University of Asia Pacific

Department of CSE

Name: Rashik Rahman

Reg ID: 17201012

Year: 4th

Semester: 2nd

Course Code: CSE 429

Course Title: Compiler Design

Date: 17.09.2021

"During Examination and upload time I will not take any help from anyone. I will give my exam all by myself."

University of Asia Pacific

Admit Card

Mid-Term Examination of Spring, 2021

Financial Clearance	PAID

Registration No : 17201012

Student Name : Rashik Rahman

Program : Bachelor of Science in Computer Science and Engineering

Sl.NO.	COURSE CODE	COURSE TITLE	CR.HR.	EXAM. SCHEDULE
1	CSE 425	Computer Graphics	3.00	
2	CSE 426	Computer Graphics Lab	1.50	
3	CSE 429	Compiler Design	3.00	
4	CSE 430	Compiler Design Lab	1.50	
5	BUS 401	Business and Entrepreneurship	3.00	
6	BUS 402	Business and Entrepreneurship Lab	0.75	
7	CSE 457	Design and Testing of VLSI	3.00	
8	CSE 458	Design and Testing of VLSI Lab	0.75	
9	CSE 400	Project / Thesis	3.00	

Total Credit: 19.50

1. Examinees are not allowed to enter the examination hall after 30 minutes of commencement of examination for mid semester examinations and 60 minutes for semester final examinations.

2. No examinees shall be allowed to submit their answer scripts before 50% of the allocated time of examination has elapsed.

3. No examinees would be allowed to go to washroom within the first 60 minutes of final examinations.

4. No student will be allowed to carry any books, bags, extra paper or cellular phone or objectionable items/incriminating paper in the examination hall.
Violators will be subjects to disciplinary action.

This is a system generated Admit Card. No signature is required.

Admit Card Generation Time: 12-Sep-2021 03:31 PM

17201012

Answer to the Q. No. 1(a)

i) Symbol table is an important data structure created and maintained by by compilers in order to store information about the occurence of various entities such as variable names, function names, objects etc. It is used for both analysis and synthesis part of compiler. It has the following roles:

→ ~~Stror~~ Store the name of ~~of~~ entities in structured format

→ Verify declaration of variables.

→ Implement type checking

→ Determine the scope of a name.

ii) Pattern is the notation used to identify the set of lexemes represented by a token. In another words it is the description of the the lexeme of token. In sytax analysis ~~And in~~ pattern is handled like the following.

Token: tok_identifier
Lexemes; buffer_size, D2
Pattern: letter followed by letters on digits.

iii) Difference between DFA and NFA:

DFA	NFA
i) DFA stands for Deterministic Finite Automata	i) NFA stands for Nondeterministic finite Automata.
ii) DFA can't use empty string transitions	ii) Can use empty string transitions.
iii) DFA can be understood as one machine	iii) NFA can be understood as multiple little machines computing at the same time
iv) DFA is more difficult to construct	iv) NFA is easier to construct

Answer to the Q.No. 1(b)

i) Identifiers = {a, zA, @ZO, _$}

∴ Alphabets for DFA, Σ = {a, zA, ZO, _$}

Here we draw a DFA where the string must contain all the elements in alphabets. Reapeted occurance of identifier would also be acceptable

He we use states 0,1,2,3,4 where 0 is start state and 4 is end state.

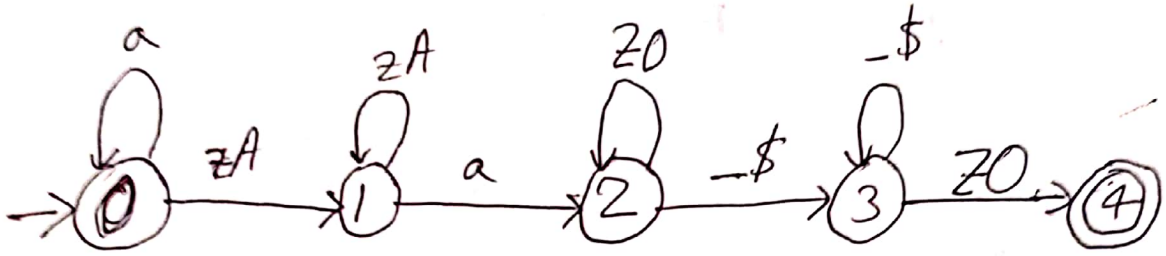


Fig: DFA diagram that contains atleast each of the alphabets.

