
Reg.No:

SNS College of Technology, Coimbatore-35.
(Autonomous)
B.E/B.Tech- Internal Assessment -I
Academic Year 2022-2023(ODD)
Fifth Semester

A

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

19CSB301 – AUTOMATA THEORY AND COMPILER DESIGN

Time: $1^{1/2}$ Hours **Maximum Marks: 50**

Answer All Questions

PART-A (5 x 2 = 10 Marks)

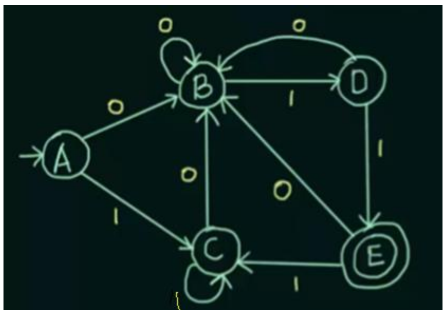
1. List the types of grammar based on Chomsky Hierarchy CO1 REM
2. Find the type of Grammar for the given production rule. Give the general rule for that type of grammar CO1 ANA
   S-> AC|CB
   C->aCb| ε
   A->aA| ε
   B->Bb| ε
3. Construct the Deterministic Finite Automata for set of strings over {a, b} which has atmost 1a CO1 CRE
4. What is sentinel and give its usage. CO2 REM
5. Differentiate tokens, patterns, lexeme. CO2 ANA

PART–B (13 X 2 = 26 Marks)

6. (a) Construct the Deterministic Finite Automata for the following by their regular language and regular expression over {0,1}/{a,b}: 13 CO1 CRE
   (i) Set of strings that begins with 0
   (ii) Set of strings that begins with 0 and ends with 1
   (iii) Set of strings that ends with bb
   (iv) Set of strings that has at least 1 a

(or)

	(b)	Construct the NFA which accepts all the strings over {0,1} ending with 01 and find the equivalent DFA	13	CO1	APP
7.	(a)	(i) Give the difference between NFA and DFA	3	CO1	ANA
		(ii) Simplify the given DFA	10	CO1	APP



(or)

	(b)	Outline the following: (i) Language Processing System (ii) Compiler Construction Tools	13	CO2	UND

PART-C (14 x 1 = 14 Marks)

8.	(a) Explain how the Turing machine is more powerful than other automata with its formal and graphical representation. Construct the Turing machine for Language 01*0	CO1	APP

(or)

	(b) Elaborate on the various phases of compiler and trace it with the program segment (position:=initial + rate * 60)	CO2	APP

****

(Note: UND-Understand REM-Remember ANA-Analyze APP-Apply CRE-Create)

Reg.No:

SNS College of Technology, Coimbatore-35.
(Autonomous)
B.E/B.Tech- Internal Assessment -I
Academic Year 2022-2023(ODD)
Fifth Semester

$\boxed{B}$

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

19CSB301 – AUTOMATA THEORY AND COMPILER DESIGN

Time: $1^{1/2}$ Hours **Maximum Marks: 50**

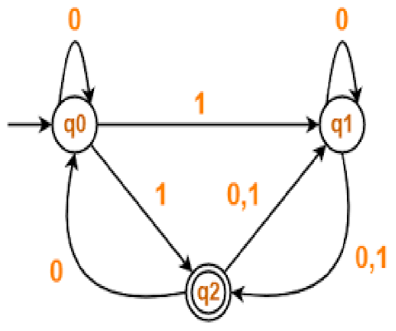
Answer All Questions

PART-A (5 x 2 = 10 Marks)

1.	Find the type of Grammar for the given production rule AB → CDB , AB → CdEB , ABcd → abCDBcd , B → b	CO1	ANA
2.	State the Regular expression for set of strings over {a,b} that has at least 1 a	CO1	APP
3.	Differentiate NFA and DFA	CO1	ANA
4.	Outline the tuple representation of Turing Machine	CO1	UND
5.	List out the features of the Compiler	CO2	REM

PART–B (13 X 2 = 26 Marks)

6.	(a)	Define Finite automata and explain on the types of finite automata and convert the following NFA to DFA	13	CO1	APP



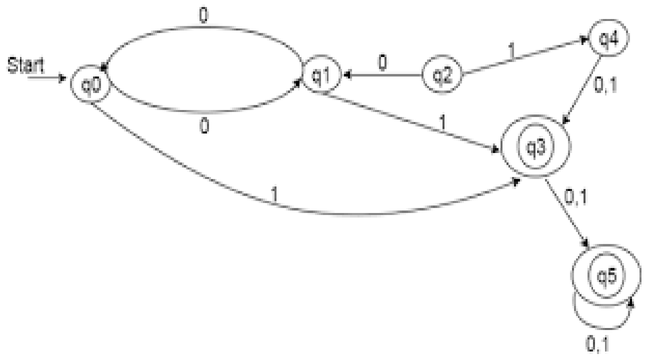
(or)

(b) Explain how Pushdown Automata is more powerful than Finite automata with its formal and graphical representation. Construct the Pushdown Automata for Language L = $\{0^n1^n|n>=0\}$ — 13 CO1 ANA

7. (a) Elaborate the various phases of compiler and trace it with the program segment (a=b+c*50.0) — 13 CO2 APP

(or)

(b) Find the minimized DFA for the given DFA — 13 CO1 ANA



PART-C (14 x 1 = 14 Marks)

8. (a) Construct the Regular Expression, DFA which accepts a string over {0,1} / {a,b} — CO1 CRE

i. which is of length 2

ii. set of strings that ends with “bb”

iii. start with “0” and ends with “1”

iv. exactly 1 a

(or)

(b) Outline on following: — CO2 UND

(i) Cousins of the compiler

(ii) Tools available for various phases of compiler

****

(Note: UND-Understand REM-Remember ANA-Analyze APP-Apply CRE-Create)

Reg.No:

SNS College of Technology, Coimbatore-35.
(Autonomous)
B.E/B.Tech- Internal Assessment -I
Academic Year 2022-2023(ODD)
Fifth Semester

A

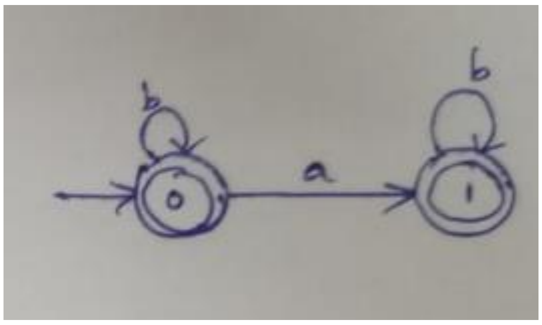
DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

19CSB301 – AUTOMATA THEORY AND COMPILER DESIGN

Time: $1^{1/2}$ Hours **Maximum Marks: 50**

Answer All Questions

PART-A (5 x 2 = 10 Marks)

1. List the types of grammar based on Chomsky Hierarchy CO1 REM
   - Type 0 is known as unrestricted grammar.
   - Type 1 is known as context-sensitive grammar.
   - Type 2 is known as a context-free grammar.
   - Type 3 Regular Grammar
2. Identify the type of Grammar for the given production rule. Give the general rule for that type of grammar CO1 ANA
   S□ AC|CB
   C□aCb| ε
   A□aA| ε
   B□Bb| ε
   Type 2
3. Construct the Deterministic Finite Automata for set of strings over {a, b} which has at most 1a CO1 CRE
   

4. What is sentinel and give its usage. CO2 REM
   A sentential form is any string derivable from the start symbol. Thus, in the derivation of a + a * a , E + T * F and E + F * a and F + a * a are all sentential forms as are E and a + a * a themselves. Sentence. A sentence is a sentential form consisting only of terminals such as a + a * a.
5. Differentiate tokens, patterns, lexeme. CO2 ANA

Token

It is basically a sequence of characters that are treated as a unit as it cannot be further broken down. In programming languages like C language- keywords (int, char, float, const, goto, continue, etc.) identifiers (user-defined names), operators (+, -, *, /), delimiters/punctuators like comma (,), semicolon(;), braces ({ }), etc. , strings can be considered as tokens. This phase recognizes **three types of tokens:** Terminal Symbols (TRM)- Keywords and Operators, Literals (LIT), and Identifiers (IDN).

