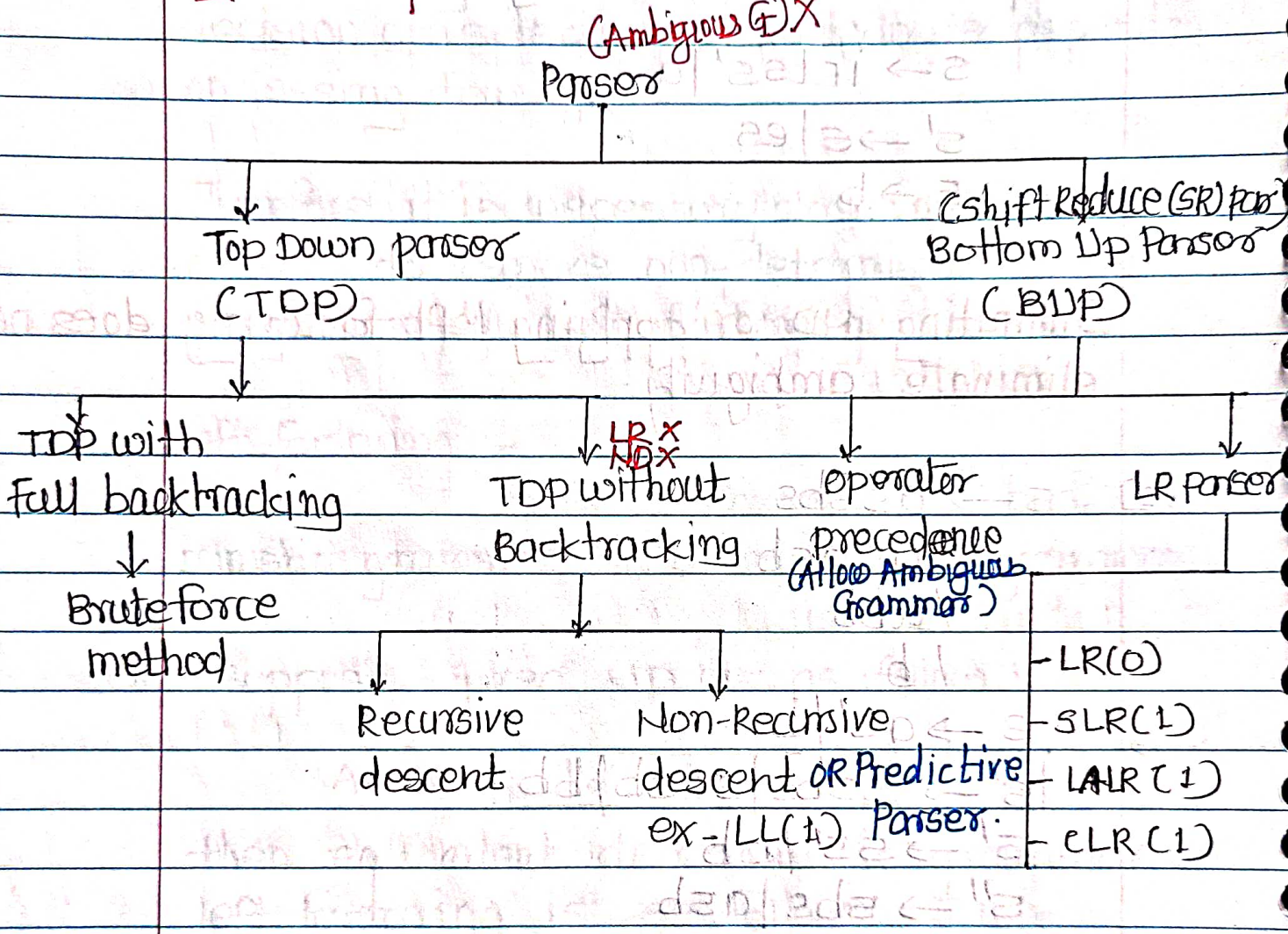
Types of parser:-



ex

S → aABe

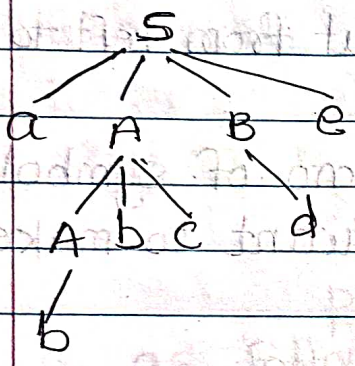
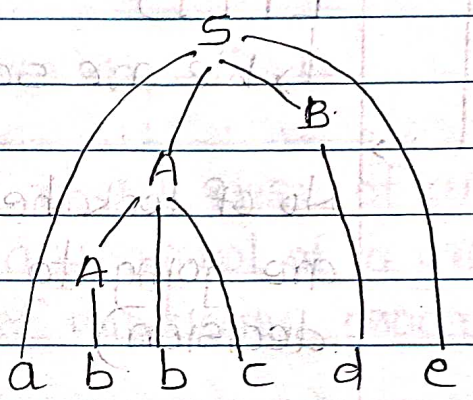
A → Abc | b

B → d

string - abbcde

shift → push.

Reduce - pop.

TDP	BUP
1] start at the root of derivation tree	1] start at the leaves
2] 	2] 
3] More emphasis on "What alternate production to use if we are having more than one production"	3] more emphasis on "When to reduce a terminal"
4] Production rules followed to generate above parse tree is as follow S → aABe [S → aABe] S → aAbcBe [A → Abc] → abbcBe [A → b] → abbcde [B → d]	4] S → aABe [S → aABe] → aAde [B → d] → aAbcde [A → Abc] → abbcde [A → b]
5] By analyzing step 4. TDP follow Left most derivation	5] BUP follow reverse of right most derivation.

LL(1) Parser :- (Top Down Parser)

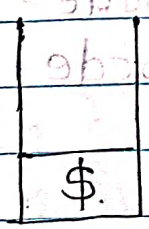
- Using Left most derivation
- LL(1)
  - ↳ We are scanning the input from left to right.
- No of lookahead (how many no. of symbol you are going to see when you want to make decision)

Components :-

1] LL(1) parser (Algorithm)

2] LL(1) parsing table :- data structure that will get constructed by using given grammar.

3] stack :- required for processing of parser.



$ is always present at the bottom of stack.

4] Input buffer :- at the end of input string $ symbol is there.

In LL(1) parsing following two functions are important:

1] First()
2] Follow()

1] First() :- If i want to find out first of any non-terminal or any symbol of a grammar present on LHS of production procedure is as follows.

