Total No. of Pages 1
VI-SEMESTER
MID SEMESTER EXAMINATION
Roll No.....
B.Tech (CO)
March- 2023
Time: 1:30 Hours CO-302 Compiler Design Max. Marks: 20
Note: Attempt all questions, assume suitable missing data if any

Total No. of Pages 1
VI-SEMESTER
MID SEMESTER EXAMINATION
Roll No.....
B.Tech.(CO)
March- 2020
Time: 1:30 Hours CO-302 Compiler Design Max. Marks: 30
Note: Attempt all questions

Total No. of Pages 1
VI-SEMESTER
MID SEMESTER EXAMINATION
Roll No.....
B.Tech.(CO)
March- 2019
Time: 1:30 Hours CO-302 Compiler Design Max. Marks: 30
Note:Attempt all questions

Total No. of Pages 01
Roll No. .....
VI- SEMESTER
MID SEMESTER EXAMINATION
B.Tech [CO]
MARCH-2018
CO-302: Compiler Design
Time: 1:30 Hours Max. Marks: 30
Note : 1) Attempt all questions.
2) All parts of a question must be attempted together.
3) Assume suitable missing data, if any.

Total No. of Pages 02
Roll No.....
VI-SEMESTER
B.Tech.(CO)
END SEMESTER EXAMINATION
May- 2023
CO302 Compiler Design
Time: 3:00 Hours Max. Marks: 40
Note: Attempt any five questions

Total Number of Pages 1
Roll No. .....
SEVENTH SEMESTER
B.E.(COE)
MID SEMESTER EXAMINATION
SEPTEMBER-2010
COE- 401 COMPLIER & TRANSLATION DESIGN
Time: 1 Hour 30 Minutes Max. Marks : 20
Note : Answer ALL questions.
Assume suitable missing data, if any.

Total No. of Pages 2
Roll No.....
VI-SEMESTER
END SEMESTER EXAMINATION
B.Tech.(CO)
May- 2019
CO-302 Compiler Design
Time: 3:00 Hours Max. Marks: 40
Note: Attempt any five questions

Total No. of Pages 2
Roll No.....
VI-SEMESTER
END SEMESTER EXAMINATION
B.Tech.(CO)
May- 2018
Time: 3:00 Hours CO-302 Compiler Design Max. Marks: 40
Note: Attempt any five questions

UNIT- 1 : INTRODUCTION

B. How is boot strapping of a compiler is done to a second machine? And also explain the function of each phase of compiler with suitable example. 2023 E / [4] [CO#1]

Page No.	
Date	

B. Explain different phases of compiler with suitable example. 2019M [3]

Explain different phases of compiler for the source code Position =initial+ rate *45 2020M [4]

The compilation process is divided into two parts :

Q2. a) Describe analysis and synthesis phases of Complier? 2018E [7]

① ANALYSIS PHASE (front end)

It is a machine indefendant and language dependent phase which breaks up the source program into constituent pieces and imposes grammatical structure to them.

② SYNTHESIS PHASE (back end)

It is a machine dependant and language indefendant phase which constructs the desired target program from the intermediate representation and information in symbol table

Q4.a) Describe machine dependent and independent phases of Compiler? 2017E [7]

Ⓐ M/C dependant

LEXICAL ANALYZER

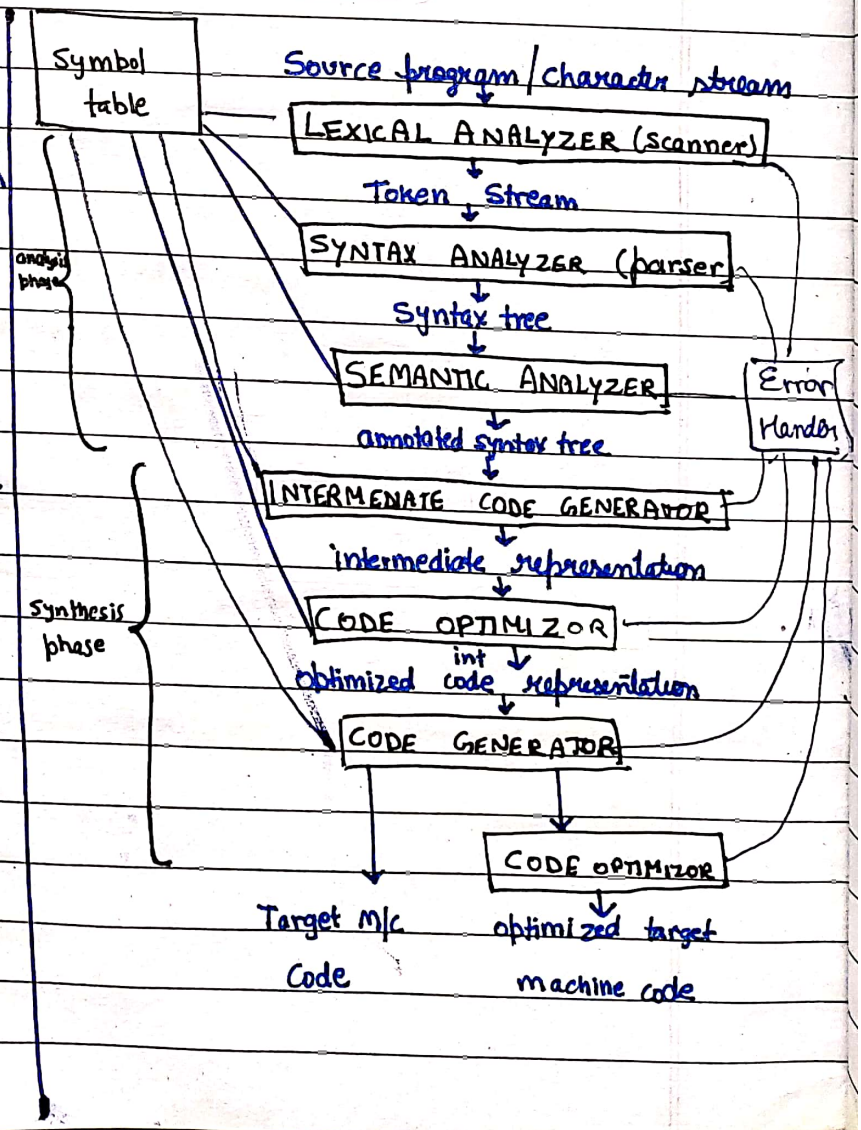
It is scanner which reads the prog. and converts it into lexemes, a stream of lexemes into a stream of tokens. Tokens are defined by regular exp which are understood by lexical analyzer

It also removes white space, comments.

SYNTAX ANALYZER

It is parser which constructs parse tree. It uses productions of CFG to construct parse tree.

It reads all tokens one by one.



Page No.
Date

SEMANTIC ANALYZER

It verifies whether the parse tree is meaningful or not.
It produces a verified / annotated parse tree.
(B) M/c Independant

INTERMEDIATE - CODE GENERATOR

It generates intermediate code, that is a form which can be readily understood by machine.
Eg.) Three address code etc.

CODE OPTIMIZER

It transforms the code so that it consumes fewer resources and produces more speed. (The code is transformed, not altered)

It can be : i.) M/c dependant
ii.) M/c independant

TARGET CODE GENERATOR

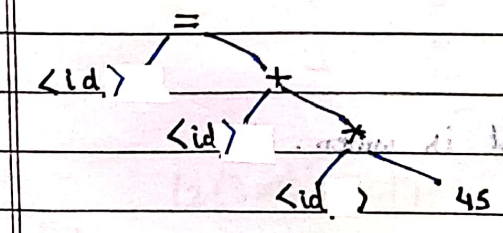
It writes a code that the m/c can understand and also register allocation, instruction selection etc.
The o/p is dependant on the type of assembler.

Eg.) Position = initial + rate * 45

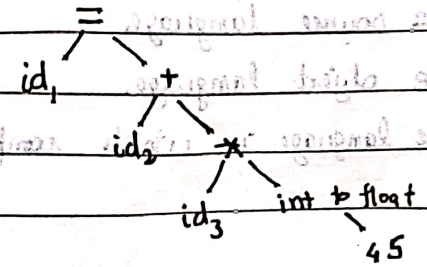
i) Lexical Analyzer

<id, 1> <assign op> <id, 2> <add op> <id, 3> <mult op> <int const 45>

ii) Syntax Analyzer



iii) Semantic Analyzer



iv.) Intermediate code generator

t_1 = int to float (45)

$t_2 = id3 * t_1$

$t_3 = id_3 * t_2$

$id_1 = t_3$

v.) Code optimizer

$t_1 = id_3 * 45.0$

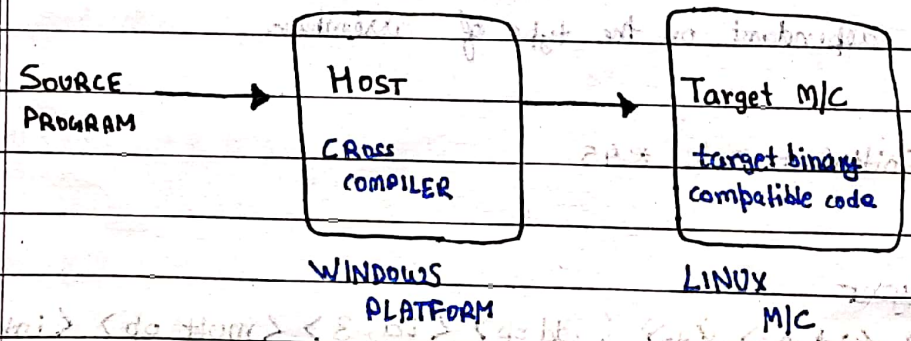
$id_1 = id_2 + t_1$

vi.) Code Generator

```
LDF   R2, id3
MULF  R2, R2, #60.0
LDF   R1, id2
ADDF  R1, R1, R2
STF   id1, R1
```

B. What is cross compiler? 2020M

A cross compiler is a compiler that runs on one machine and produces object code for another machine.