Lexeme

It is a sequence of characters in the source code that are matched by given predefined language rules for every lexeme to be specified as a valid token.

Pattern

It specifies a set of rules that a scanner follows to create a token.

PART–B (13 X 2 = 26 Marks)

6. (a) Construct the Deterministic Finite Automata for the following by their regular language and regular expression: CO1 CRE

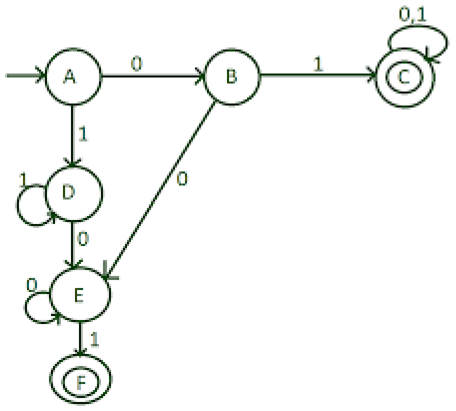
3

3

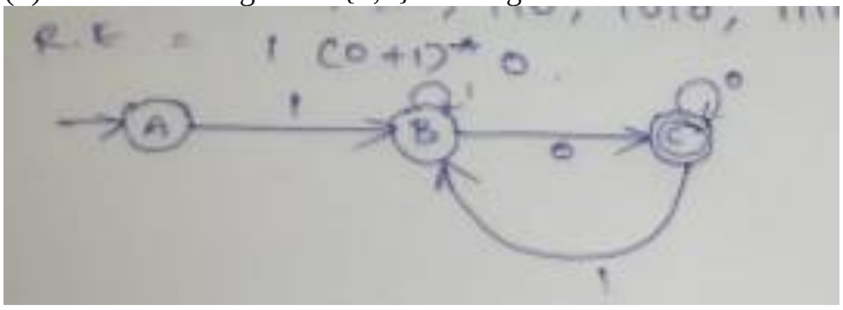
3

4

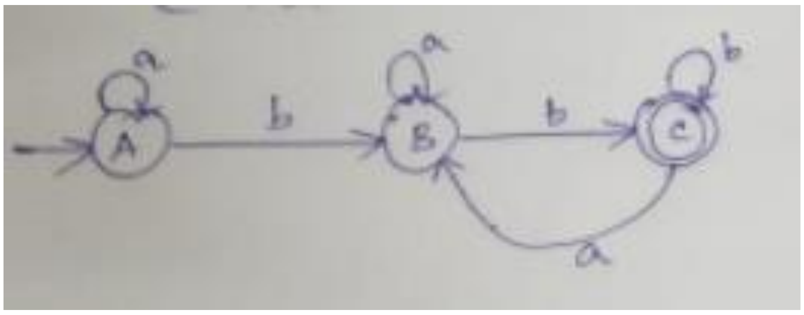
(i) Set of strings over {0,1} that begins with 0



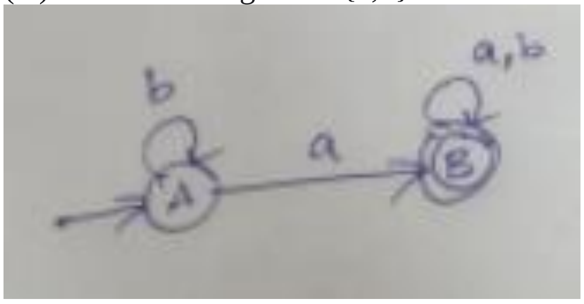
(ii) Set of strings over {0,1} that begins with 0 and ends with 1



