mid G$

$G \to id$

$T \to FT'$

$T' \to \epsilon \mid +FT'$

$F \to GF'$

$F' \to \epsilon \mid *GF'$

$G \to id$

Ex: $S \to X \mid Xb \mid Ya$

$X \to Xa \mid a \mid b$

$Y \to X \mid a$

$S \to X \mid Xb \mid Ya$

$X \to aX' \mid bX'$

$X' \to \epsilon \mid aX'$

$Y \to X \mid a$

$A \rightarrow A \mid Ab \mid b$

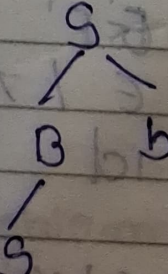
$A \rightarrow bA'$
$A' \rightarrow \epsilon \mid bA'$

$S \rightarrow A$
$A \rightarrow Ad \mid Ae \mid aB \mid aC$
$B \rightarrow bBC \mid f$
$C \rightarrow g$

$S \rightarrow A$
$A \rightarrow aBA' \mid aCA'$
$A' \rightarrow dA' \mid eA' \mid \epsilon$
$B \rightarrow bBC \mid f$
$C \rightarrow g$

(3rd Issue)
(Imdirect Left Recursion)

$S \rightarrow Bb \mid a$
$B \rightarrow Bc \mid Sd \mid e$



$S \rightarrow Bb \mid a$
$B \rightarrow Sd B' \mid eB'$
$B' \rightarrow cB' \mid \epsilon$

$S \rightarrow Bb \mid a$
$B \rightarrow Bc \mid Bbd \mid ad \mid e$

$\Downarrow$

$S \rightarrow Bb \mid a$
$B \rightarrow adB' \mid eB'$
$B' \rightarrow cB' \mid bdB' \mid \epsilon$

Page
Date

x: $Q \rightarrow QED \mid q$
$E \rightarrow e$
$D \rightarrow NFA \mid d$
$N \rightarrow DFA \mid m$
$F \rightarrow f$
$A \rightarrow a$

$Q \rightarrow QED \mid q$
$E \rightarrow e$
$D \rightarrow NFA \mid d$
$N \rightarrow NFAFA \mid dFA \mid m$
$F \rightarrow f$
$A \rightarrow a$

$Q \rightarrow qQ'$
$Q' \rightarrow EDQ' \mid \epsilon$
$E \rightarrow e$
$D \rightarrow NFA \mid d$
$N \rightarrow dFAN' \mid mN'$
$N' \rightarrow \epsilon \mid FAFAN'$
$F \rightarrow f$
$A \rightarrow a$

Ex: $A \rightarrow B \mid a \mid CBD$
$B \rightarrow C \mid \epsilon \mid b$
$C \rightarrow A \mid z$
$D \rightarrow d$

$A \rightarrow B \mid a \mid ABD \mid zBD$
$B \rightarrow C \mid b$
$C \rightarrow A \mid z$
$D \rightarrow d$

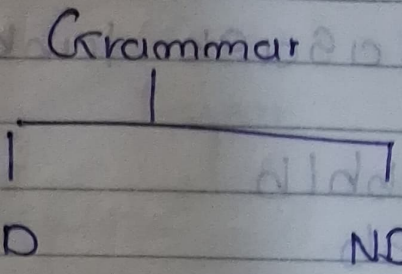
~~$A \rightarrow BA' \mid aA' \mid zBDA'$~~
$A' \rightarrow BDA' \mid \epsilon$
~~$B \rightarrow C \mid b$~~

$A \rightarrow C \mid b \mid ABD \mid zBD \mid a$
$A \rightarrow A \mid z \mid b \mid ABD \mid zBD \mid a$
$B \rightarrow C \mid b$
$C \rightarrow A \mid z$
$D \rightarrow d$

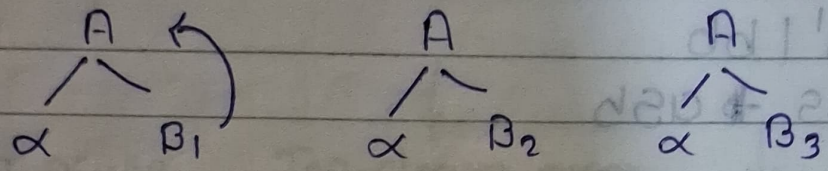
$A \rightarrow zA' \mid bA' \mid zBDA' \mid aA'$
$A' \rightarrow BDA' \mid \epsilon$
$B \rightarrow C \mid b$
$C \rightarrow A \mid z$
$D \rightarrow d$

→ Determimstic : Only one path for output

→ Non-Determimstic : Multiple path for output



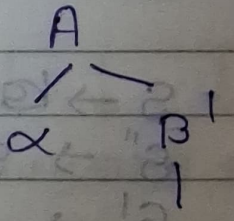
$A \to \alpha B_1 | \alpha B_2 | \alpha B_3$ (Left factoring)



No of backtracking occured (First Value same, so multiple option, backtracking occured)

