KTU NOTES

The learning companion.

KTU STUDY MATERIALS | SYLLABUS | LIVE NOTIFICATIONS | SOLVED QUESTION PAPERS

Website: www.ktunotes.in

MODULE - 2

SYNTAX ANALYSIS

Context Free Grammar

(V, T, P, S)

V - Set of Non Terminals (Capital Letter)

T - Set of Terminals (Small Letter)

P - Production Rule

S - Start Symbol

Terminals

Lower case letter (a, b, c)

Operators (+, -, *, etc)

Punctuation (( ),), etc)

Digits (0, . . . , 9)

Boldface strings if or id

Non Terminals

Upper case letters (A, B, C)

Start symbol (S)

Lower case italics (expression or string)

1. $E \longrightarrow E+E / (E) / -E / id$

$A \longrightarrow +/-/*/ / / \uparrow$

Derive the string -(id + id)

$E \rightarrow -E$

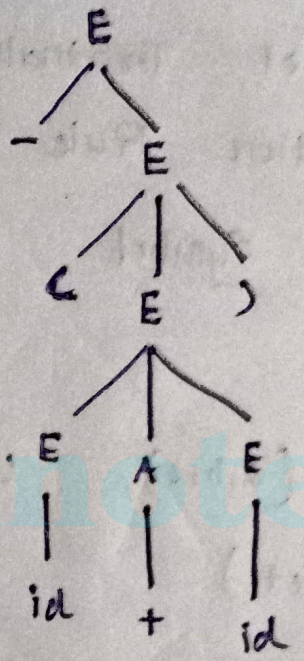
$E \rightarrow -(E)$

$E \rightarrow -(E+E)$

$E \rightarrow -(id+E)$

$E \rightarrow -(id+E)$

$E \rightarrow -(id+id)$



2. Construct a parse tree for the string aabaa from the following production

$S \rightarrow aAS/aSS/\epsilon$

$A \rightarrow SbA/ba$

$S \rightarrow aAS$

~~$\rightarrow aaSS$~~

$S \rightarrow aSS$

$\rightarrow aaASS$