(iii) Set of strings over {a,b} that ends with bb

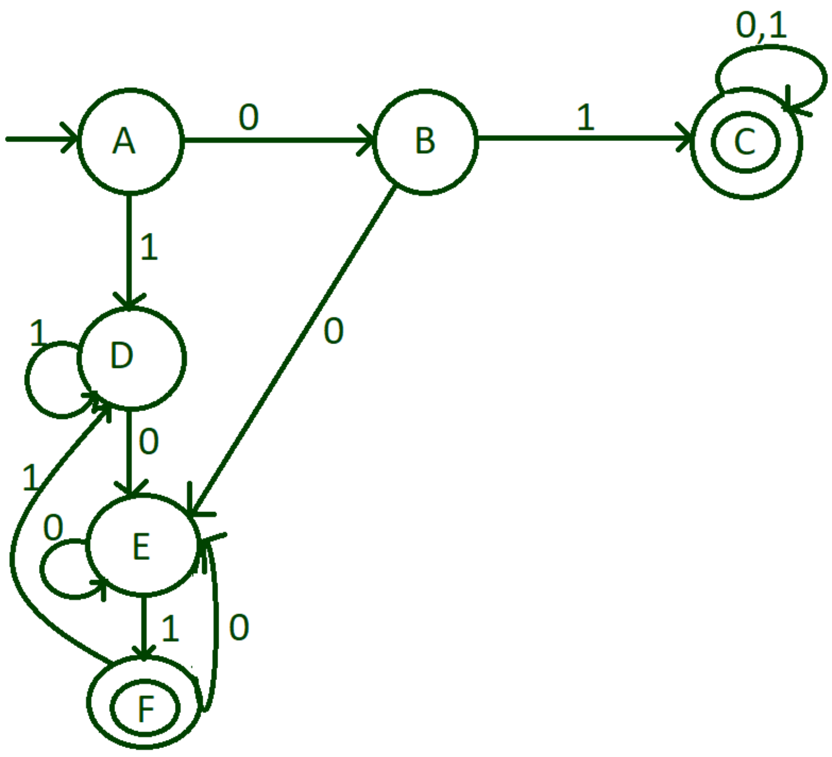


(iv) Set of strings over {a,b} that has at least 1 a

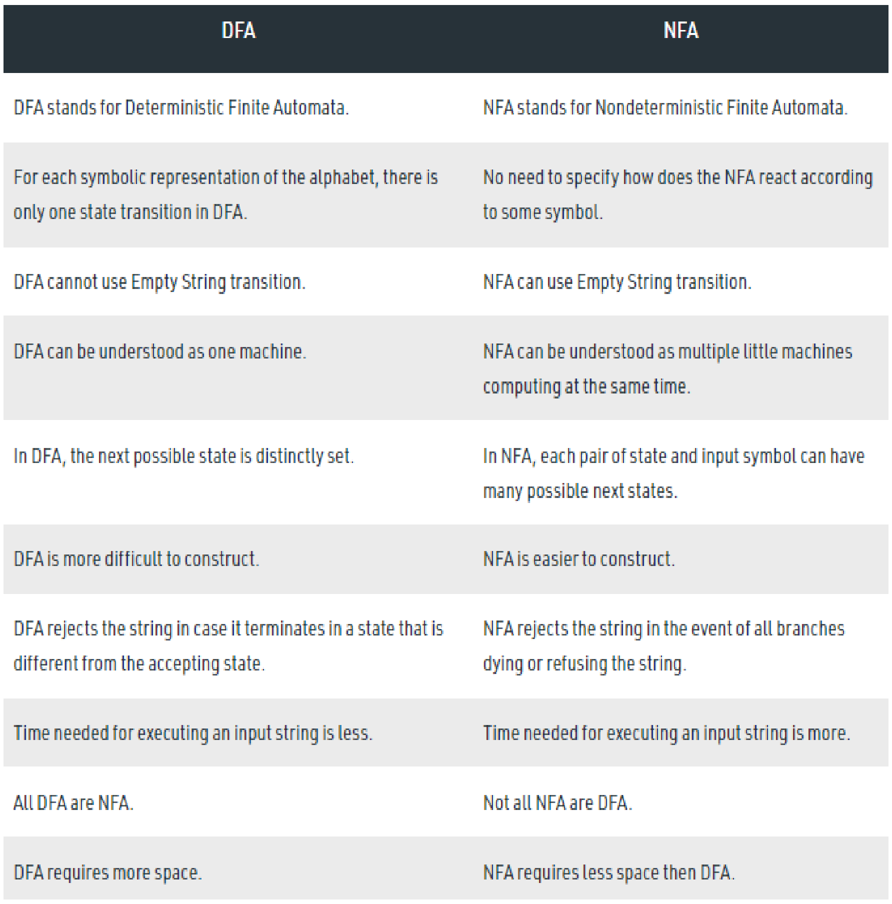


(or)

(b)	Construct the NFA which accepts all the strings over {0,1} ending with 01 and find the equivalent DFA	13	CO1	APP

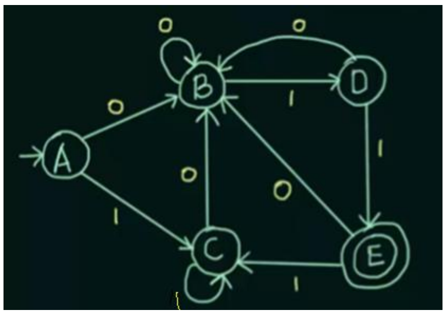


7. (a) (i) Give the difference between NFA and DFA 3 10 CO1 ANA



DFA	NFA
DFA stands for Deterministic Finite Automata.	NFA stands for Nondeterministic Finite Automata.
For each symbolic representation of the alphabet, there is only one state transition in DFA.	No need to specify how does the NFA react according to some symbol.
DFA cannot use Empty String transition.	NFA can use Empty String transition.
DFA can be understood as one machine.	NFA can be understood as multiple little machines computing at the same time.
In DFA, the next possible state is distinctly set.	In NFA, each pair of state and input symbol can have many possible next states.
DFA is more difficult to construct.	NFA is easier to construct.
DFA rejects the string in case it terminates in a state that is different from the accepting state.	NFA rejects the string in the event of all branches dying or refusing the string.
Time needed for executing an input string is less.	Time needed for executing an input string is more.
All DFA are NFA.	Not all NFA are DFA.
DFA requires more space.	NFA requires less space then DFA.

(ii) Find the minimized number of states in the given DFA CO1 APP



Q□{A,B,C,D,E} q0=A, F=E, inputs={0,1}

0 – Equivalence □ {A,B,C,D} {E}

1 – Equivalence □{A,B,C} {D}{E}

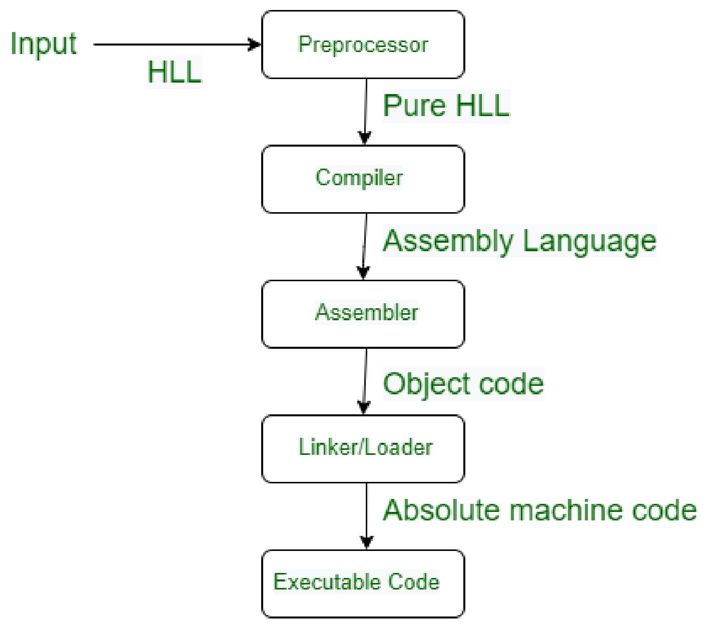
2 – Equivalence □{A,C}{B}{D}{E}

3 – Equivalence □{A,C}{B}{D}{E}

(or)

(b) Outline the following: 13 CO2 UND

(i) Language Processing System



(ii) Compiler Construction Tools

The compiler writer can use some specialized tools that help in implementing various phases of a compiler. These tools assist in the creation of an entire compiler or its parts. Some commonly used compiler construction tools include:

1. **Parser Generator** –
It produces syntax analyzers (parsers) from the input that is based on a grammatical description of programming language or on a context-free grammar. It is useful as the syntax analysis phase is highly complex and consumes more manual and compilation time. Example: PIC, EQM

2. **Scanner Generator** –
It generates lexical analyzers from the input that consists of regular expression description based on tokens of a language. It generates a finite automaton to recognize the regular expression. Example: Lex

3. **Syntax directed translation engines** –
It generates intermediate code with three address format from the input that consists of a parse tree. These engines have routines to traverse the parse tree and then produces the intermediate code. In this, each node of the parse tree is associated with one or more translations.

4. **Automatic code generators** –
It generates the machine language for a target machine. Each operation

of the intermediate language is translated using a collection of rules and then is taken as an input by the code generator. A template matching process is used. An intermediate language statement is replaced by its equivalent machine language statement using templates.

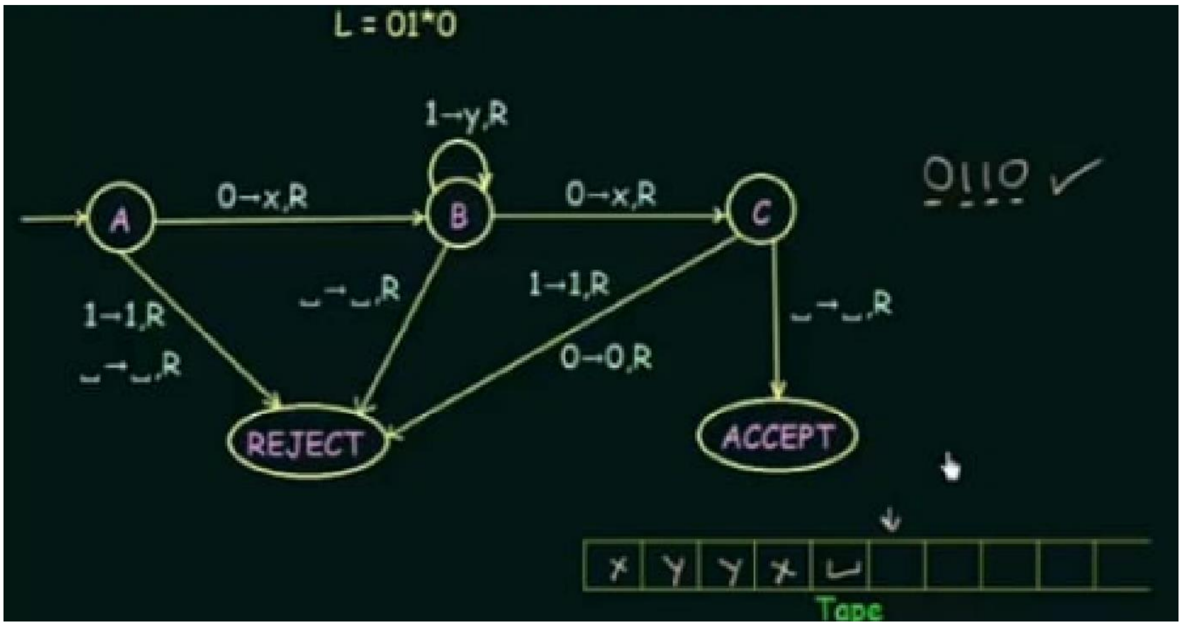
5. **Data-flow analysis engines** –
It is used in code optimization.Data flow analysis is a key part of the code optimization that gathers the information, that is the values that flow from one part of a program to another. Refer – data flow analysis in Compiler

6. **Compiler construction toolkits** –
It provides an integrated set of routines that aids in building compiler components or in the construction of various phases of compiler.