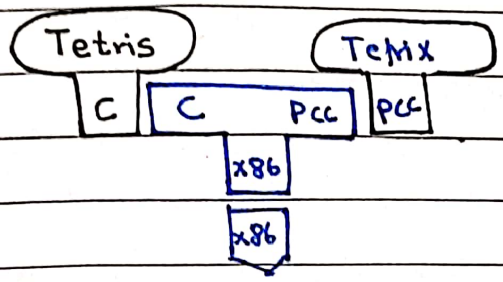


The cross-compiler is used to implement the compiler, which is categorized by three languages:

1. The source language
2. The object language
3. The language in which compiler itself is written.

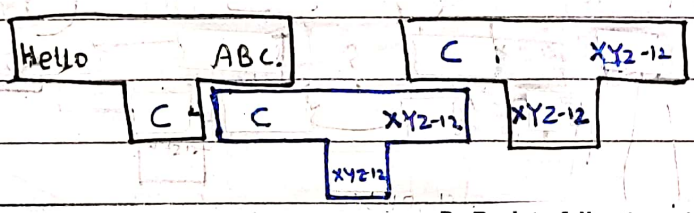
Page No.
Date

Eg)



Q.No. 1

What is cross compiler? Create a cross compiler for ABC using C compiler, written in XYZ-12, producing code in XYZ-12 and a ABC language producing code for text formatter, HELLO written in C. 2023M [4][CO#1]



[b] Boot strapping. 2010M

Bootstrapping is an arrangement which refers to writing a compiler in the same language it intends to compile. It is the process of using a compiler written in a particular lang to compile a new version of the compiler written in same lang.

$$L\,C^{A}_{S} \longrightarrow S\,C^{A}_{A} \rightarrow L\,C^{A}_{A}$$

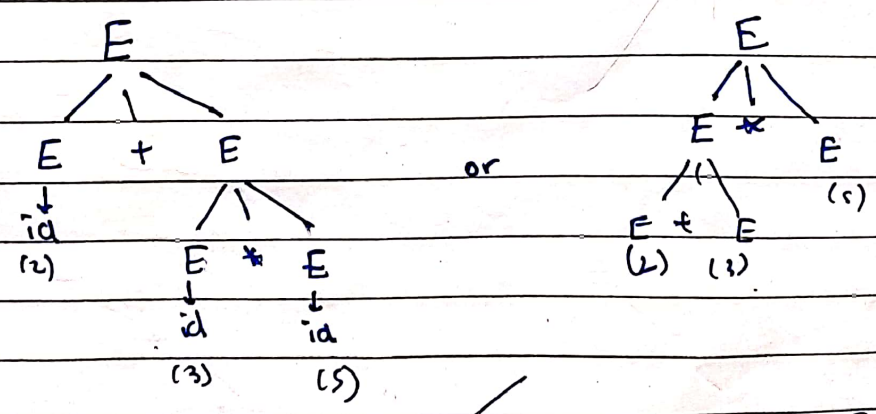
B. Explain following with suitable examples 2019E
a. Ambiguous grammar and cross compiler

B. Explain following with suitable examples
a. Ambiguous grammar and cross compiler 2019S

AMBIGUOUS GRAMMAR

An ambiguous grammar is a context free grammar for which there exists a string that can have more than one leftmost derivation or parse tree. or more than one rightmost derivation or parse tree.

Eg.) w = id + id * id



(2+(3*5)) = 17 ✓

(2+3)*5 = 25 ✗

B. Design a context free grammar(CFG) for the language L= $\{x^c y^c z^d u^d$ for c>=1,d>=1$\}$ 2019M [5]

Page No. | Date

$G = (N, T, P, S)$

$N = \{S, A, B\}$

$T = \{x y z v\}$

$S = \{S\}$

P:

$$\begin{cases} S \rightarrow AB \\ A \rightarrow xAy \mid xy \\ B \rightarrow zBv \mid zv \end{cases}$$

refer. TOC

[c] Regular Expression

B. Design a DFA with input alphabet {a,b} for the language [2x2=4]

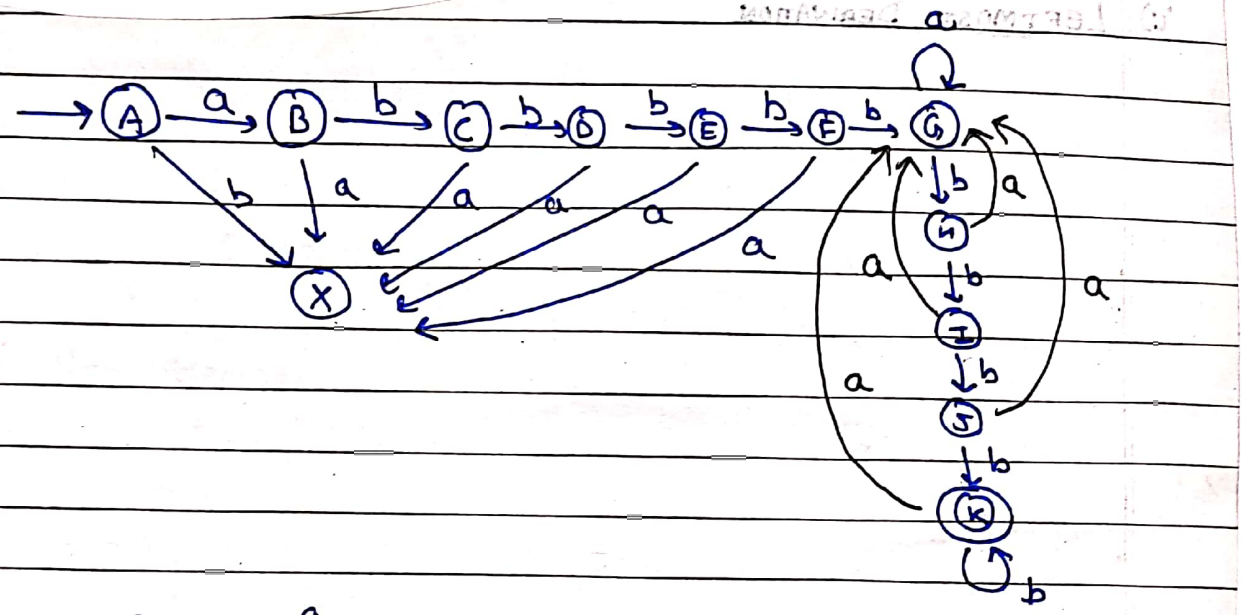
2018E

i) $L = \{w \in (a,b)^* : n_b(w) \bmod 3 > 1\}$
Where $n_b(w)$ is number of b's in w

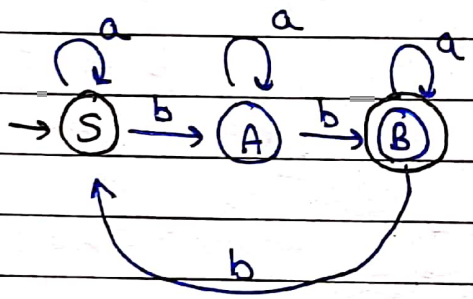
ii) $L = \{ab^5 w b^4 : w \in (a,b)^*\}$

Page No.
Date

ii)

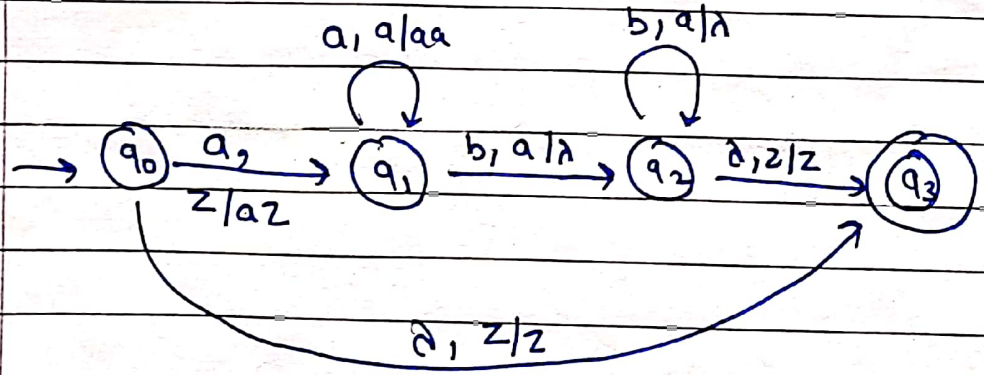


i)



b. Design a PDA accepting language $L = \{a^n b^n \mid n \geq 0\}$ 2018M (4)

$L = \{ a^n b^n \mid n \geq 0 \}$



Page No. | Date

Q.No. 1

2018 S

A. Construct DFA accepting strings of binary digits which are divisible by 3. [3]

let DFA be $M = \{Q, \Sigma, F, S, \delta\}$

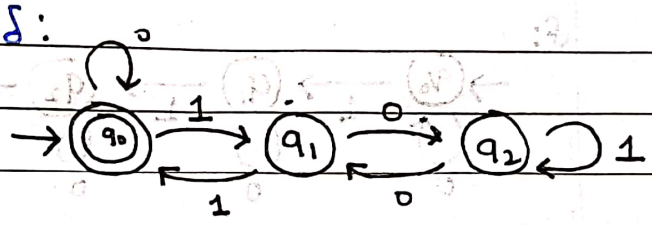
$Q : \{q_0, q_1, q_2\}$

$\Sigma : \{0,1\}$

$F : \{q_0\}$

$S : \{q_0\}$

δ:



b) Design a DFA (with input alphabet {0,1}) that accepts [7]
i) The set of strings where number of 0's is multiple of 3.
ii) $L = \{(0\ 1)^i 1^{2j} / i>=1, j>=1\}$ 2017 E

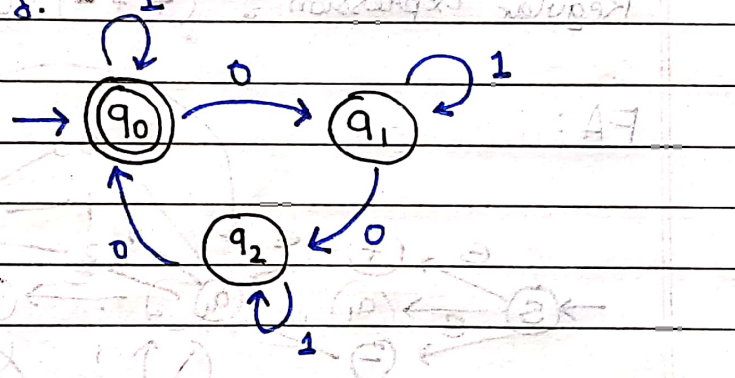
i) $Q : \{q_0, q_1, q_2\}$

$\Sigma : \{0,1\}$

$F : \{q_0\}$

$S : \{q_0\}$

δ:



ii) $L = \{ (01)^i 1^{2j} \mid i>=1, j>=1 \} = \{ 0111, 010111, 01011111 \ldots \}$

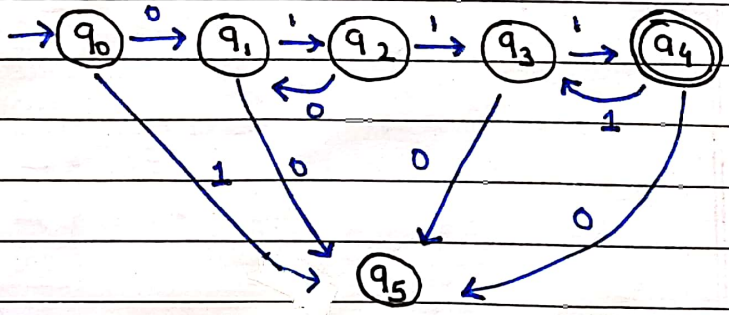
$Q : \{q_0, q_1, q_2, q_3, q_4, q_5\}$

$\Sigma : \{0,1\}$

$F : \{q_4\}$

$S : \{q_0\}$

δ:



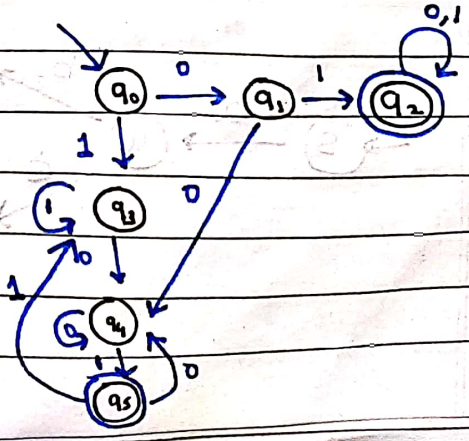
b) Design a DFA (with input alphabet {0,1}) that accepts 2018 E [7]
i) The set of strings which either starts with 01 or end with 01.
ii) The set of strings containing four 1's in every string.