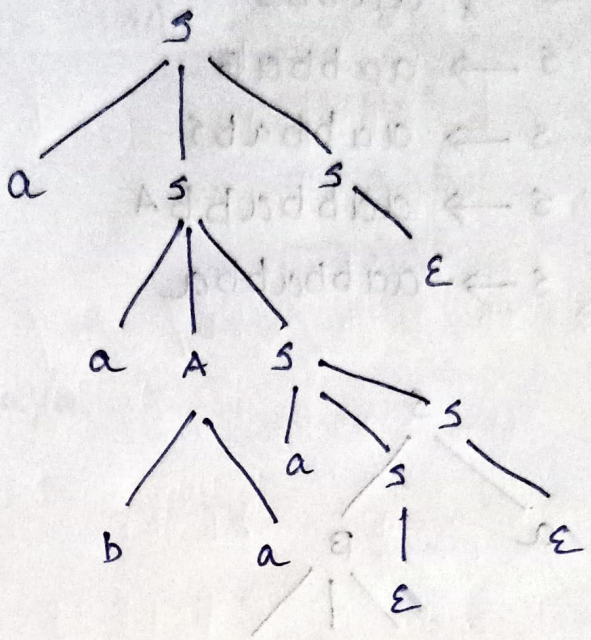
$\rightarrow aabaSS$

$\rightarrow aabaaSSS$

$\rightarrow$ aabaassε

$\rightarrow$ aabaasε

$\rightarrow$ aabaaε $\longrightarrow$ aabaa



Leftmost Derivation

S $\rightarrow$ ass

S $\rightarrow$ aaAss

S $\rightarrow$ aabass

S $\rightarrow$ aabaasss

S $\rightarrow$ aabaass

S $\rightarrow$ aabaas

S $\rightarrow$ aabaa

Rightmost Derivation

S $\rightarrow$ ass

S $\rightarrow$ as

S $\rightarrow$ aaAs

S $\rightarrow$ aaAass

S $\rightarrow$ aaAas

S $\rightarrow$ aaAa

S $\rightarrow$ aabaa

3. S $\rightarrow$ aB/bA

A $\rightarrow$ a/as/bAA

B $\rightarrow$ b/bS/aBB

aabbabba

$S \rightarrow aB$

$S \rightarrow aaBB$

$S \rightarrow aabB$

$S \rightarrow aabbS$

$S \rightarrow aabbaB$

$S \rightarrow aabbabS$

$S \rightarrow aabbabbA$

$S \rightarrow aabbabba$

$S \rightarrow aB$

$S \rightarrow aaBB$

$S \rightarrow aaBbS$

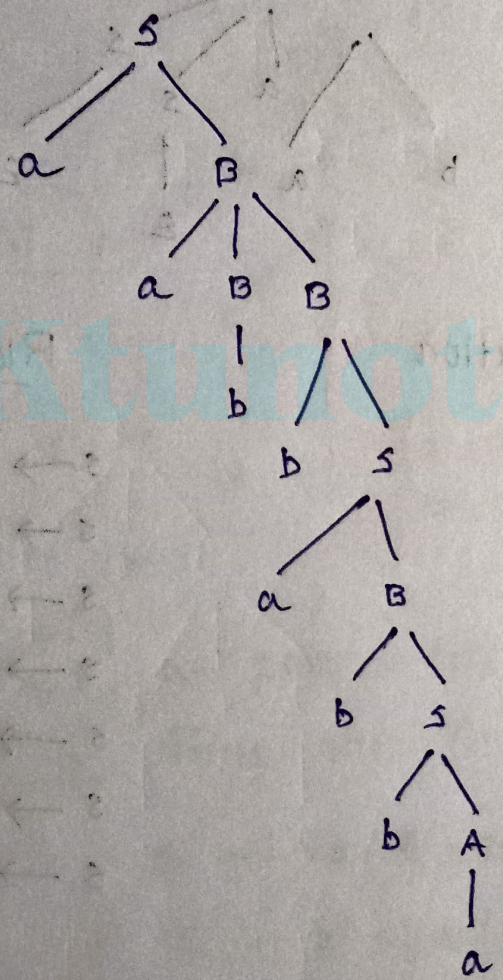
$S \rightarrow aaBbbA$

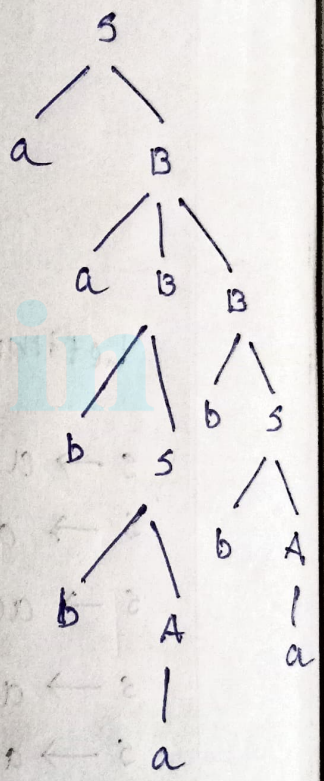
$S \rightarrow aaBbba$

$S \rightarrow aabSbba$

$S \rightarrow aabbAbba$

$S \rightarrow aabbabba$





Ex:

A grammar that produces more than one parse tree for the same sentence

1. $S \rightarrow AB/aaB$

$A \rightarrow a/Aa$

$B \rightarrow b$

$w = aab$

i) $S \rightarrow aaB$

$S \rightarrow aab$

ii) $S \rightarrow AB$

$\rightarrow AaB$

$\rightarrow aaB$

$\rightarrow aab$

2. $A \rightarrow aA/Aa/a$

$S \rightarrow A$

$w = aaa$

$A \rightarrow aA$

$\rightarrow aaA$

$\rightarrow aaa$

RMD

$A \rightarrow Aa$

$\rightarrow Aaa$

$\rightarrow aaa$

LMD

Ambigous

3. $S \rightarrow S+S/S*S/a$

$w = a+a+a$

$S \rightarrow S+S$

$\rightarrow S+S*S$

$\rightarrow a+S*S$

$\rightarrow a+a*S$

$\rightarrow a+a*a$

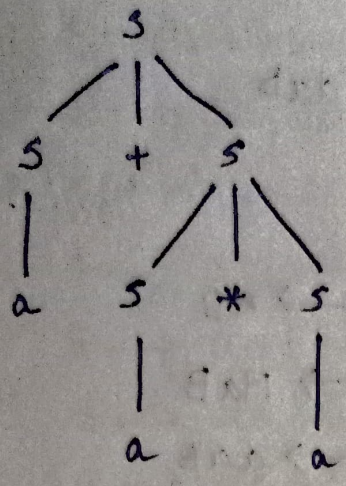
$S \rightarrow S*S$

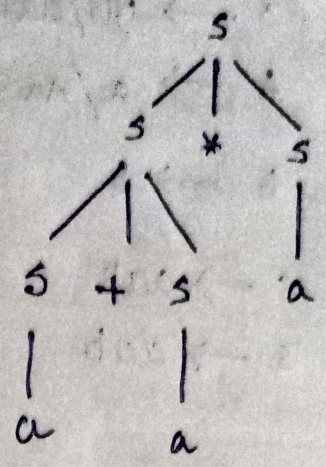
$\rightarrow S+S*S$

$\rightarrow S+S*a$

$\rightarrow S+a*a$

$\rightarrow a+a*a$





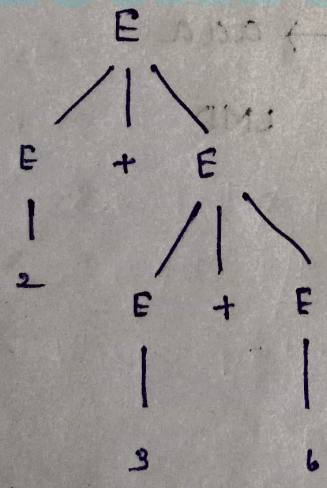
4. $E \longrightarrow E+E / E-E / E*E / E/E / num$