PART-C (14 x 1 = 14 Marks)

8. (a) Explain how Turing machine is more powerful than other automata with its formal and graphical representation. Construct the Turing machine for Language 01*0 CO1 APP

- Turing Machine recursively enumerable language
- Alan Turing (1936)
- Unrestricted Grammar
- ***Tape***
- TM what can be computed
- ***Model for Computer***
- Algorithm TM can do its computation



- Applications
- Computer Networks
- Artificial Intelligence
- Machine Learning

(or)

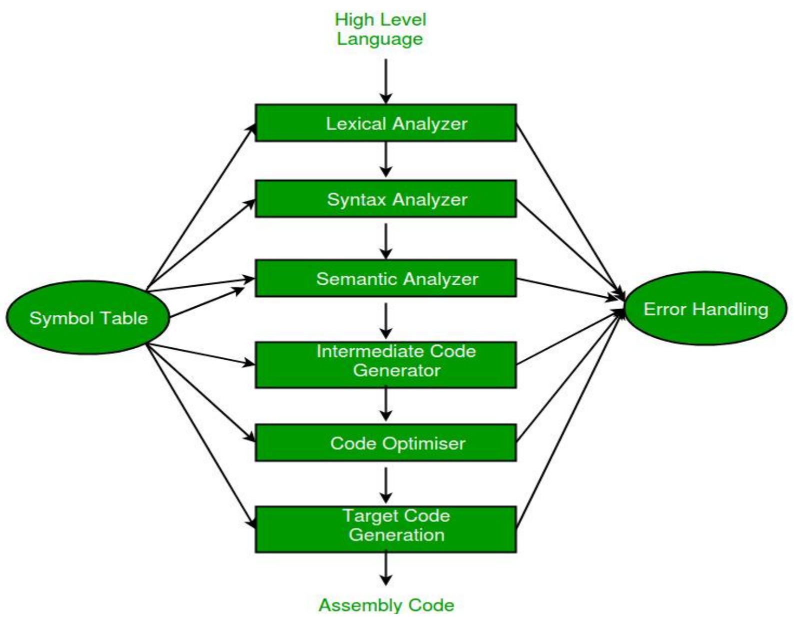
(b) Describe the various phases of compiler and trace it with the program segment CO2 APP (position:=initial + rate * 60)

We basically have two phases of compilers, namely the

1. Analysis phase
2. Synthesis phase.

The analysis phase creates an intermediate representation from the given source code.

The synthesis phase creates an equivalent target program from the intermediate representation.



Symbol Table – It is a data structure being used and maintained by the compiler, consisting of all the identifier's names along with their types. It helps the compiler to function smoothly by finding the identifiers quickly.

The analysis of a source program is divided into mainly three phases. They are:

Linear Analysis-

Hierarchical Analysis-

Semantic Analysis-

Lexical Analyzer –

It is also called a scanner. It takes the output of the preprocessor (which performs file inclusion and macro expansion) as the input which is in a pure high-level language. It reads the characters from the source program and groups them into lexemes (sequence of characters that "go together"). Each lexeme corresponds to a token. Tokens are defined by regular expressions which are understood by the lexical analyzer. It also removes lexical errors (e.g., erroneous characters), comments, and white space.

1. **Syntax Analyzer** – It is sometimes called a parser. It constructs the parse tree. It takes all the tokens one by one and uses Context-Free Grammar to construct the parse tree.

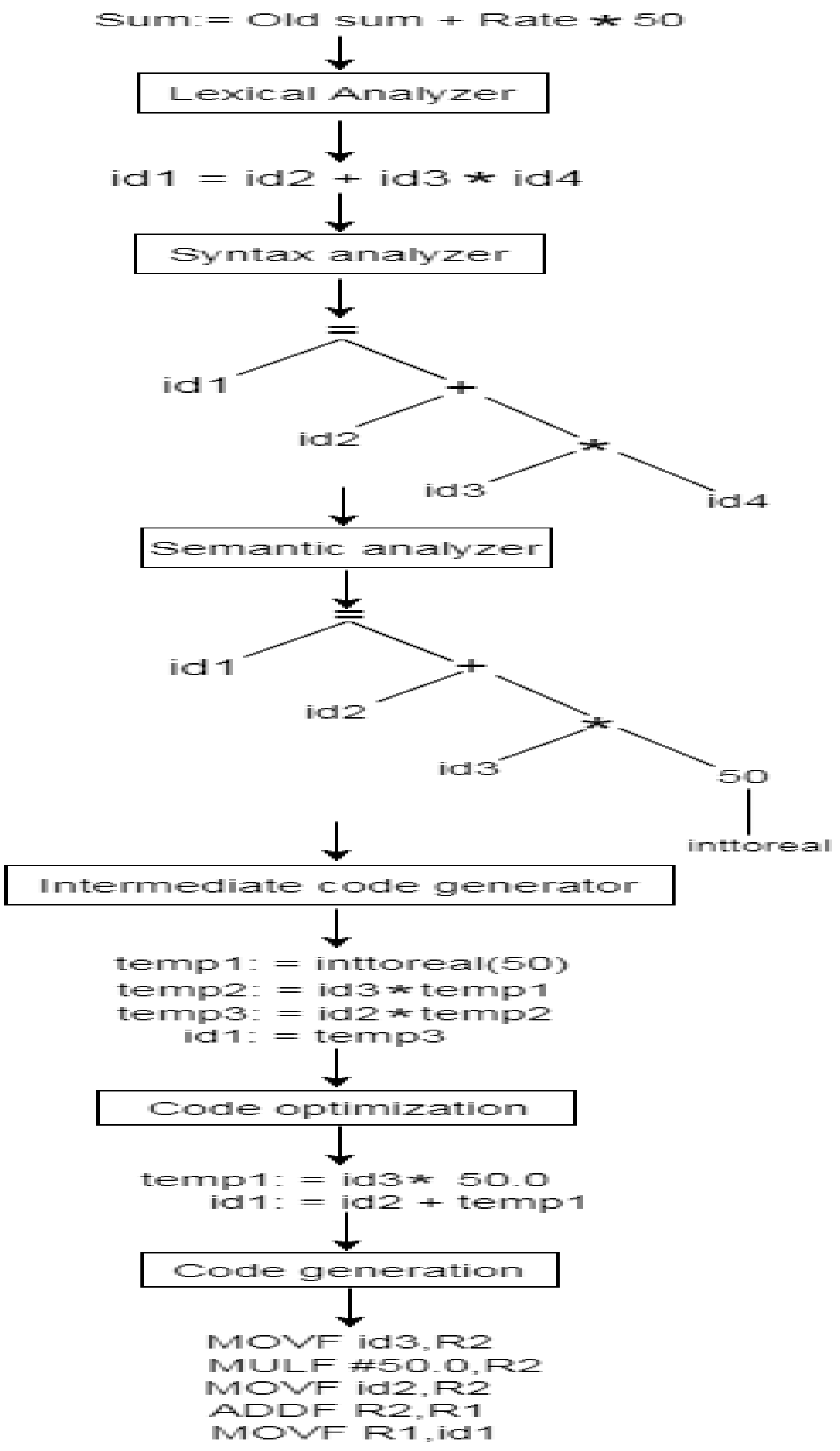
Why Grammar?