a] Find all possible strings that get generated starting from that non-terminal symbol.

b] Find all possible first symbol or terminal of for those many number of strings.

c] Whatever output symbols we are getting in step b indicate the first of that symbol.

First of any variable may contain ϵ.

If any production directly derive ϵ then include ϵ in first of that non-terminal.

2) Follow():- What is the terminal which can follow a variable in the process of derivation.

In LL parser I/p is alway followed by $ so when we start derivation $ is always at the end. i.e. S$

follow will never contain ϵ

If variable or Non-terminal is at the right most side then follow of that variable is the follow of Left hand side variable or non-terminal.

Ex $A \rightarrow BC$

here, follow of C is follow of A.
because suppose Aabc is a string, where A is followed by a string abc if i replace $A \rightarrow BC$ then

BCabc, here also C is followed by string abc.
that why follow of C is follow of A.

ex. $S \rightarrow aABCD$

$A \rightarrow b$

$B \rightarrow c$

$C \rightarrow d$

$D \rightarrow e$

→ First of S is {a}

First of A is {b}

B {c}

C {d}

D {e}

Follow of S is {$}.

then we try to search S on right hand side of production rules. If we found S on RHS then we find follow for that

Follow of A is {c}

B {d}

C {e}

D {$}

2]

	First	Follow
$S \rightarrow ABCDE$	{a, b, c}	{$}
$A \rightarrow a \mid \epsilon$	{a, ϵ}	{b, c}
$B \rightarrow b \mid \epsilon$	{b, ϵ}	{c}
$C \rightarrow c$	{c}	{d, e, $}
$D \rightarrow d \mid \epsilon$	{d, ϵ}	{e, $}
$E \rightarrow e \mid \epsilon$	{e, ϵ}	{$}

2]