$w = 2+3+6$

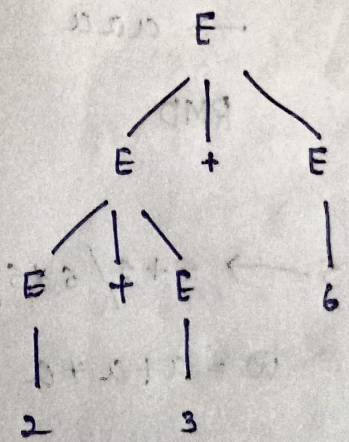
~~$w =$~~ $E \longrightarrow E+E$

$\longrightarrow E+E+E$

$\longrightarrow 2+3+6$



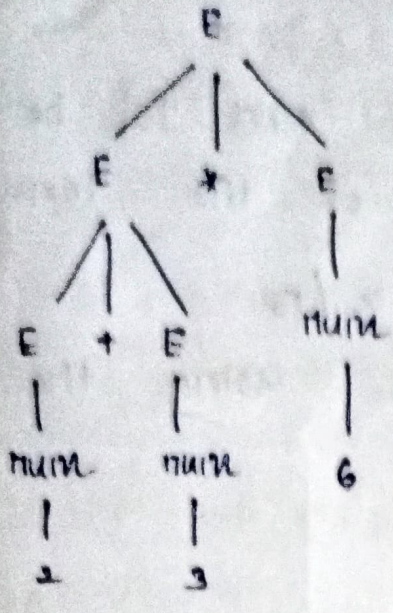
Right Associative



Left Associative

5. $w = 2+3*6$

$= 20$



= 30

L.H

E → E+E / E-E / E*E / E/E / num

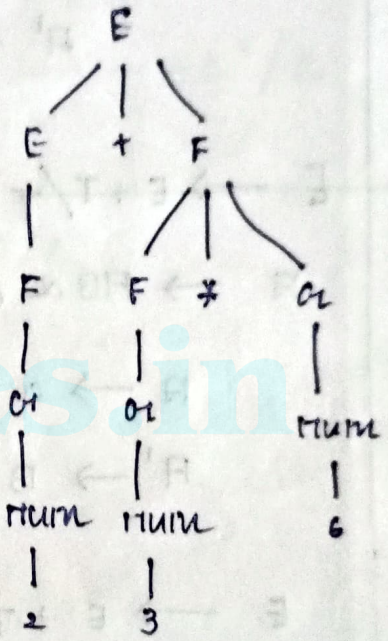
↓ levels

E → E+F / E-F / F

F → F*G1 / F/G1 / G1

G1 → num

↓



B → B OR B / B AND B / NOT B / T/F

↓

B → B OR C / C

C → C AND D / D

D → NOT D / E

E → T/F

OR
AND
NOT
T/F

H.P

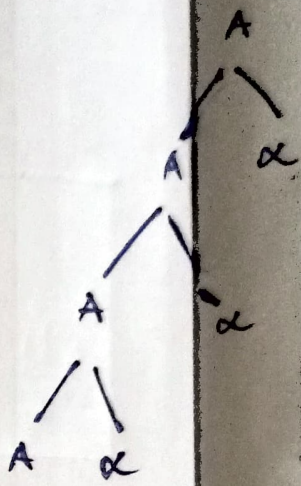
Left Recursion

A grammar is said to be left recursive if it is of the form

$$A \longrightarrow A\alpha / \beta$$

A can be eliminated using the production

$$A \longrightarrow \beta A'$$
$$A' \longrightarrow \alpha A' / \varepsilon$$



1. ~~$E \longrightarrow E+T/T$~~

1. $A \longrightarrow AB\alpha / a$

$$A \longrightarrow \alpha A'$$
$$A' \longrightarrow B\alpha A' / \varepsilon$$

$$A \longrightarrow aC / a$$
$$C \longrightarrow B\alpha C / B\alpha$$

2. $E \longrightarrow E+T / T$

$A \quad \alpha \quad \beta$

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

$$E \longrightarrow TC / T$$
$$C \longrightarrow +TC / +T$$

3. $S \longrightarrow S0S1S / 01$

$$S \longrightarrow 01S'$$
$$S \longrightarrow 0S1SS' / \varepsilon$$

$$S \longrightarrow 01 / 01C$$
$$C \longrightarrow 0S1SC / 0S1S$$

4. $S \longrightarrow (L) / a$

$L \longrightarrow L, S / S$

$A \quad \alpha \quad \beta$

$A \longrightarrow \beta A'$

$A' \longrightarrow \alpha A' / \varepsilon$

$S \longrightarrow (L) / aL$

$L \longrightarrow SL'$

$L' \longrightarrow , SL' / \varepsilon$

5. $E \longrightarrow E+E / E*E / a$

$E \longrightarrow aE'$

$E \longrightarrow +EE' / \varepsilon$

$E \longrightarrow aE'$

$E \longrightarrow *EE' / \varepsilon$

$E \longrightarrow aE'$

$E \longrightarrow +EE' / *EE' / \varepsilon$

6. $E \longrightarrow E+T / T$

$T \longrightarrow T*F / F$

$F \longrightarrow id$

$E \longrightarrow TE'$

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

$T \longrightarrow FT'$

$T' \longrightarrow *FT' / \varepsilon$

$F \longrightarrow id$

Left Factoring

A grammar is said to be left factoring if it is of the form

$A \longrightarrow \alpha\beta_1 / \alpha\beta_2 / \alpha\beta_3 / \ldots$

If more than one grammar production rule has a common prefix string, then top down parser cannot make a choice to which production to use.