The rules of programming can be entirely represented in a few productions. Using these productions we can represent what the program actually is. The input has to be checked whether it is in the desired format or not.

The parse tree is also called the derivation tree. Parse trees are generally constructed to check for ambiguity in the given grammar. There are certain rules associated with the derivation tree.

2. **Semantic Analyzer –** It verifies the parse tree, whether it's meaningful or not. It furthermore produces a verified parse tree. It also does type checking, Label checking, and Flow control checking.

- **Intermediate Code Generator –** It generates intermediate code, which is a form that can be readily executed by a machine We have many popular intermediate codes. Example – Three address codes etc. Intermediate code is converted to machine language using the last two phases which are platform dependent. Till intermediate code, it is the same for every compiler out there, but after that, it depends on the platform. To build a new compiler we don't need to build it from scratch. We can take the intermediate code from the already existing compiler and build the last two parts.
- **Code Optimizer –** It transforms the code so that it consumes fewer resources and produces more speed. The meaning of the code being transformed is not altered. Optimization can be categorized into two types: machine-dependent and machine-independent.
- **Target Code Generator –** The main purpose of the Target Code generator is to write a code that the machine can understand and also register allocation, instruction selection, etc. The output is dependent on the type of assembler. This is the final stage of compilation. The optimized code is converted into relocatable machine code which then forms the input to the linker and loader.



(Note: UND-Understand REM-Remember ANA-Analyze APP-Apply CRE-Create)

Prepared By **Verified By** **HoD**

Reg.No:

SNS College of Technology, Coimbatore-35.
(Autonomous)
B.E/B.Tech- Internal Assessment -I
Academic Year 2022-2023(ODD)
Fifth Semester

B

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

19CSB301 – AUTOMATA THEORY AND COMPILER DESIGN

Time: $1^{1/2}$ Hours **Maximum Marks: 50**

Answer All Questions

PART-A (5 x 2 = 10 Marks)

1. Identify the type of Grammar for the given production rule CO1 ANA

$AB \rightarrow CDB$, $AB \rightarrow CdEB$, $ABcd \rightarrow abCDBcd$, $B \rightarrow b$

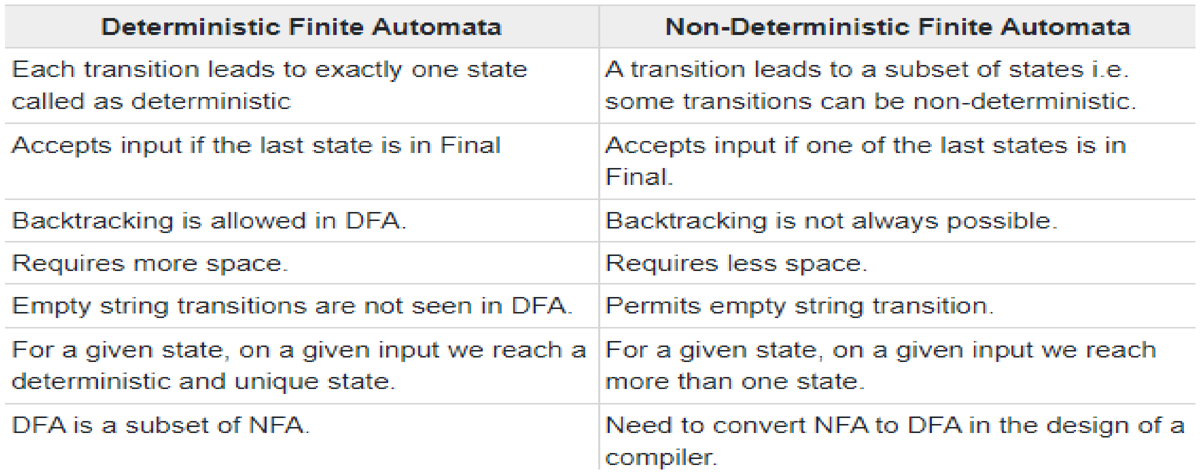
Type 1

2. State the Regular expression for the set of strings over {a,b} that has atleast 1 a CO1 APP

R.L = {a,ba,ab,aa,aaa,aba,bbbba,bba,bbbaa,bbaaa,…..}

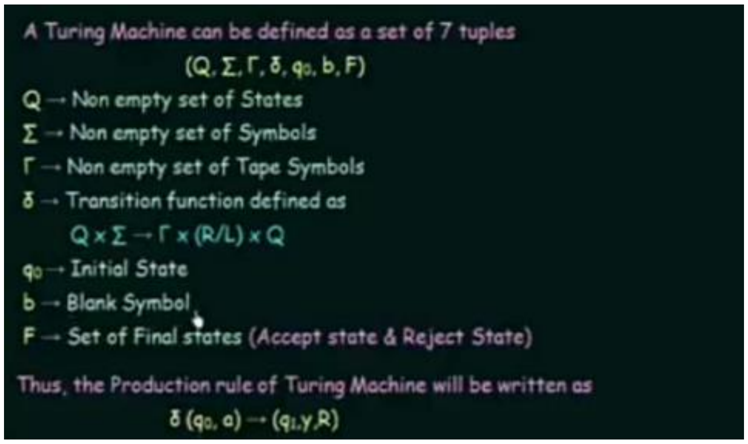
R.E = (a+b)* a (a+b)*

3. Differentiate NFA and DFA CO1 ANA



Deterministic Finite Automata	Non-Deterministic Finite Automata
Each transition leads to exactly one state called as deterministic	A transition leads to a subset of states i.e. some transitions can be non-deterministic.
Accepts input if the last state is in Final	Accepts input if one of the last states is in Final.
Backtracking is allowed in DFA.	Backtracking is not always possible.
Requires more space.	Requires less space.
Empty string transitions are not seen in DFA.	Permits empty string transition.
For a given state, on a given input we reach a deterministic and unique state.	For a given state, on a given input we reach more than one state.
DFA is a subset of NFA.	Need to convert NFA to DFA in the design of a compiler.

4. Give the tuple representation of Turing Machine CO1 UND


A Turing Machine can be defined as a set of 7 tuples

$(Q, \Sigma, \Gamma, \delta, q_0, b, F)$

$Q \to$ Non empty set of States

$\Sigma \to$ Non empty set of Symbols

$\Gamma \to$ Non empty set of Tape Symbols

$\delta \to$ Transition function defined as

$Q \times \Sigma \to \Gamma \times (R/L) \times Q$

$q_0 \to$ Initial State

$b \to$ Blank Symbol

$F \to$ Set of Final states (Accept state & Reject State)

Thus, the Production rule of Turing Machine will be written as

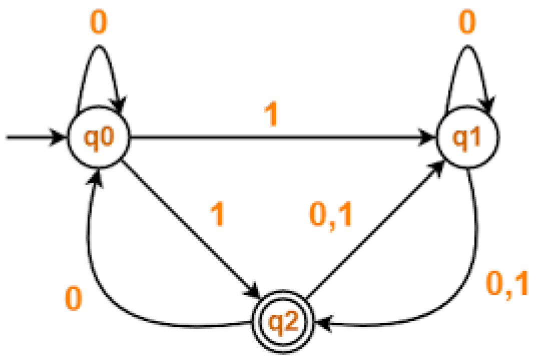
$\delta(q_0, a) \to (q_1, y, R)$

5. List out the features of the Compiler CO2 REM

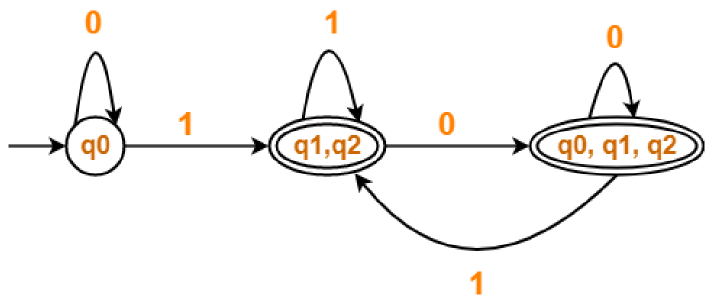
- Compilation speed.
- The correctness of machine code.
- The meaning of code should not change.
- Speed of machine code.
- Good error detection.
- Checking the code correctly according to grammar.

