Assignment 2

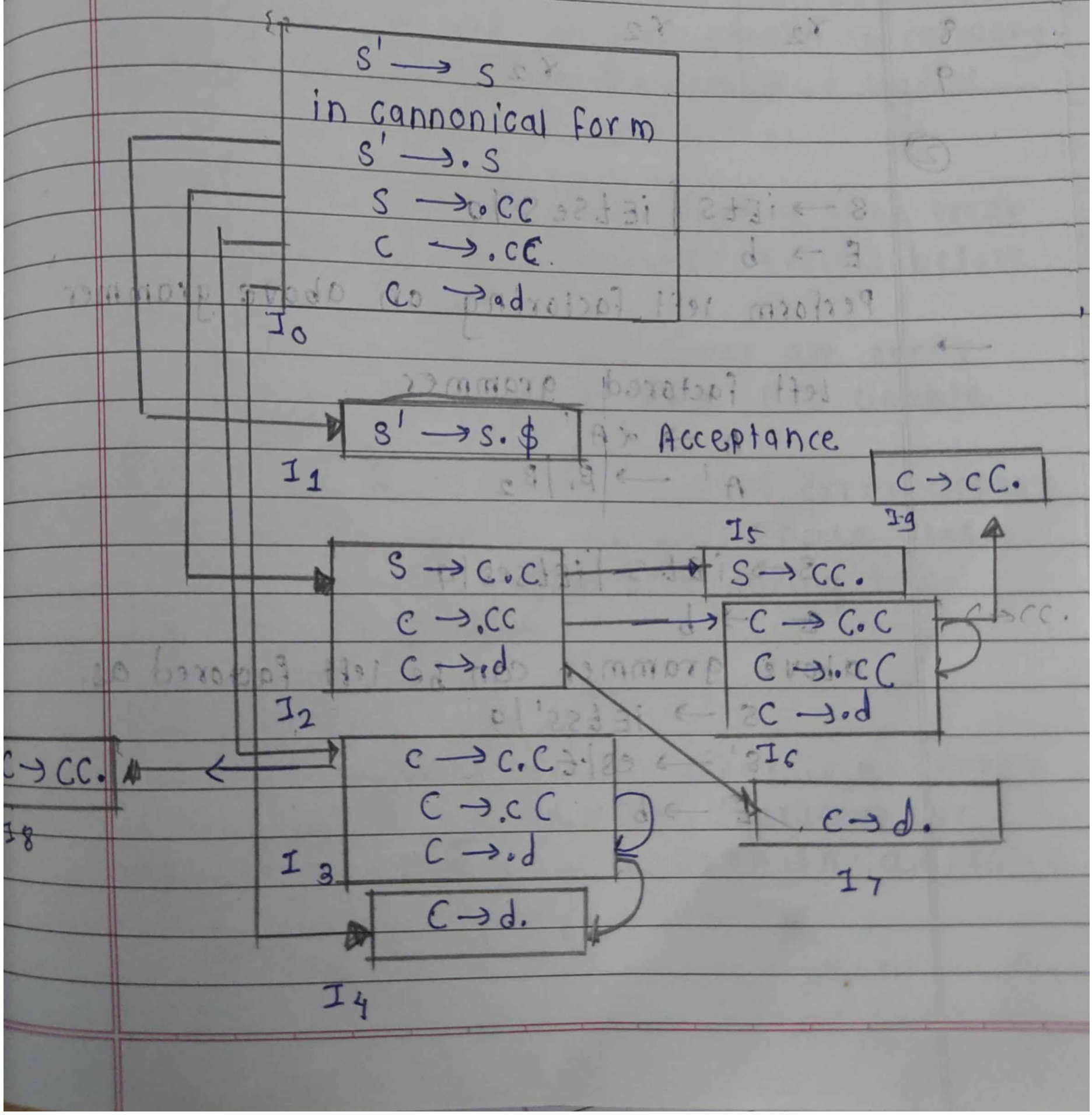
GoodLuck | Page No.
Date

(1) Construct LR(1) Parsing table for following grammer

$S \rightarrow CC$
$C \rightarrow CC$
$C \rightarrow d$

step 1 : $S' \rightarrow S$
$S \rightarrow CC$
$C \rightarrow CC$
$C \rightarrow d$

step 2 :