i) $Q : \{q_0, q_1, q_2, q_3, q_4, q_5\}$

$\Sigma : \{0,1\}$

$F : \{q_2, q_5\}$

$S : \{q_0\}$



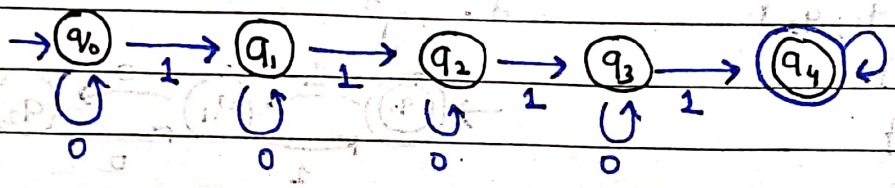
(ii) $Q: \{q_0, q_1, q_2, q_3, q_4\}$

$\Sigma: \{0, 1\}$

$q_0: q_0$

$q_f: \{q_4\}$

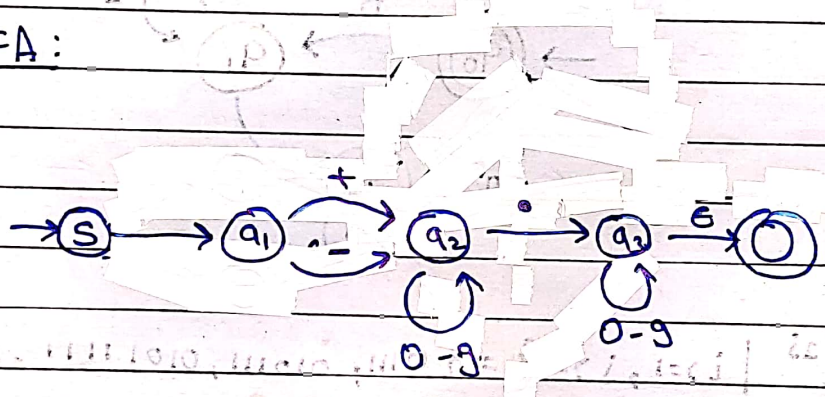
δ:



b. Write a regular expression for real number and draw FA for that regular expression? 2018M (3)

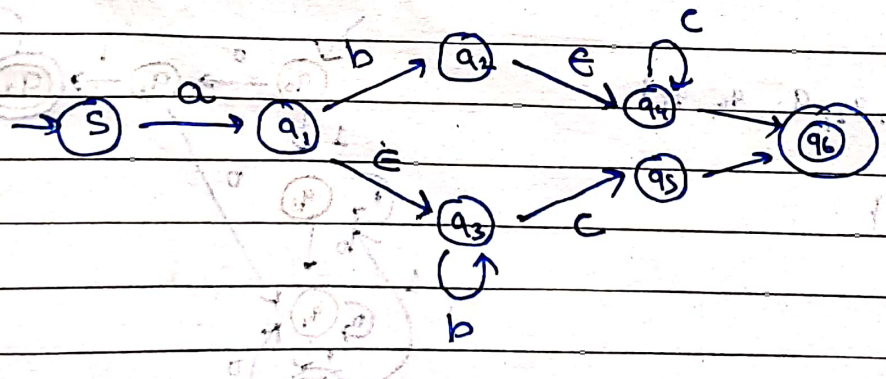
Regular Expression : $(+ + -)(0+1+2+3+\ldots+9)^*(.)(0+1+2+\ldots+9)^*$

FA:



1 Convert the following regular expression into minimal state equivalent DFA. 5

$(ab^*c)|(abc^*)$



Unit 2: Syntax Analysis

Page No.	
Date	

c. How a compiler differentiate between keywords and identifiers? 2018M (3)

In the lexical analysis phase, the compiler distinguishes keywords and identifiers by comparing scanned tokens with a list of reserved keywords for matches stored in a keyword table. While identifiers are managed using symbol table, checking for existing entries and adding new ones as needed.

Q.No. 2

a. Rectify the problems with the following grammers to make them suitable for LL(1) parsing: 2018M (4)

i) S → Sa/Sb/AB/C/DEF

ii) A → da/acB ; B → abB/daA/Af

i) we have to remove left recursion.

∴ S → ABS' / CS' / DEFS'

S' → aS' / bS' / ϵ

ii) Given

A → da / acB

B → abB / daA / Af

it can be written as.

A → da / acB

B → abB / dada / daacB / daf / acBf

now B exhibits left factoring.

A → da / acB

B → abB / Cda / CacB / Cf / acBf

C → da

Page No.
Date

2 A production of the form $A \rightarrow A\alpha$ is said to be left recursive. Similarly a production of the form $B \rightarrow \beta B$ is said to be right recursive. Show that any grammer that contains both left & right recursive productions with the same left hand side symbol must be ambiguous. 2012M 3

Let us consider the grammar

$S \rightarrow A|B$

$A \rightarrow Aa / B / \epsilon$

$B \rightarrow bB / A / \epsilon$

∴ we can derive "bbaa" as

1.) $A \rightarrow B \rightarrow bB \rightarrow bbB \rightarrow bbA \rightarrow bbaA \rightarrow bbaa$

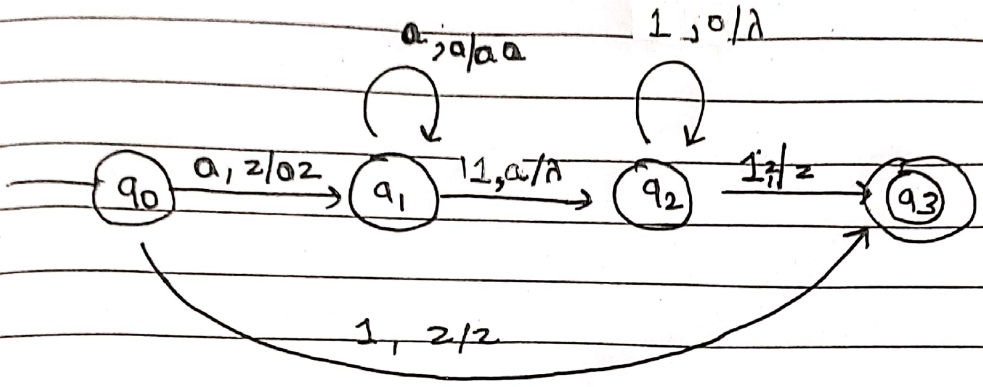
2.) $A \rightarrow Aa \rightarrow Aaa \rightarrow Baa \rightarrow Bbaa \rightarrow bbaa$

Hence, the grammar is ambiguous as more than one way to generate a given string exists.

Q.No. 1

a. Give an example of grammer to show that determinism cannot eliminate ambiguity? 2018M (2)

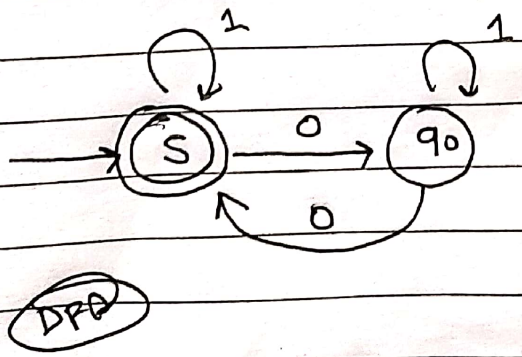
B. Explain importance of PDA in compiler design and Design a PDA for Language L={ $0^n 1^n 1$ for n>=0}. 2018S [5]



Q.No, 5

A. What is the use of FA in lexical analysis? Design a DFA for strings over {0, 1} having an even number of 0's and any no. of 1's. 2018S [3]

Finite automata in lexical analysis represents the language's structure using regular expressions, facilitating efficient tokenization of input strings into tokens by recognizing patterns and transitioning between states based on i/p symbols



3 Consider the following CFG:

$S \rightarrow a|\wedge|(T)$

$T \rightarrow T,s|s$

2010 M

[a] Compute the operator precedence relations for this grammer. 3

[b] Eliminate left recursion from the grammer. 1

[c] Show the steps of a Top-down passer without back tracking i.e. predictive passer for the string (((a,a), ∧, (a)), a) 4

ii) Expain common prefix problem with example?

It is also called left factoring, it is a problem as it is not clear which production to choose for NT because multiple productions begin with the same T/NT/lookahead.

It gives problem ~~in~~ backtracking

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

it is removed by rewriting prodn rules as

$$A \to \alpha A'$$
$$A' \to \beta_1 / \beta_2$$

Eg.) $S \to ictS$
$S \to ictSeS$
$S \to a$
$E \to d$

after Elemination

$S \to ictSS'$
$S' \to eS/\epsilon$
$S \to a$
$C \to d$

Q1. 2017E [14]

a) (0* / 1)* & (0 /1*)* are equivalent or not?
b) G= {N, { (,) } , { S-> SS, S-> (S), S-> e },S} is a CFG to produce_____?
c) Recursive descent parser is a predictive or non-predictive parser?
d) All CFL's are regular languages or not?
e) YACC is a tool for _____?
f) _____ is a solution for common prefix problem?
g) Number of states in SLR and CLR are same or not?
h) _____ is used to identify common sub expressions?
i) Type conversion automatically done by complier is called _____?
j) Folding is the technique for _____?
k) Syntax tree is the form of _____?
l) A group of statements with single entry and single exit is called _____? (with respect to code generation)
m) Two adjacent transposed characters is an example of _____ error?
n) _____ is a hashing technique in which there is no limit on number of entries that can be made to the symbol table.

a:) yes
b:) 1
c:) predictive parser
d:) No
e:) generating parser
f:) Trie
g:) No
h:) global optimization tech.
i:) implicit TC
j:) optimizing exp
k:) parse tree
l:) basic block
m:) typographical
n:) Linear Probing

B What is the role of context free grammar (CFG) in compiler design? Construct CFG for Language L={0^a 1^b | a≠b} and Eliminate Left recursion from following grammar 2023E [4] [CO#1]

B. Eliminate Left recursion from following grammar 2019E [2]

S→A
A→ Ad|Ae|aB|aC
B→bBC| f
C→g

S→A
A→ Ad|Ae|aB|aC
B→bBC| f
C→g

[5] 2020M

recursion from following grammar
S→A
A→Ad | Ae| aB| aC
B→bBC | f
C→g

Its left recursive as it contains

$A \to Ad \mid Ae \mid aB \mid aC$

we replace these with

$A \to aBD \mid aCD$

$D \to dD \mid eD \mid \epsilon$

∴ new resultant grammar.

$S \to A$

$A \to aBD \mid aCD$

$D \to dD \mid eD \mid \epsilon$

$B \to bBC \mid f$

$C \to g$

Construct a CFG for language L={ $0^a 1^b$ | a≠b} and eliminate Left

$P :=$

$S \to AT \mid TB$

$T \to 0T1 \mid \epsilon$

$A \to 0A \mid 0$

$B \to 1B \mid 1$

$G = \{N, T, P, S\}$

$S = \{S\}$

$N = \{S, T, A, B\}$

$T = \{0, 1\}$

remove left recursion from following CFG : 2023M [4][CO#2]

S → Bb | a
B → Be | Sd | e

It is left left recursive as it contains indirect and direct LR

① remove indirect calls.