PART–B (13 X 2 = 26 Marks)

6. (a) Define Finite automata and explain on the types of finite automata and convert the following NFA to DFA 13 CO1 APP



Now, Deterministic Finite Automata (DFA) may be drawn as-

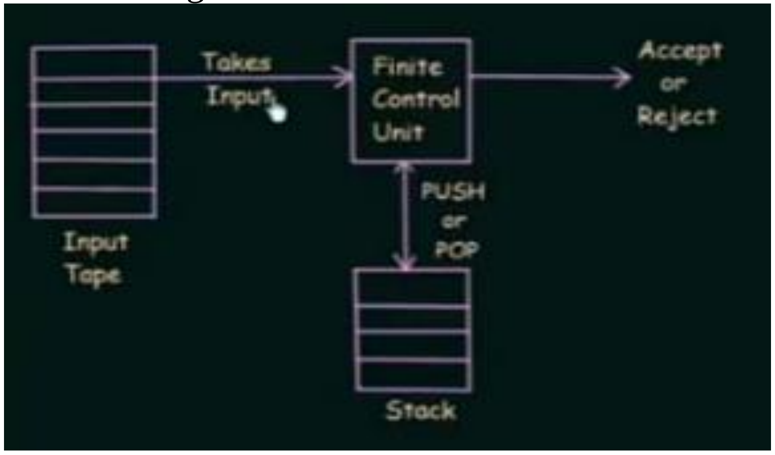


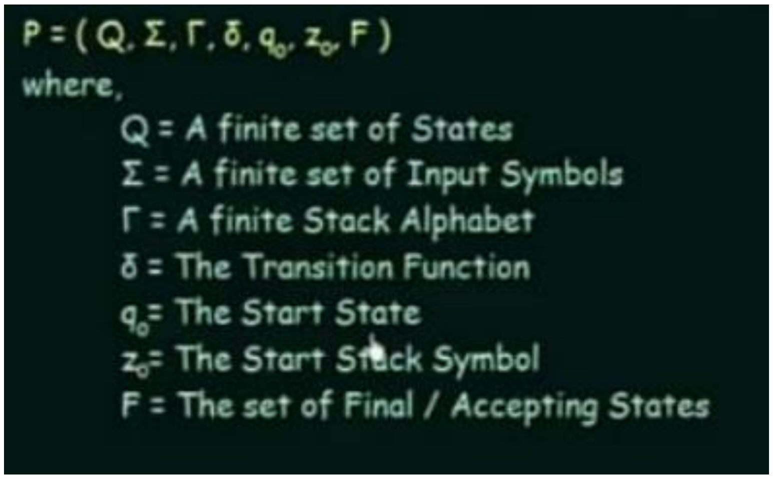
Deterministic Finite Automata (DFA)

(or)

(b) Explain how Pushdown Automata is more powerful than Finite automata with its formal and graphical representation. Construct the Pushdown Automata for Language L = {0n1n|n>=0} 13 CO1 ANA

- FSA
  - not applicable for all domains
  - Limited Memory
- PDA
  - FSA + Stack
  - Applications
    - Calculator
    - Java / C Program




$P = (Q, \Sigma, \Gamma, \delta, q_0, z_0, F)$

where,

Q = A finite set of States

Σ = A finite set of Input Symbols

Γ = A finite Stack Alphabet

δ = The Transition Function

q_0= The Start State

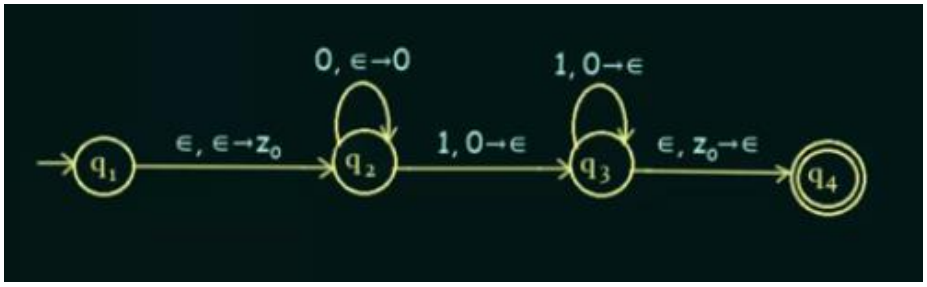
z_0= The Start Stack Symbol

F = The set of Final / Accepting States

Transition:

a,b->c

- a Input Symbol
  - ε empty
- b Top of the Stack which is to be popped
  - ε the stack is neither read nor popped
- c Symbol to be pushed into Stack
  - ε No symbols are pushed



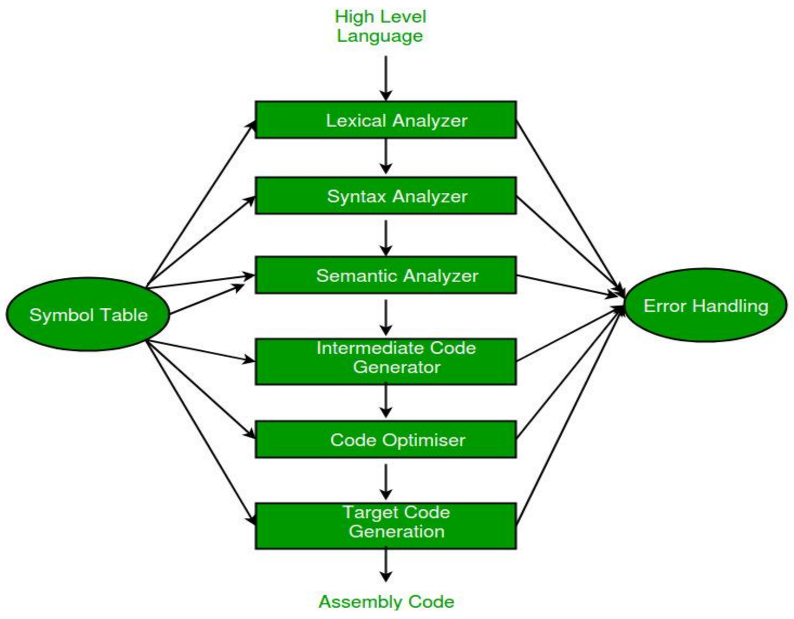
7. (a) Elaborate the various phases of compiler and trace it with the program segment (a=b+c* 50) 13 CO2 APP

We basically have two phases of compilers, namely the

1. Analysis phase
2. Synthesis phase.

The analysis phase creates an intermediate representation from the given source code.

The synthesis phase creates an equivalent target program from the intermediate representation.



Symbol Table – It is a data structure being used and maintained by the compiler, consisting of all the identifier's names along with their types. It helps the compiler to function smoothly by finding the identifiers quickly.

