Date: 09/04/2025

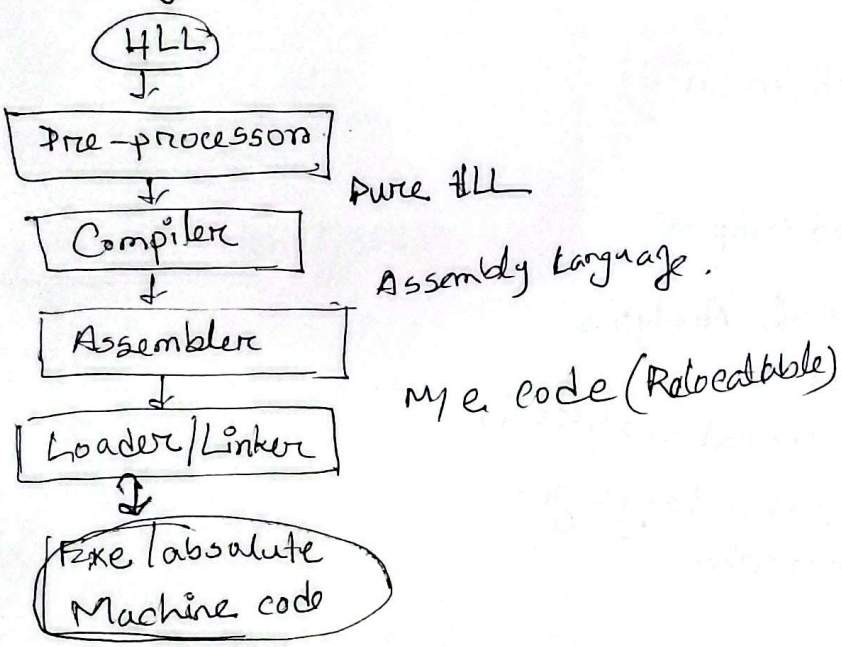
→ Introduction class.

Date: 13/04/25

"Compiler design".

→ Translator.

→ Byte code.
→ Machine code.
→ Execution of a Program.



* Phases or structure of a Compiler

Date: 20/04/25

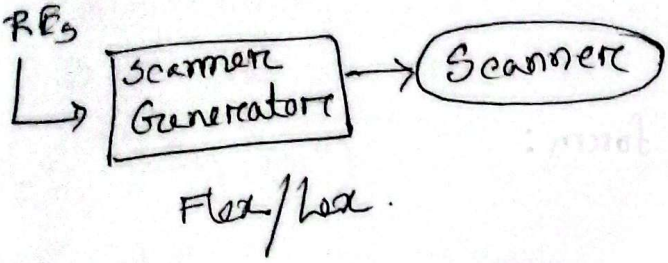
→ Lexical Analysis (Scanning)
→ Syntax Analysis (Parsing)
→ AST for b * b - 4 * a * c
(Abstract syntax tree)
→ Intermidiate code generation.
→ Code optimization.
→ Code generation phase.
→ Front-end, Back-end.
→ General structure of a compiler.
→ Conceptual structure: two major phases.
→ m x n compilers with m+n components.
→ Qualities of a Good Compiler.
→ Introduction of Lexical Analysis
- The big picture.
  - What does lexical analysis do?
  - Recognises the language's parts of speech.
- Language.
- Token, lexemes, Patterns.
- What is attribute for token.
- Why study lexical analysis
- Build a lexical analyzer by hand.
- Scanners Generator.

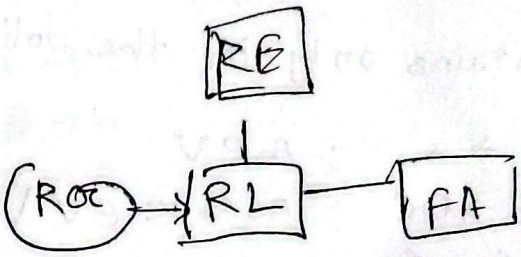
Date: 24/4/25

Lexical structure of tokens.

RE
↓
lexical analyzer (FSM)
↓
FSA's

FSA's ⟺ RE, RG ⟺ FSA's





Context-Free-Grammer's —

Backus-Norem Form (BNF)

Natural → SVO
↓
SOV

E → E+E / E*E

Chomskay-Normal Form (CNF)

4-tupple (CFG)

G = (V, Σ, δ, S)

CYK → Kassami
↓
Cocke Younger

Date: 27/04/25

CFGr BNF

A special way to write CFGr introduced by Noam chomsky is called chomsky Normal form. (CNF)

A CFGr say 'Gr' is in CNF say Gr'

$Gr = (V, \Sigma, R, S)$
$Gr' = (V', \Sigma, R', S')$

$Gr = (V, \Sigma, R, S)$

R contains only in the following form:

(i) $A \to a$; $A \in V$
(ii) $A \to BC$; $A, B, C \in V \setminus \{S\}$
(iii) $S \to \varepsilon$

$$\frac{\alpha \to \beta}{\alpha \in V \cdot \beta \in (V \cup \Sigma)^*}$$

→ "Bad Rules"

1. $A \to wSw$; $A \in V$, $w \in (V \cup \Sigma)^*$
   "start symbol rule".

2. $A \to \varepsilon$; $A \in V \setminus \{S\}$
   "ipsilon rule".

3. $A \to B$; $A, B \in V$
   "Unit Rule"

4. $A \to w$; $A \in V$ and $w \in (V \cup \Sigma)^*$; w must contain one terminal and one nonterminal
   "mixed rule"

5. $A \to w$; $w \in V^*$ and $|w| > 2$
   "long-rule"

(i) $A \to a$
(ii) $A \to BC$
(iii) $S \to \varepsilon$

→ Folbw না violat হবে।

Transformation of CFGs to CNF :—

R:
$S \to ASA \mid aB$
$A \to B \mid S$
$B \to b \mid \epsilon$

R_1 : ("Fixing start symbol rule")
$S_1 \to S$
$S \to ASA \mid aB$
$A \to B \mid S$
$B \to b \mid \epsilon$

R_2 : ("Fixing B")
$S_1 \to S$
$S \to ASA \mid aB \mid a$
$A \to S \mid B \mid \epsilon$
$B \to b$

R_3 : ("Fixing A")
$S_1 \to S$
$S \to ASA \mid AS \mid SA \mid aB \mid a$
$A \to S \mid B$
$B \to b$

R_4 : ("Fixing Units")
$S_1 \to ASA \mid AS \mid SA \mid aB \mid a$
$S \to ASA \mid AS \mid SA \mid aB \mid a$
$A \to ASA \mid AS \mid SA \mid aB \mid a \mid b$
$B \to b$

R_5 : ("Fixing mix rule")
$S_1 \to ASA \mid AS \mid SA \mid XB \mid a$
$S \to ASA \mid AS \mid SA \mid XB \mid a$
$A \to ASA \mid AS \mid SA \mid XB \mid a$
$B \to b$
$X \to a$

$A \to BCDE$
$A \to BY_1$
$Y_1 \to CY_2$
$Y_2 \to DE$

→ Breaking "Long Rule"

R_6 : ("Fixing Long rule")
$S_1 \to AY \mid AS \mid SA \mid XB \mid a$
$S \to AY \mid AS \mid SA \mid XB \mid a$
$A \to AY \mid AS \mid SA \mid XB \mid a \mid b$
$Y \to SA$
$B \to b$
$X \to a$

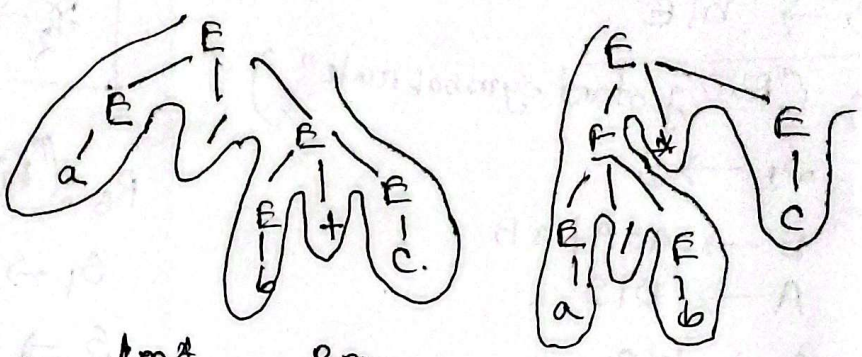
04/05/2025

Syntax analyzer / Parser:

(a) If check the stream of tokens is syntactically correct or not.

a+/b;
id+/id;

(b) a/b*c;



Derivation denoted by : $\overset{lm}{\Rightarrow}$ or $\overset{lm*}{\Rightarrow}$ or $\overset{Rm}{\Rightarrow}$.
is a process to confirm if a string is form a language.