State	ACTION			GOTO	
	c	d	$	S	C
0	S_3	S_4		1	2
1			accept		
2	S_6	S_7			5
3	S_3	S_4			8
4	r_3	r_3			
5			r_1		
6	S_6	S_7			9
7					
8	r_2	r_2			
9			r_2		

(2)

$S \rightarrow iEtS \mid iEtSeS \mid a$

$E \rightarrow b$

Perform left factoring on above grammer

→

Left factored grammer

$A \rightarrow \alpha A'$

$A' \rightarrow \beta_1 \mid \beta_2$

$S \rightarrow iEtS \mid iEtSeS \mid a$

$E \rightarrow b$

above grammer can be left factored as

$S \rightarrow iEtSS' \mid a$

$S' \rightarrow eS \mid \epsilon$

$E \rightarrow b$

GoodLuck | Page No.
Date

(8) List and explain various errors and error recovery Strategies in syntax Analysis

→ Type of error

(1) Run-time error
(2) compile-time error

(1) Run-time error →

A Run time error is one that occur during the execution of program and is usually caused by incorrect Syetem parameters or invalid input data. Example can be alack of memory to run an application a memory conflict with another program and logical error

(2) Compile-time error → compile time error increase during the compilation process before program execute

(4) lexical error → lexical error are error that your lexer throws when it is unable to continue.

(B) Sematic error → This type of error appear during the sematic analysis phase. These types of error are detected during the compilation Process

(3) ~~Syntax error~~

(3) Syntatic error →

~~In~~ ~~Compil~~ Syntatic error is an error in syntax of sequence of character or token intended to be written in a specific lang.

(4) What is left Recursion? explain with an example

→

A Grammer is said to be left recursive if it has non-terminal A such that their is derivation in a form of $A \rightarrow A\alpha$ for some String α

$A \rightarrow$ Terminal - Non Terminal

$\alpha \rightarrow$ Set of String

OR

if the left side of non-Terminal symbol is equal to or some as left most Symbol Non-Terminal Symbol of RHS Then the grammer is said to be left recursive.

Rule for eliminating left recursion

$A \rightarrow A\alpha \mid \beta$ —(1)

Then left Recursion can be

$A \rightarrow \beta A'$ —(2)

$A' \rightarrow \alpha A' \mid \epsilon$ —(3)

e.g

$E \rightarrow E+T \mid T$

$T \rightarrow T*F \mid F$

$F \rightarrow (E) \mid id$

→

$E \rightarrow TE'$

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

$T \rightarrow FT'$

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

$F \rightarrow (E) \mid id$

GoodLuck | Page No.
Date

Q5 Is it the below grammer contain left recursion? if yes remove the left recursion from the grammer

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

→ Non-Terminal E appear at the left most symbol This production is left recursive.

2. $E \rightarrow E+T \mid T$. — 1st Rule $A \rightarrow A\alpha \mid \beta$
$E \rightarrow TE'$ — 2nd Rule $A \rightarrow \beta A'$
$E' \rightarrow +TE'$ — 3rd Rule: $A' \rightarrow \alpha A' \mid \epsilon$

2. $T \rightarrow T*F \mid F$ — 1st Rule

$T \rightarrow FT'$ — 2nd Rule
$T' \rightarrow *FT' \mid \epsilon$ — 3rd Rule.

3. $F \rightarrow (E) \mid id$

~~ref~~ $E \rightarrow TE'$
$E' \rightarrow +TE' \mid \epsilon$
$T \rightarrow FT'$
$T' \rightarrow *FT' \mid \epsilon$
$F \rightarrow (E) \mid id$

Q6 What is Recursive Descent Parsing Technique explain with an e.g

→

Recursive Descent Parsing technique sometime requires backtracking process to find Correct A-production

As backtraking process increses the time and reduces the efficiency. . normally recursive descent parsing technique not used practically
example.