$S \to Bb \mid a$

$B \to Be \mid Bbd \mid ad \mid e$

② remove B→Be and B→Be

$S \to Bb \mid a$

$B \to eB' \mid adB'$

$B' \to eB' \mid bdB' \mid \epsilon$

B. Eliminate Left recursion from following grammar [2]

2018E

S→aBDh
B→Bb | c
D→EF
E→g | ε
F→f | ε

Page No.
Date

B. Eliminate Left recursion from following grammar [3]

2019S

S→aBDh
B→ Bb|c
D→ EF; E→ g | ϵ ; F→f | ϵ

The grammar is left-recursive because it contains a pair of productions $B \rightarrow Bb \mid c$. To eleminate this, replace this pair with

$B \rightarrow cC$

$C \rightarrow bC \mid \epsilon$

∴ New grammar :

$S \rightarrow aBDh$

$B \rightarrow cC$

$C \rightarrow bC \mid \epsilon$

$D \rightarrow EF$

$E \rightarrow g \mid \epsilon$

$F \rightarrow f \mid \epsilon$

B. What is the role of Push down automata (PDA) in syntax analysis?
Eliminate Left recursion from following grammar 2019M [5]

S→aBDh
B→Bb | c
D→EF
E→g | ϵ
F→f | ϵ

B. What is the role of Context free grammar (CFG) in Compiler Design?

PDAs with stacks are used to implement parsing algorithms for context free grammars enabling recognition CFLs and generation of parse trees,

So, CFGs define the syntax of programming languages and guide various stages of compilation, particularly syntax analysis, by specifying how valid programs are structured and facilitating the construction of parsers to analyse program syntax

Q.No. 3

a. For the string id+id*id+id and the grammer (4)

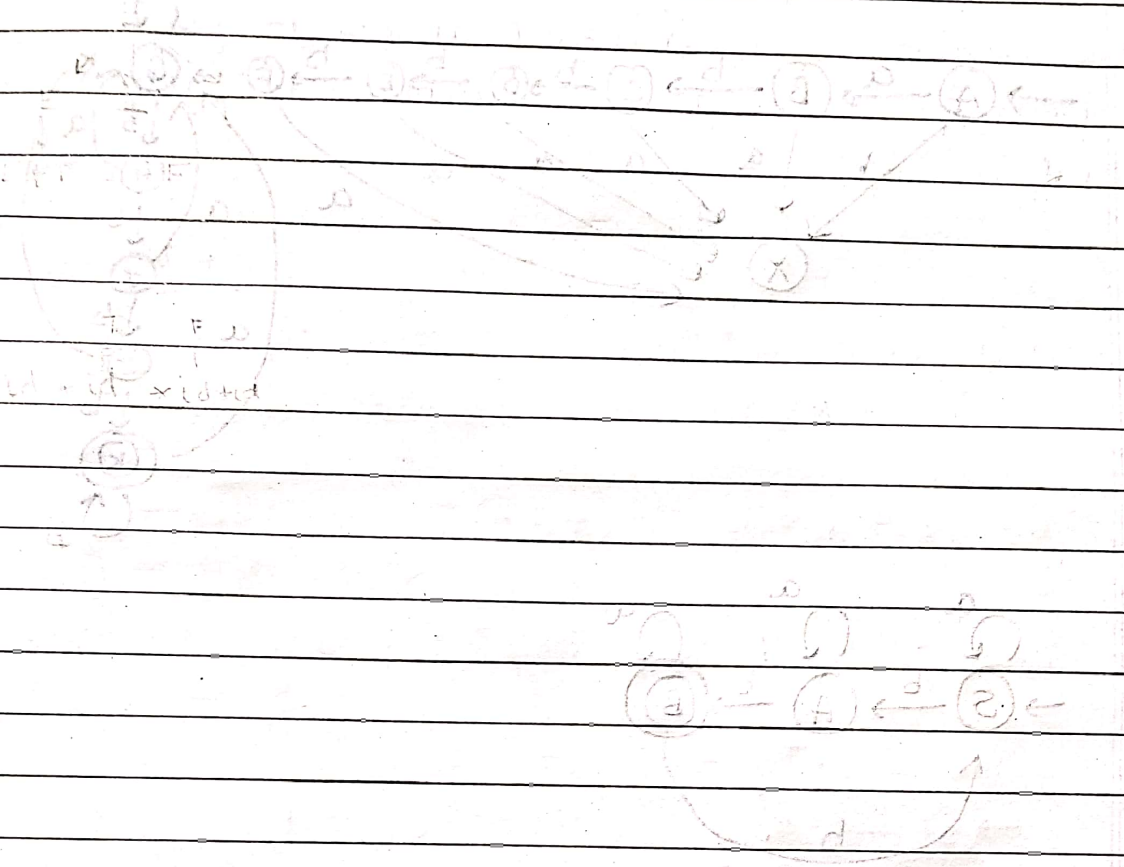
E →E+T/T T→T*F/F F→id 2018M

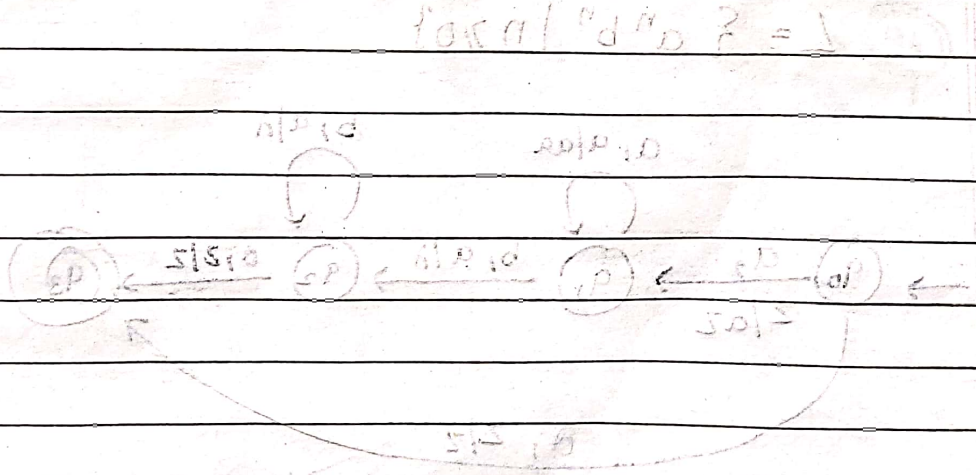
Find i) leftmost derivation ii)rightmost derivation iii) parse tree

iv) Is the grammer ambiguous ?

Page No.
Date

i) LEFTMOST DERIVATION





Page No.
Date

Q.No. 2
Explain functions of various phases of compiler design and Compute FIRST & FOLLOW for the following grammar: 2023M [4][CO#1]

S'→ S -①
S→ fAh | gBh | fBk | gAk -②
A→ d -③
B→ d -④ (S' is the start symbol of grammar)

FIRST

① S' → S

FIRST (S') = First (S') if Fir(S) is not having ϵ
= {f, g} - Ⓐ using Ⓑ

② S → fAh | gBh | fBk | gAk

FIRST (S) = {f, g} - Ⓑ as X → ad and First (X) = a

③ A → d

FIRST (A) = d

④ B → d

First (B) = d

FOLLOW

① for S'

Follow (S') = {$}

② for S

Follow(S) = {follow (S')} = {$}

③ for A

follow (A) = {h, k}

④ for B

Follow (B) = {h, k}

	FIRST	FOLLOW
S → S'	{f, g}	{$}
S → fAh/gBh/fBk/gAk	{f, g}	{$}
A → d	{d}	{h, k}
B → d	{d}	{h, k}

B. Compute FIRST and FOLLOW sets and Construct a predictive parsing table for the following grammar,where S is the start symbol. [5]

2018 S

S→aBDh
B→cC | ϵ
C→bC | ϵ
D→EF
E→g | ϵ
F→ f | ϵ

(where 'ϵ' denotes epsilon)

Q.No. 1

A. Compute FIRST & FOLLOW and Construct a predictive parsing table for the following grammar,where S is the start symbol. 2020M [6]

S→aBDh
B→Bb | c
D→EF
E→g | ϵ
F→f | ϵ

① Removal of left recursion in $B \to Bb|c$

∴ Grammar : $S \to aBD$

$B \to cC$

$C \to bC|\epsilon$

$D \to EF$

$E \to g|\epsilon$

$F \to f|\epsilon$

Q.No. 2

A. How top down parsing is different from bottom up parsing? Design predictive parsing table for the following grammar: [4] [CO# 2]

2023E

S → aBDh
B → Bb|c
D → EF
E → g /c
F → f/c

(where 'S' is start symbol)

② FIRST

FIRST (S) = {a}

FIRST (B) = {c}

FIRST (C) = {b, ϵ}

FIRST (D) = {FIRST (E) − {} ∪ {FIRST (F)}
= {g, f, ϵ}

FIRST (E) = {g, ϵ}

FIRST (F) = {f, ϵ}

FOLLOW

Follow (S) = {$}

Follow (B) = {FIRST (D) − ϵ} ∪ First(h)
= {g, f, h}

Follow (C) = Follow (B) = {g, f, h}

Follow (D) = follow (h) = {h}

Follow (E) = {FIRST (F) − ϵ} ∪ follow (D)
= {f, h}

Follow (F) = Follow (D) = {h}

③ PREDICTIVE PARSING TABLE

	a	b	c	g	f	h	$
S	S→aBDh	err	err	err	err	err	err
B	err	err	B→cC	err	err	err	err
C	err	C→bC	err	C→ϵ	C→ϵ	C→ϵ	err
D	err	err	err	D→EF	D→EF	D→EF	err
E	err	err	err	E→g	E→g	E→g	err
F	err	err	err	err	F→f	F→f	err

Q.No. 1

A. Compute FIRST and FOLLOW sets and Construct a predictive parsing table for the following grammar, where S is the start symbol. [7]

S→iEtS | a
S→ iEtSeS
E→ b

2019M

Page No.
Date

(1) Grammar is unambiguous but has left factoring

$S \to iEtS \mid a$
$S \to iEtSeS$
$E \to b$

$\Rightarrow$

$S \to iEtSS' \mid a$
$S' \to eS \mid \epsilon$
$E \to b$

(2)

FIRST

First (S) = {i, a}
First (S') = {e, ϵ}
First (E) = {b}

FOLLOW

Follow (S) = {First (S)} = {i, a}
Follow (S') = {i, a}
Follow (E) = {$}

(3) Predictive Parsing Table

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

Q.No. 1

A. Compute FIRST, FOLLOW sets, and Construct a predictive parsing table for the following grammar, where S is the start symbol. [5]

S→iEtSS$_1$ | a
S$_1$→eS | ϵ
E→ b

(where 'ϵ' denotes epsilon)

2019S

Q.No. 2

A. Compute FIRST and FOLLOW sets for the following grammar

D→ T L;
L→ id M;
M→ id M /ϵ
T→ int / float

(where 'ϵ' denotes epsilon) [4]

2018E

FIRST

First (T) = {int, float}
First (D) = First (T) = {int, float}
First (L) = {id}
First (M) = {id, ϵ}

FOLLOW

Follow (T) = First (L) = {id}
Follow (D) = {$}
Follow (L) = {Follow (D) = {$}}
Follow (M) = follow (L) = {$}

Page No.
Date

Q.No. 4
What is shift reduce parsing? Design predictive parsing table for the following grammar: 2023M [4][CO#2]

A → dBcAD | f
D → tA | ε
B → g

(where 'A' is the start symbol.)

