COSC 340: THEORY OF COMPUTATION REVISION

CHUKA UNIVERSITY PAST PAPERS

COMPUTER SCIENCE

PAPER 1

QUESTION ONE [30 MARKS]

a) Explain the purpose of the Theory of Computation [2 Marks]
The purpose of the Theory of Computation is to understand the fundamental principles and capabilities of computers. It explores concepts such as algorithms, formal languages, automata theory, and the limits of computation, providing a theoretical foundation for comprehending what computers can and cannot do.

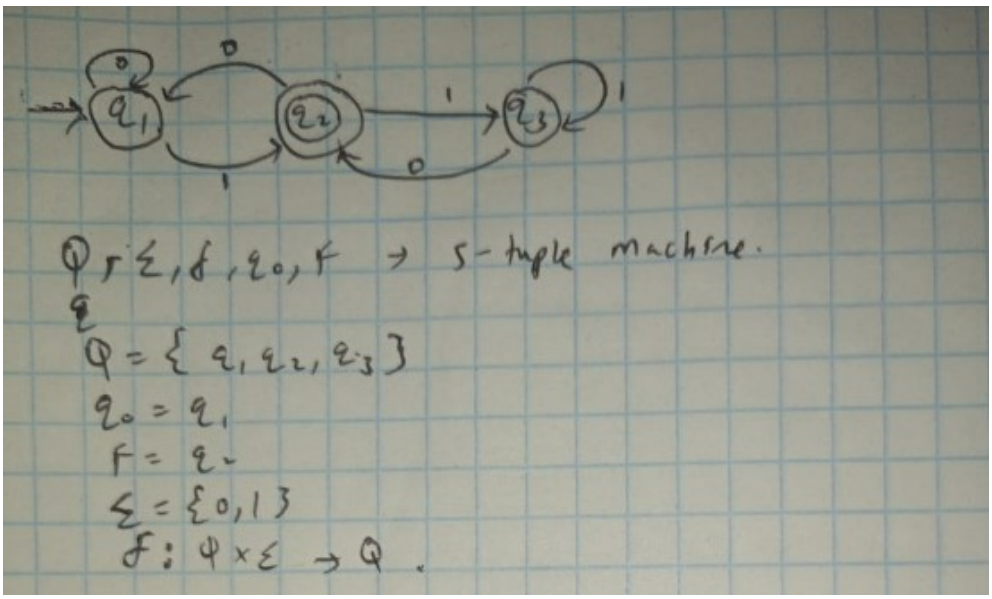
b) Briefly describe each of the three areas of the theory of computation and explain the benefits of each to the field of computing [6 Marks]
Automata Theory: This area focuses on abstract machines and their computational abilities, such as Finite Automata and Turing Machines. The benefit of computing lies in understanding the theoretical basis for designing efficient algorithms and recognizing the limits of computation.

Formal Languages: This field deals with the study of languages and grammar, offering a way to represent and describe the syntax of programming languages and other formal systems. The benefit is in developing formal methods for language specification, aiding in the design and analysis of programming languages.

Computability Theory: This area investigates the concept of computability and decidability, exploring what problems can be solved algorithmically and what cannot. The benefit is in understanding the theoretical limits of computation, contributing to the development of algorithms that solve practical problems

c) Using your knowledge from the theory of computation explain what an algorithm is [2 Marks]
An algorithm is a step-by-step procedure or set of rules designed to perform a specific task or solve a particular problem. It's a sequence of well-defined instructions that, when executed, produces the desired output or result.

d) Draw and define an example Deterministic Finite Automaton [6 Marks] (diagram needed)



A DFA consists of a finite set of states, a set of input symbols, a transition function, a start state, and a set of accepting states. The DFA reads input symbols and transitions between states based on the transitions defined in the transition function. It accepts a string if it reaches an accepting state after processing the entire input.

e) Explain any three areas where Push Down Automaton are used [6 Marks]

Parsing: PDAs are employed in the syntax analysis phase of compilers to parse and analyze the structure of programming languages.

Natural Language Processing: PDAs can model certain aspects of natural language processing, particularly in handling nested structures within sentences.

Database Systems: PDAs are used in database systems to validate and manipulate nested structures, such as nested queries.

Programming languages: PDAs are used to build compilers and interpreters for programming languages.

Syntax analysis in compilers: PDAs are employed in the front-end phase of a compiler to check the syntax of a program written in a high-level programming language.

Automated theorem proving: PDAs play a crucial role in automated theorem proving systems, which are used in logic and mathematics to prove theorems and mathematical statements.

f) Giving examples define the three regular expressions on languages [6 Marks]

1. Regular expression: a*b
   Language: L = {b, ab, aab, aaab, aaaab, ...}
   This regular expression represents all strings that start with zero or more 'a's followed by a 'b'. Examples of strings in this language are: b, ab, aab, aaab, aaaab, etc.

2. Regular expression: (01)*
   Language: L = {ε, 01, 0101, 010101, ...}

This regular expression represents all strings that can be formed by repeating the pattern '01'. Examples of strings in this language are: ε (empty string), 01, 0101, 010101, etc.