S → cAd
A → ab | a

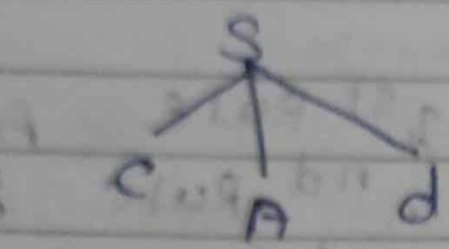


fig (a)

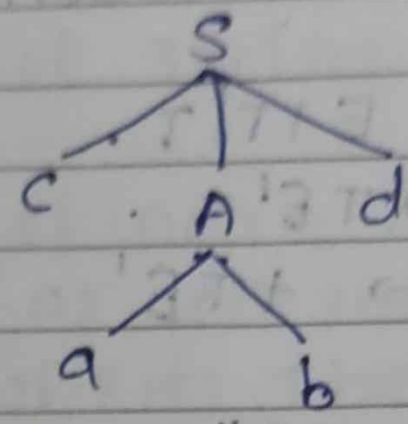


fig (b)

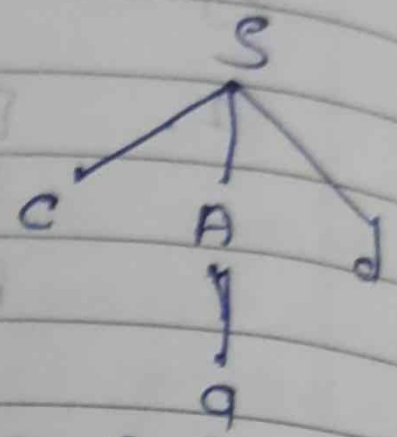
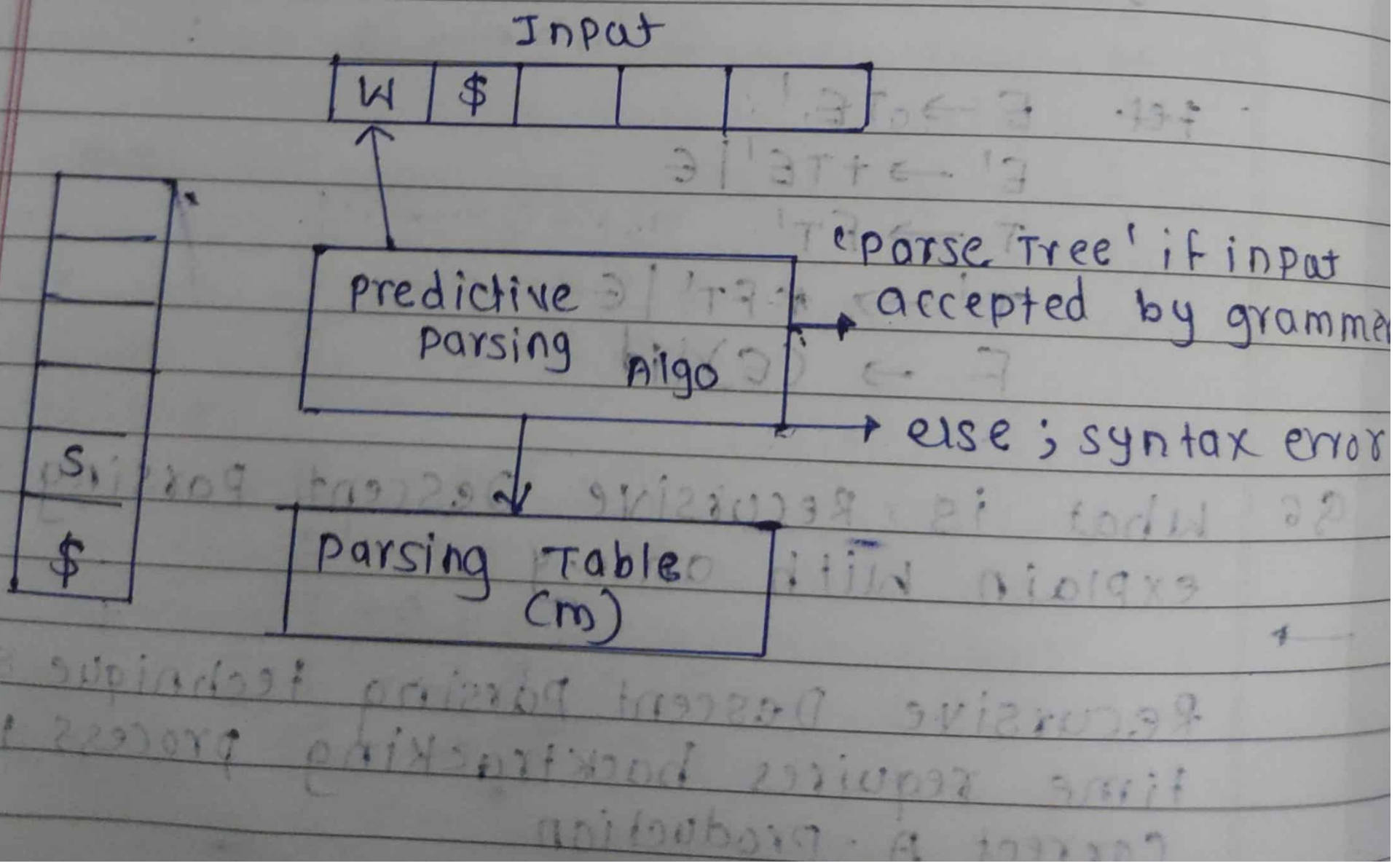