1. E → I.
2. E → E+E
3. E → E*E
4. E → (E)
5. I → a
6. I → 5
7. I → Ia
8. I → Ib
9. I → I0
10. I → I1

language.

a*(a+b00) — এই string language এর মধ্যে আছে কিনা confirm

(lm → left most) (Rm → Right most)

E ⇒ E*E
$\overset{lm}{\Rightarrow}$ I*E
$\overset{lm}{\Rightarrow}$ a*E
$\overset{lm}{\Rightarrow}$ a*(E)
⇒ a*(E+E)
⇒ a+(I+E)
⇒ a*(a+1)
⇒ a*(a+I0)
⇒ a*(a+I00)
⇒ a*(a+b00)

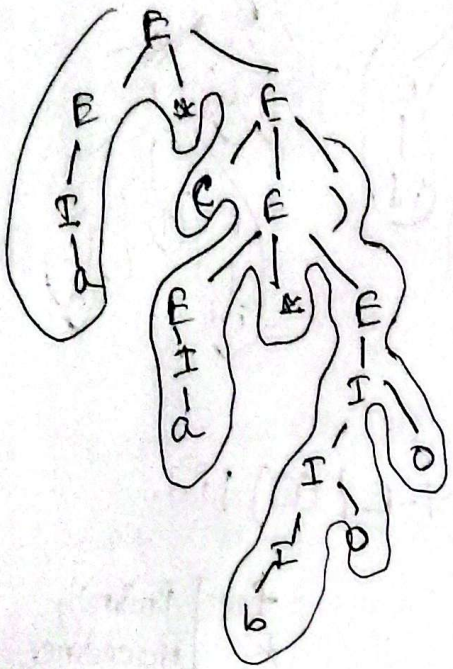
$E \overset{Rm}{\Rightarrow} E*E$
⇒ E*(E)
⇒ E*(E+E)
⇒ E*(E+I)
⇒ E*(E+I0)
⇒ E*(E+I00)
⇒ E*(E+b00)
⇒ E*(I+b00)
⇒ E*(a+b00)

left most sentential form ⇒ I*(a+b00)
⇒ a*(a+b00)

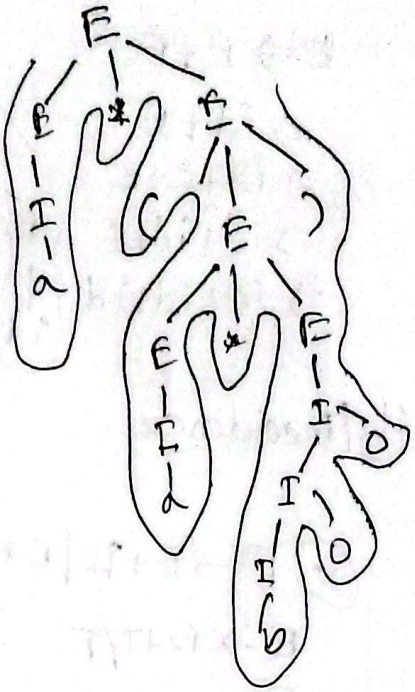
Nm terminals and/or terminal

Right sentential form

LM



RM



Date: 08/05/25

Ambiguous Grammer :

$E \rightarrow EAE \mid (E) \mid id.$

$A \rightarrow + / * / \uparrow$

" id + id * id "

$E \overset{lm}{\Rightarrow} EAE$

$\Rightarrow EAEAE$

$\Rightarrow idAEAE$

$E \overset{lm}{\Rightarrow} E+E$

$\Rightarrow id+E$

$\Rightarrow id+E*E$

$\Rightarrow id+id*E$

$\Rightarrow id+id*id$

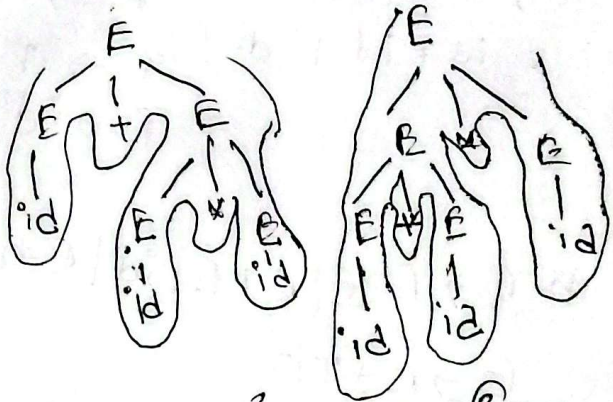
$E \Rightarrow E*E$

$= E+E*E$

$= id+E*E$

$= id+id*E$

$= id+id*id.$

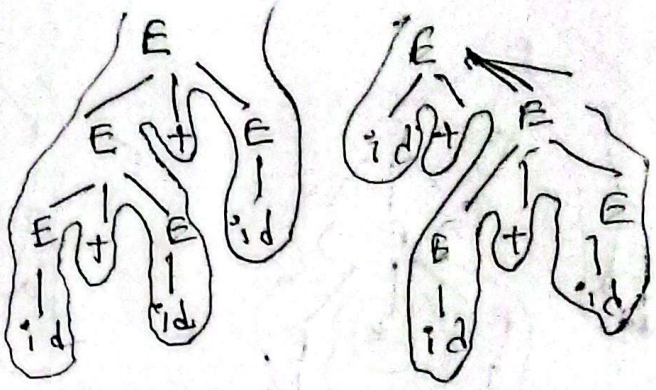


একটি string এর চেয়ে বা তার-অধিক parse tree হলে সেটি Ambiguous Grammer হবে।

"id+id+id"

$E \Rightarrow E+E$
$\Rightarrow E+E+E$
$\Rightarrow id+E+E$
$\Rightarrow id+id+E$
$\Rightarrow id+id+id$

$E \Rightarrow E+E$
$\Rightarrow id+E$
$\Rightarrow id+E+E$
$\Rightarrow id+id+E$
$\Rightarrow id+id+id$



priority/Precidence.

$E \to E+T \mid T$
$T \to id.$

$E \Rightarrow E+T$
$\Rightarrow E+E+T$
$\Rightarrow T+E+T$
$\Rightarrow id+T+T$
$\Rightarrow id+id+T$
$\Rightarrow id+id+id.$

(*) $E \to E+E \mid E*E \mid E\uparrow E \mid -E \mid (E) \mid id.$

$E \to E+T/T$
$T \to T*F \mid F$
$F \to -F/id.$

+ , * , ↑ — Priority / Precedence.