Solution: $A \to \alpha B'$

$B' \to B_1 | B_2 | B_3$



Ex: $S \to iEtS | iEtSeS | a$

~~$S \to S'$~~ ~~$S' eS | a$~~ $S \to iEtSS' | a$

~~$S' \to iEtS$~~ $S' \to \epsilon | eS$

Ex: $A \to xByA | xByAzA | a$

$B \to b$

$A \to xByAA' | a$

$A' \to zA | \epsilon$

$B \to b$

Ex: $S \to aSSbS \mid aSaSb \mid abb \mid b$

$S \to aSS' \mid abb \mid b$
$S' \to SbS \mid aSb$

$S \to aS'' \mid b$
$S'' \to SS' \mid bb$
$S' \to SbS \mid aSb$

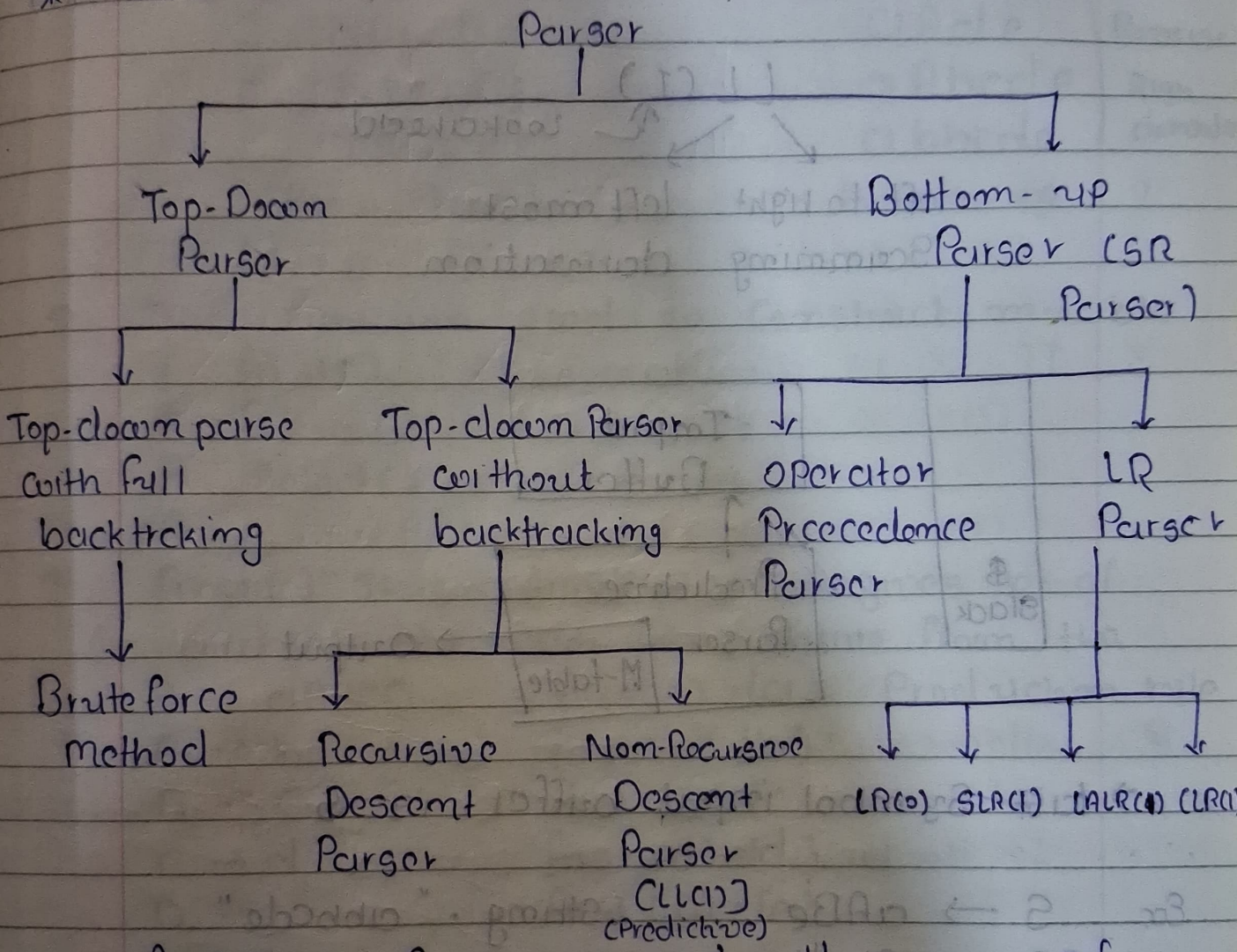
Ex $S \to bSSaaS \mid bSSaSb \mid bSb \mid a$

$S \to bSSaS' \mid bSb \mid a$
$S' \to aS \mid Sb$

$S \to bSS'' \mid a$
$S'' \to SaS' \mid b$
$S' \to aS \mid Sb$

Page
Date

* Parser :



→ Ambiguos Grammar not allow any of Parser.

→ Operator Precedence parser only allow ambiguos grammar.

→ LR : Left to Rigt Reduce Right most derivation

→ Top down : starting Symbol to given string
Bottom up : given string to starting Symbol

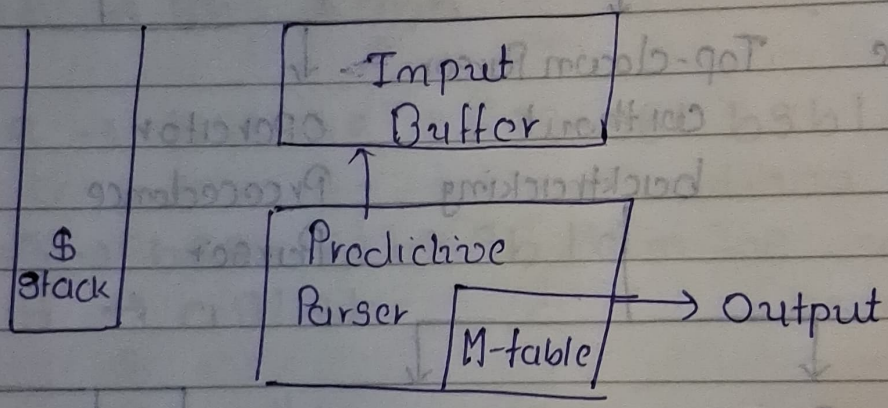
Top-down → when to use
bottom-up → What to use

* Predictive Parser (LL(1)) (Top-down Method)

LL(1) ← lookahead

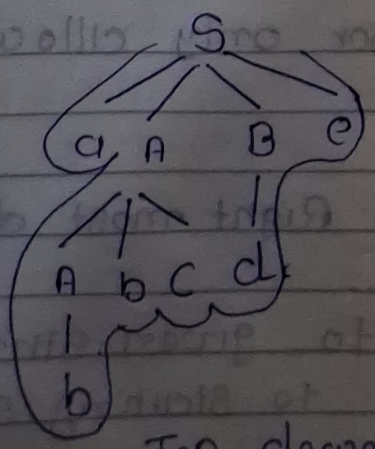
left to right Scanning

left most derivation

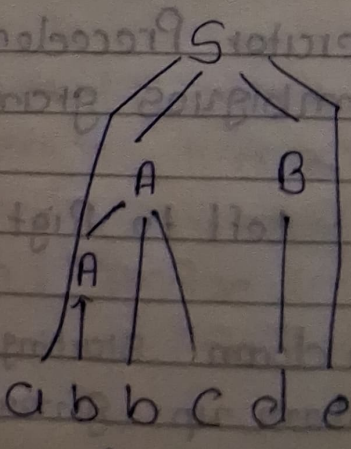


→ last Symbol input buffer is $

Ex S → aABe String : "abbcde"
A → Abc|b
B → d



Top-down



bottom-up

↳Top-down approch follow left-most derivation.
↳Bottom-up approch follow Righ-most derivation in Reverse order.

$S \rightarrow aABe$	$S \rightarrow aABe$
$aAbcBe$	$aAde$
$abbcBe$	$aAbcde$
$abbcde$	$abbcde$
Top-down	bottom-up

Reverse Righ derivation

⇒ Three function used to construct m-table
1. first()
2. follow()
3. Select()

1. First(): It is a function which gives set of terminals that begin string derive from the

Ex: $S \rightarrow aABCD$ first(S) = {a} Production rule
$A \rightarrow b$ first(A) = {b}
$B \rightarrow c$ first(B) = {c}
$C \rightarrow d$ first(C) = {d}
$D \rightarrow e$ first(D) = {e}

Ex: $S \rightarrow AB$ first(S) = {b}
$A \rightarrow b$ first(A) = {b}
$B \rightarrow c$ first(B) = {c}

Ex: $S \rightarrow AB$ first(S) = {b, c}
$A \rightarrow b \mid \epsilon$ first(A) = {b, ε}
$B \rightarrow c$ first(B) = {c}

↳ Starting Symbol (S) follow(S) always contain {$}.