fig (c)

Q7 Write algorithm for predective parsing Technique

→



Algorithm

Input: A string 'w' a parsing table m and grammer G.

output: if 'w' is in L(G) a leftmost derivation of 'w' otherwise on error indication

method: intially the parser is in a configuration with w$ in the input buffer and the start symbol S of G on top of stack above $.
let a be the first symbol of w;
let x be the top of stack symbol;
While (x ≠ $) {/* stack not empty */
if (x = a) pop the stack and let a be the next symbol of w;
else if (x is a terminal) error();
else if (m[x,a] is an error entry) error();
else if (m[x,a] = x → y1, y2 .... yk) {
output the production x → y1, y2 .. yk;
Pop the stack
push yk yk-1 .... y1 onto the stack
with y1 on top;
}
}

Q8. Compute FIRST and Follow set for the given grammer below.

1. $E \rightarrow TE'$
   $E' \rightarrow +TE' | \epsilon$
   $T \rightarrow FT'$
   $T' \rightarrow *FT' | \epsilon$
   $F \rightarrow (E) | id$

(1) FIRST SET →

$E \rightarrow TE'$

FIRST (E) = FIRST (T)
= FIRST (F)

FIRST (T) = { (, id }

$E' \rightarrow +TE | \epsilon$

FIRST (T') = { #, ∈ }

(2) Follow set →

Follow (E) = { $,) }

Rule 1
$A \rightarrow \alpha B \beta$ = FIRST (B)

Follow (E') = Follow (E)
= { $,) }

Follow (T) = { FIRST (E') - ∈ } ∪ { FOLLOW (E) }
= { +, ∈ - ∈ } ∪ { $,) }
= { +, $,) }

Follow (T') = { follow (T) }
= { +, $,) }

Follow(F) = { FIRST(T') - ∈ } ∪ { FOLLOW(T) }
= { *, ∈ - ∈ } ∪ { +, $,) }
= { *, +, $,) }

Eg check the below grammer is in LL -
② S' → S$
S → Ss+ | ss* | a

→ ① FIRST SET →
S' → S
FIRST(S') = FIRST(S)
= FIRST(S)
FIRST(S) = { S, a }
FIRST(S) = { S, a, $ }.

② Follow set →
Follow(S) = { $ }
① follow(S) = { FIRST(S) - ∈ } ∪ { follow(S) }
= { S, a - ∈ } ∪ { $, ∈ }
= { S, a, $ }.