(1) $A \longrightarrow aAB/aA$

$B \longrightarrow bB/b$

$A \longrightarrow aA'$

$A' \longrightarrow AB/A$

$B \longrightarrow bB'$

$B' \longrightarrow B/\varepsilon$

$A \longrightarrow \alpha A'$

$A' \longrightarrow \beta_1/\beta_2/\beta_3$

(2) $A \longrightarrow aAB/aA/a$

$A \longrightarrow aA'$

$A' \longrightarrow AB/A/\varepsilon$

$A \longrightarrow aA'$

(3) $A' \longrightarrow AB'/\varepsilon$

$B' \longrightarrow B/\varepsilon$

(3) $S \longrightarrow iEtS/iEtSeS/a$

$E \longrightarrow b$

$S \longrightarrow iEtSS'/a$

$S' \longrightarrow eS/\varepsilon$

$E \longrightarrow b$

(3) $S \longrightarrow iEtS/iEtSeS/a$

$E \longrightarrow b$

$S \longrightarrow iS'/a$

$S' \longrightarrow EtS/EtSeS/$

$\alpha = iEtS$

(4) $S \longrightarrow aSSbS / aSaSb / abb / b$

$S \longrightarrow aS' / b$

$S' \longrightarrow SSbS / SaSb / bb$

$S' \longrightarrow SB' / bb$

$B' \longrightarrow SbS / aSb$

$S \longrightarrow aS' / b$

$S' \longrightarrow SB' / bb$

$B' \longrightarrow SbS / aSb$

(5) $S \longrightarrow a / ab / abc / abcd$

$S \longrightarrow aS'$

$S' \longrightarrow b / bc / bcd / \varepsilon$

$S' \longrightarrow bB / \varepsilon$

$B \longrightarrow c / cd / \varepsilon$

$B \longrightarrow cC / \varepsilon$

$C \longrightarrow d / \varepsilon$