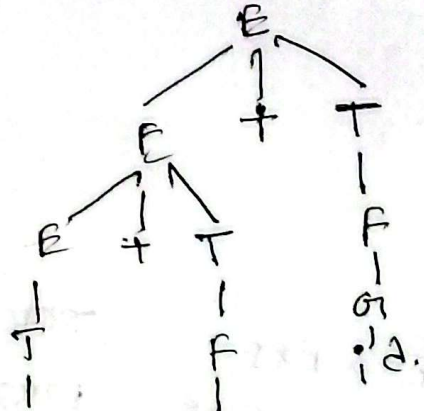
Unimus

(*) $E \to E+E \mid E\uparrow E \mid id$

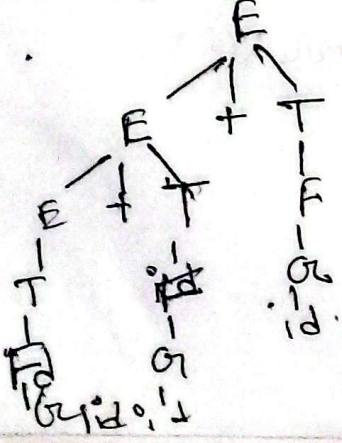
$E \Rightarrow E+T \mid T$
$T \to T\uparrow F \mid F$
$F \to id.$

(*) $E \to E+E \mid E\uparrow E \mid id \mid E*E$

$E \to E+T \mid T$
$T \to T*F \mid F$
$F \to G\uparrow F \mid G$
$G \to id.$



$E \overset{lm}{\Rightarrow} E+T$
$\Rightarrow E+F$
$\Rightarrow E+id$
$\Rightarrow E+T+id$
$\Rightarrow E+id+id$
$\Rightarrow T+id+id$
$\Rightarrow id+id+id.$



BISON

RE → Lex → Scanner

CFG → Bison → Parser

$S \to aAC$

$A \to Ab \mid \varepsilon$

$L = \{ab^n c \mid n \geq 0\}$

- SDT

Date: 15/05/25

- $R \to R+R \mid RR \mid R^* \mid a \mid b \mid \varepsilon$

$\Sigma = \{p\}$ $r_1 = a$, $r_2 = b$

E.P

$r = r_1 + r_2 = a+b$ (a | b)

$r = r_1 r_2 = ab$

$R \to R+R \mid RR \mid R^* \mid a \mid b \mid \varepsilon$

$R \to R+T \mid T$

$T \to TF \mid F$

$F \to F^* \mid a \mid b \mid \varepsilon$

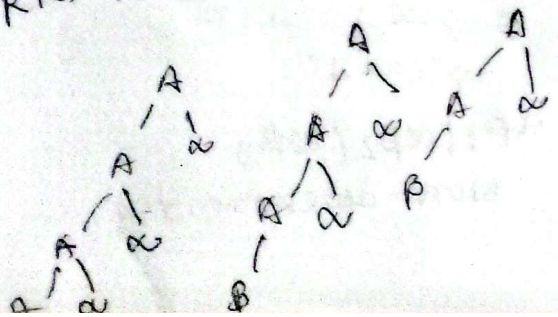
RGL

2R → RR

$L = \{\beta\alpha^n \mid n \geq 0\}$

2R: $A \to A\alpha \mid \beta$

RR: $A \to \alpha A \mid \beta$



Elimination of Left recursion:

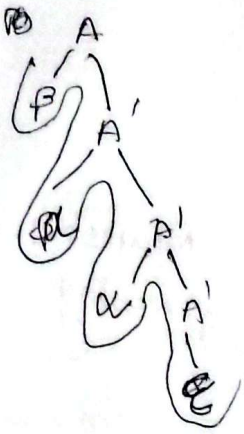
$A \to A\alpha \mid \beta$ "$\beta\alpha\alpha$"

$A \to \beta A'$

$A' \to \alpha A' \mid \varepsilon$

$L = \beta\alpha^*$

$L = \alpha^*\beta$



$E \to E+T \mid T$

$T \to id$

$E \to TE'$

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

$T \to id$

$R \to TR'$

$R' \to +TR' \mid \varepsilon$

$T \to FT'$

$T' \to FT' \mid \varepsilon$

$F \to F^* \mid a \mid b \mid \varepsilon$

$\alpha, \beta \in (V \cup T)^*$

$A \to \beta A'$

$A' \to \alpha A' \mid \varepsilon$

$A \to A\alpha_1 \mid \beta_1 \mid A\alpha_2 \mid \beta_2 \ldots$

$A \to A\alpha_1 \mid A\alpha_2 \mid \ldots A\alpha \mid \beta_1 \mid \beta_2 \ldots \beta_n$

$A \to \beta_1 A' \mid \beta_2 A' \ldots \beta_n A'$

$A' \to \alpha_1 A' \mid \alpha_2 A' \mid \ldots \alpha_n A' \mid \varepsilon$

$A \to B\alpha \mid \epsilon$

$D \to AB \mid D$

$A \to (AB \mid D)\alpha \mid \epsilon$

$A \to AB\alpha \mid D\alpha \mid \epsilon$

$A \to D\alpha A' \mid \epsilon A'$

$A' \to B\alpha A' \mid \epsilon$

Date: 18/05/25

$E \to E+T \mid T$

$T \to T*F \mid F$

$F \to id \mid (E)$

$E \to TE'$

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

$T \to FT'$

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

$F \to id \mid (E)$

Recursive_decent parsing.

```
procedure E();
  begin
    T();
    EP();
  end
procedure EP();
  if input_symbol == '+' then
  begin
    Advance();
    T();
    EP();
  end
Procedure T()
  begin
    F();
    TP();
  end
```

```
Procedure T()
  if input-symbol = '*' then
  begin
    Advance();
    F();
    TP'();
  end
Procedure F();
  if input-symbol = 'id';
    Advance();
  else if input-symbol = '(' then
    begin
      Advance();
      E();
    end.
```

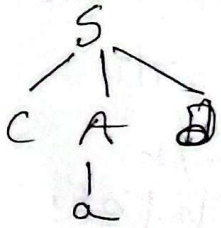
$S \to cAd$

$A \to ab \mid a$

$w = cad$

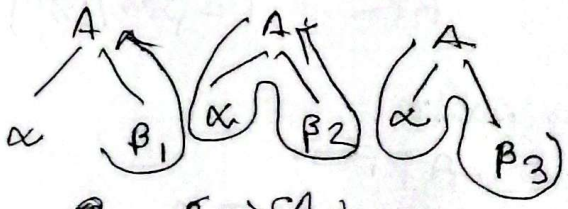
$S \Rightarrow cAd$

$\Rightarrow cad$



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

'$\alpha\beta_3$'



$S \Rightarrow cAd$

$\Rightarrow$ "cabd"

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

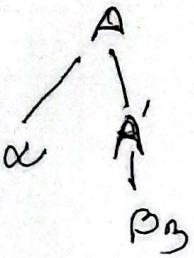
Non-deterministic.

Left-factoring

$A \to \alpha(\beta_1 | \beta_2 | \beta_3)$

$A \to \alpha A'$

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



$S \to bSSaaS | bSSaSb | bSb | a$

$S \to bSS' | b$

$S' \to SaaS | SaSb | b$

$S'' \to aS | Sb$

Multi-level - Non-determinism

Ll grammer.

Difficulty/Problem of top down parsing:-

1. Left-recursive
2. Backtracking
3. Error handler.
4. Totally fail to specify sentictechilly correct or not.

LL(1) → left factoring করে grammer.