GoodLuck | Page No. | Date

Q9 Check the below grammer is in LL(1) or not
S → iEtS | iEtSeS | a
E → b

→

	FIRST	Follow
S	{i}	{$,e}
S'	{e,ϵ}	{$,e}
E	{b}	{t}

Predictive Parsing Table.

	m	i	t	e	b	a	$
S		S→iEtSS'					
S'				S'→eS S'→ϵ			S'→ϵ
E					E→b		

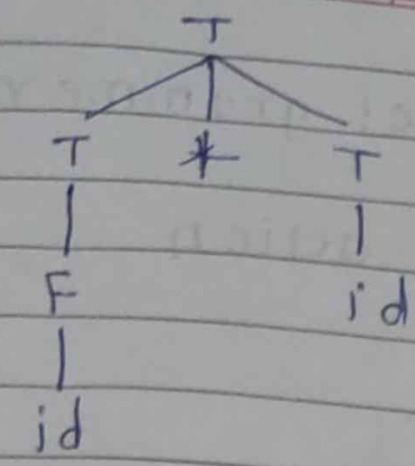
Q10 What is bottom-up parsing technique? explain Shift-Reduce parsing Technique with ex

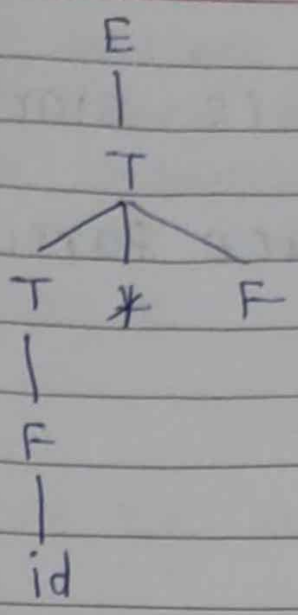
→ Bottom up parse corresponds to the construct of a parse tree for an input string beginning at the lower (bottom) and working up towards the root (the top)
e.g

E → E+T | T
T → T*F | F
F → (E) | id

id * id

F * id	T * id	T * F
|	|	| |
id	F	F id
	|	|
	id	id





Bottom up parsing is also known as process of reduction

ex. id * id
F * id
T * id
T * F
T
E

* shift reduce parsing ~~is of~~

shift reduce parsing is a form of bottom-up parsing in which a stack holds grammer symbol and an input buffer holds input string to be parsed

We use $ to mark the bottom of the stack and also to represent end of the input string

intially the stack is empty and string is on input as follow

STACK	Input
$	w$

final configuration of stack and input after succesful completion of parsing the input strin is

STACK	INPUT
$S	$

Where S is start symbol of grommer

Shift - Reduce Parsing use 4-action

① Shift
② Reduce
③ Accept
④ error

① Shift → Shift the input symbol on top of stack
② Reduce → When top of the stack matches RHS of production then it can be reduced to left side of production
③ Accept → if input is succesfully parsed then it is an accept action
④ error → else an error and call an error recovery routine.

example →

$S \rightarrow AB$
$A \rightarrow a$
$B \rightarrow b$

and input string ab

Stack	Input	Action
$	ab$	Shift a
$a	b$	Reduce A → a
$A	b$	Shift b
$Ab	$	Reduce B → b
$AB	$	Reduce S → AB
$S	$	Accept

Total action used to parse input is 6

GoodLuck | Page No. | Date

Q11) explain Removing left factori recursion and left factorial of grammer with help of example

→ ① A grammer is left recursive if it has a non-terminal A such that there is derivation $A \to A\alpha$ for some string α.

if the production is

$$A \to A\alpha | \beta$$

then left recursion can be eliminated by rewriting the above grammer as

$$A \to \beta A'$$
$$A' \to \alpha A' | \epsilon$$

} free from left recursion

example →

$$E \to E+T | T$$
$$T \to T*F | F$$
$$F \to (E) | id$$

→ removing a left recursion

$$E \to TE'$$
$$E' \to +TE' | \epsilon$$
$$T \to FT'$$
$$T' \to *FT' | \epsilon$$
$$F \to (E) | id$$

② left factorial →

left factorial is a grammer that is useful for producing grammer suitable for predictive or top-down parsing technique.

e if grammer is

$$A \to \alpha B_1 | \alpha B_2$$

removing left factorial

$$A \to \alpha A'$$
$$A' \to B_1 | B_2$$

example →

S → iEts | iEtses | a

E → b

→

S → iEts S' | a

S' → es | ∈

E → b

Q13) Give leftmost and rightmost derivation of string daab write following grammer Also draw the derivation tree

