WWW.KTUNOTES.IN

MODULE-2.

CFG

E → E+E | E*E | (E) | -E | id.

Derive the tree. for -(id+id)

Use left most derivation

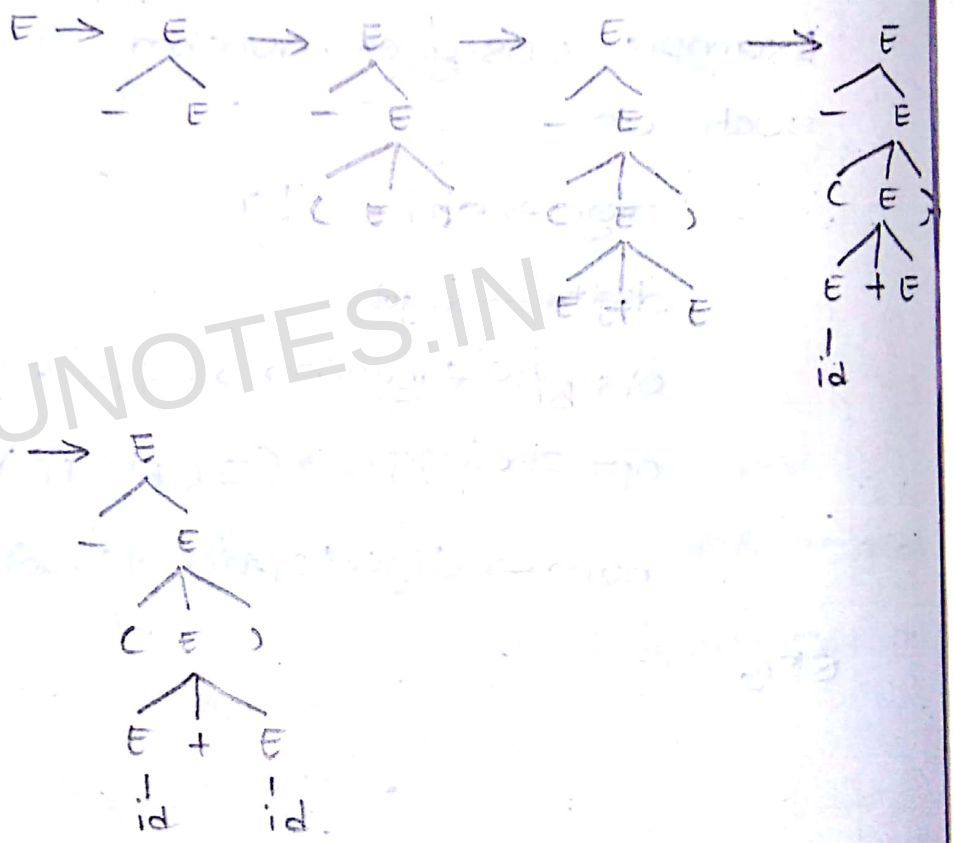
E → -E

→ -(E)

→ -(E+E)

→ -(id+E)

→ -(id+id)



Many programming language constructs have an inherently recursive structure (syntax) that can be defined by CFG

CFG consists of terminals, non-terminals, start symbol

production.

DERIVATION

A production is treated as rewriting rule in which non-terminal on left is replaced by the string on the right side of the production.

PARSE TREE

A parse tree may be viewed as a graphical representation for a derivation that filters out the choice regarding the replacement order.

AMBIGUITY

id + id * id.

LMD.

E → E + E
→ id + E
→ id + E * E
→ id + id * E
→ id + id * id

E → E * E
→ E + E * E
→ id + E * E
→ id + id * E
→ id + id * id.

A grammar that produces more than one parse tree for the same string is said to be ambiguous i.e., an ambiguous grammar is one that produces more than one left most or more than one right most for the same string.

LEFT FACTORING

: is a grammer transformation that is useful for producing a grammar suitable for predictive parsing.

Eg: Consider the CFG

Stmt → if Expr then Stmt

Stmt → if Expr then Stmt Else stmt

Stmt → a

Expr → b

Consider general form of grammar

$A \rightarrow \alpha\beta_1 | \alpha\beta_2$

It can be left factored as

$A \rightarrow \alpha A'$

$A' \rightarrow \beta_1 | \beta_2$

30.1.18.

S → iEtS

S → iEtSeS

S → a

E → b.

After left factoring

S → iEtSS'

S' → ε | eS

S → a

E → b

If there are more productions in a CFG which is non-deterministic we can write general form as given below:

$A \to \alpha\beta_1 | \alpha\beta_2 | \ldots | \alpha\beta_n | \gamma$

where γ represents all alternative that do not begin with α.