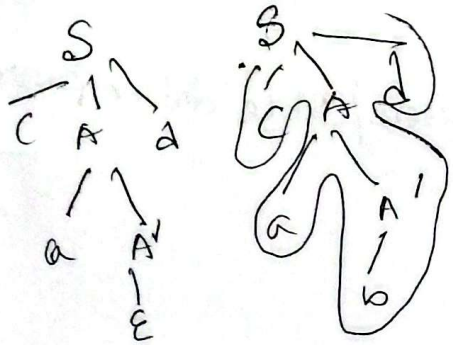
$S \to cAd$

$A \to a | ab$

$S \to cAd$

$A \to aA'$

$A' \to b | \varepsilon$



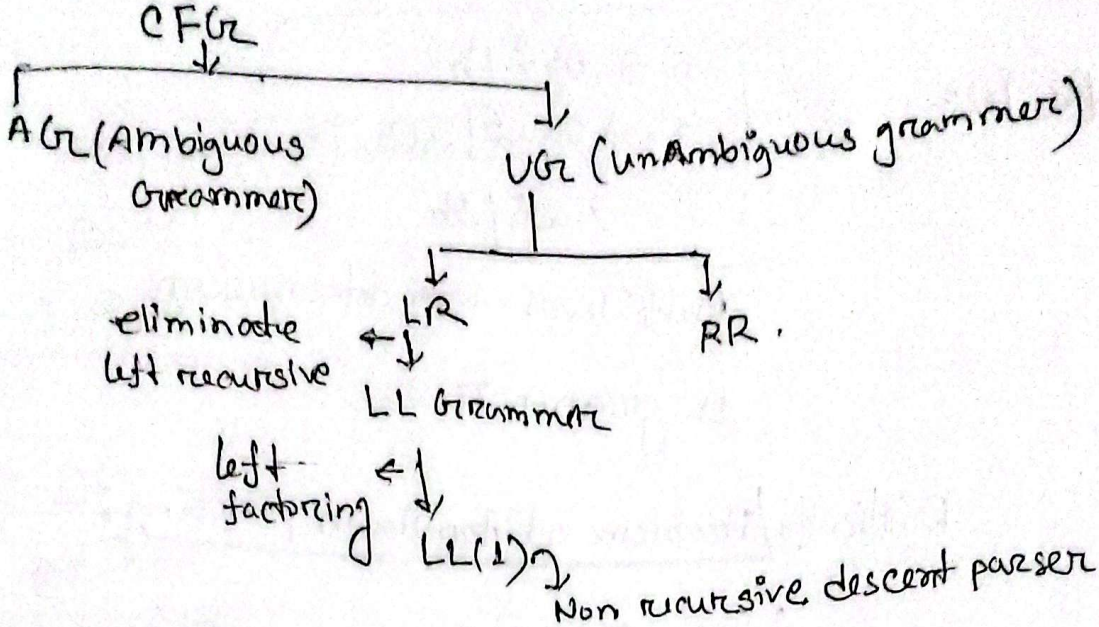
$S \to iE+S | iE+SeS | a$

~~$S \to iE+S(E|eS) | a$~~

$S \to iESS' | a$

$S' \to E | eS$

Date: 22/05/25



$S \rightarrow aB \mid ac$

$B \rightarrow b$

$C \rightarrow c$.

First(S) = First(aB) ∪ First(ac)

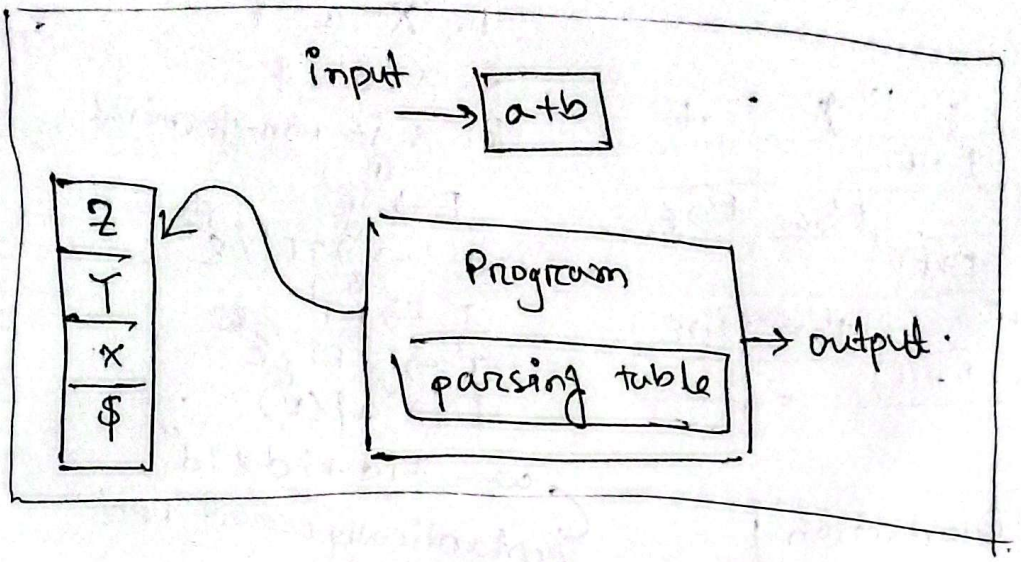
Properties of LL(1) grammer:—

- UnAmbiguous
- Non-left recursive.
- deterministic.

Predictive parser / Table driven parser

Predictive Parser:—

Predictive is a Non recursive table driven top-down parser that maintain stack activation explecitly by parser itself.



predictive parser

M[X, a]

$	a	b	c
A	α→β		
B		α→β	
C	α→β		α→β

1. X = a = $

declaration successfully complete

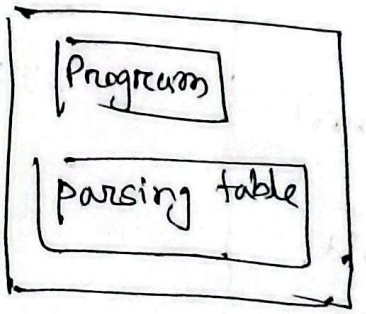
2. X = a ≠ $

a.

b. X

X → UVW.

LL(1) → Bison →



Date: 25/05/25

M [X, a]

X \ a	id	+	*	(	)	$
E	E→TE'			E→TE'		
E'		E'→+TE'			E'→ε	E'→ε
T	T→FT'			T→FT'		
T'		T'→ε	T'→*FT'		T'→ε	T'→ε
F	F→id			F→(E)		

X, a

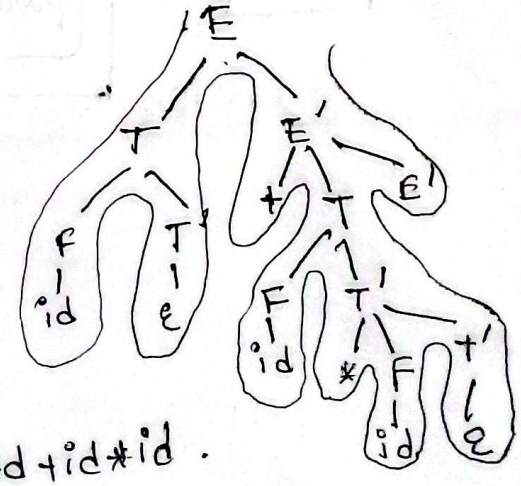
1. X = a = $
2. X = a = $
3. X is non-terminal

E → ① TE' ⑤
E' → ② +TE' / ε ③
T → ④ FT' ⑥
T' → ⑤ *FT' / ε ⑦
F → ⑧ id / (E) ⑨

w = "id + id * id"
syntactically correct input.