Shift reduce parsing is a bottom up parsing method which uses reverse of rightmost derivation to reduce the string into start symbol using
→ input buffer for input string
→ Stack for storing and accessing handle.

Its basic actions are:
- Shift
- Reduce
- Accept
- Error.

① Grammar: (unambiguous)
A → dBcAD | f
D → tA | ε
B → g.

② FIRST
First (A) = {d, f}
First (D) = {t, ε}
First (B) = {g}

FOLLOW
Follow (A) = {First (D) - {}} ∪ {Follow A}
= {t, $}
Follow (D) = {Follow (A)} = {t, $}
Follow (B) = First (c) = {c}

③ TABLE

	c	d	f	g	t	$
A		A → dBcAD	A → f			
D					D → ε ? D → t :	D → ε
B				B → g		

Page No.
Date

Q.No. 2

A. Design predictive parsing table for the following grammar and differentiate between top down and bottom up parsing. [4

S → X
X → aY|Xd
Y → bYZ /f
Z → g (where 'S' is start symbol)

2019 E

TOP-DOWN	BOTTOM-UP
Parse tree can be built from root to leaves	Parse tree can be built from leaves to root.
Simple to implement and less power	Complex to impl. and more powerful.
Less efficient parsing techniques due to ambiguity and left recursion	More efficient parsing technique, efficiently handles ambiguity in parse tree
Applicable to small class of lang.	Applicable to broader class of lang
Eg.) Recursive Descent Parser Predictive Parser	Eg.) Shift reduce parser Operator precedence Parser LR parser

(1) Grammar is ambiguous, X → Xd ⟹ S → X

X → aY X'
X' → d X' | ϵ
Y → b Y Z | f
Z → g

(2)

FIRST	Follow
First(S) = {a} = First(X)	Follow(S) = {$}
First (X') = {d, ϵ}	Follow (X) = {$}
First (Y) = {b, f}	Follow (X') = {$}
First (Z) = {g}	Follow (Y) = {g}
	Follow (Z) = {g}

(3) Table

	a	b	d	f	g	$
S	S→X	—	—	—	—	—
X	X→aYX'	—	—	—	—	—
X'	—	—	X'→dX'	—	—	X'→ϵ
Y	—	Y→bYZ	—	Y→f	—	—
Z	—	—	—	—	Z→g	—

3

Page No.
Date

B. Design predictive parsing table for the following grammar and differentiate between top down and bottom up parsing. [4]

S → aAcD|BCe
A → b| c
B → Cf | d
C → fe (where 'S' is start symbol)

2019S

(1) Grammar is LL(1)

(2)

FIRST	FOLLOW
First (S) = {a, f, d}	Follow (S) = {$}
First (A) = {b, ε}	Follow (A) = {c}
First (B) = {f, d}	Follow (B) = {f}
First (C) = {f}	Follow (C) = {e, f}

(3) Predictive parsing TABLE

	a	b	c	d	e	f	$
S	S→aACD			S→BCE		S→BCE	
A		A→b	A→ε				
B				B→d		B→Cf	
C						C→fe	

B. Construct parsing table for the following grammar, where S is the start symbol [5]

S→AaAb
S→BbBa
A→ ϵ
B→ ϵ

2018S

① Grammar is LL(1) ✓

② First

First (S) = First (Aa Ab) ∪ First (Bb Ba)

First (S) = {a} ∪ {b} = {a, b}

First (A) = {ϵ}

First (B) = {ϵ}

Follow

Follow (S) = {$}

Follow (A) = First (aAb) = {a} ∪ {b} = {a, b}
∪ First (b)

Follow (B) = First (bBa) ∪ First (a) = {b, a}

③ Predictive parsing table

	a	b	$
S	S → Aa Ab	S → Bb Ba	
A	A → ϵ	A → ϵ	
B	B → ϵ	B → ϵ	

B. Consider following grammar [5]

S→aAb| bB
A→ Aa | ϵ
B→ Bb| ϵ

and test whether the grammar is LL(1) or not?

2018S

test

① for S → aAb | bB

First (aAb) ∩ First (bB) = ϕ ✓

② for A → Aa | ϵ

First (Aa) ∩ First (ϵ) = ϕ ✓

and First (Aa) ∩ Follow (A) = {a, ϵ} ∩ {a, b} = {a} ✗

Not LL(1)

Q.No. 2

A. Consider following grammar and test whether the grammar is LL(1) or not ? [4]

2019 S

S→1AB| ϵ
A→ 1AC|0C
B→ 0S
C→ 1

Page No.
Date

Q.No. 3 .

A. Consider following grammar and test whether the grammar is LL(1) or not ? [5]

2019 M

S→1AB| ϵ
A→ 1AC|0C
B→ 0S
C→ 1

$A \rightarrow \alpha | \beta$

for a grammar to be LL(1) ;

for every pair of productions :

{

FIRST (α) ∩ FIRST (β) = ϕ

and

if FIRST (B) contains ϵ, and FIRST (α) dont contain ϵ

then

FIRST (α) ∩ FOLLOW (A) = ϕ

for pair of productions $S \rightarrow 1AB | \epsilon$

FIRST (1AB) ∩ FIRST (ϵ) = $\{1\} \cap \{\epsilon\} = \phi$, and

FIRST (1AB) ∩ FOLLOW (S) = $\{1\} \cap \{\$\} = \phi$

for pair of prodn $A \rightarrow 1AC | 0C$:

FIRST (1AC) ∩ FIRST (0C) = $\{1\} \cap \{0\} = \phi$

Hence , its LL(1) grammar .

[e] LL(1) grammer. 2010 M

A context free grammar $G = (V_T, N, P, S)$ whose parsing table has no multiple entries is said to be LL(1). here in LL(1),

→ first L stands for scanning the input from left to right.

→ second L stands for producing a leftmost derivation.

→ 1 stands for using 1 symbol of lookahead at each step to make parsing action decision.

Page No.
Date

8. Explain recursive descent parsing and shift reduce parsing with suitable examples. 2020M [5]

b) Explain recursive decent parser? 2018E

[d] Recursive descent passing. 2010M

A recursive descent parser is a top-down parser built from a set of mutually recursive procedures where each such procedure implements one of the non-terminals of the grammar. Execution starts with procedure corresponding to start symbol.

Eg) $A \rightarrow X_1 X_2 X_3 \cdots X_n$

$\therefore X_i \rightarrow ab$

```
Void A()
{
    take A → X1 X2 X3 --- Xn
    for (i=0 to n)
    {
        if Xi is a non terminal
            then call procedure Xi();
        else if Xi is a terminal a
            then enhance ptr to next input.
        else
            error
    }
```

SHIFT REDUCE PARSING

It is a simple bottom up parsing technique which uses:

→ an input buffer for storing the input string

→ a stack for storing and accessing the prodn rules.

It has basic operations:

(1) Shift: Moves symbols from input buffer to stack

(2) Reduce: finds handle and replace with NT

(3) Accept: Pair ($S, $)

(4) Error: Can't reduce nor shift

Page No.
Date

Eg.) $E \to E + E$
$E \to E * E$
$E \to (E)$
$E \to id_1 \mid id_2 \mid id_3$
String, $\omega = id_1 + id_2 * id_3$

$E \to E + E$
$\to E + E * E$
$\to E + E * id_3$
$\to E + id_2 * id_3$
$\to id_1 + id_2 * id_3$

STACK	INPUT	ACTION
$	$id_1 + id_2 * id_3 \$$	Shift
$ id_1	$+ id_2 * id_3 \$$	Reduce $E \to id_1$
$ E	$+ id_2 * id_3 \$$	Shift
$ $E+$	$id_2 * id_3 \$$	Shift
$ $E+id_2$	$* id_3 \$$	Reduce $E \to id_2$
$ $E+E$	$* id_3 \$$	Shift
$ $E+E*$	$id_3 \$$	Shift
$ $E+E*id_3$	$	Reduce $E \to id_3$
$ $E+E*E$	$	Reduce $E \to E*E$
$ $E+E$	$	Reduce $E \to E+E$
$ E	$	Accept as $S

$\therefore$ successful parsing $\to$ HALT

4 Explain the following terms (any four):-
[a] Handle pruning 2010M

Handle pruning is the process of removing the children of the left-hand side non terminal from the parse tree, i.e. it is the replacement of handle by NT.

$S \to \alpha A \beta \to \alpha \gamma \beta$

↳ handle → replaced by NT in left side of prodn $A \to \gamma$

Page No.
Date

b. For the grammer 2018M (10)

S→aBDh B→cC C→bC/ϵ D→EF
E→g/ϵ F→f/ϵ

Check whether the string "acbbgfh" is parsable by LL(1) parser or not
(ϵ is epsilon . Show all steps involved in parsing)

At is parsable

First (S) = {a}	Follow (S) = {$}
First (B) = {c}	Follow (B) = {
First (C) = {b, ϵ}	Follow (C) = {Follow(Dh)} = {g, f, h}
First (D) = {g, f, ϵ}	Follow (D) = {h}
First (G) = {g, ϵ}	Follow (E) = {f, h}
First (F) = {f, ϵ}	Follow (F) = {h}

TABLE

	a	b	c	g	f	h	$
S	S→aBDh						
B			B→cC				
C		C→bC		C→ϵ	C→ϵ	C→ϵ	
D				D→EF	D→EF	D→EF	
E				E→g	E→ϵ	E→ϵ	
F					F→f	F→ϵ	

STACK	INPUT	MOVE
$ S	a c b b g f h	S→ a BDh
$ h D B ~~a~~	~~a~~ c b b g f h	POP
$ h D B	c b b g f h	B→ c C
$ h D C ~~c~~	~~c~~ b b g f h	POP
$ h D C	b b g f h	C→ b C
$ h D C C ~~b~~	~~b~~ b g f h	POP
$ h D C ~~b~~	~~b~~ g f h	C→ b C
$ h D C	g f h	C→ ϵ
$ h D	g f h	C→ ϵ
$ h F E	g f h	D→ EF
$ h F ~~g~~	~~g~~ f h	E→ g
$ h F	f h	POP
$ h ~~f~~	~~f~~ h	F→ f
$ h	h	POP

Accept

Q.No. 3

A. What are the problems associated with Top down parsing? Is the following grammar LL(1)? 2020M [5]

Problems with top down parsing are:-

1) 1.) Left Recursion: Occurs when a non-terminal directly or indirectly produces itself. It causes infinite loops in parsing.

Eg:) $A \rightarrow A\alpha / \beta$

can be reduced to

$A \rightarrow \beta A'$

$A' \rightarrow \alpha A' / \epsilon$

2.) Left Factoring: Technique to remove common prefixes in grammar rules, reducing ambiguity and aiding predictive parsing.