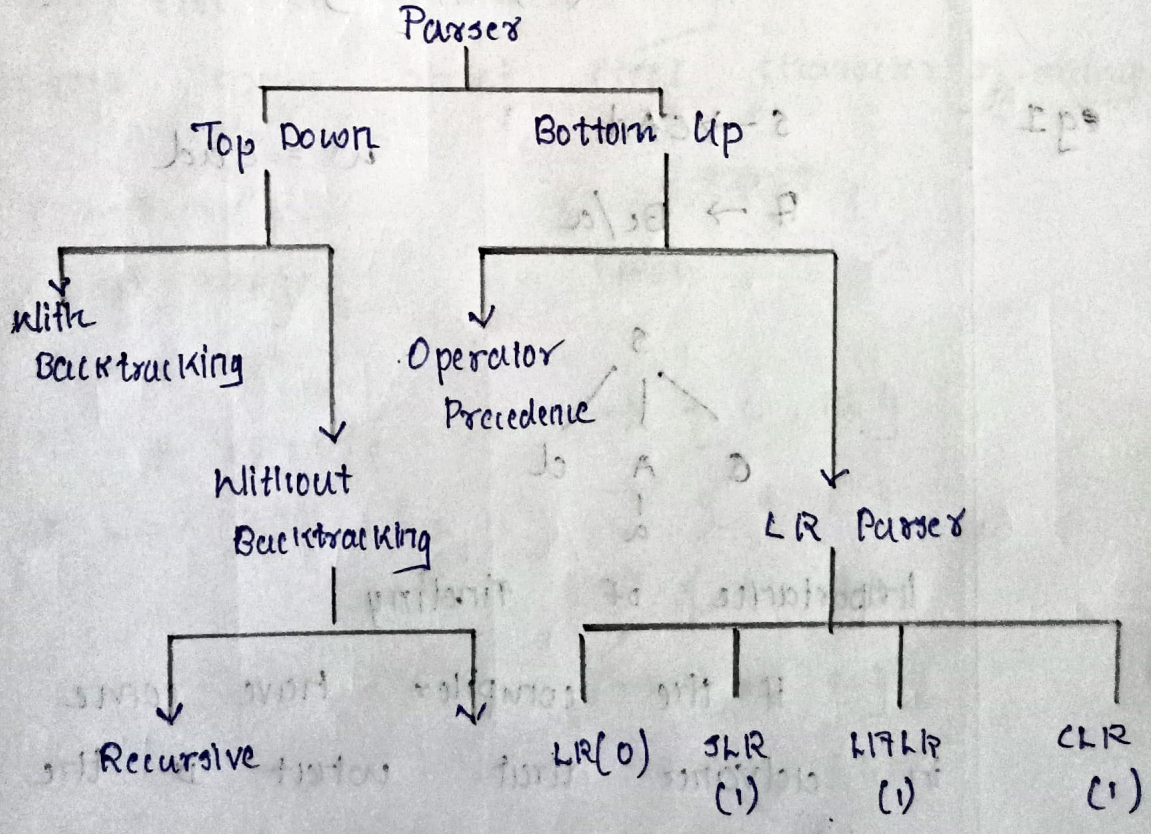
$S \longrightarrow aS'$

$S' \longrightarrow bB / \varepsilon$

~~$B \longrightarrow cd / cd /$~~

$B \longrightarrow cC / \varepsilon$

$C \longrightarrow d / \varepsilon$



FIRST

1. If X is a terminal, FIRST(X) = {X}
2. If X is a non-terminal, and $X \to \varepsilon$ is a production, FIRST(X) = {ε}
3. If X is a NT, and $X \to Y_1, Y_2, \ldots Y_n$

FIRST(X) = FIRST(Y_1)

IF $Y_1 = \varepsilon$

FIRST(X) = FIRST(Y_2)

IF $Y_2 = \varepsilon$

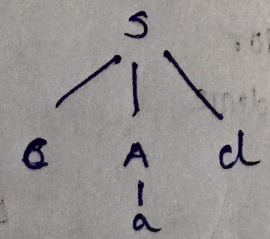
FIRST(X) = FIRST(Y_3)

IF $Y_3 = \varepsilon$

FIRST(X) = {ε}

eg 1 $S \to cAd$ $w = cad$

$A \to Bc/a$

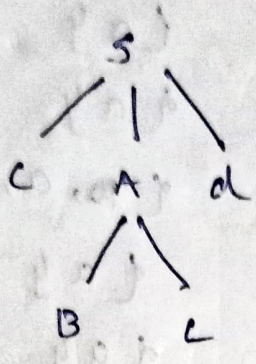


Importance of Finding

If the compiler have come to know in advance that what is the first character of the string produced when

a production rule is applied. It can take decisions wisely on which production rule to apply

As in eg 1 if we take



we will not obtain the string, so we backtrack and ~~then~~ apply production $A \rightarrow a$

If compiler knew after reading character c and next character is A, then it would have ignored the production rule $A \rightarrow Bc$ and directly use production rule $A \rightarrow a$.

This can be achieved only if compiler or parser knows about first character of string