S → dAB

A → aA | a

B → bB | b

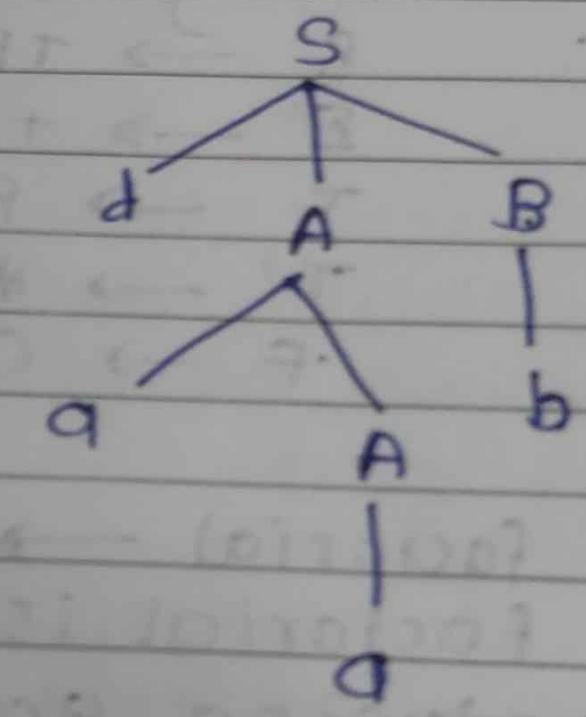
leftmost

S → dAB

→ daAB

→ daaB

→ daab



GoodLuck | Page No.
Date

$S \xrightarrow{\text{leftmost}} dAB$

$\xrightarrow{\text{leftmost}} daAB$

$\xrightarrow{\text{leftmost}} daaB$

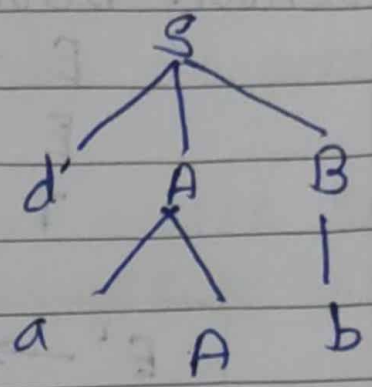
$\xrightarrow{\text{leftmost}} daab$

$S \xrightarrow{\text{Rightmost}} dAB$

$S \longrightarrow dAb$

$S \longrightarrow daAb$

$S \longrightarrow daab$



Q(4) What are the conditions to check SR-conflict and RR-Conflict in SLR(1) Parsing technique.

→

(1) If SLR(1) contain a state like

$A \to \alpha . a\beta$

$B \to \gamma .$

We cannot directly say that state is S-R conflict instead we need to find

Follow(B)

if follow(B) = {a} (symbol after '.')

then we say that state ~~consider~~ contain SR conflict and hence grammer is not in SLR(1)