Eg:) $A \rightarrow \alpha \beta_1$

$A \rightarrow \alpha \beta_2$

rewrite as

$A \rightarrow \alpha A'$

$A' \rightarrow \beta_1 / \beta_2$

3.) Backtracking: Strategy used in parsing to explore multiple parsing paths. Can be inefficient, especially for ambiguous grammars.

Page No.
Date

Q.No. 5
A. What is an operator precedence parsing? Explain operator precedence parsing Algorithm. 2023E [4] [CO#2]

Operator precedence parser is an efficient shift reduce parser which can be constructed for both ambiguous and unambiguous grammars. It is done on operator precedence grammar.
(Operator grammar is one where no production of the grammar has null or two adjacent non-terminal in RHS of grammar).
These are useful for scientific applications

Parsing Algo

Set pointer i/p to first symbol of w$
while {
  if TOS is $ and i/p points to $
    return
  else {
    let terminal a is on TOS and i/p points to terminal
    if a<b or a=b then {
      push b on TOS
      advance i/p to next symbol
    }
    if a > b {
      while (top terminal of stack is related by < to
        most recently popped terminal) {
        pop
      }
    }
  }
}

Eg: Table for $E \to E+E$
$E \to E*E$
$E \to id$

θ_1 \ θ_2	+	*	id	$
+	>	<	<	>
*	>	>	<	>
id	>	>	-	>
$	<	<	<	-

Page No.
Date

for string, ω = id+id*id

STACK		input buffer	Action
$	<	id+id*id$	Shift id
$ id	>	+id*id$	pop
$	<	+id*id$	Shift +
$ +	<	id*id$	Shift id
$ +id	>	*id$	Pop
$ +	<	*id$	Shift *
$ + *	<	id$	Shift id
$ + * id	>	$	Pop
$ + *	>	$	Pop
$ +	>	$	Pop
$		$	Accept

Successful parsing → HALT.

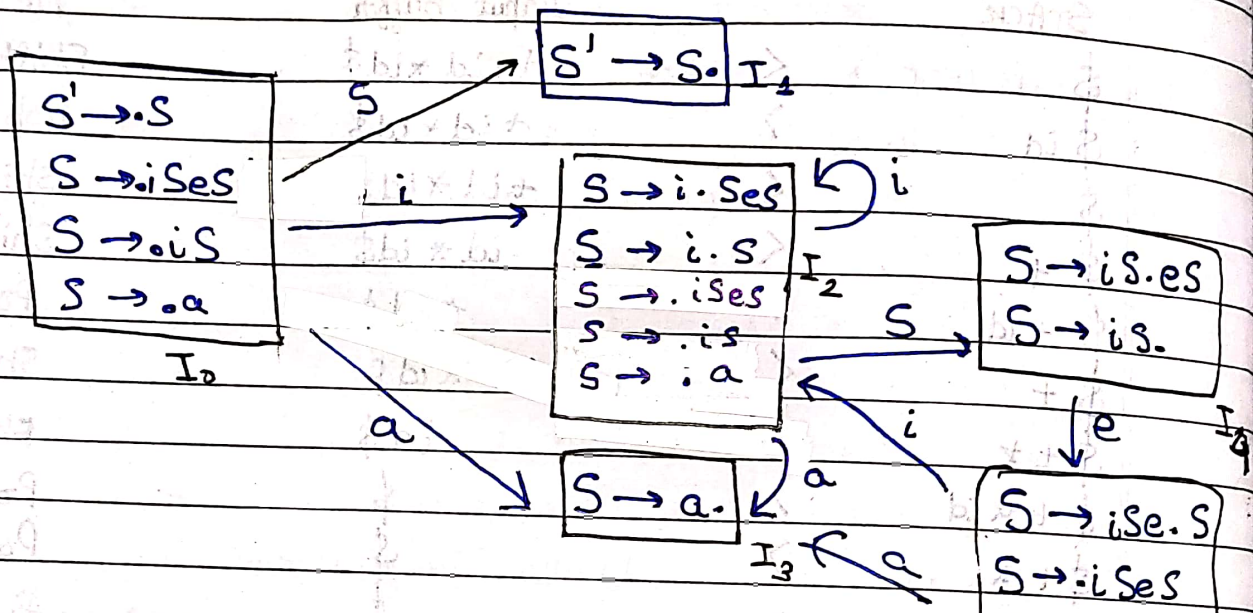
Page No.
Date

Q.No. 3

A. Define CLOSURE(I) and GOTO(I,X) functions and construct the sets of LR(0) items for the following grammar. [4][CO#3]

2023E

S' → S
S → iSeS|iS|a (where 'S' is start symbol)



CLOSURE (I_0) = {
S'→.S
S→.iSeS
S→.iS
S→.a
}

Closure (I_1) = {
S'→S.
}

Closure (I_2) = {
S→i.SeS
S→i.S
S→.iSeS
S→.iS
S→.a
}

Closure (I_3) = {
S→a.
}

Closure (I_4) = {
S→iS.eS
S→iS.
}

Closure (I_5) = {
S→iSe.S
S→.iSeS
S→.iS
S→.a
}

Closure (I_6) = {
S→iSeS.
}

GoTo (I_0, S) = I_1
GoTo (I_0, i) = I_2
GoTo (I_0, a) = I_3
GoTo (I_2, i) = I_2
GoTo (I_2, S) = I_4
GoTo (I_2, a) = I_3
GoTo (I_4, e) = I_5
GoTo (I_5, S) = I_6
GoTo (I_5, a) = I_3

Page No.
Date

Q3. a i) Explain type of conflicts that occur in LR parsing with example? [7.

The two types of conflicts in LR parsing are SR and RR conflict:

1.) **Shift Reduce Conflict:**
It is caused when grammar allows a production rule to be reduced in a state and in the same state another production rule is shifted for same token.

2.) **Reduce - Reduce Conflict:**
It is when two or more productions apply to the same sequence of inputs. This grammar becomes ambiguous because a program can be interpreted in more than one way.

Eg.) $A \to B.$ -③ $\xrightarrow{id}$ I_2

I_1

	id	+	*
I_1	S_2/r_3	r_3	r_3

$A \to B.$ ②
$B \to id.$ ③

I_1

	.	t_1	t_2	t_3
I_1		r_2/r_3	r_2/r_3	r_2/r_3

Page No.
Date

No. 3
Explain algorithm for constructing SLR parsing table and

Algorithm for construction of SLR parsing table.

① find canonical collection of set of items / LR(0) items
$C = \{ I_0, I_1, I_2 \text{ ---- } \}$ i.e. Set of states of DFA.

② Follow following rules for construction of Action and Goto table.

for ACTION TABLE

a.) If $A \to \alpha . a\beta$ is in I_i and $Goto(I_i, a) = I_j$
then set action $[I_i, a] = S_j$ (shift j)

b.) If $A \to \alpha .$ is in I_i then for every b in follow(A)
Set action $[I_i, b] = R_k$ or Reduce by $A \to \alpha$
↳(production no.)

c.) If $S' \to S$, where S': initial start system of aug. grammar
S: Start state of given grammar.
Set Action $[I_i, \$] =$ accept

for GOTO TABLE

d.) If $Goto(I_0, A) = I_j$ then
set $Goto[I_i, A] = j$.

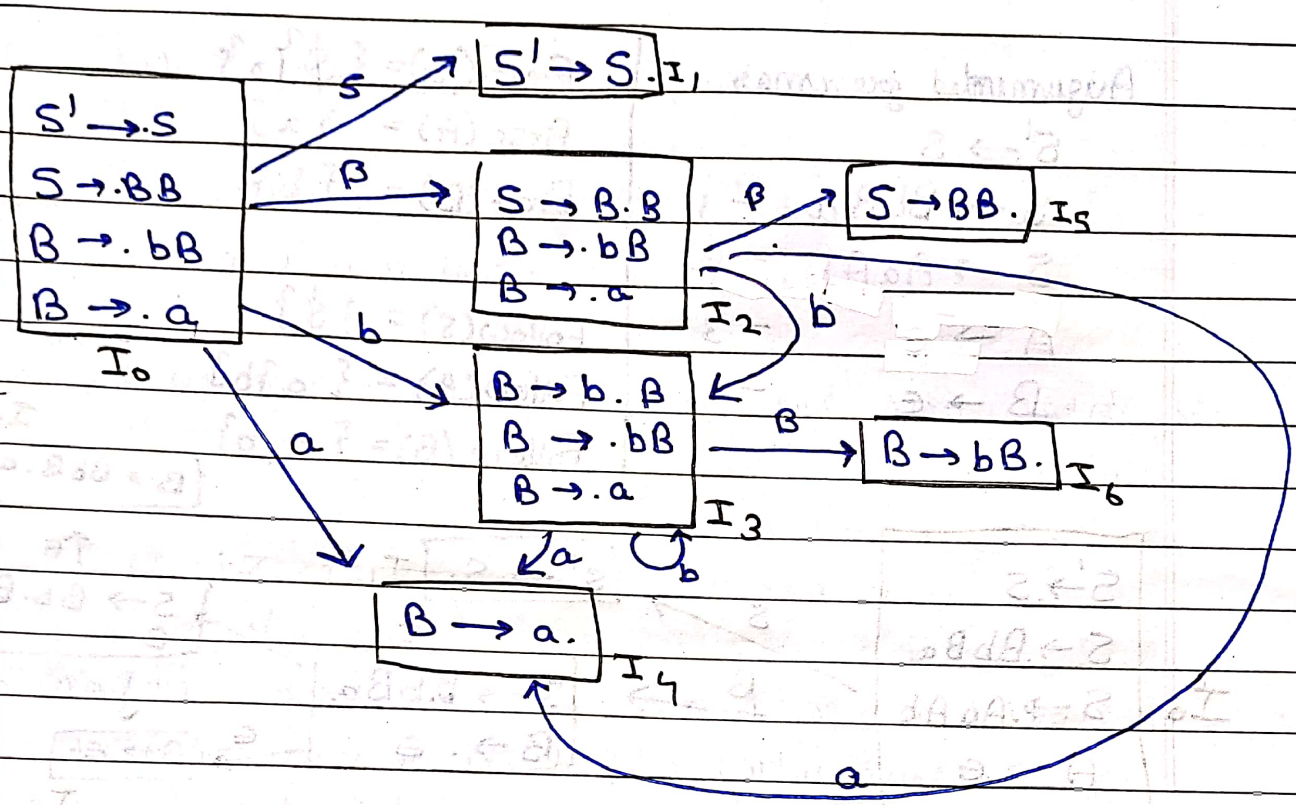
e.) All undefined entries in Action and Goto table are an error.

Q.No. 2
A. Construct SLR(1) parsing table for the following grammar, where S is the start symbol 2019M [5]
S→BB
B→bB | a

Augumented grammar
$S' \to S$
$S \to BB$ — R_1
$B \to bB$ — R_2
$B \to a$ — R_3

First (B) = {a, b}
First (S) = {a, b}

Follow (B) = {$}
Follow (S) = {$}



States of DFA	Action			GoTo	
	$	a	b	S	B
I_0	—	S_4	S_3	1	2
I_1	accept	—	—	—	—
I_2	—	S_4	S_3	—	5
I_3	—	S_4	S_3	—	6
I_4	R_3	—	—	—	—
I_5	R_1	—	—	—	—
I_6	R_2	—	—	—	—

SLR(1) Parsing table.