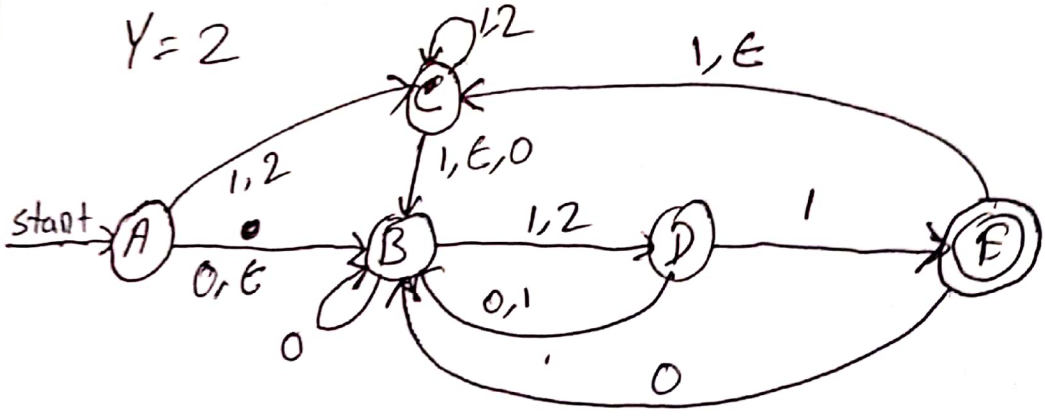
Answer to the Q. NO. 2(a)

(i)

Here,

$x_0 = 12 \% 3 = 0$

$Y = 2$



(4)

17201012

Transition table for the updated FA.

states \ alphabets	0	1	2	∈
A	B	C	C	B
B	B	D	D	–
C	B	C, B	C	B
D	B	B, E	–	–
E	B	C	–	C

(ii)

Eclosure (A) = {A, B}

Eclosure (B) = {B}

Eclosure (C) = {C, B}

Eclosure (D) = {D}

Eclosure (E) = {C, E}

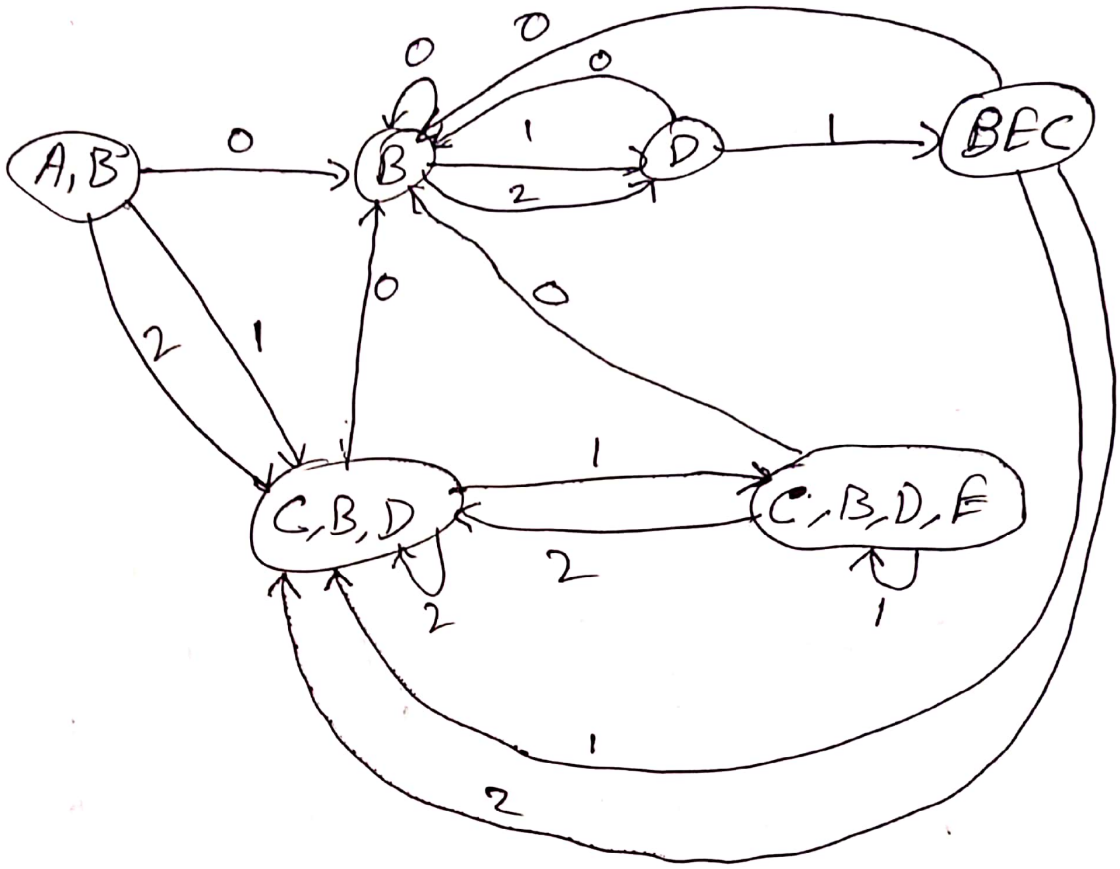
Eclosure (C, D) = {C, B, D}

17201012

Updated table

States	0	1	2
Start {A,B}	{B}	{C,B,D}	Eclose(C,D) = {C,B,D}
{B}	{B}	{D}	{D}
{C,B,D}	{B}	~~{C,B,D,F~~ Eclose(C,B,D,E) = {C,B,D,E}	{C,B,D}
{D}	{B}	Eclose(B,F) = {B,F,C}	{ }
{C,B,D,E}	{B}	{C,B,D,E}	{C,B,D}
{B,F,C}	{B}	{C,B,D}	{C,B,D}

