Parse – 1, 2, 3

FIRST:

1) $A \to aB$

FIRST (A) = {a}

2) $A \to a \mid w$

FIRST (A) = {a, w}

3) $A \to aB \mid w$

FIRST (A) = {a, w}

4) $A \to (aB) \mid w$

FIRST (A) = {(, w}

5) $A \to Ta$

$T \to *FT' \mid w$

FIRST (A) = FIRST (T)
= {*}

Note

Small = terminal
Capital = Non terminal
operation = terminal
digit = terminal

$A \to a$
$A \to w$

6) $A \to Ta$

$T \to *FT' \mid w$

FIRST (A) = FIRST (T)
= {*, w}

Now, $A \to *a$

FIRST (A) = {*, a}

Note
w remove করতে হবে

* Start variable এর জন্য follow তে always $ দিয়ে শুরু হবে।

FIRST and Follow Table

	FIRST (w ∨ y)	FOLLOW (w x y)
$E \to TE'$	{id, (}	{$,)}
$E' \to +TE' \mid w$	{+, w}	{$,)}
$T \to FT'$	{id, (}	{+, $,)}
$T' \to *FT' \mid w$	{*, w}	{+, $,)}
$F \to id \mid (E)$	{id, (}	{*, +, $,)}

FIRST (E)
= FIRST (T)
= FIRST (F)
= {id, (}

Follow rules

1) যদি কোনো variable এর E' কে কেউ follow করতে না তাহলে E এর follow বসাও। কোনো E' আসলে E থেকে

2) TE' বসাতে পারব না FIRST (E') বসাতে হবে

FOLLOW(T) = FIRST(E') = { +, ω }

FOLLOW(F) = FIRST(T') = { *, ω }

Parsing Table

Non terminal ২টো column

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

Rule 1:

1) E → TE' : E এর first বের করব, তার পরবর্তী T এর first বের করব

2) E' → +TE' | ω

E' → ω

E' এর follow বসাবো

3) T → FT'

→ first F (Non-Terminal)

→ first of (F) এইবার)

{ id, (}

4) T' → *FT' | ω

i) T' → *FT'

ii) T' → ω

T' এর follow বসাব

5) F → id | (E)

F → id

F → (E)

Remove left recursion

$A \to A\alpha / \beta$

Formula

$A \to \beta A'$

$A' \to \alpha A' / \omega$

Rule

$A \to A a / b$

i) $\alpha = a, \beta = b$

$A \to bA'$

$A' \to aA' / \omega$

ii) $A \to Ad / b$

$\alpha = d, \beta = b$

$A \to bA'$

$A' \to dA' / \omega$

iii) $\beta \to \beta b / c$

$\alpha = b, \beta = c$

$\beta = c\beta'$

$\beta' = b\beta' / \omega$

iv) $C \to Cc / g$

$\alpha = c, \beta = g$

$C = gC'$

$C' = cC' / \omega$

Example:

$S \to ABC$

$A \to Aa / Ad / b$

$B \to Bb / C$

$C \to Cc / g$

Soln:

$S \to ABC$

$A \to bA'$

~~$A' \to aA' / \omega$~~

~~$A \to bA'$~~

$A' \to dA' / \omega / dA'$

$B \to CB'$

$B' \to bB' / \epsilon$

$C = gC'$

$C' = cC' / \omega$

α এবং β যেভাবে নেওয়া হয়েছে।

$E \to E+T / T$

$\alpha \quad \beta$

A

$\alpha = +T, \beta = T$

$A \alpha / \beta$

Example: (1)

$E \rightarrow \underset{A}{E} \underset{\alpha}{+T} / \underset{\beta}{T}$

formula

$A \rightarrow \beta A'$

$A' \rightarrow \alpha A' \omega / \alpha A'$

$E \rightarrow TE'$

$E' \rightarrow \omega / +TE'$

Example: (2)