E ⇒ TE' (using ①)
⇒ FT'E' (using ④)
⇒ id T'E' (using ⑧)
⇒ id E' (using ⑥)
⇒ id + TE' (using ②)
⇒ id + FT'E' (using ④)
⇒ id + id T'E' (using ⑧)
⇒ id + id * FT'E' (using ⑤)
⇒ id + id * id T'E' (using ⑧)
⇒ id + id * id E' (using ⑥)
⇒ id + id * id (using ⑨)

Stack	input	Production
$E	id + id * id $	
$E'T	id + id * id $	E→TE'
$E'T'F	id + id * id $	T→FT'
$E'T'id	id + id * id $	F→id
$E'T'	+ id * id $	
$E'	+ id * id $	T'→ε
$E'T+	+ id * id $	E'→+TE'
$E'T	id * id $	
$E'T'F	id * id $	T→FT'
$E'T'id	id * id $	F→id.
$E'T'	* id $	
$E'T'F*	* id $	T'→*FT'
$E'T'F	id $	
$E'T'id	id $	F→id
$E'T'	$	
$E'	$	T'→ε
$	$	E'→ε



id + id * id.

FIRST, FOLLOW (Next class)

Date: 19/06/25

Parsing Table of LL(1) grammer

First set, follow set

$S \to A$

$A \to aB \mid Ad$

$B \to bBC \mid f$

$C \to g$

Rules	first set	follow set
$S \to ACB \mid CbB \mid Ba$	{a, b, d, g, h, ε}	{$}
$A \to da \mid BC$	{d, g, h, ε}	{h, g, $}
$B \to g \mid \varepsilon$	{g, ε}	{a, h, g, $}
$C \to h \mid \varepsilon$	{h, ε}	{g, b, h, $}

$fro(S) = fro(ACB) \cup fn(CbB) \cup fro(Ba)$

$= fro(A) \cup fro(C) \cup fro(B)$

$= \{d, g\} \cup Fro(C)$

Rules	First set	follow set
$E \to TE'$	{id, (}	{), $}
$E' \to +TE' \mid \varepsilon$	{+, ε}	{), $}
$T \to FT'$	{id, (}	{+,), $}
$T' \to *FT' \mid \varepsilon$	{*, ε}	{+,), $}
$F \to id \mid (E)$	{id, (}	{*, +,), $}

Parse table

	id	+	*	(	)	$
E	E→TE'			E→TE'		
E'		E'→+TE'			E'→ε	E'→ε
T	T→FT'			T→FT'		
T'		T'→ε	T'→*FT'		T→ε	T→ε
F	F→id			F→(E)		

	first	Follow
$S \to asbS \mid bSaS \mid \varepsilon$	{a, b, ε}	{b, a$}

	a	b	$
S	S→asbS	S→bSaS	

① Make the given grammer LL(1)

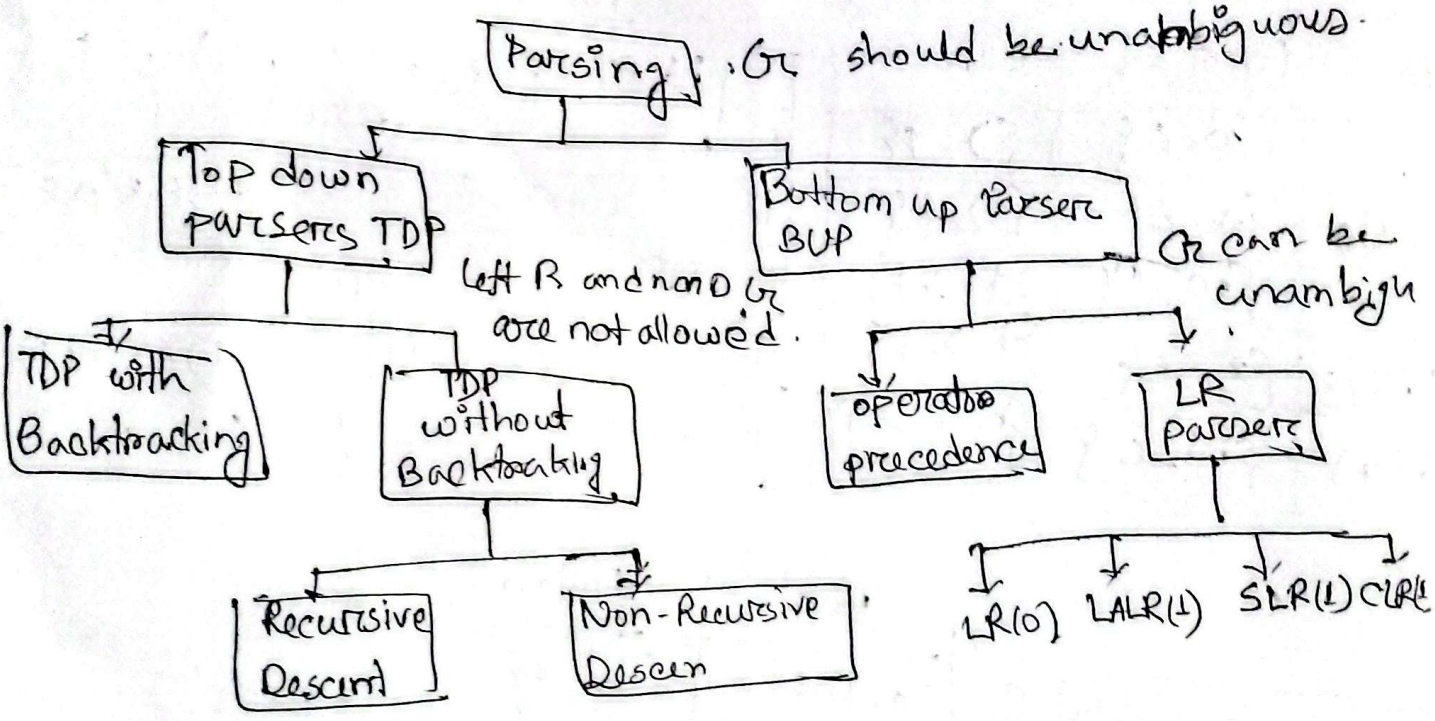
② Define first and follow

③ Construct Parse table

④ Write Parsing algorithm / Program

Date: 22/06/25

→ Parsing (coming from parts of speech)
→ operator precedence
→ Recursive descent
→ Ways of parsing.
  • Top Down parsing
  • Bottom-Up parsing (Shift Reduce parsing)
→ parsing algorithms
  • CYK Algorithm.
  • Earley's Algorithm.
  • LL Parsing Algorithm
  • LR parsing algorithm.



① $S \to aABe$

~~$A \to bBc \mid b$~~

② $A \to Abc \mid b$ ③

④ $B \to d$

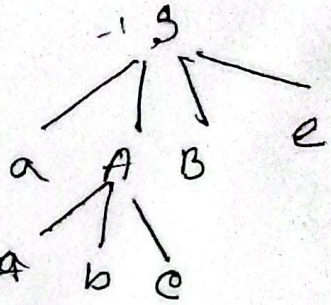
$w =$ "abbcde"

$S \Rightarrow aABe$ [①]

$\Rightarrow aAbcBe$ [②]

$\Rightarrow abbcBe$ [③]

$\Rightarrow abbcde$ [④]



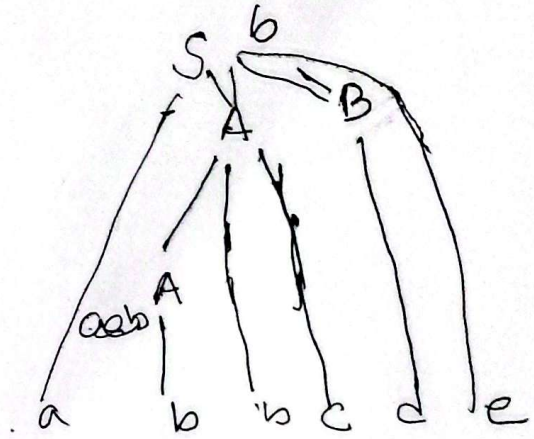
$S \Rightarrow aABe$ [①]

$\Rightarrow aAde$ [④]

$\Rightarrow aAbcde$ [②]

$\Rightarrow abbcde$ [③]

(RMD)



LR Parser → Shift reduce parser.