Q.No. 2

A. Construct SLR(1) parsing table for the following grammar, where S is the start symbol [5] 2020M

S→BbBa A→ϵ

S→AaAb B→ϵ

Page No. Date

Augumented grammar

$S' \to S$

$S \to BbBa$ – 1

$S \to AaAb$ – 2

$A \to \epsilon$ – 3

$B \to \epsilon$ – 4

First (S) = { a, b }

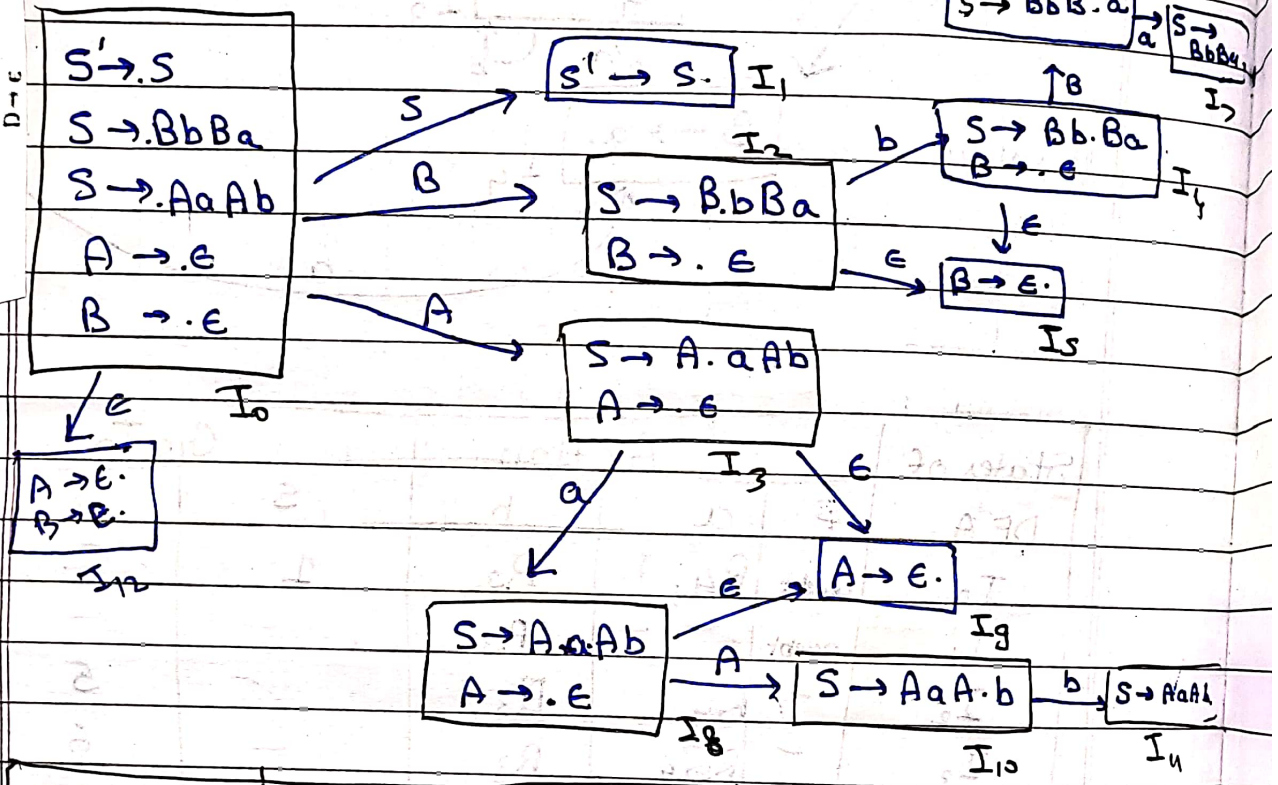
First (A) = { a }

First (B) = { b }

Follow(S) = { $ }

Follow(A) = { a, b }

Follow (B) = { a, b }



States of DFA	Action				Goto		
	a	b	ϵ	$	S	A	B
I_0	–	–	S_{12}	–	1	3	2
I_1	–	–	–	accept	–	–	–
I_2	–	S_4	S_5	–	–	–	–
I_3	S_8	–	S_9	–	–	–	–
I_4	–	–	S_5	–	–	–	6
I_5	R_4	R_4	–	–	–	–	–
I_6	S_7	–	–	–	–	–	–
I_7	–	–	–	R_1	–	–	–
I_8	–	–	S_9	–	–	10	–

Page No. | Date

I_9	R_3	R_3	—	—	—	—	—
I_{10}	—	S_{11}	—	—	—	—	—
I_{11}	—	—	—	R_2	—	—	—
I_{12}	R_3/R_4	R_3/R_4	—	—	—	—	—

Q.No. 1

A. Construct SLR (1) parsing table and compute FIRST & FOLLOW for the following grammar: [6]

S→xAy|xBy|xAz
A→ aS /q
B→ q

2019E

Q.5 Construct SLR(1) parsing table for the following grammer : [14]
S→xAy /xBy / xAz
A→aS / q
B→ q
(Also show canonical collection diagram & follow sets)

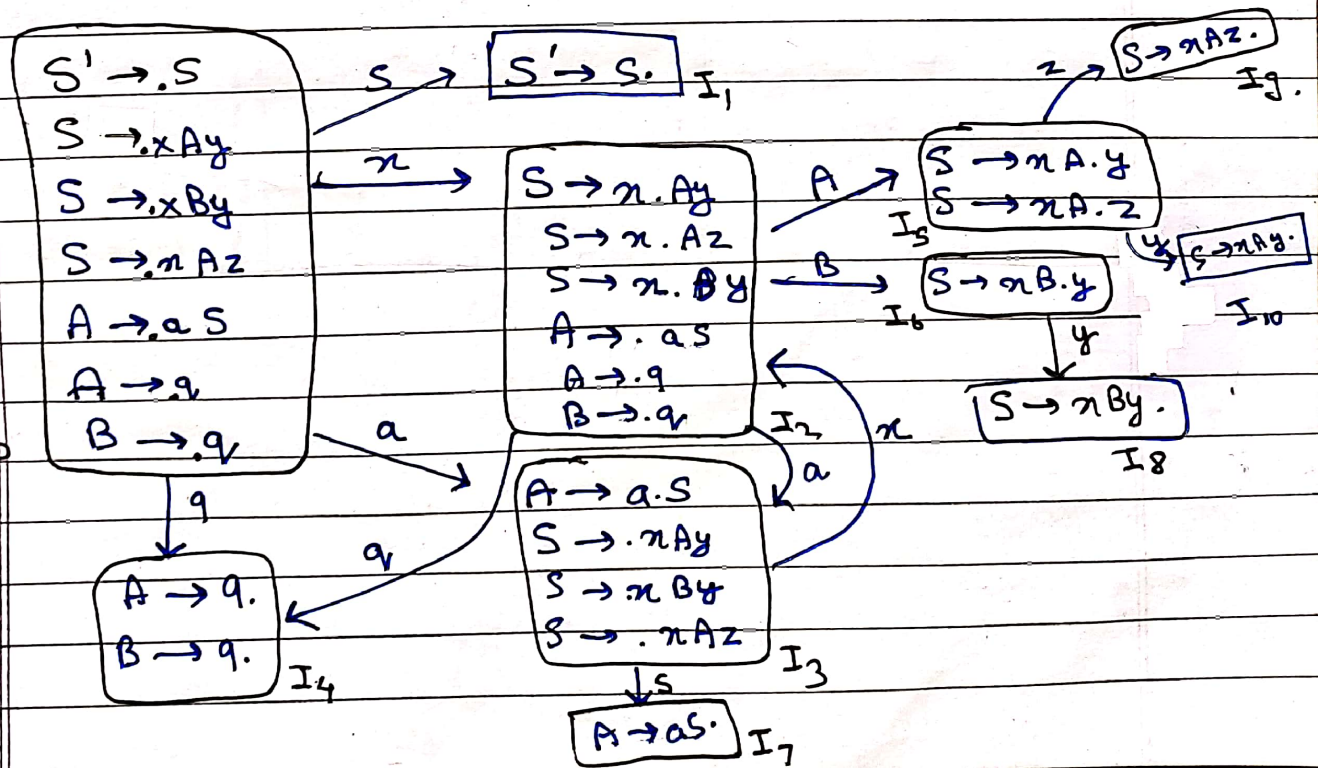
2017E

Augumented Grammar

$S' \to S$
$S \to xAy$ – 1
$S \to xBy$ – 2
$S \to xAz$ – 3
$A \to aS$ – 4
$A \to q$ – 5
$B \to q$ – 6

First (S) = {x}
First (A) = {a, q}
First (B) = {q}

Follow (S) = {$, y, z}
Follow (A) = {y, z}
Follow (B) = {y}



State of DFA	Action						Goto		
	z	x	y	a	q	$	S	A	B
I_0		S_2	-	S_3	S_4	-	1		
I_1		-	-	-	-	accept			
I_2		-	-	S_3	S_4	-		5	6
I_3		S_2	-	-	-	-	7		
I_4	R_5	-	R_5/R_6	-	-	-			
I_5	S_9	-	S_{10}	-	-	-			
I_6	-	-	S_8	-	-	-			
I_7	R_4	-	R_4	-	-	-			
I_8	R_2	-	R_2	-	-	R_2			
I_9	R_3	-	R_3	-	-	R_3			
I_{10}	R_1	-	R_1	-	-	R_1			

Q.No. 4

A. What is the advantage of left recursive grammar over right recursive grammar in LR parsing. Explain with suitable example. 2018 S [3]

We should use left recursive grammar as we can parse a sequence of any number of elements with bounded stack space.

With right recursion, no reduction takes place until the entire list of elements has been read; with left recursion, a reduction takes place as each new list element is encountered. It saves space.

In LR(1), left recursive grammar elimination allows for easier construction of LR parsing tables and avoids the need for backtracking during parsing. Right recursive requires extra work for conversion

Eg.