	First	Follow
$S \to Bb \mid Cd$	$\{a, b, c, d\}$	$\{\$\}$
$B \to aB \mid \epsilon$	$\{a, \epsilon\}$	$\{b\}$
$C \to cC \mid \epsilon$	$\{c, \epsilon\}$	$\{d\}$

3]

	First	Follow
$E \to TE'$	$\{id, c\}$	$\{\$,)\}$
$E' \to +TE' \mid \epsilon$	$\{+, \epsilon\}$	$\{\$,)\}$
$T \to FT'$	$\{id, c\}$	$\{+, \$,)\}$
$T' \to *FT' \mid \epsilon$	$\{*, \epsilon\}$	$\{+, \$,)\}$
$F \to id \mid (E)$	$\{id, c\}$	$\{*, +, \$,)\}$

4]

	First	Follow.
$S \to ACB \mid CbB \mid Ba$	$\{d, g, h, b, a, \epsilon\}$	$\{\$\}$
$A \to da \mid BC$	$\{d, g, h, \epsilon\}$	$\{h, g, \$\}$
$B \to g \mid \epsilon$	$\{g, \epsilon\}$	$\{\$, a, h, g\}$
$C \to h \mid \epsilon$	$\{h, \epsilon\}$	$\{g, \$, b, h\}$

ex.5]

	First	Follow.
$S \to aABb$	$\{a\}$	$\{\$\}$
$A \to c \mid \epsilon$	$\{c, \epsilon\}$	$\{d, b\}$
$B \to d \mid \epsilon$	$\{d, \epsilon\}$	$\{b\}$

6]

	First	Follow
$S \to aBDh$	$\{a\}$	$\{\$\}$
$B \to cC$	$\{c\}$	$\{g, f, h\}$
$C \to bC \mid \epsilon$	$\{b, \epsilon\}$	$\{g, f, h\}$
$D \to EF$	$\{g, f, \epsilon\}$	$\{h\}$
$E \to g \mid \epsilon$	$\{g, \epsilon\}$	$\{f, h\}$
$F \to f \mid \epsilon$	$\{f, \epsilon\}$	$\{h\}$

Construction of LL parser table:- To construct LL parser table we required First & follow.

Consider example 3.

a] For given grammar find first & follow.

Rules 1] Place the production in first of right hand side.

$E \to TE'$

above production is place in row E & the column is equal to first of (TE') i.e first of T. i.e $\{id, (\}$

2] When production rule contain ϵ

ex.

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

In order to place $E' \to \epsilon$, row = E' & column = follow of E' i.e $\{\$,)\}$

All ϵ production has to place in follows of LHS.

row ← LHS Variable

Column ← first of RHS.

	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)		

If we see ϵ in first of RHS then you place the production in follow of LHS.

If production is E→ϵ then place this under column = follow of E. Why?

ex (id E) $

suppose we want the string (id).

When to vanish E only when follow of E is)

While constructing parse tree parsing table is used.

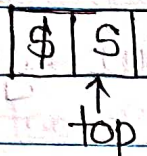
ex

	First	Follow
S→(S)|ϵ	{(, ϵ}	{$,)}

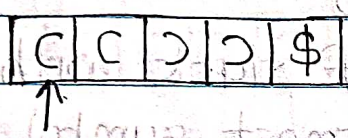
	(	)	$
S	S→(S)	S→ϵ	S→ϵ

suppose we want to generate (( )) $

→ Initially, always bottom of stack is $ & top of stack is start symbol i.e S.



Input buffer :-



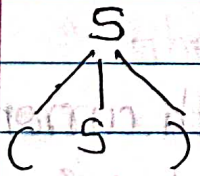
Top of stack is S, on current i/p C

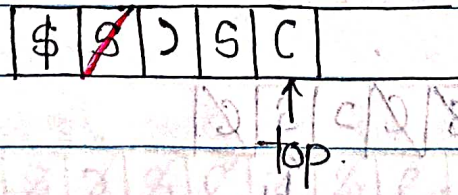
∴ find entry in LL(1) parsing table in row S & column C i.e $S \rightarrow (S)$

∴ replace the top of the stack with RHS

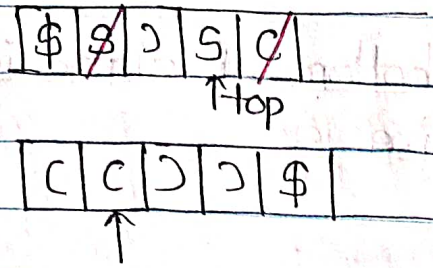
∴ top of stack S is replaced by (S)

such that ~~Right~~ Left most symbol of RHS should appear on the top of the stack.