(2) if SLR(1) contain a state like

$A \to \alpha .$

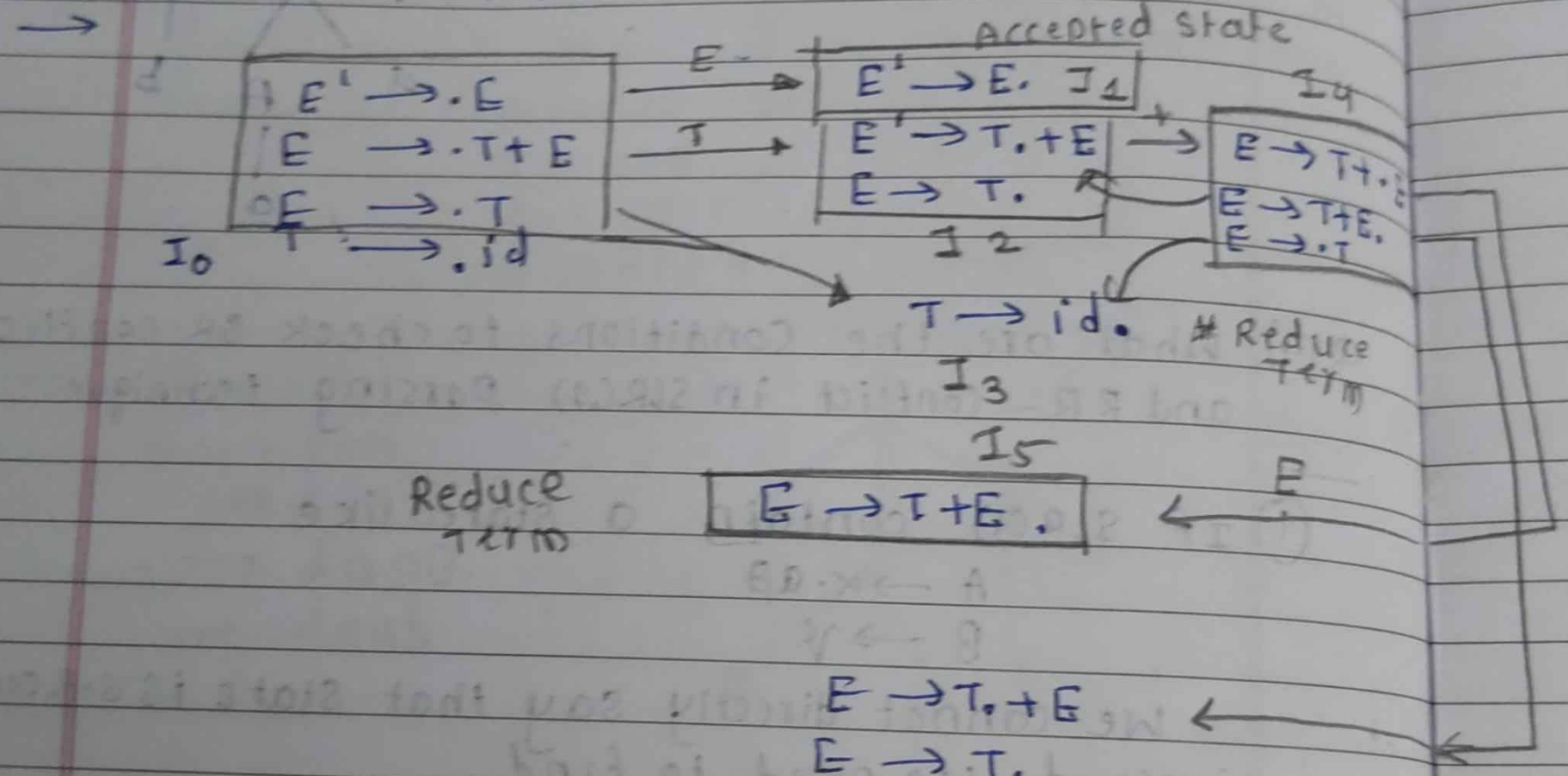
$B \to \gamma .$

We cannot directly say that state is RR-conflict instead we need to find

Follow(A) ∈ follow(B)
if follow(A) ∩ follow(B) ≠ {ϕ}
then we say that state consider RR conflict
and hence grammer is not in SLR(1)

Q15) check below grommer is in LR(0) or not

$E \to T+E$
$E \to T$
$T \to id$



State	id	+	$	E	T
0	S_3			1	2
1			Accepted		
2					
3					
4					
5					

GoodLuck | Page No.
Date

Q16 Write algorithm for e LR-Parsing Technique.

→

Algorithm for LR(0)

Input: An input string 'w' and LR-Parsing table m with ACTION and GOTO function and a grammer G.

Output: If 'w' is in L(G) the Reduction steps of bottom-up parse for w. otherwise an error indicat' method:

initially the parser has 0 on its stack where '0' is intial state and w$ in the input buffer

```
let a be the first symbol of w$;
while(1) { /* repeat forever */
   let s be the state on top of stack;
 if (ACTION[s,a] = shift t) {
        Push 't' onto the stack;
   let a be the next input symbol;
} else if (ACTION[s,a] = reduce A → β) {
    POP |β| symbol of the stack;
  let State 't' now be on top of stack;
   Push GOTO[t,A] onto the stack
} else if (ACTION[s,a] = Accept) break;
    else call error-recovery technique;
}
```