⇒ Algorithm: First ( )

1 If x is terminal, First(x) is {x}

2 If $x \to \epsilon$ is production, then add ϵ to set of first(x).

3 If x is non-terminal,
(a) $x \to Y$, first(Y) is an element of the set of first(x)
(b) If first(Y) has ϵ as an element and $x \to YZ$ then,
First(X) = First(Y) − {ε} ∪ First(Z)

2 Follow(): It is function which gives set of terminal that can be appered immediately to the right of given symbol

Ex: $S \to ABCD$
$A \to b \mid \epsilon$
$B \to C$
$C \to d$
$D \to e$

follow(A) = First(BCD) = {c}
follow(B) = {d}
follow(C) = {e}
follow(D) = follow(S)

follow(S) = {$}

Page
Date

Exc: $S \to AB$ follow(S) = {$}
$A \to a \mid \epsilon$ follow(A) = {a, b, $}
$B \to b \mid \epsilon$ follow(B) = $

⇒ Algorithm: follow()

1. $ is an element of follow(S), where S is starting symbol and $ indicate the end of Input

2 if $A \to \alpha X Y$ is a production, the set First(Y) is in set follow(X), excluding ϵ.

3 If $A \to \alpha X$ or $A \to \alpha X Y$ are production, the First(Y) has an element ϵ, the set follow(X) is in set follow(A)

Exc: 1 $E \to TA$ First(E) = { (, id }
$A \to +TA \mid \epsilon$ First(A) = { +, ε }
$T \to FB$ First(T) = { (, id }
$B \to *FB \mid \epsilon$ first(B) = { *, ε }
$F \to (E) \mid id$ First(F) = { (, id }

Follow(E) = {$,)}
Follow(A) = {$,)}
Follow(T) = {+, $,)}
Follow(B) = {+, $,)}
follow(F) = {+, *,), $}

Ex: $S \to AaAb \mid BbBa$
$A \to \epsilon$
$B \to \epsilon$

first (S) = {a, b}
first (A) = {ϵ}
first (B) = {ϵ}

Follow (S) = {$, $, $}
follow (A) = {a, b}
follow (B) = {a, b}

Ex:2 $S \to aBDh$
$B \to cC$
$C \to bC \mid \epsilon$
$D \to EF$
$E \to g \mid \epsilon$
$F \to f \mid \epsilon$

First (S) = {a}
first (B) = {c}
first (C) = {b, ϵ}
first (D) = {g, f, ϵ}
first (E) = {g, ϵ}
first (F) = {f, ϵ}

follow (S) = {$}
follow (B) = {g, f, h}
follow (C) = {g, f, h}
follow (D) = {h}
follow (E) = {f, h}
follow (F) = {h}

Exc: $S \rightarrow ABBA$
$A \rightarrow a \mid \epsilon$
$B \rightarrow b \mid \epsilon$

$First(S) = \{a, b, \epsilon\}$ $follow(S) = \{\$\}$
$First(A) = \{a, \epsilon\}$ $follow(A) = \{b, a, \$\}$
$first(B) = \{b, \epsilon\}$ $follow(B) = \{a, b, \$\}$

Exc $S \rightarrow aSe \mid B$
$B \rightarrow bBe \mid C$
$C \rightarrow cCe \mid d$

$first(S) = \{a, b, c, d\}$ $follow(S) = \{\$, e\}$
$first(B) = \{b, c, d\}$ $follow(B) = \{\$, e\}$
$first(C) = \{c, d\}$ $follow(C) = \{\$, e\}$

3 Select():

$A \rightarrow \omega$
$Select(A \rightarrow \omega) = First(First(\omega) \cdot follow(A))$

Exc: Example 1 in follow

$Select(\epsilon \rightarrow TA) = First(First(TA) \cdot follow(\epsilon))$
$= First(\{c, id\} \times \{\$,)\})$
$= \{c, id\}$

Select (A → +TA) = First (First (+TA) × follow(A))

= First ({+} × {$,)}

= {+}

Select (A → ε) = First ({ε} × follow(A))

= First ({ε} × {$,)}

= {$,)}

Select (T → FB) = First (First (FB) · follow(T))

= First ({(, id} · {+, $,)}

= {(, id}

Select (B → *FB) = First ({*} · {+, $,)}

= {*}

Select (B → ε) = First ({ε} · {+, $,)}

= {+, $,)}

Select (F → (E)) = First ({(} · {+, *,), $})

= {(}

Select (F → id) = First ({id} · {+, *,), $})

= {id}

⇒ Construct m-table

	+	*	(	)	id	$
E			E→TA		E→TA	
T			T→FB		T→FB	
A	A→+TA			A→ϵ		A→ϵ
F			F→(E)		F→id	
B	B→ϵ		B→*FB		B→ϵ	B→ϵ

→ If any cell contain multiple production than it is not LL(1) grammar.

→ id + id * id accept or not

Stack	IIP String	Production Rule
$	id+id*id$	
E$	id+id*id$	
TA$	id+id*id$	E→TA
FBA$	id+id*id$	T→FB
idBA$	id+id*id$	F→id
BA$	+id*id$	
A$	+id*id$	B→ϵ
+TA$	+id*id$	A→+TA
TA$	id*id$	
FBA$	id*id$	T→FB
idBA$	id*id$	F→id
BA$	*id$	
*FBA$	*id$	B→*FB

FBA$	id$	
idBA$	id$	F→id
BA$	$	
A$	$	B→ε
$	$	A→ε

String is accepted.

Exc:2 Comstruct m table. check "acbbgfh" String accept or not.

→ Select (S→aBDh) = first (first (aBDh). follow(S)
= {a}

→ Select (B→cC) = first (first (cC) · follow(B)
= {c}

→ Select (C→bC) = first (first (bC) · follow(C)
= {b}

→ Select (C→ε) = first (first (ε) · follow(C)
= {g, f, h}

→ Select (D→EF) = first (first (EF) · follow(D)
= {g, f, h}

FBA$	id$	
idBA$	id$	F → id
BA$	$	
A$	$	B → ε
$	$	A → ε

String is accepted.

Ex:2 Construct m table. check "acbbgfh" String accept or not.

→ Select (S → aBDh) = first (first (aBDh) · follow(S))
= {a}

→ Select (B → cC) = first (first (cC) · follow(B))
= {c}

→ Select (C → bC) = first (first (bC) · follow(C))
= {b}

→ Select (C → ε) = first (first (ε) · follow(C))
= {g, f, h}

→ Select (D → EF) = first (first (EF) · follow(D))
= {g, f, h}

→ Select (E → g) = {g}

Select (E → ϵ) = {f, h}

Select (F → f) = {f}

Select (F → ϵ) = {h}

	a	b	c	f	g	h	$
S	aBDh						
B			cC				
C		bC		ϵ	ϵ	ϵ	
D				EF	EF	EF	
E				ϵ	g	ϵ	
F				f		ϵ	

Stack	Imput string	Production
$		
S $	acbbgfh $	
aBDh$	acbbgfh $	S → aBDh
BDh$	cbbgfh $	
cCDh$	cbbgfh $	B → cC
CDh$	bbgfh$	
bCDh$	bbgfh $	C → bC
CDh$	bgfh $	
bCDh$	bgfh $	C → bC
CDB $	gfh $	
Dh$	gfh $	C → ϵ

EFh$	gfh$	D → EF
gFh$	gfh$	E → g
Fh$	fh$	
fh$	fh$	F → f
h$	h$	
$	$	

→ String is accepted.