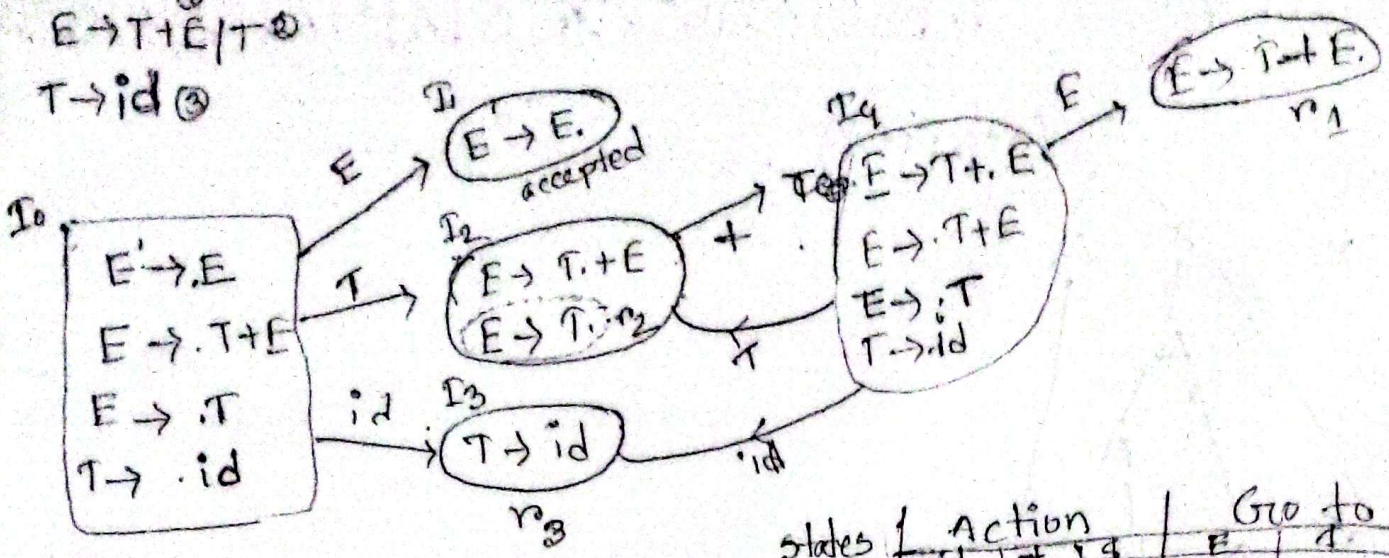
Date: 03/07/25

LR(0) + Follow = LR(1) = SLR(1)

LR(0) SLR(1)

$E \to T+E$ ① / T ②

$T \to id$ ③



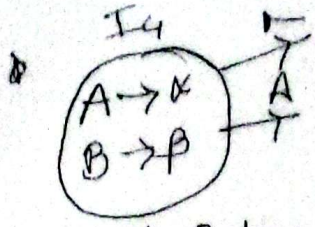
States	Action			Go to	
	id	+	$	E	T
0	S3			1	2
1			accept		
2	r2	S4	r2		
3	r3	r3	r3		
4	S3			5	2
5	r1	r1	r1		

LR(0) + Follow = LR(1) = SLR(1)

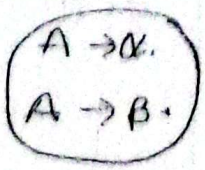
Follow(E) = {$}

Follow(T) = {+, $}

Date: 10/07/25



Shift - Reduce conflict



Reduce Reduce conflict

Then SLR(1) will not applicable.

CLR(1)