3. Regular expression: (0+1)*0
Language: L = {0, 1, 00, 01, 10, 11, 000, 001, ...}
This regular expression represents all strings that can be formed by any combination of 0s and 1s, ending with a 0. Examples of strings in this language are: 0, 1, 00, 01, 10, 11, 000, 001, etc.

4. Regular expression: (ab+c)*
Language: L = {ε, abc, aabc, abbc, abbbc, ...}
This regular expression represents all strings that can be formed by repeating the pattern 'abc' zero or more times. Examples of strings in this language are: ε (empty string), abc, aabc, abbc, abbbc, etc.

g) Identify any two differences between the input alphabet and the tape alphabet of a deterministic Turing Machine [2 Marks]

1. The input alphabet of a deterministic Turing Machine consists of symbols that can be read from the input tape, while the tape alphabet includes symbols that can be written on the tape.

2. The input alphabet is finite and fixed for a particular Turing Machine, whereas the tape alphabet can be infinite or variable, depending on the specific Turing Machine.

3. The input alphabet typically includes special symbols such as a blank symbol to mark the end of the input, whereas the tape alphabet can have additional symbols that are used for specific purposes such as indicating the current position of the tape head.

QUESTION TWO [20 MARKS]

a) Consider the following substitution rules of a Context Free Grammar
S → AB
A → a
A → aA
B → b
B → bB

b) Use the above to identify:
i. Terminal [2 Marks]
Terminals are symbols that do not have any production rules. In this context-free grammar, the terminals are 'a' and 'b'.
ii. Start variable [1 Mark]
The start variable is the variable from which the derivation of the language begins. In this case, the start variable is 'S'.
iii. Variables [2 Marks]
Variables are symbols that can be replaced by other symbols according to the production rules. In this grammar, the variables are 'S', 'A', and 'B'.

c) Use the rules of the grammar in a) above to derive any three strings of the format {a, b} *
[3 Marks]

The rules of the grammar are:

S → AB

A → a

A → aA

B → b

B → bB

Using these rules, we can derive strings of the format {a, b}*:

i. $S \Rightarrow AB \Rightarrow aB \Rightarrow ab$

ii. $S \Rightarrow AB \Rightarrow aB \Rightarrow abB \Rightarrow abb$

iii. $S \Rightarrow AB \Rightarrow aB \Rightarrow abB \Rightarrow abbB \Rightarrow abbb$

Strings derived:

ab

abb

abbb

d) Generate a parse tree to derive each of the strings in b) above [3 Marks]

i. Parse tree for

ab:

```
  S
 / \
A   B
|   |
a   b
```

ii. Parse tree for

abb:

```
  S
 / \
A   B
|   |
a   B
    |
    b
```

iii. Parse tree for

abbb:

S

**/ **

A B

| |

a B

|

B

|

b

e) Let n be a positive integer. If n2 is even, then n is even.

a. Prove the theorem using proof by contradiction [5 Marks]

i. **Proof by Contradiction:**

Assume n is a positive integer, and n^2 is even, but n is odd. Then, n can be expressed as $n = 2k + 1$ for some integer k. Substituting this into n^2, we get $n^2 = (2k + 1)^2 = 4k^2 + 4k + 1$, which is an odd number. However, we assumed n^2 is even, leading to a contradiction. Therefore, our assumption that n is odd must be false, so n is even.

b. Prove the theorem using proof by induction [4 Marks]

ii. **Proof by Induction:**

Base case: For $n = 1, n^2 = 1$, which is even.

Inductive step: Assume that for some positive integer m, if m^2 is even, then m is even.

Now, consider $(m + 1)^2 = m^2 + 2m + 1$. Since m^2 is even by the inductive assumption, $2m$ is even. Adding an even number to an even number gives an even result. Therefore, $(m + 1)^2$ is even.

QUESTION THREE [20 MARKS]

a) You are given the following language

L = {0n 1 n | n>=0}

i. Describe this language [2 Marks]

The language $L=\{0^n1^n|n\geq0\}$ consists of strings where the number of '0's is equal to the number of '1's, and both can appear in any order. In other words, the language contains strings with a balanced number of '0's and '1's, including the possibility of having no '0's or '1's.

ii. Give examples of any two strings of this language [2 Marks]

ii. **Examples of two strings in this language:**

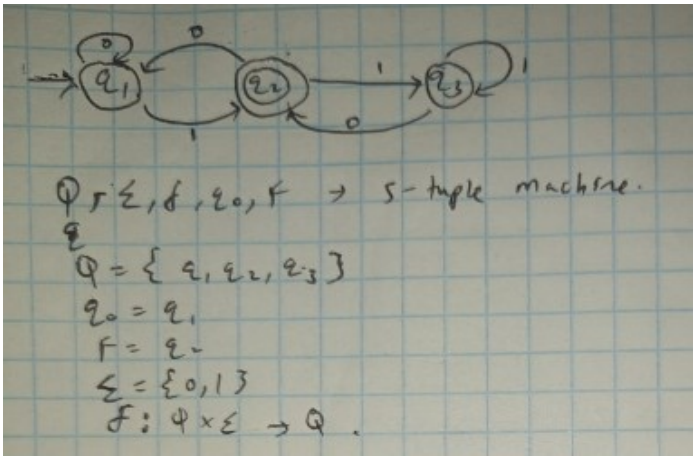
- $n = 0$: The string is ε (the empty string).
- $n = 2$: The string is 0011.