Exc:3 S → (L) | a
L → L,S | S

Construct m table and check following is accepted? (a,(a,a))

1. Remove Left Recursion

S → (L) | a
L → SL'
L' → ,SL' | ∈

First(S) = {(, a} follow(S) = {$, ,,)}
First(L) = {(, a} follow(L) = {)}
First(L') = {, , ∈} follow(L') = {)}

Select (S → (L)) = First (First[(L)] · follow (S)
= {(}

Select (S → a) = {a}

Select (L → SL') = First (First (SL') · follow(L))
= {(, a}

Select (L' → , SL') = first (first (, SL') · follow(L')
= {,}

Select (L' → ∈) = {)}

	(	a	)	,	$
S	(L)	a			
L	SL'	SL'			
L'			∈	,SL'	

Stack	IIP String	Production
$		
S$	(a,(a,a))$	
(L)$	(a,(a,a))$	S → (L)
L)$	a,(a,a))$	
SL')$	a,(a,a))$	L → SL'
aL')$	,(a,a))$	S → a
L')$	,(a,a))$	
,SL')$	,(a,a))$	L' → ,SL'
SL')$	(a,a))$	

Top-down: Expand the non terminal to terminal

Bottom-up: Reduce the terminal to non-terminal upto starting symbol

(L)L') $	(a,a))$	S→(L)
L)L') $	a,a))$	
SL')L')$	a,a))$	L→SL'
aL')L')$	a,a))$	S→a
L')L')$	,a))$	
,SL')L')$	,a))$	L'→,SL'
SL')L')$	a))$	
aL')L')$	a))$	S→a
L')L')$	))$	
)L')$	))$	L'→ϵ
L')$	)$	
)$	)$	L'→ϵ
$	$	

String accepted.

⇒ Note:

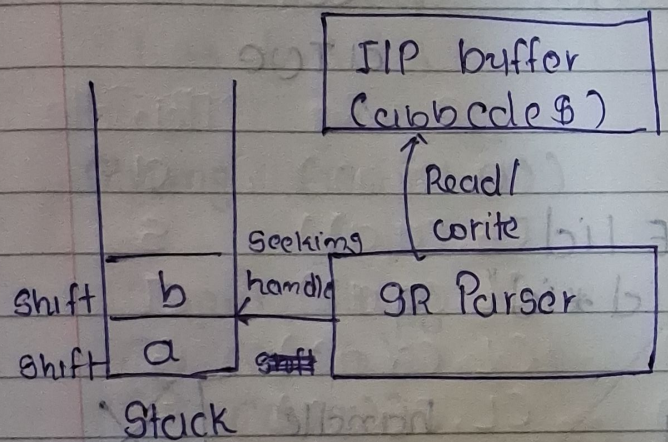
1. Check first grammer is left Recursion then remove left recursion
2. Check grammer is left factoring, then remove it

[Non-determinstic grammer is left factor to Convert determinstic first remove left factoring

* Bottom-up Parser

SR Parser

1. Shift 2. Reduce (4 - operation / function)
(Push) (Pop)
3. Accept : String accept
4. Error : Error in parsing



$S \rightarrow aTUe$
$T \rightarrow Tbc \mid b$
$U \rightarrow d$

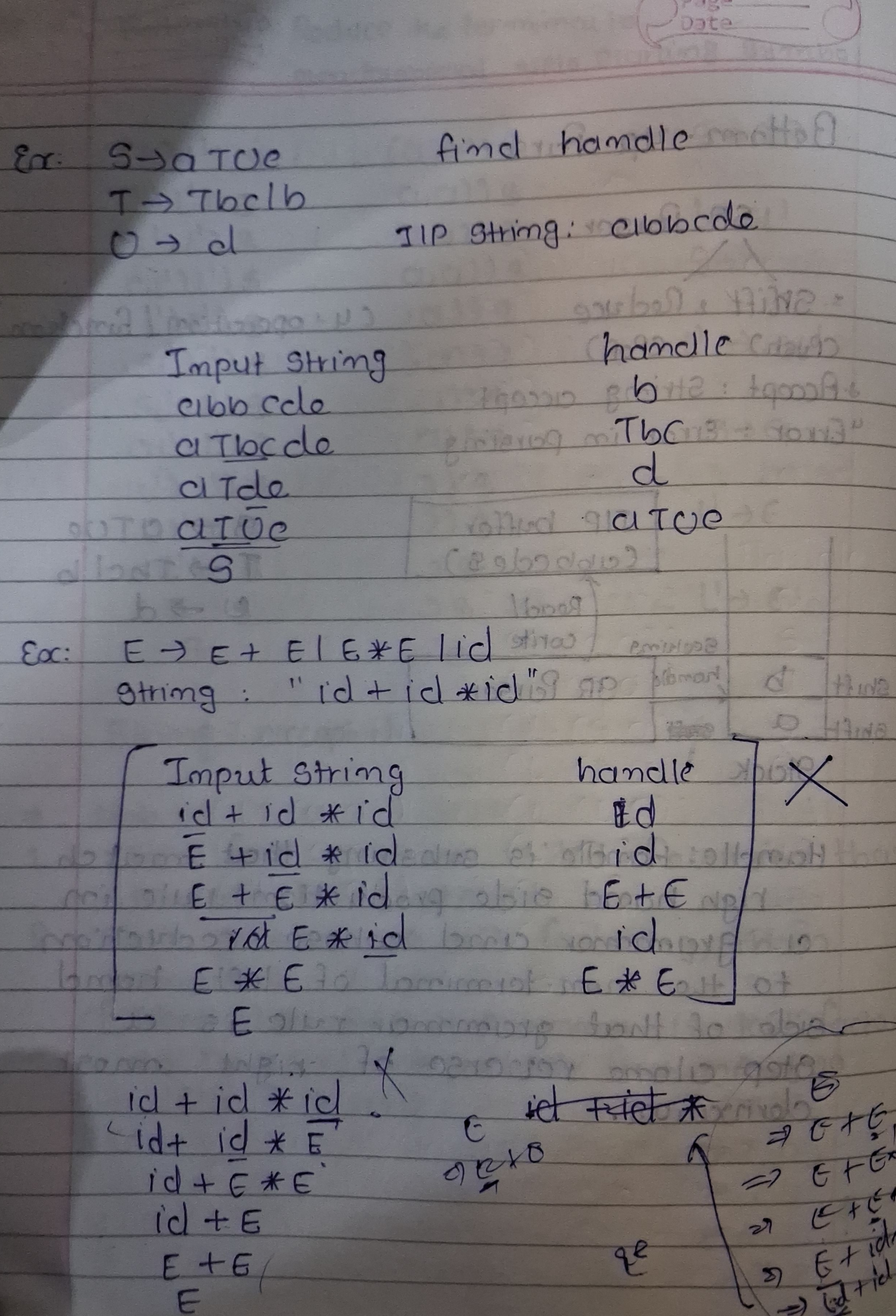
Handle: Handle is substring that match righ hand side production rule in a grammar and whose reduction to the non-terminal of left hand side of that grammar rule is a step alone reverse of right most derivation.