$A \to \alpha A'$

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

$A \to \gamma$

It can be written as

$A \to \alpha A' | \gamma$

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

ELIMINATION OF LEFT ~~FACTORING~~ RECURSION

A grammar is left recursive if it has an non-terminal A such that there is a derivation $A \overset{+}{\Rightarrow} A\alpha$ for some string α.

Top-down parsing method cannot handle left recursive grammar so a transformation that eliminates left recursion is needed.

The production of the form $A \to A\alpha | \beta$ can be replaced by the non-left recursive productions as:

$$A \rightarrow \beta A'$$
$$A' \rightarrow \alpha A' \mid \varepsilon$$

Consider the grammar

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

$A \rightarrow A\alpha \mid \beta$ $\Rightarrow$ $A \rightarrow \beta A'$, $A' \rightarrow \alpha A' \mid \varepsilon$

$E \rightarrow E+T \mid T \Rightarrow E \rightarrow TE'$, $E' \rightarrow +TE' \mid \varepsilon$

$T \rightarrow T*F \mid F \Rightarrow T \rightarrow FT'$, $T' \rightarrow *FT' \mid \varepsilon$

$F \rightarrow (E) \mid id \Rightarrow F \rightarrow (E) \mid id$

More generally it can be written as

$$A \rightarrow A\alpha_1 \mid A\alpha_2 \mid \ldots \mid A\alpha_m \mid \beta_1 \mid \beta_2 \mid \ldots \mid \beta_n$$

The left recursion can be eliminated as:

$$A \rightarrow \beta_1 A' \mid \beta_2 A' \mid \ldots \mid \beta_n A'$$
$$A' \rightarrow \alpha_1 A' \mid \alpha_2 A' \mid \ldots \mid \alpha_m A' \mid \varepsilon$$

S1.1.18

FIRST AND FOLLOW

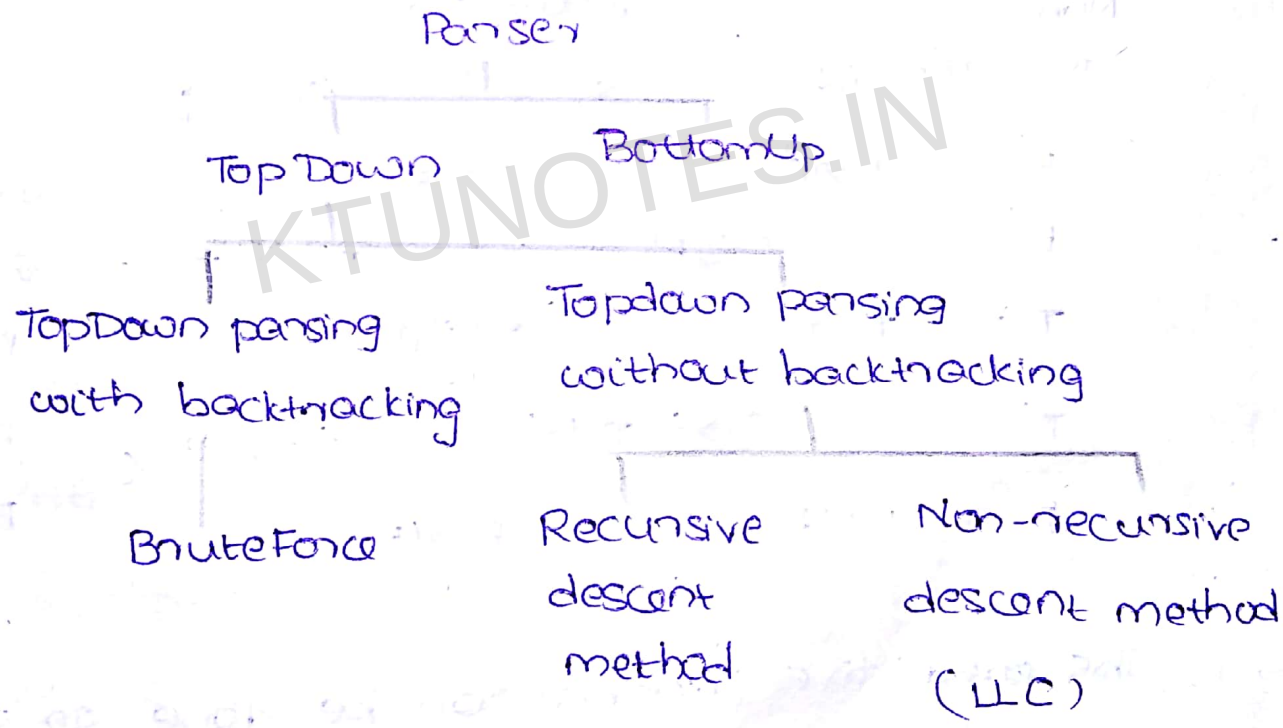
		FIRST	FOLLOW
$S \rightarrow ABCD$	S	{a,b,c}	{$}
$A \rightarrow a \mid \varepsilon$	A	{a,ε}	{b,c}
$B \rightarrow b \mid \varepsilon$	B	{b,ε}	{c}
$C \rightarrow c$	C	{c}	{d,$}
$D \rightarrow d \mid ?$	D	{d,ε}	{$}

2.2.18

Q Write first and follow of all non-terminals of given CFG.