$T \rightarrow \underset{A}{T} \underset{\alpha}{*F} / \underset{\beta}{F}$

$T \rightarrow FT'$

$T' \rightarrow \omega / *FT'$

Example: 3

$F \rightarrow (E) / id$

Example: 4

$E \rightarrow E+T / T$

$T \rightarrow T*F / F$

$F \rightarrow (E) / id$

$E \rightarrow TE'$

$E' \rightarrow \omega / +TE'$

$T \rightarrow FT'$

$T' \rightarrow \omega / *FT'$

$F \rightarrow (E) / id$

Example: (5)

$S \rightarrow (L) / a / b$

$L \rightarrow \underset{A}{L} , \underset{\alpha}{S} / \underset{\beta}{S}$

Soln: $S \rightarrow (L) / a / b$

$L \rightarrow SL'$

$L' \rightarrow \omega / , SL'$

Example: (6)

$\underset{A}{S} \rightarrow \underset{A}{S} \underset{\alpha}{0} \underset{\alpha}{S} 1 S / \underset{\beta}{01}$

$S \rightarrow 01S'$

$S' \rightarrow \omega / 0S1SS'$

Example: (7)

$\underset{A}{A} \rightarrow \underset{A}{A} \underset{\alpha}{+} \underset{\alpha}{B} / \underset{\beta}{B}$

$B \rightarrow int / (A)$

$A \rightarrow BA'$

$A' \rightarrow \omega / +BA'$

* এবার দেখি Left recursion থাকলে.

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

<u>formula:</u> $A \to \beta A'$

$A' \to \omega \mid \alpha A'$

<u>Example:</u> $A \to A\alpha_1 \mid A\alpha_2 \mid A\alpha_3 \cdots A\alpha_m \mid \beta_1 \mid \beta_2 \cdots \beta_n$

<u>Soln:</u>

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

$A' \to \omega \mid \alpha_1 A' \mid \alpha_2 A' \mid \cdots \alpha_n A'$

Left fectoring

* common prefix problem solve করা হয় left fectoring দিয়ে।

<u>Example:</u> (i)

$A \to \alpha\beta_1 \mid \alpha\beta_2 \mid \alpha\beta_3 \mid \alpha\beta_4$

$A \to \alpha A'$

$A' \to \beta_1 \mid \beta_2 \mid \beta_2 \mid \beta_4$

→ কমনটা নিলে যেমন। এখানে α কমন।
আর তার পর rest আছে $\beta_2 \cdots \beta_4$ এগুলো
A' এর নিচে β rest গুলো বসিয়ে দিব।

<u>Example: (ii)</u>

$S \to iEtS$

$S \to iEtSeS$

$S \to a$

$E \to b$

<u>common</u> iEtS

<u>Soln:</u>

$S \to iEtSS' \mid a$

$S' \to \omega \mid eS$

$E \to b$

(এমন থাকলে হবে।

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

$A \to \alpha A'$

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

Top down and Bottom up

Grammer $G = (V, T, P, S)$, $E \to E+E \mid E*E \mid id$

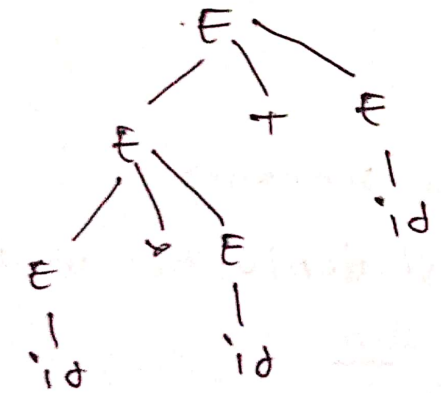
$V = \{E\}$, $S = E$, $T = \{id, +, *\}$

Obtain Parse tree for,

$w = id*id+id$

Top down parsing

* Parsers starts constructing the parse tree from start symbol and tries to transform the start symbol to the input symbol (string)
  * (Left to Right scanning performed)