The analysis of a source program is divided into mainly three phases. They are:

Linear Analysis-

Hierarchical Analysis-

Semantic Analysis-

Lexical Analyzer –

It is also called a scanner. It takes the output of the preprocessor (which performs file inclusion and macro expansion) as the input which is in a pure high-level language. It reads the characters from the source program and groups them into lexemes (sequence of characters that "go together"). Each lexeme corresponds to a token. Tokens are defined by regular expressions which are understood by the lexical analyzer. It also removes lexical errors (e.g., erroneous characters), comments, and white space.

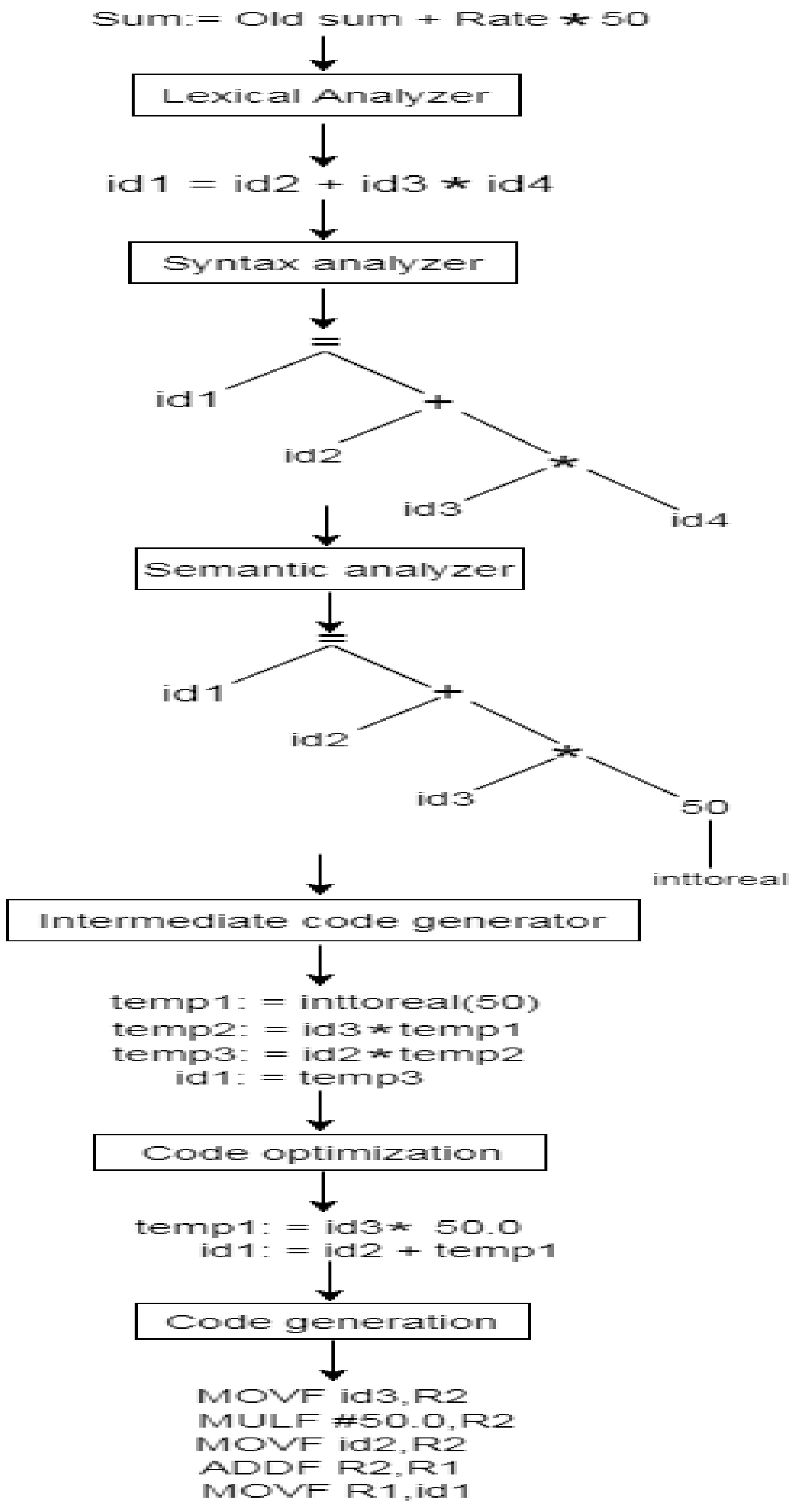
1. Syntax Analyzer – It is sometimes called a parser. It constructs the parse tree. It takes all the tokens one by one and uses Context-Free Grammar to construct the parse tree.
Why Grammar?
The rules of programming can be entirely represented in a few productions. Using these productions we can represent what the program actually is. The input has to be checked whether it is in the desired format or not.
The parse tree is also called the derivation tree. Parse trees are generally constructed to check for ambiguity in the given grammar. There are certain rules associated with the derivation tree.

2. Semantic Analyzer – It verifies the parse tree, whether it's meaningful or not. It furthermore produces a verified parse tree. It also does type checking, Label checking, and Flow control checking.

- Intermediate Code Generator – It generates intermediate code, which is a form that can be readily executed by a machine We have many popular intermediate codes. Example – Three address codes etc. Intermediate code is converted to machine language using the last two phases which are platform dependent.
Till intermediate code, it is the same for every compiler out there, but after that, it depends on the platform. To build a new compiler we don't need to build it from scratch. We can take

the intermediate code from the already existing compiler and build the last two parts.

- Code Optimizer – It transforms the code so that it consumes fewer resources and produces more speed. The meaning of the code being transformed is not altered. Optimization can be categorized into two types: machine-dependent and machine-independent.
- Target Code Generator – The main purpose of the Target Code generator is to write a code that the machine can understand and also register allocation, instruction selection, etc. The output is dependent on the type of assembler. This is the final stage of compilation. The optimized code is converted into relocatable machine code which then forms the input to the linker and loader.



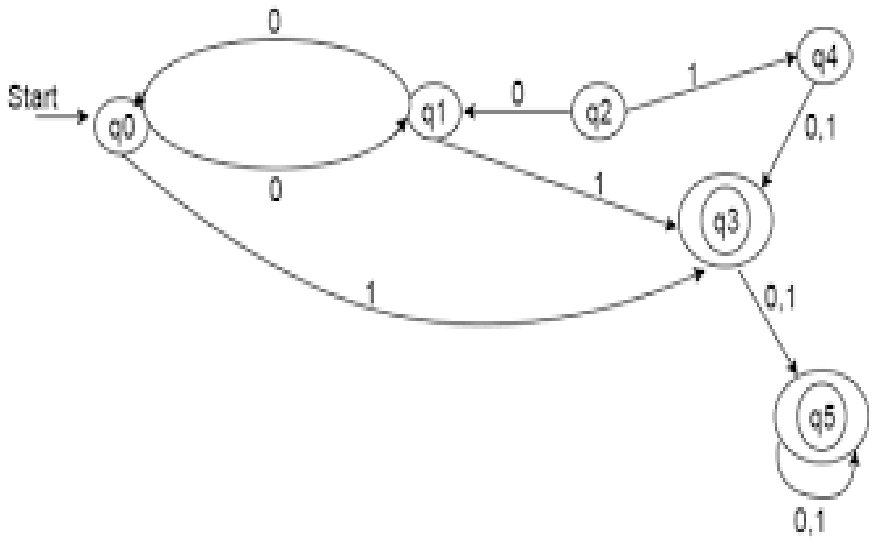
```
temp1: = inttoreal(50)
temp2: = id3*temp1
temp3: = id2*temp2
    id1: = temp3
```

```
temp1: = id3*  50.0
    id1: = id2 + temp1
```