Ex: S → a T U e find handle
T → T b c | b
U → d IP string: a b b c d e

Input string	handle
a b b c d e	b
a T b c d e	T b c
a T d e	d
a T U e	a T U e
S	

Ex: E → E + E | E * E | id
string : "id + id * id"

Input string	handle
id + id * id	id
E + id * id	id
E + E * id	E + E
E * id	id
E * E	E * E
E	



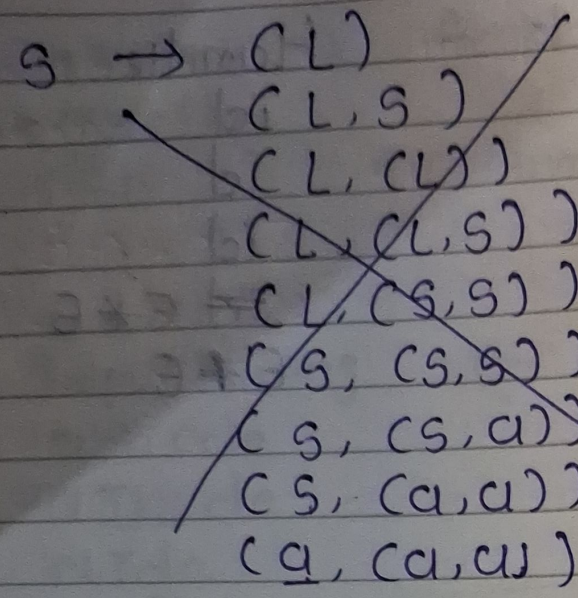
Input string	Handle
id + id * id	id
E + id * id	id
E + E * id	id
E + E * E	~~id~~ E * E
E + E	E * E
E	

Ex: $S \rightarrow (L) \mid a$

$L \rightarrow L, S \mid S$

String: (a, (a, a))

Input String	Handle
(a, (a, a))	a
(S, (a, a))	a
(S, (S, a))	a
(S, (S, S))	S
(L, (S, S))	S
(L, (L, S))	(L, S)
(L, (L))	(L)
(L, S)	(L, S)
(L)	(L)
S	



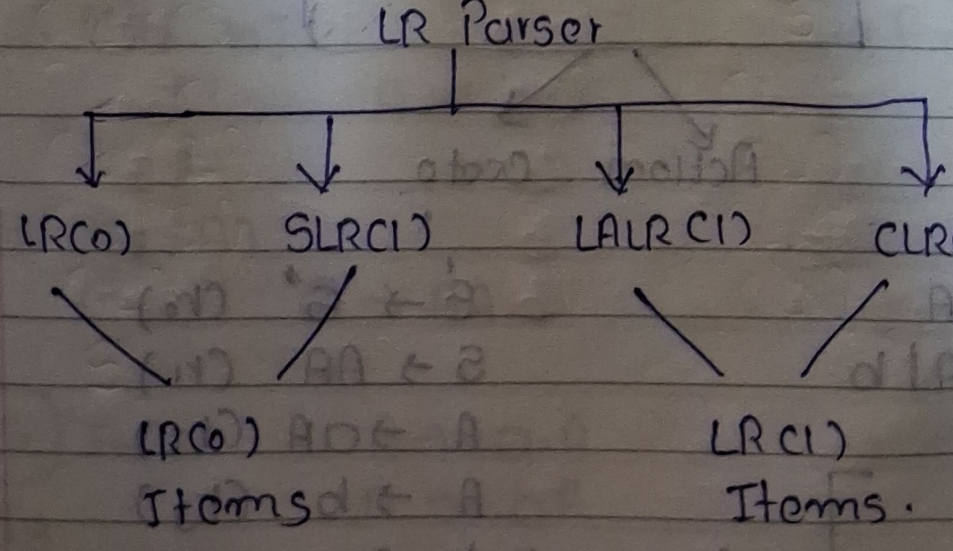
S → (L)	I/P string	Handle
(L,S)	(a,(a,a))	a
(L,(L))	(S,(a,a))	S
(L,(L,S))	(L,(a,a))	a
(L,(L,a))	(L,(S,a))	S
(L,(S,a))	(L,(L,a))	S
(L,(a,a))	(L,(L,S))	L,S
(S,(a,a))	(L,(L))	(L)
(a,(a,a))	(L,S)	(L,S
	(L)	(L).
	S	

Ex: E → E+E / E*E / (E) / id

String: id * (id+id) $ Accep or not

Stack	I/P String	Action
$	id * (id+id) $	Shift id
$id	* (id+id) $	Reduce E→id
$E	* (id+id) $	Shift *
$E*	(id+id) $	Shift (
$E*(	id+id) $	Shift id
$E*(id	+id) $	Reduce E→id
$E*(E	+id) $	Shift +
$E*(E+	id) $	Shift id
$E*(E+id	) $	Reduce E→id
$E*(E+E	) $	Reduce E→E+E
$E*(E	) $	Shift)
$E*(E)	$	Reduce E→(E)
$E*E	$	Reduce E→E*E
$E	$	Accept

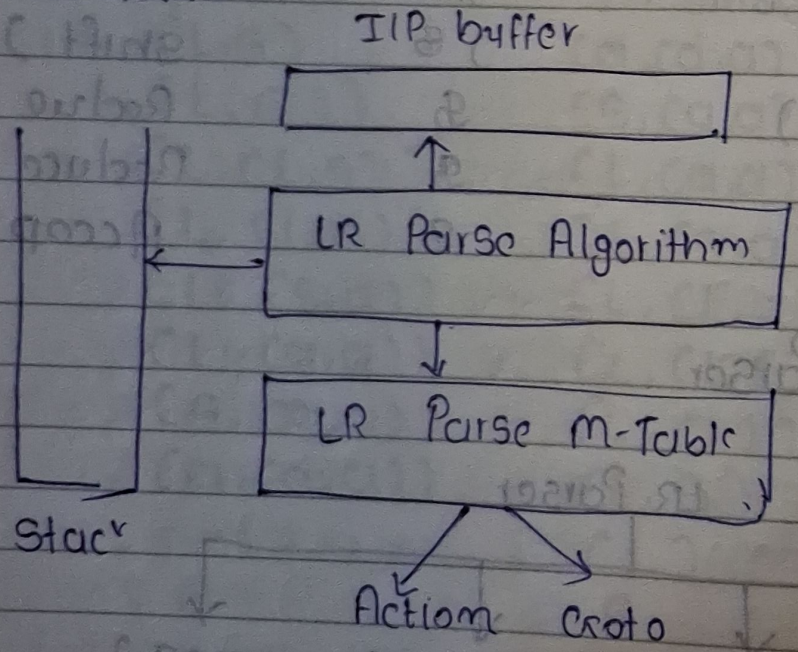
* LR Parser



If we put "." in production. its called items.