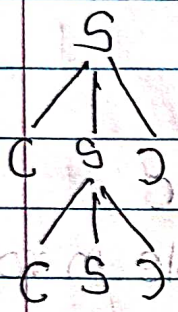
Now ~~stack~~ top of stack & the current input read get matched so, increment input buffer pointer & also pop the top of stack.



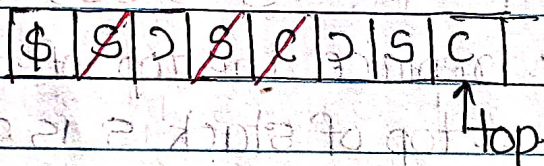
Now stack of top is S & input symbol read is (

∴ use production rule of row S & column (

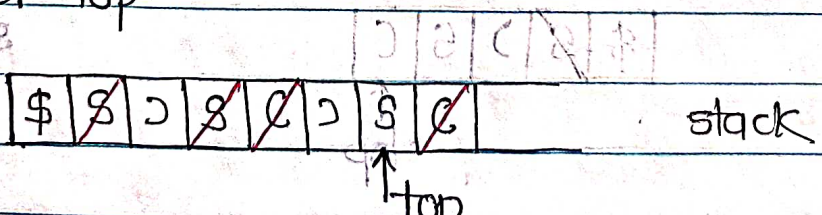
S → (S)



replace top of stack with (S) (RHS). such that Leftmost symbol appear on top



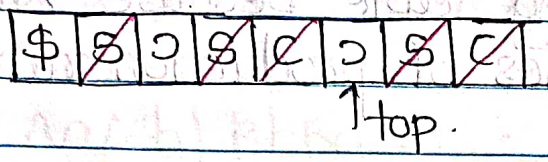
Now, top of stack is matching with current i/p read. therefore increment input pointer & pop the stack of top.



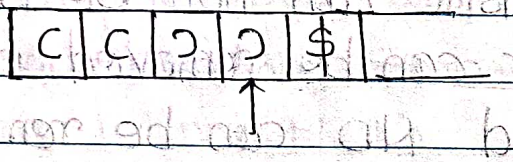
Now, top of stack is S & i/p is), therefore use production of row S & column)

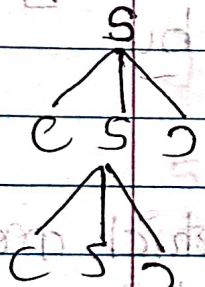
i.e $S \rightarrow \epsilon$

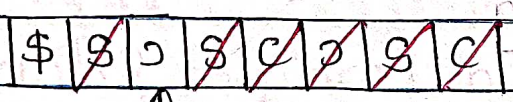
then replace top of stack S by ϵ. i.e pop top of stack.



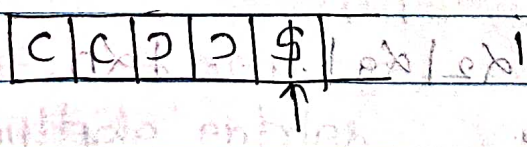
Now, top of stack is matching with input read. therefore, increment i/p & pop top of stack.

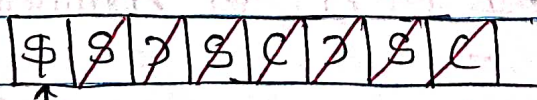






Now, top of stack, & i/p read is matching. therefore increment i/p pointer & pop the top of stack





Now, symbol read is $ & top of stack is also $ ∴ the string is accepted by grammar.

Is for every grammar parsing table created?

→ No, for some grammar we are unable to create parsing table because we might get multiple entries of production rule for same cell.

Any grammar that is Left rearsive & non-deterministic can not be used for LL(1) LR grammar can be removed using RR. (Right recursion. and ND can be removed by Left factoring.