	FIRST	FOLLOW
$E \to TE'$	{C, id}	{$,)}
$E' \to +TE'/\varepsilon$	{+, ε}	{$,)}
$T \to FT'$	{C, id}	{+, $,)}
$T' \to *FT'/\varepsilon$	{*, ε}	{+, $,)}
$F \to (E)/id$	{C, id}	{*, +, $,)}

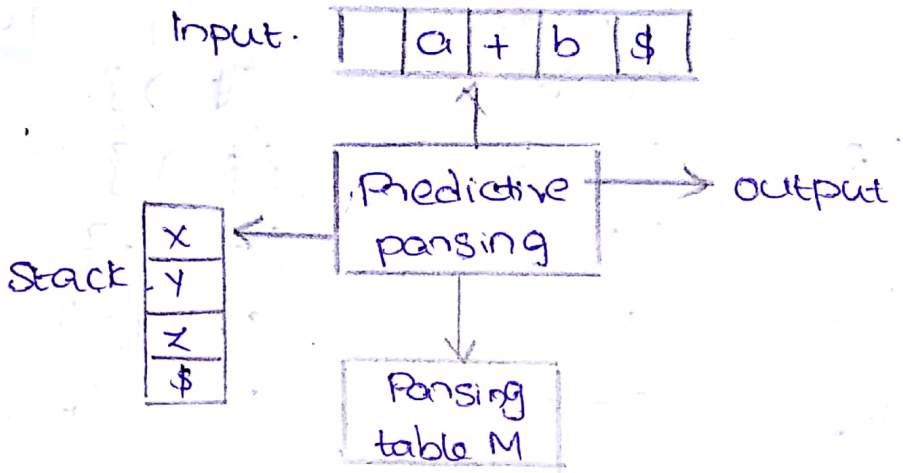
TOP DOWN PARSING



PREDICTIVE

A special form of recursive descent parsing in which lookahead symbol unambiguously descent the procedure selected for each non-terminal.

MODEL OF NON-RECURSIVE PREDICTIVE PARSING



Non-terminal	input Stack					
	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'		$T' \to \varepsilon$	$T' \to *FT'$		$T' \to \varepsilon$	$T' \to \varepsilon$
F	$F \to id$			$F \to (E)$		

$E \to TE'$

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

$T \to FT'$

$T' \to *FT' | \varepsilon$

$F \to (E) | id$

The predictive parsing can be done on the input id + id * id as follows

Stack	input	output
$E	id+id*id	—
$E'T	id+id*id	$E \to TE'$
$E'T'F	id+id*id	$T \to FT'$

$E'T'id	id+id*id	F→id
$E'T'	+id*id	T'→ε
$E'	+id*id	E'→+TE'
$E'T+	+id*id	—
$E'T	id*id	T→FT'
$E'T'F	id*id	F→id.
$E'T'id	id*id	—
$E'T'	*id	T'→*FT'
$E'T'F*	*id	—
$E'T'F	id	—
$E'T'id	id	F→id
$E'T'	$	—
$E'	$	T'→ε
$	$	E'→ε

9.8.18

NON-RECURSIVE PREDICTIVE PARSING

It is possible to build a non-recursive predictive parser by maintaining a stack explicitly rather than implicitly via recursive calls. The main problem during predictive parsing is that of

determining a production to be applied for a non-terminal. The non-recursive parser looks up the production to be applied in a parsing table. A table driven predictive parser has an input

(i) input buffer

(ii) stack.

(iii) parsing table

(iv) output stream.

LL(1) : Every cell has single entry.

1. $S \to iEtSS' | a$.

$S' \to eS | \varepsilon$

$E \to b$

Check whether given grammar is LL(1).

Step 1: Find first and follow of each non-terminal.

	First	Follow
$S \to iEtSS' \mid a$	$\{i, a\}$	$\{\$, e\}$
$S' \to eS \mid \varepsilon$	$\{e, \varepsilon\}$	$\{\$, e\}$
$E \to b$	$\{b\}$	$\{t\}$

Non-terminal	input stack. i	t	a	e	b	$
S	$S \to iEtSS'$		$S \to a$			
S'				$S' \to eS$		$S' \to \varepsilon$
E					$E \to b$	