$E \to E+E$

$\Rightarrow E*E+E$

$\Rightarrow id*E+E$

$\Rightarrow id*id+E$

$\Rightarrow id*id+id$

→ Expansion

→ LL parser

↓

Left to right scan

↳ Left most derivation

Ambiguity

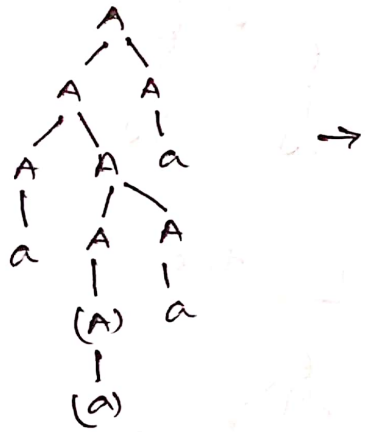
If there is ~~two~~ put more than one parse tree, so the grammer is called ambiguous.

Example: check whether the grammer is ambiguous or not.

$A \rightarrow AA$
$A \rightarrow (A)$
$A \rightarrow a$

Soln: to the string "a(a)aa".

Left Most Derivation



a(a)aa

Right Most Derivation

a(a)aa

so, the grammer is ambiguous grammer.

— o —

Ambugity will be done as three ways:

1.

Example 02: Check whether the given grammer is ambiguous or not.

$E \to E+E$

$E \to E-E$

$E \to id$

string: "id+id-id"

$E \to E+E$

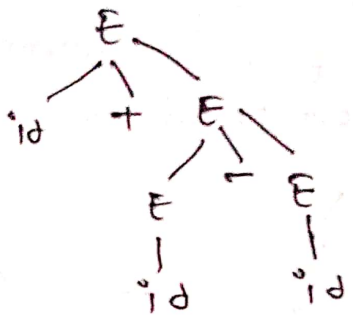
$\Rightarrow id+E$

$\Rightarrow id+E-E$

$\Rightarrow id+id-E$

$\Rightarrow id+id-id$

Parse tree



আরেকভাবে,

$E \to E-E$

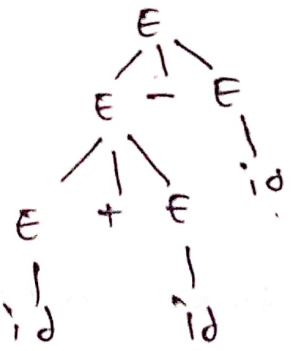
$\Rightarrow E+E-E$

$\Rightarrow id+E-E$

$\Rightarrow id+id-E$

$\Rightarrow id+id-id$

Parse tree



There are more than ~~two~~ one parse tree. So this grammer is ambiguous.

Derivation

Leftmost derivation:

input string :

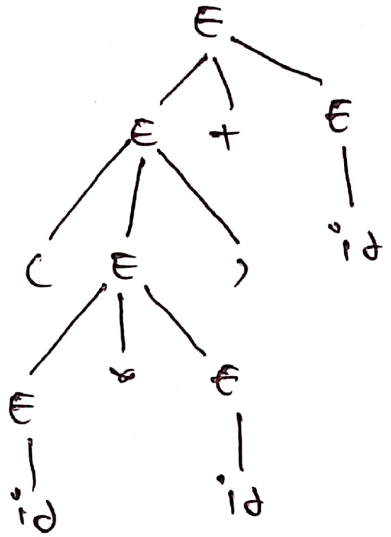
(id * id) + id

Rules

E → E + E
E → E * E
E → (E)
E → - E
E → ID

E + E
⇒ (E) + E
⇒ (E * E) + E
⇒ (id * E) + E
⇒ (id * id) + E
⇒ (id * id) + id

Parse tree



Right most Derivation

E
⇒ E + E
⇒ E + id
⇒ (E) + id
⇒ (E * E) + id
⇒ (E * id) + id
⇒ (id * id) + id

Parse tree