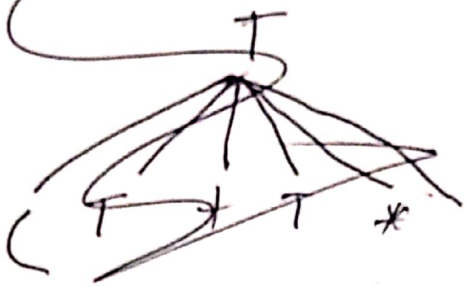
Updated FA:



17201012

(a)

Parse tree (L.M.D) =



i) L.M.D = S = S*T = T*T

= (S)*T

= (S*T)*T

= (T+T)*T

= (S+T*T)*T

= (T+T*T)*T

= (x+T*T)*T

= (x+y*T)*T

= (x+y*z)*T

17201012

i) R.M.D = ~~T * T~~ S = S * T = S * x

= T * x

= (S) * x

= (S * T) * x

= (S * Z) * x

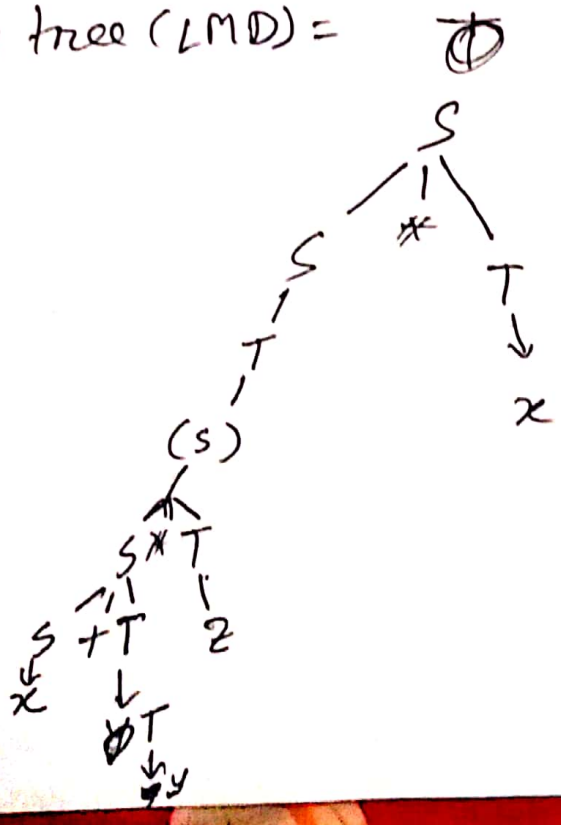
= (S + T * Z) * x

= (S + y * Z) * x

= (x + y * Z) * x

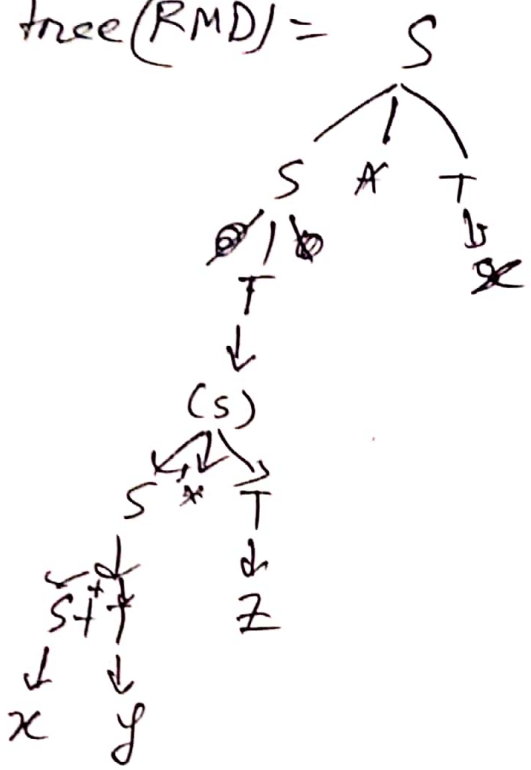
ii)

Parse tree (LMD) =



17201012

Parsee tree (RMD) =



∴ Parsee tree (LMD) == Parsee tree (RMD)

∴ The grammer isn't ambiguous.