$S \to \cdot XYZ$ (Items)
$X \cdot YZ$
$XY \cdot Z$
$XYZ \cdot$ ← Entire seen production rule then we perform Reduce operation

⇒ Architecture:



Exc: $S \to AA$
$A \to aA \mid b$

$S' \to S^{*}$ (r_0)
$S \to AA$ (r_1)
$A \to aA$ (r_2)
$A \to b$ (r_3)

$S' \to S$
$S \to AA$
$A \to aA \mid b$
→ Augmented Grammar

→ I_0 : $S' \to .S$
$S \to .AA$
$A \to .aA$
$A \to .b$

→ I_1 : (I_0, S)
$S' \to S.$

→ I_2 : (I_0, A)
$S \to A.A$
$A \to .aA$
$A \to .b$

→ I_3 : (I_0, a)
$A \to a.A$
$A \to .aA$
$A \to .b$

→ I_4 (I_0, b)
$A \to b.$

→ I_5 (I_2, A)
$S \to AA.$

→ I_6 (I_2, a) : I_3
$A \to a.A$
$A \to .aA$
$A \to .b$

→ I_6 : (I_2, b) ⇓ I_4
$A \to b.$

→ I_6 : (I_3, A)
$A \to aA.$

→ I_7 : (I_3, a) I_3
$A \to a.A$
$A \to .aA$
$A \to .b$

→ I_7 : (I_3, b) ⇓ I_4 $A \to b.$

→ m-table :

	Action			Goto	
	a	b	$	S	A
I_0	S_3	S_4		1	2
I_1			Accept/r_0		
I_2	S_3	S_4			5
I_3	S_3	S_4			6
I_4	r_3	r_3	r_3		
I_5	r_1	r_1	r_1		
I_6	r_2	r_2	r_2		

String ← aabb$

Stack	Input buffer	Action
$0	aabb$	S_3
$0a3	abb$	S_3
$0a3a3	bb$	S_4
$0a3a3b4	b$	r_3
$0a3a3A6	b$	r_2
$0a3A6	b$	r_2
$0A2	b$	S_4
$0A2b4	$	r_3
$0A2A5	$	r_1
$0S1	$	Accept

Eoc: $S \to dA \mid aB$
$A \to bA \mid c$
$B \to bB \mid c$

$S' \to S$
$S \to dA \mid aB$
$A \to bA \mid c$
$B \to bB \mid c$

r_0 $S' \to S$
r_1 $S \to dA$
r_2 $S \to aB$
r_3 $A \to bA$
r_4 $A \to c$
r_5 $B \to bB$
r_6 $B \to c$

I_0: $S' \to .S$
$S \to .dA$
$S \to .aB$

I_1 : (I_0, S)
$S' \to S.$

I_2: (I_0, d)
$S \to d.A$
$A \to .bA$
$A \to .C$

I_3 : (I_0, a)
$S \to a.B$
$B \to .bB$
$B \to .C$

I_4 : (I_2, A)
$S \to dA.$

I_5 : (I_2, b)
$A \to b.A$
$A \to .bA$
$A \to .C$

I_6 : (I_2, C)
$A \to C.$

I_7 : (I_3, B)
$S \to aB.$

I_8 : (I_3, b)
$B \to b.B$
$B \to .bB$
$B \to .C$

I_9 : (I3, C)

$B \to C.$

I_{10} : (I5, A)

$A \to bA.$

I_{11} : (I5, b)

I5 $A \to b.A$

$A \to .bA$

$A \to .C$

I_{11} : (I5, C)

I6 $A \to C.$

I_{11} : (I8, B)

$B \to bB.$

I_{12} : (I8, b)

I8

$B \to b.B$

$B \to .bB$

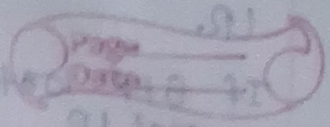
$B \to .C$

I_{12} : (I8, C)

I8

$B \to C.$

LR(0) coo can't check follow, it possible to take unnessary step



⇒ m-table

	Action					Goto		
	d	a	b	c	$	S	A	B
I_0	S_2	S_3				1		
I_1					Accept			
I_2			S_5	S_6			4	
I_3			S_8	S_9				7
I_4	r_1	r_1	r_1	r_1	r_1			
I_5			S_5	S_6			10	
I_6	r_4	r_4	r_4	r_4	r_4			
I_7	r_2	r_2	r_2	r_2	r_2			
I_8			S_8	S_9			11	
I_9	r_6	r_6	r_6	r_6	r_6			
I_{10}	r_3	r_3	r_3	r_3	r_3			
I_{11}	r_5	r_5	r_5	r_5	r_5			

"dbbba"

Stack	IP buffer	Action
$0	dbbbac$	$2
$0d2	bbbac$	$5
$0d2b5	bbac$	$5
$0d2b5b5	bac$	$5
$0d2b5b5b5	ac$	Error

"dbbbc"

multiple entry in table then not LR.

If SLR both together then not LR

Stack	I/P buffer	Action
$0	dbbbbc$	$_2
$0d2	bbbc$	$_5
$0d2b5	bbc$	$_5
$0d2b5b5	bc$	$_5
$0d2b5b5b5	c$	$_6
$0d2b5b5b5C6	$	r_4
$0d2b5b5b5A10	$	r_3
$0d2b5b5A10	$	r_3
$0d2b5A10	$	r_3
$0d2A4	$	r_1
$0S1	$	Accept

* SLR(1):

Eoc: $S \to AA$
$A \to aA|b$

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

	action			goto	
	a	b	$	S	A
I_0	S_3	S_4		1	2
I_1			accept		
I_2	S_3	S_4			5
I_3	S_3	S_4			6
I_4	r_3	r_3	r_3		
I_5			r_1		
I_6	r_2	r_2	r_2		

Ex: $E \to T + E \mid T$

$T \to i$

$S' \to E$ (r0)

$E \to T + E$ (r1) $\mid T$ (r2)

$T \to i$

I_0: $S' \to .E$

$E \to .T + E$

$T \to .i$

$E \to .T$

I_1: (I0, E)

$S' \to E.$

I_2: (I0, T)

$E \to T. + E$

$E \to T.$

I_3 (I0, i)

$T \to i.$

I_4: (I4, +)

$E \to T + .E$

$E \to .T + E$

$E \to .T$

$T \to .i$

I_5: (I4, E)

$E \to T + E.$

(I6): (I4, T)

I_2 $E \to T. + E$

$E \to T.$

(I6): (I4, i)