$E \to E+T \mid T$	$E \to TE'$
$T \to T*F \mid F$	$E' \to +TE' \mid \epsilon$
$F \to (E) \mid id$	$T \to FT'$
	$T' \to *FT' \mid \epsilon$
	$F \to (E) \mid id$
left	right

Page No.
Date

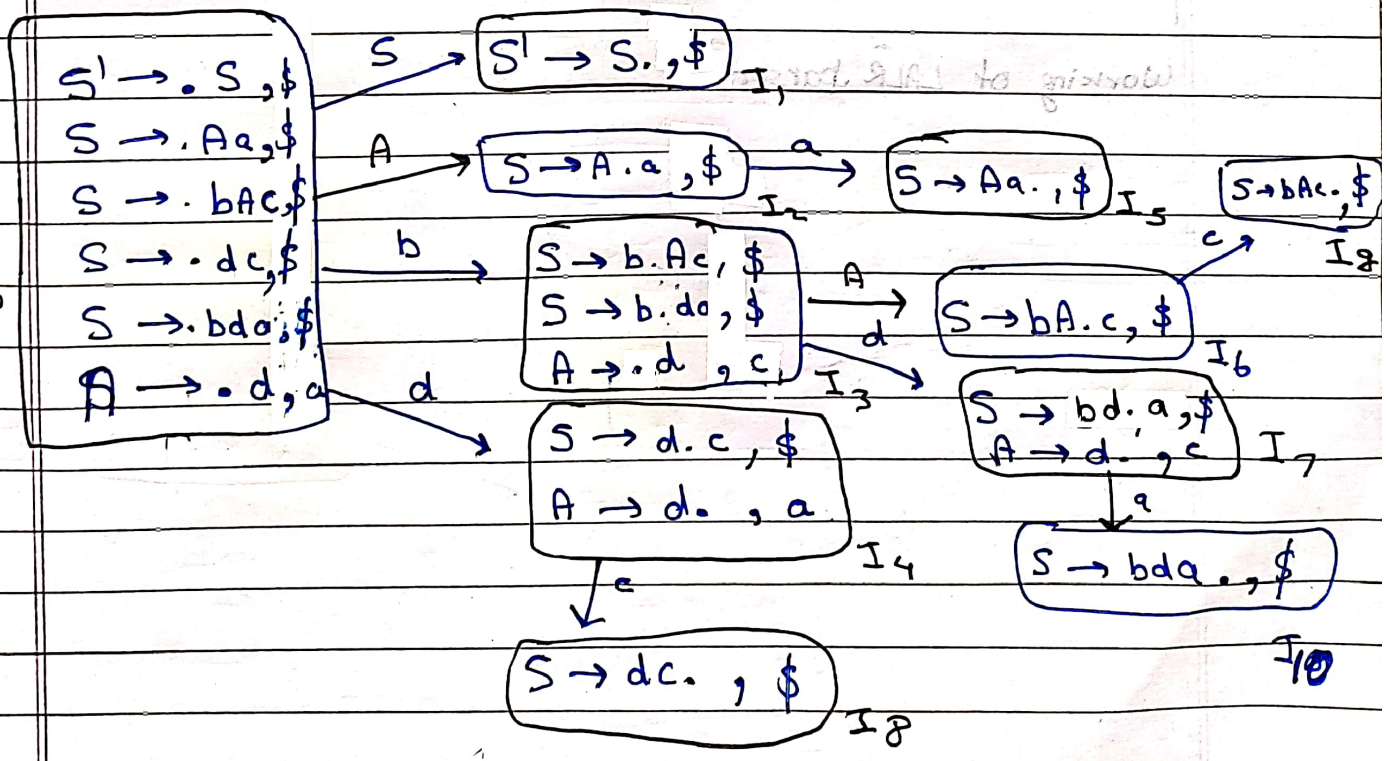
B. How LALR is different from CLR? explain and find canonical collection of sets of LR(1) items for following grammar [4]
S → Aa| bAc |dc |bda
A → d

2019 E

In LALR(1) parsing, the LR(1) items with equal productions but different look ahead are grouped to form an individual set of items. It is frequently the similar as CLR(1) except for the one difference that is the parsing table.

LALR Parser	CLR Parser
It is cheap and easy to implement.	It is expensive and difficult to imp
Less states / ~~more ICFC~~	More states
Error detection isn't immediate	
power is less	power is more

Grammar : S' → S
S → Aa | bAc | dc | bda
A → d



Q.No. 3

A. Discuss algorithm for computation of the sets of LR(1) items. Shows that the following grammar is LR(1) but not LALR(1) [3]

S→Aa|bAc|Bc|bBa
A→d
B→ d

2018 S

B. Explain the working of LALR parser and Construct canonical LR parsing table for following grammar [4] [CO#4]

S → Aa| bAc |dc |bda
A → d

2023 E

Parsing Table (CLR(1))

It's not LALR(1) as it can't be reduced w/o conflict ($I_4 - I_5$)

States of DFA	ACTION					GOTO	
	a	d	c	d	$	S	A
I_0		S_3		S_4		1	2
I_1					accept		
I_2	S_5						
I_3				S_7		1	
I_4	R_5		S_8				
I_5					R_1		
I_6	S_{10}		S_9				
I_7			R_5				
I_8					R_3		
I_9					R_2		
I_{10}					R_4		

Working of LALR parser

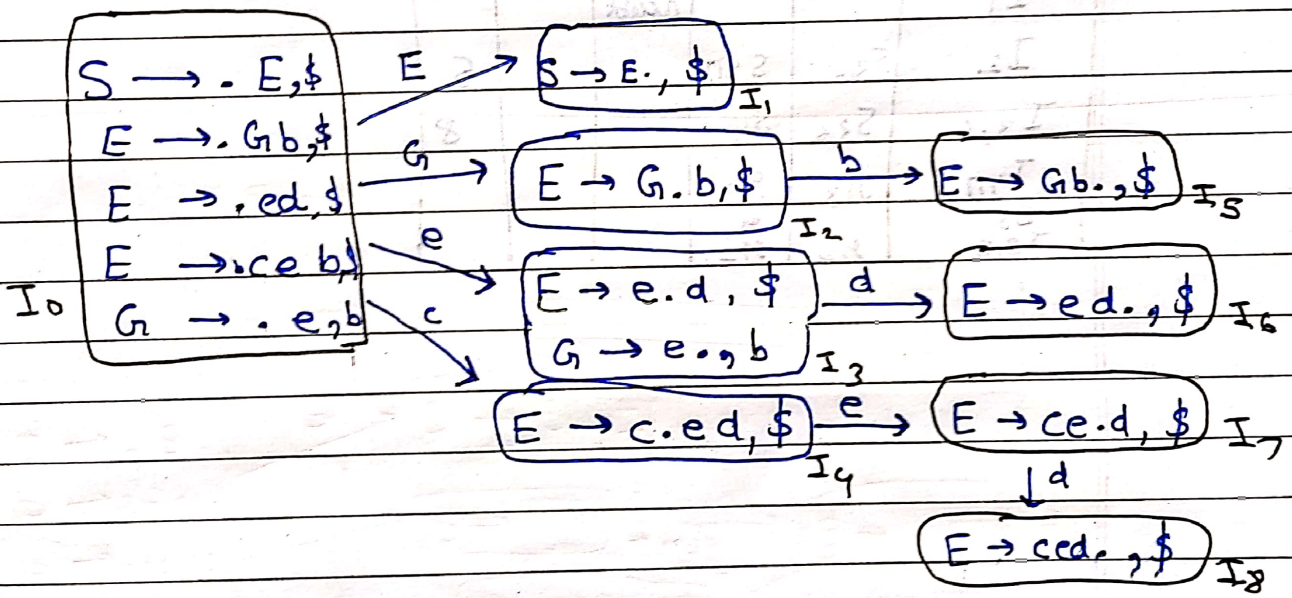
Page No.
Date

Q.No. 5
How LALR is different from CLR? explain and find sets of LR (1) items for following grammar 2023M {4}{CO#4}

E → Gb| cGd |ed |ceb
G → e .(where 'E' is the start symbol)

augumented grammar

S → E
E → Gb
E → cGd
E → ed
E → ceb
G → e



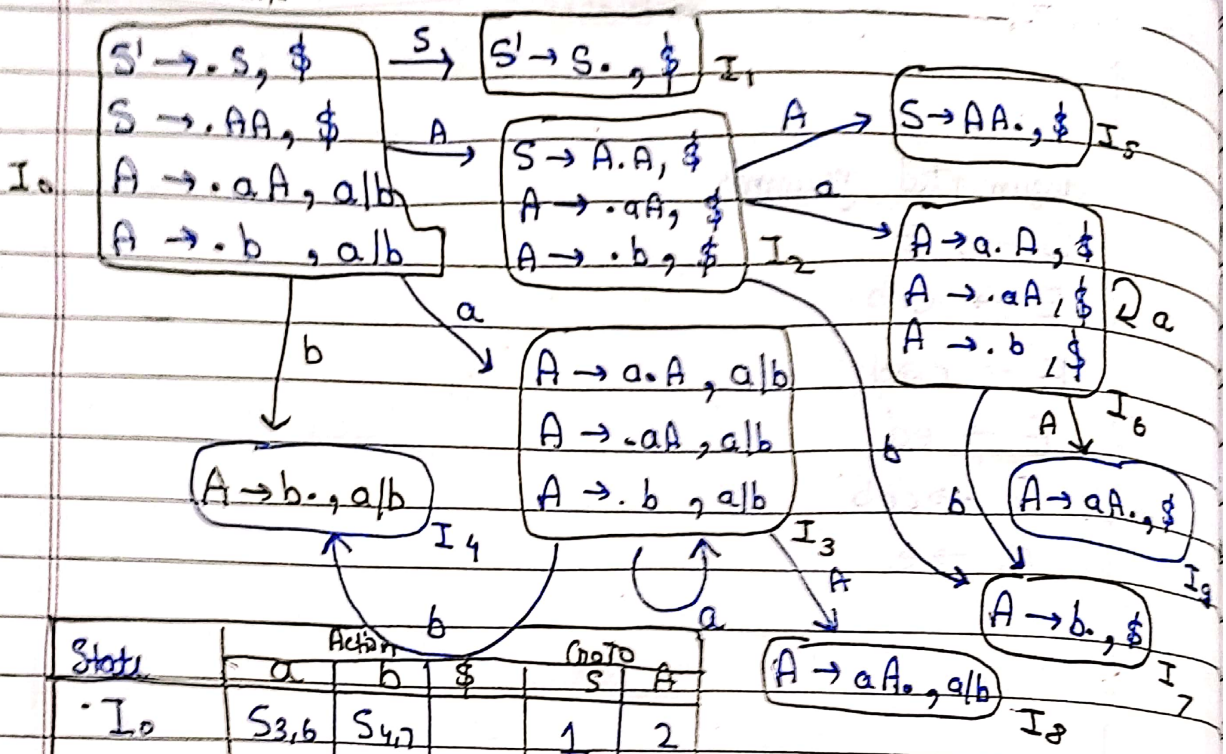
$S' \to S$
$S \to AA$
$A \to aA$
$A \to b$

Q.No. 1

A- How LALR is different from CLR? Construct LALR (1) parsing table for the following grammar: (6)

$S \to AA$
$A \to aA / b$

2018E



State	Action			GoTo	
	a	b	$	S	A
I_0	$S_{3,6}$	$S_{4,7}$		1	2
I_1			accept		
I_2	$S_{3,6}$	$S_{4,7}$			5
$I_{3,6}$	$S_{3,6}$	$S_{4,7}$			8
$I_{4,7}$	r_3	r_3			
I_5		r_1			
$I_{8,9}$	r_2	r_2			

Q.1 Construct CLR(1) parsing table for the following grammer : [14]

$E \to T / E-T$
$T \to F / *F$
$F \to id / (E)$

2018E

augumented grammar

$S \to E$
$E \to T / E-T$
$T \to F / *F$
$F \to id / (E)$

Page No.
Date

$S \rightarrow .E \;,\; \$$
$E \rightarrow .T \;,\; \$$
$E \rightarrow .E-T \;,\; -$
$T \rightarrow .F \;,\; \$$
$T \rightarrow .*F \;,\; id/($
$F \rightarrow .id \;,\; \$$
$F \rightarrow .(E) \;,\; -$

?