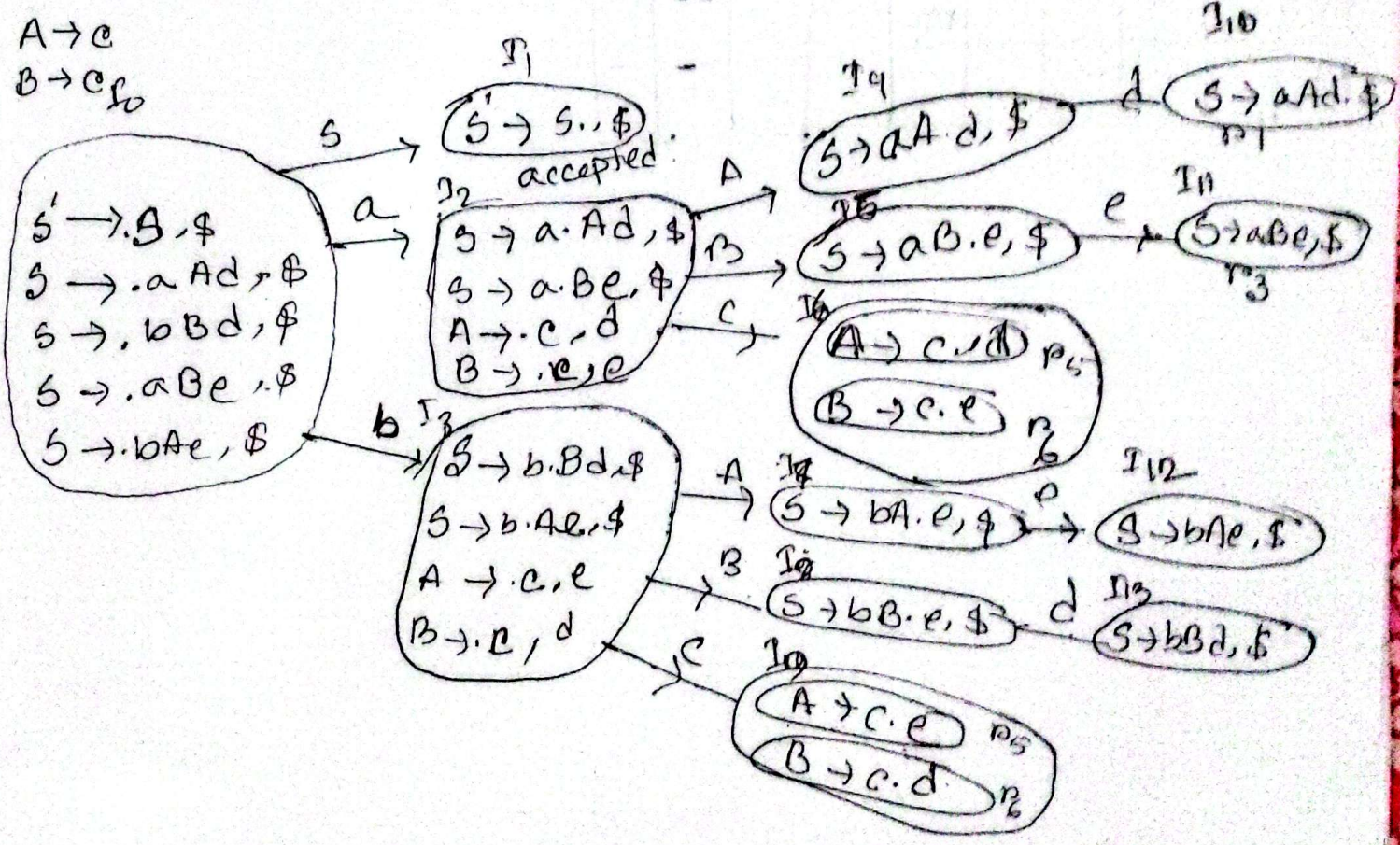
Look ahead

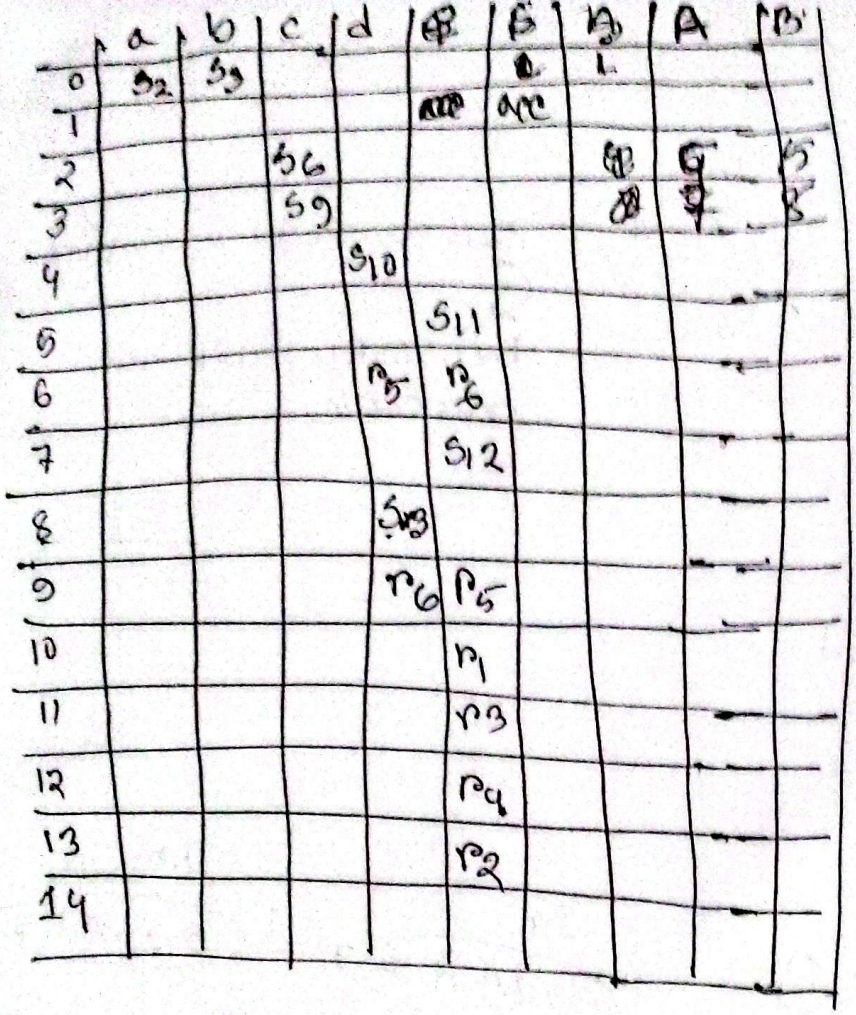
LR(1) = LR(0) + Look ahead

S → aAd | bBd | aBe | bAe

A → c

B → c




a
b
c
d
$
E
A
B
0
1
2
3
4
5
6
7
8
9
10
11
12
13
14
s_2
s_3
s_6
s_9
s_{10}
s_{11}
r_5
r_6
s_{12}
s_{13}
r_6
r_5
r_1
r_3
r_4
r_2

Date: 17/07/25

→ php Notation
→ laravel ~~php~~ Notation framework.

Notation → lexical analyzer (RE)
→ Syntactic analyzer (CFGr)

Notational framework.

Notation: System of symbols that represents concepts, action on data.

Notational framework:- structured on systematic methodology on methodology that governs how symbols designed, represented and used.

grammer + subroutine = SDT (Syntax directed Translation scheme)

CFGr + Actions = SDT subroutine, actions, semantics.

$E \to E + T \{ \}$

$S \overset{*}{\Rightarrow}$

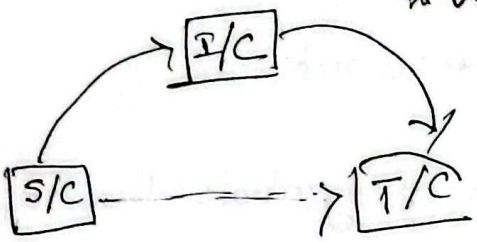
Date: 20/07/25

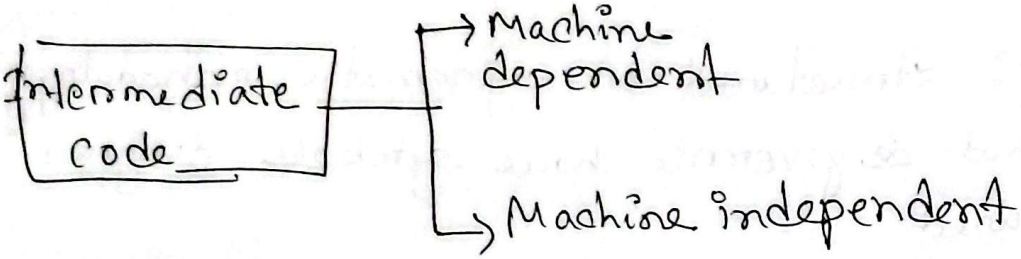
Syntax Directed Translation (SDT)

Grammer + semantic action = SDT

$A \overset{*}{\Rightarrow} XYZ \overset{*}{\Rightarrow} \omega$

* Why use intermidiate code?*





Intermediate code:-

classification:-

(i) Postfix Notation: infix, ~~postfix~~ prefix.

(ii) Syntax tree

(iii) Three-address code → Quadraple, Triple

Postfix Notation:-

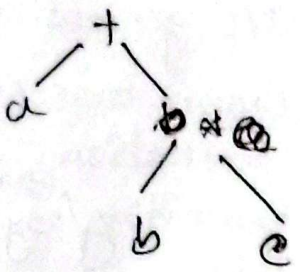
if a then if b-c then a:=b+c
else a:=b-c else
a:=b*c

if a then
if b-c then
bc-
a:=b+c → bc+
else
a:=b-c → bc-
else
a:=b*c → bc*

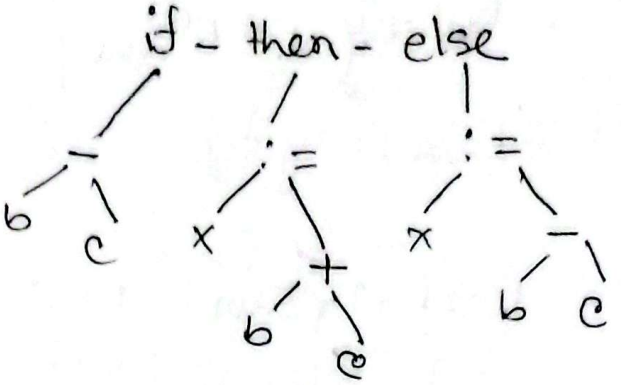
bc+ bc+ bc- ?
ternary operator.
abc-bc+bc-?bc*?

Syntax tree:

a + b * c.



if - then - else



Three-address-code:

Two data structure use Quadraple and Triple.

a := b op c

↓ operators

X + Y * Z

$T_1 := Y * Z$

$T_2 := X + T_1$

a := -b + c * d.

$T_1 := -b$

$T_2 := c * d$

$T_3 := T_1 + T_2$

$a := T_3$

Quadruple representation:-

	OP	AR1	AR2	Result
(0)	-	b		T_1
(1)	*	c	d	T_2
(2)	+	T_1	T_2	T_3
(3)	:=	T_3		a

Triple Representation:- (No temporary variable)

	OP	ARG1	ARG2
(0)	-	b	
(1)	*	c	d
(2)	+	(0)	(1)
(3)	:=	a	(2)

Syntax Directed Translation (SDT)

Grammar + Semantic action.