I_3 $T \to i.$

	Action			Goto	
	+	i	$	E	T
I_0		S_3		1	2
I_1			Accept		
I_2	S_4/r_2	r_2	r_2		
I_3	r_3	r_3	r_3		
I_4		S_3		5	2
I_5	r_1	r_1	r_1		

[Conflict so, not parsing LR(0)]

⇒ SLR(1)

follow(E) = {$}

follow(T) = {+, $}

	Action			Goto	
	+	i	$	E	T
I_0		S_3		1	2
I_1			Accept		
I_2	S_4		r_2		
I_3	r_3		r_3		
I_4		S_3		5	2
I_5			r_1		

Bottom-up not required to remove Left Recursion

Ex: $E \to E + T \mid T$
$T \to T * F \mid F$
$F \to (E) \mid id$

$E' \to E$
$E \to E + T \mid T$
$T \to T * F \mid F$
$F \to (E) \mid id$

$E' \to E$ r_0
$E \to E + T$ r_1
$E \to T$ r_2
$T \to T * F$ r_3
$T \to F$ r_4
$F \to (E)$ r_5
$F \to id$ r_6

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

I_1: (I_0, E)
$E' \to E.$

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

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

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

I_5: (I_0, F)
$F \to (.E)$
$E \to .E + T$
$E \to .T$
$T \to .T * F$
$T \to .F$
$F \to .(E)$
$F \to .id$

Is

I_4: (Io, id)

F → id.

Ic

I_5: (I_2, +)

E → E + . T

T → . T * F

T → . F

F → . (E)

F → . id

I_7

I_6: (I_3, *)

T → T * . F

F → . (E)

F → . id

I_8 I_4

I_{11}: (I_8, E)

F → (E .)

I_9: (I_6, T)

E → E + T .

I_{10} I_7

I_{12}: (I_6, F)

T → T * F .

I_{11} I_8

I_{10}: (I_9,)

F → (E) .

Follow(E) = { +,), $ }

follow(T) = { *, +,), $ }

Follow(F) = { *, +,), $ }

SR-Conflict (Shift-Reduce)

RR-Conflict (Reduce-Reduce)

⇒ LR(0)

	Action						Goto		
	+	*	(	)	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	r_2	r_2	r_2			
I_3	r_4	r_4	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	r_6	r_6			
I_6			S_4		S_5			9	3
I_7			S_4		S_5				10
I_8				S_{11}					
I_9	r_1	r_1	r_1	r_1	r_1	r_1			
I_{10}	r_3	r_3	r_3	r_3	r_3	r_3			
I_{11}	r_5	r_5	r_5	r_5	r_5	r_5			

⇒ SLR(1)

	+	*	(	)	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_{11}					
I_9	r_1			r_1		r_1			

SR - Conflict (Shift - Reduce)
RR - Conflict (Reduce - Reduce)

⇒ LR(0)

	Action						Goto		
	+	*	(	)	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	r_2	r_2	r_2			
I_3	r_4	r_4	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	r_6	r_6			
I_6			S_4		S_5			9	3
I_7			S_4		S_5				10
I_8				S_{11}					
I_9	r_1	r_1	r_1	r_1	r_1	r_1			
I_{10}	r_3	r_3	r_3	r_3	r_3	r_3			
I_{11}	r_5	r_5	r_5	r_5	r_5	r_5			

⇒ SLR(1)

	+	*	(	)	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_{11}					
I_9	r_1			r_1		r_1			
I_{10}	r_3	r_3		r_3		r_3			

Page
Date

Ex $S \to AaAb \mid BaBb$

$A \to \epsilon$

$B \to \epsilon$

$S' \to S \quad r_0$

$S \to AaAb \quad r_1 \mid BaBb \quad r_2$

$A \to \epsilon \quad r_3$

$B \to \epsilon \quad r_4$

I_0: $S' \to .S$
$S \to .AaAb$
$S \to .BaBb$
$A \to .$
$B \to .$

I_1 : (I_0, S)
$S' \to S.$

I_2 : (I_0, A)
$S \to A.aAb$

I_3 : (I_0, B)
$S \to B.aBb$

I_4 : (I_2, a)
$S \to Aa.Ab$
$A \to .$

I_5: (I_3, a)
$S \to Ba.Bb$
$B \to .$

I_6 : (I_4, A)
$S \to AaA.b$
|
I_7 : (I_5, B)
$S \to BaB.b$

I_8 : (I_6, b)
$S \to AaAb.$

I_9 : (I_7, b)
$S \to BaBb.$

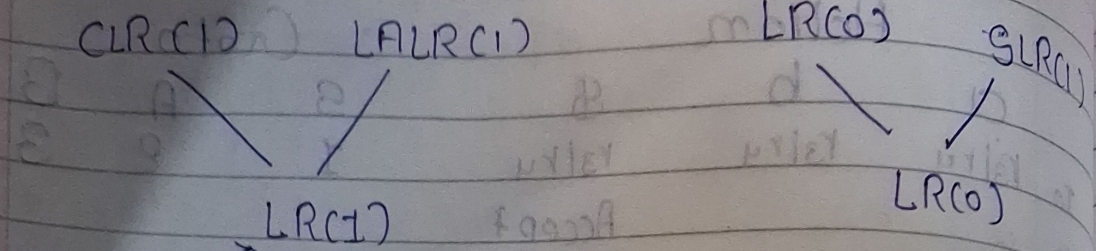
LR(0)

	Action			Goto		
	a	b	$	S	A	B
I_0	r_3/r_4	r_3/r_4	r_3/r_4	1	2	3
I_1			Accept			
I_2	S_4					
I_3	S_5					
I_4	r_3	r_3	r_3		6	
I_5	r_4	r_4	r_4			7
I_6		S_8				
I_7		S_9				
I_8	r_1	r_1	r_1			
I_9	r_2	r_2	r_2			

SLR(1)

	Action			Goto		
	a	b	$	S	A	B
I_0	r_3/r_4			1	2	3
I_1			Accept			
I_2	S_4					
I_3	S_5					
I_4	r_3					
I_5	r_4					
I_6		S_8			6	
I_7		S_9				7
I_8			r_1			
I_9			r_2			
π						

⇒



left to right

↑ look ahead

left most derivation reverse order

LR(0) + lookahead = LR(1)

$S \to .AA, \$$

↑ look ahead

CLR(1) → Reduction only in look ahead part

Ex: $S \to AA \; r_1$ $\quad S' \to S \; r_0$

$A \to aA \; r_2$

$A \to b \; r_3$

I_0: $S' \to .S, \$$

$S \to .AA, \$$

$A \to .aA, a/b$

$A \to .b, a/b$

I_1: $S' \to S., \$$

$I_2(I_0, A)$

$S \to A.A, \$$

$A \to .aA, \$$

$A \to .b, \$$

I_3: (I_0, a)

$A \to a.A, a/b$

$A \to .aA, a/b$

$A \to .b, a/b$

CLR not any conflict RR and SR
If any conflict than Grammar ambiguos.