But still there are some grammar which are not LR & ND still they cannot used for LL(1) parser because we get more than one entry for given symbol on some terminal.

ex. $A \rightarrow \alpha_1 | \alpha_2 | \alpha_3 | \ldots | \alpha_n$

parsing table

	a	b	c	d	$
A					

If suppose to insert $A \rightarrow \alpha_1$ we are entering it in column First of A & row A for ex.

First of A = {a, b}

Now to insert $A \rightarrow \alpha_2$ again colum is {a, b} & row = A multiple entries in single cell

∴ Given grammar is not LL(1)

ex Identify whether given grammar is LL(1) or Not.

1] $S \to AaAb \mid BbBa$

$A \to \in$

$B \to \in$

If for any variable we are having more than one production then there is chance of that the grammar is not LL(1)

$S \to AaAb$ } is placed in row = S & column = First of
$S \to BbBa$ } S = {a, b} & Col = {b} respectively.

$A \to \in$ is placed in row = A column = follow of A
= {a, b}

$B \to \in \Rightarrow$ row = B, col = {b, a}

No multiple entries.

∴ Grammar is LL(1) & parsing table is created.

2] placed under

	Row	Col.
$S \to aSbS$	{S}	{a} $S \to aSbS$ First(S)
$/ bSaS$	{S}	{b} $S \to bSaS$ First(S)
$/ \in$	{S}	{b, a, $} Follow(S)

G is Not LL(1) ∴ It is ambiguous

Placed under

2]

	Row	Col.
$S \to aABb$	$\{S\}$	$\{a\}$
$A \to c \mid \epsilon$	$\{A\}$	$A \to c$ $\{c\}$, $A \to \epsilon$ $\{d, b\}$
$B \to d \mid \epsilon$	$\{B\}$	$B \to d$ $\{d\}$, $B \to \epsilon$ $\{b\}$

Grammar is LL(1)

3]

	row	col	
$S \to A \mid a$	$\{\$\}$	$S \to A$ $\{a\}$	$S \to a$ $\{a\}$
$A \to a$	$\{A\}$	$\{a\}$	

Grammar is not LL(1) :- Ambiguous.

4]

	Row	col.	
$S \to aB \mid \epsilon$	$\{S\}$	$S \to aB$ $\{a\}$	$S \to \epsilon$ $\{\$\}$
$B \to bC \mid \epsilon$	$\{B\}$	$B \to bC$ $\{b\}$	$B \to \epsilon$ $\{\$\}$
$C \to cS \mid \epsilon$	$\{C\}$	$C \to cS$ $\{c\}$	$C \to \epsilon$ $\{\$\}$

LL(1) grammar

5]

	row	col		
$S \to AB$	$\{S\}$	$\{a, b, \$\}$		
$A \to a \mid \epsilon$	$\{A\}$	$A \to a$ $\{a\}$	$A \to \epsilon$ $\{b, \$\}$	
$B \to b \mid \epsilon$	$\{B\}$	$B \to b$ $\{b\}$	$B \to \epsilon$ $\{\$\}$	

LL(1) grammar

6]

	row	col.
$S \to aSA \mid \epsilon$	{S}	$S \to aSA$ {a} $S \to \epsilon$ {c, $}
$A \to c \mid \epsilon$	{A}	$A \to c$ {c} $A \to \epsilon$ {c, $}

Not LL(1)

7]

$S \to A$	{S}	$S \to A$ {a, b, c, d}
$A \to Bb \mid Cd$	{A}	$A \to Bb$ {a, b} $A \to Cd$ {c, d}
$B \to aB \mid \epsilon$	{B}	$B \to aB$ {a} $B \to \epsilon$ {b}
$C \to cC \mid \epsilon$	{C}	$C \to cC$ {c} $C \to \epsilon$ {d}

LL(1) grammar.

8]

	row	col.
$S \to aAa \mid \epsilon$	{S}	$S \to aAa$ {a} $S \to \epsilon$ {a, $}
$A \to abs \mid \epsilon$	{A}	$A \to abs$ {a} $A \to \epsilon$ {a}

Not LL(1) grammar.

9]

	row	col.
$S \to iEtSS' \mid a$	{S}	$S \to iEtSS'$ {i} $S \to a$ {a}
$S' \to eS \mid \epsilon$	{S'}	$S' \to eS$ {e} $S' \to \epsilon$ {e}
$E \to b$	{E}	$E \to b$ {b}

Not LL(1) grammar.