1. $E \rightarrow TE'$

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

   $T \rightarrow FT'$

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

   $F \rightarrow (E) / id$

~~FIRST~~

FIRST

$\rightarrow \{ c, id \}$

$\rightarrow \{ +, \varepsilon \}$

$\rightarrow \{ c, id \}$

$\rightarrow \{ *, \varepsilon \}$

$\rightarrow \{ c, id \}$

2. A → abB / Ce — {a, f, e}
B → d — {d}
C → f / ε — {f, ε}

3. S → AB — {a}
A → a — {a}
B → C / b — {b, a, b}
C → D — {a}
D → E — {a}
E → a — {a}

4. S → Abc / ad → {e, f, p, a}
A → eS / cr → {e, f, p}
C → f / p → {f, p}

5. S → ABD → {a, b, d, ε}
A → a / BSB → {a, b, d, ε}
B → b / D → {b, d, ε}
D → d / ε → {d, ε}

6. S → TS / [S]S /)S / ε → {(, [,), ε}
T → (x) → {(}
X → TX / [X]X / e → {(, [, ε}

7. $S \rightarrow AaAb \rightarrow \{a\}$

$S \rightarrow Bb \rightarrow \{b\}$

$A \rightarrow \varepsilon \rightarrow \{\varepsilon\}$

$B \rightarrow \varepsilon \rightarrow \{\varepsilon\}$

8. $A \rightarrow BCx/y \rightarrow \{y, x\}$

$B \rightarrow yA/\varepsilon \rightarrow \{y, \varepsilon\}$

$C \rightarrow Ay/x \rightarrow \{y, x\}$

9. $A \rightarrow BC \rightarrow \{x, y\}$

$B \rightarrow Ax/x/\varepsilon \rightarrow \{x, y, \varepsilon\}$

$C \rightarrow yC/y \rightarrow \{y\}$

10. $S \rightarrow BAa \rightarrow \{b, a, c\}$

$A \rightarrow \varepsilon/BcA \rightarrow \{c, b, \varepsilon\}$

$B \rightarrow \varepsilon/b \rightarrow \{\varepsilon, b\}$

Follow

● Need for finding Follow

$A \rightarrow aBb$ String = ab

$B \rightarrow c/\varepsilon$

As the first character is 'a', the parser produces $A \rightarrow aBb$. Now the parser checks for the second character. ie b for that we apply production $B \rightarrow \varepsilon$. If that

is applied, then B will vanished and parser get input ab. But parser can apply it only when it knows that character that follow b is same as the current character in the input. As a conclusion we apply First and follow so that parser can properly apply needed rule at the correct position.

2.

	FIRST	FOLLOW
S → Bb/Cd	{a, b, c, d}	{$}
B → aB/ε	{a, ε}	{b}
C → cC/ε	{c, ε}	{d}

2.

	FIRST	FOLLOW
E → TE'	{id, (}	{$,)}
E' → +TE'/ε	{+, ε}	{$,)}
T → FT'	{id, (}	{+, $,)}
T' → *FT'/ε	{*, ε}	{+, $,)}
F → id/(E)	{id, (}	{*, +, $,)}

3.

$S \rightarrow aABb$	{a}	{$, b}
$A \rightarrow c \mid \varepsilon$	{c, ε}	{a, b}
$B \rightarrow d \mid \varepsilon$	{d, ε}	{b, $}

4.

$S \rightarrow HCB \mid CbB \mid Ba$	{d, g, h, ε, b, a}	{$}
$H \rightarrow da \mid BC$	{d, g, h, ε}	{h, g, $}
$B \rightarrow g \mid \varepsilon$	{g, ε}	{$, a, h, g}
$C \rightarrow h \mid \varepsilon$	{h, ε}	{g, b, g, $, h}

5.

$S \rightarrow aBDh$	{a}	{$}
$B \rightarrow cC$	{c}	{g, f, h}
$C \rightarrow bC \mid \varepsilon$	{b, ε}	{g, f, h}
$D \rightarrow EF$	{g, f, ε}	{h}
$E \rightarrow g \mid \varepsilon$	{g, ε}	{f, h}
$F \rightarrow f \mid \varepsilon$	{f, ε}	{h}

Construction Of Parsing Table

	a	h	c	g	f	$	$
S	$S \rightarrow aBDh$						
B			$B \rightarrow cC$				
C		$C \rightarrow bC/\varepsilon$		$C \rightarrow bC/\varepsilon$	$C \rightarrow bC/\varepsilon$	$C \rightarrow bC/\varepsilon$	
D		$D \rightarrow EF$		$D \rightarrow EF$	$D \rightarrow EF$		
E		$E \rightarrow g/\varepsilon$		$E \rightarrow g/\varepsilon$	$E \rightarrow g/\varepsilon$		
F		$F \rightarrow f/\varepsilon$			$F \rightarrow f/\varepsilon$		

Consider a parsing table for the Production

$E \rightarrow TE'$	{id, (}	{$,)}
$E' \rightarrow +TE'/\varepsilon$	{+, ε}	{$,)}
$T \rightarrow FT'$	{id, (}	{+, $,)}
$T' \rightarrow *FT'/\varepsilon$	{*, ε}	{+, $,)}
$F \rightarrow id(E)$	{id, (}	{*, +, $,)}

Parsing Table

	id	*	+	(	)	$
E						
E'						
T						
T'						
F						