I_4: (I_0, b)
$A \to b., a/b$

I_5: (I_2, A)
$A \to AA., \$$

I_6: (I_2, a)
$A \to a.A, \$$
$A \to .aA, \$$
$A \to .b, \$$

I_7: (I_2, b)
$A \to b., \$$

I_{10} I_6 (I_6, a)
$A \to a.A, \$$
$A \to .aA, \$$
$A \to .b, \$$

I_8: (I_3, A)
$A \to aA., a/b$

I_9 I_3: (I_3, a)
$A \to a.A, a/b$
$A \to .aA, a/b$
$A \to .b, a/b$

I_4
I_9 (I_3, b)
$A \to b., a/b$

I_9: (I_6, A)
$A \to aA., \$$

I_7
I_{10} (I_6, b)
$A \to b., \$$

	Action			Goto	
	a	b	$	S	A
I_0	S_3	S_4		1	2
I_1			Accept		
I_2	S_6	S_7			5
I_3	S_3	S_4			8
I_4	r_3	r_3			
I_5			r_1		
I_6	S_6	S_7			9
I_7			r_3		

[CLR > LR(0) = SLR(1) = LALR(1)] Iteam Set

↳ CLR more Powerfull otte Parsing

	a	b	$	S	A
I8	r2	r2			
I9			r2		

→ LALR(1) is shrinking of CLR(1) $\begin{cases} I_3 = I_6 = I_{36} \\ I_4 = I_7 = I_{47} \\ I_8 = I_9 = I_{89} \end{cases}$

⇒ Parsing Table for LALR(1)

	Action			Goto	
	a	b	$	S	A
I_0	S_3	S_4		1	2
I_1			Accept		
I_2	S_{36}	S_{47}			5
I_{36}	S_{36}	S_{47}			89
I_{47}	r_3	r_3	r_3		
I_5			r_1		
I_{89}	r_2	r_2	r_2		
~~I_{47}~~					
~~I_7~~					
~~I_{89}~~					
~~I_{89}~~					

Exc: r_1 r_2 r_3 r_4

$S \to Ba \mid bBc \mid dc \mid bda$ $\quad S' \to S \; r_0$

$B \to d \; (r_3)$

I_0 : $S' \to .S, \$$

$S \to .Ba, \$$

$S \to .bBC, \$$

$S \to .dC, \$$

$S \to .bda, \$$

$B \to .d, a$

I_1: (I_0, S)

$S' \to S.,\ \$$

I_2: (I_0, B)

$S \to B.a,\ \$$

I_3: (I_0, b)

$S \to b.BC,\ \$$

$B \to .d,\ c$

$S \to b.da,\ \$$

I_4: (I_0, d)

$S \to d.C,\ \$$

$B \to d.,\ a$

I_5: (I_2, a)

$S \to Ba.,\ \$$

I_6: (I_3, B)

$S \to bB.C,\ \$$

I_7: (I_3, d)

$B \to d.,\ C$

$S \to bd.a,\ \$$

I_8: (I_4, C)

$S \to dC.,\ \$$

I_9: (I_6, C)

$S \to aBC.,\ \$$

I_{10}: (I_7, a)

$S \to bda.,\ \$$

	Action				Goto	
	a	b	c	d	$	S	B
I_0		S_3		S_4		1	2
I_1					Accept		
I_2	S_5						6
I_3			S_7				
I_4	r_5		S_8				
I_5					r_1		
I_6			S_9				
I_7	S_{10}		r_5				
I_8					r_3		
I_9					r_2		
I_{10}					r_4		

Exc: $E \xrightarrow{r_1} E + T \;/\; T \xrightarrow{r_2}$ $\quad E' \to E \; (r_0)$

$T \xrightarrow{r_3} T * F \;/\; F \xrightarrow{r_4}$

$F \to id \; (r_5)$

I_0: $E' \to .E, \$$.
$E \to .E + T, \$ / +$
$E \to .T, \$ / +$
$T \to .T * F, \$ / + / *$
$T \to .F, \$ / + / *$
$F \to .id, \$ / + / *$

$I_1 \; (I_0, E)$
$E' \to E., \$$
$E \to E. + T, \$ / +$

I_2, (I_0, T)

E → T. , $ | +
T → T. * F, $ | + | *

I_3, (I_0, F)

T → F. , $ | + | *

I_4: (I_0, id)

F → id. , $ | + | *

I_5: (I_1, +)

E → E + . T , $ | +
T → . T * F , $ | + | *
T → . F , $ | + | *
F → . id , $ | + | *

I_6: (I_2, *)

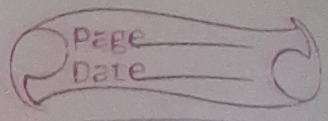
T → T * . F , $ | + | *
F → . id , $ | + | *

I_7: (I_5, T)

E → E + T. , $ | +
T → T. * F, $ | + | *

I_8: (I_5, F)

I_3 T → F. $ | + | *



(I8): (I5, id)
I4
F → id. , $/+/*

I8 : (I6, F)

T → T*F. , $/+/*

(I9): (I6, id)
I4
F → id. , $/+/*

(Ia): (I7, *)
I6
E → .E T → T*.F , $/+/*
F → .id , $/+/*

Ia

	Action				Goto		
	+	*	id	$	E	T	F
I_0			S_4		1	2	3
I_1	S_5			Accept			
I_2	r_2	S_6		r_2			
I_3	r_4	r_4		r_4			
I_4	r_5	r_5		r_5			
I_5			S_4			7	3
I_6			S_4				8
I_7	r_1			r_1			
I_8	r_3	r_3		r_3			

* Chomsky Hirechy :

⇒ 4 type of Grammar

1 type-0 : Unristricted Grammar
2 type-1 : Context Sensitive Grammar
3 type-2 : Context Free Grammar
4 type-3 : Regular Grammar

1. Type : 0

$$\alpha \rightarrow \beta$$

$\alpha \in (Vn \cup Vt)+$

$\beta \in (Vn \cup Vt)*$

e.x. $AabB \rightarrow cdF$

2. Type : 1

$Gr = \{Vn, Vt, S, P\}$

$\alpha \rightarrow \beta$,

where $|\alpha| \leq |\beta|$

and $\beta \neq Null$

ex: $S \rightarrow ab$ (Type 0 and Type 1)

$S \rightarrow \epsilon$ (Type 0 only)

3 Context Free Grammar

$\alpha \rightarrow \beta$,

$\alpha \in V_n$

$\beta \in (V_n \cup V_t)^*$

Left Side only 1 non terminal

4 Regular Grammar

Regular

Left Liner Right linear

$A \rightarrow WB$ (left most position Contain non terminal)

$A \rightarrow BW$ (Right most position Contain non terminal)

Ex. $S \rightarrow Sa$ (Left linear)

$S \rightarrow aS$ (Right linear)

Ex. $0(0|1)^*$ corite left linear and Right linear grammar

$S \rightarrow 0A$ (Right linear)
$A \rightarrow 0A|1A|\epsilon$

$S \rightarrow S0|S1|0$

Intermidiate
Code Optimization
Error detection
Symbol Table