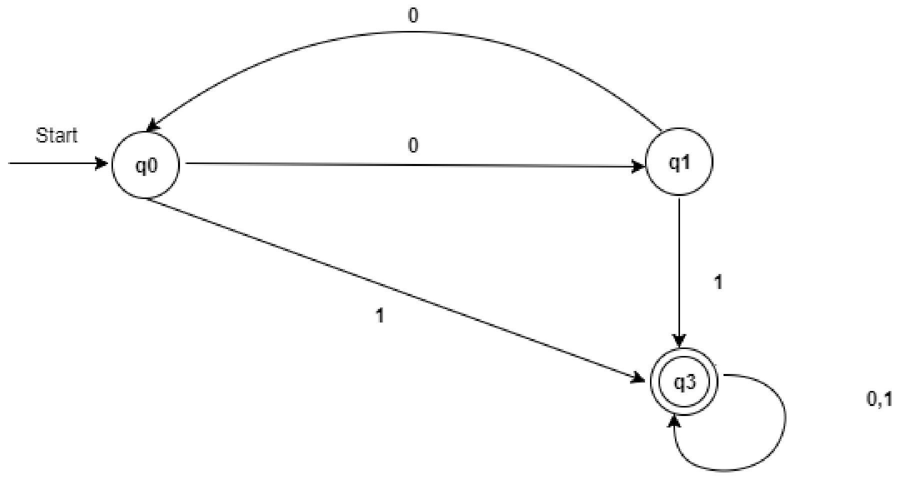
```
MOVF id3,R2
MULF #50.0,R2
MOVF id2,R2
ADDF R2,R1
MOVF R1,id1
```

(or)

(b) Find the minimized DFA for the given DFA 13 CO1 ANA



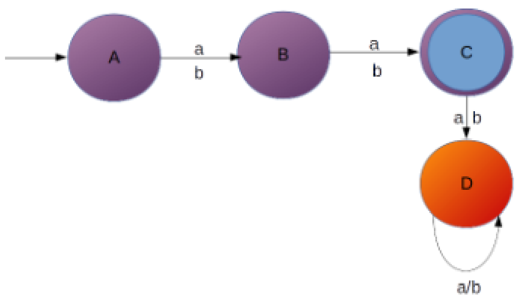
Solution:



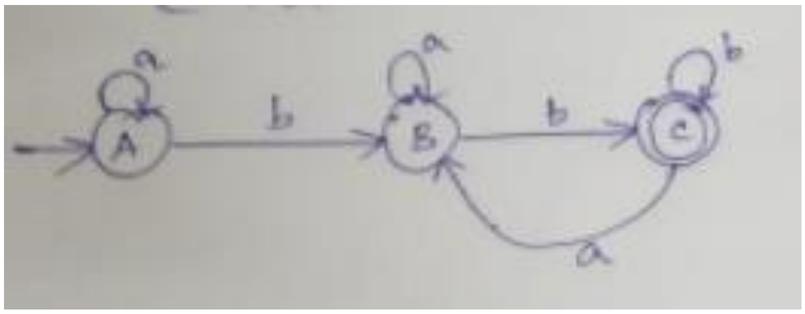
PART-C (14 x 1 = 14 Marks)

8. (a) Construct the Regular Expression, DFA which accepts a string over {0,1} / {a,b} CO1 CRE

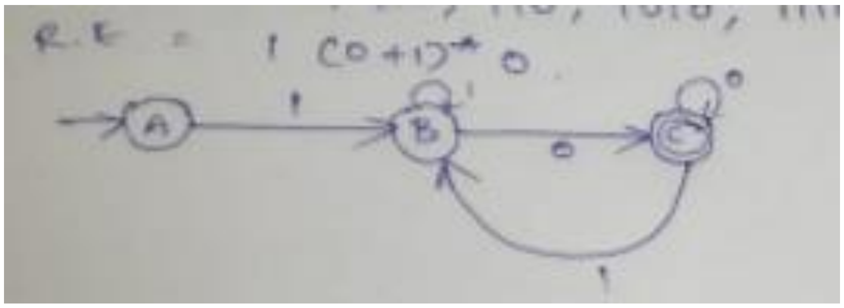
i)which is of length 2



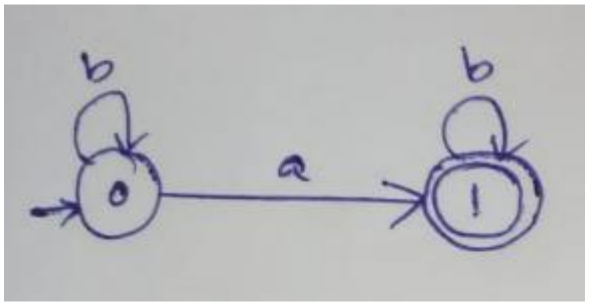
ii)set of strings that ends with "bb"



iii)start with "0" and ends with "1"



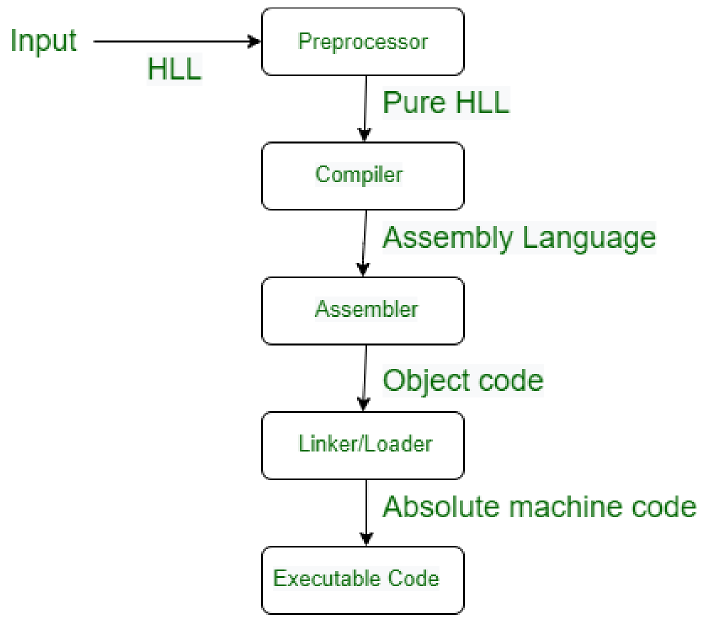
iv)exactly 1 a



(or)

(b) Outline about the following: CO2 UND

(i) Cousins of the compiler



(ii) Tools available for various phases of the compiler
The compiler writer can use some specialized tools that help in implementing various phases of a compiler. These tools assist in the creation of an entire compiler or its parts. Some commonly used compiler construction tools include:

1. Parser Generator –
It produces syntax analyzers (parsers) from the input that is based on a grammatical description of programming language or on a context-free grammar. It is useful as the syntax analysis phase is highly complex and consumes more manual and compilation time. Example: PIC, EQM
2. Scanner Generator –
It generates lexical analyzers from the input that consists of regular expression description based on tokens of a language. It generates a finite automaton to recognize the regular expression.
Example: Lex
3. Syntax directed translation engines –
It generates intermediate code with three address format from the input that consists of a parse tree. These engines have routines to traverse the parse tree

and then produces the intermediate code. In this, each node of the parse tree is associated with one or more translations.

4. Automatic code generators – It generates the machine language for a target machine. Each operation of the intermediate language is translated using a collection of rules and then is taken as an input by the code generator. A template matching process is used. An intermediate language statement is replaced by its equivalent machine language statement using templates.

5. Data-flow analysis engines – It is used in code optimization.Data flow analysis is a key part of the code optimization that gathers the information, that is the values that flow from one part of a program to another. Refer – data flow analysis in Compiler

6. Compiler construction toolkits – It provides an integrated set of routines that aids in building compiler components or in the construction of various phases of compiler.

****

(Note: UND-Understand REM-Remember ANA-Analyze APP-Apply CRE-Create)

Prepared By **Verified By** **HoD**