$E \to E+E \quad \{ E.val := E^{(1)}.val + E^{(2)}.val \}$

$E \to digit \quad \{ E.val := digit.val \}$

"2+3*4"

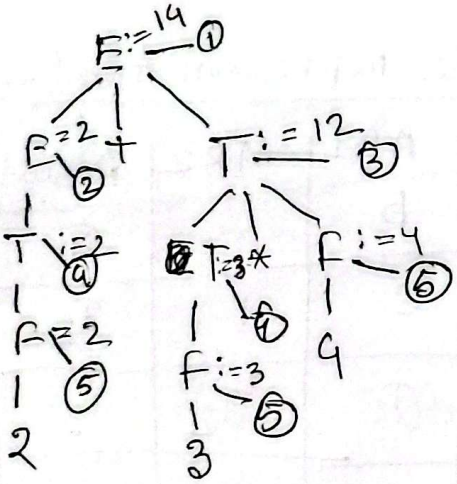
$E \to E^{(1)} + T \quad \{ E.val := E.val + T.val \}$

$\to T_{(1)} \quad \{ E.val := T.val \}$

$T \to T * F \quad \{ T.val := T.val * F.val \}$

$\to F \quad \{ T.val := F.val \}$

$F \to digit \quad \{ F.val := name.lval \}$



Evaluation.

Top Down.

Semantic Actions :-

(i) Evaluation *

(ii) I/C generation *

(iii) Error massege generation

(iv) Rewriting/update symbol Table.

Semantic Errors :

* Type missmatch.
* Multiple variable declaration.
* undeclaration of variable.
* Reserve word/keyword.

Date: 28/07/25

SDT (Grammere + Action)

$E \to E * T$ { $E_{val} := E^{(1)}.val * T.val$ } — ①

$\quad | T$ { $E.val := T.val$ } — ②

$T \to F - T$ { $T.val := f.val - T^{(1)}.val$ } — ③

$\quad | F$ { $T.val := f.val$ } — ④

$F \to 2$ { $F.val :=$ num.lval } — ⑤

$F \to 4$ { $F.val :=$ num.lval } — ⑥

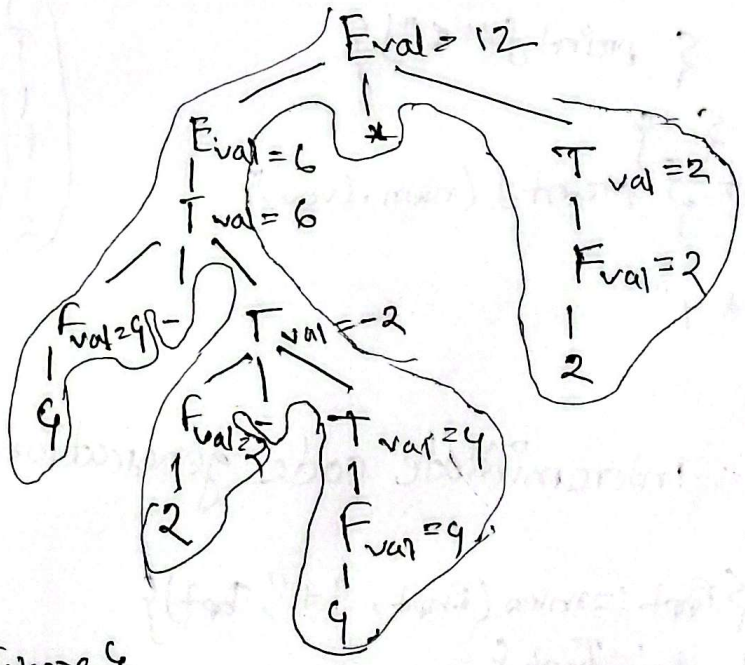
$w =$ "(4 − (2 − 4)) * 2"

$= (4 - (2)) * 2$

$= 6 * 2$

$= 12$

Bottom up



(*) $E \to E + T$ { ~~Eval :=~~

$\quad | T$ { $E_{type} == T.type$ }

$T \to T * f$ { If ($T_{type} == f_{type}$) && ($T_{type} ==$ ~~float~~ int) or ($T_{type} ==$ float) Then $T_{type} := f_{type}$; else error. }

$\quad | F$ { $T_{type} := f.type$ }

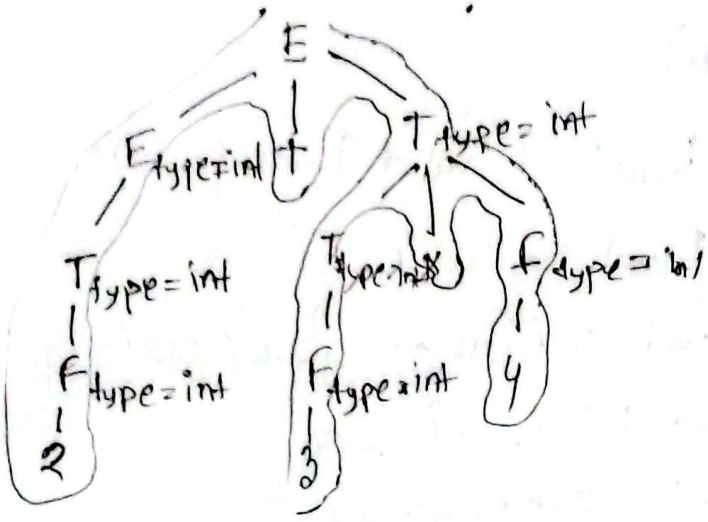
$F \to id$ { $F.Type :=$ int | float | char | bool }

$E \to E + T$ { ($E_{type} == T_{type}$) && ($E_{type} ==$ int) or ($E_{type} ==$ float)

then $E_{type} := T_{type}$; else error; }

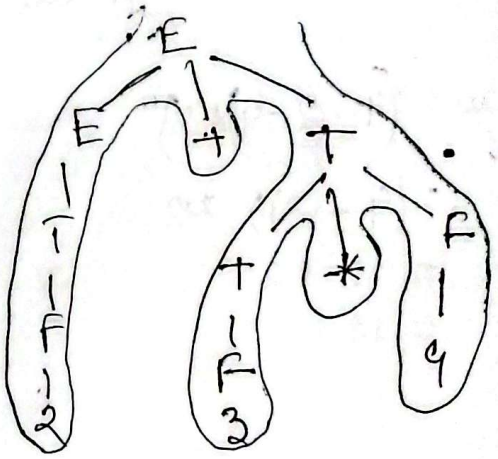
"2 + 3 * 4"

- Code optimization
- Symbol table
- analysis



E ⇒ E + T { Printf("+"); }
| T { }
T → T * F { printf("*") }
| F { }
F → digit { printf (num.lval) }

w = "2 + 3 * 4"

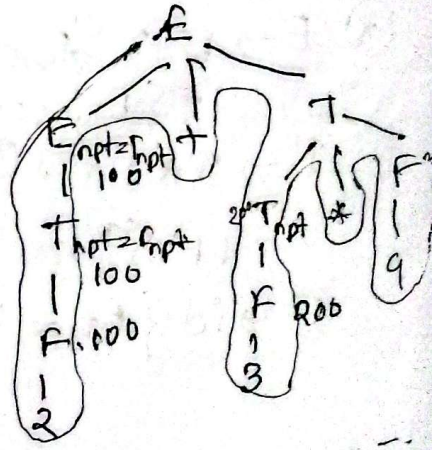


2 3 4 * +.

∴ Intermidiate code generation [Postfix Notation]

E → E + T { Enpt := mkn (Enpt, "+", Tnpt) }
| T { Enpt := Tnpt }
T → T * F { Tnpt = mkn (Tnpt, '*', fnpt) }
| F { Tnpt = fnpt; }
F → digit { fnpt = mkn (null, digit, null); }

w = "2 + 3 * 4"



Left | info | Right

Syntax tree