iii. Explain if the language is regular or irregular. Justify your answer [2 Marks]

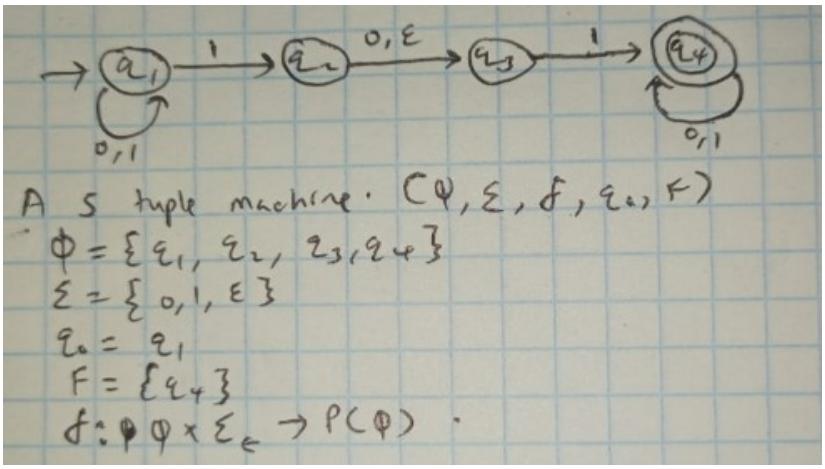
The language $L=\{0_n1_n|n\geq 0\}$ is not a regular language. Regular languages can be recognized by finite automata, but this language involves comparing the counts of two different symbols ('0' and '1'), which goes beyond the capabilities of regular languages. It requires a higher level of computational power, typically associated with context-free languages.

b) Using appropriate diagrams to differentiate between Deterministic Finite Automaton and Non-Deterministic Finite Automaton [5 Marks]

Deterministic Finite Automaton (DFA):



Non-Deterministic Finite Automaton (NFA):



In a DFA, each transition is uniquely determined by the current state and the input symbol, leading to a single next state. In contrast, an NFA can have multiple transitions for a given state and input symbol, allowing for more flexibility in recognizing languages.

c) Making reference to Computer storage highlights the different computational models [4 Marks]

Computational Models and Computer Storage:

Turing Machines: Have an infinite tape for storage, symbolized as a theoretically infinite memory.

Finite Automata: Have a finite set of states and use transitions to move between them. Storage is limited to the current state.

Random-Access Machines (RAM): Have a finite set of registers, each with its address, providing direct access to any register for storage. This model is more reflective of modern computer architecture.

The choice of a computational model affects the types of problems that can be efficiently solved and influences the theoretical understanding of computation.

QUESTION FOUR [20 MARKS]

a) Discuss the pigeonhole principle in the theory of computation [5 Marks]

The Pigeonhole Principle is a fundamental concept in combinatorics that has implications in the theory of computation. It states that if you want to distribute 1n+1 items into n containers, at least one container must contain more than one item. In the context of computation, this principle is often applied to analyze algorithms and data structures.

Example: If you have n pigeonholes and 1n−1 pigeons, and each pigeon must be placed in a pigeonhole, at least one pigeonhole will have more than one pigeon.

Application in Computation: In algorithms, the Pigeonhole Principle can be used to analyze time and space complexity. For example, when sorting n elements into m buckets, the principle implies that at least one bucket must contain more than one element.

b) Assume we have two regular languages L (A) = {0, 1} and L (B) = {010,001,110}. Show the results of the regular operations below in the two languages:

Given:

$$L(A) = \{0, 1\}$$
$$L(B) = \{010, 001, 110\}$$

i. Conjunction of Language L(A) and Language L(B) [3 Marks]

i. **Conjunction (Intersection) of Language $L(A)$ and $L(B)$:**
$L(A) \cap L(B) = \{\}$ (Empty set, as there are no common strings between the two languages.)

ii. Star of Language L(B) [3 Marks]

ii. **Star (Kleene Closure) of Language $L(B)$:**
$L(B)^* = \{\epsilon, 010, 001, 110, 010010, 001001, 110110, \ldots\}$ (Includes all possible concatenations and repetitions of strings in $L(B)$, including the empty string.)

iii. Union of Language L(A) to L(B) [3 Marks]

iii. **Union of Language $L(A)$ and $L(B)$:**

$L(A) \cup L(B) = \{0, 1, 010, 001, 110\}$ (All distinct strings from both languages.)

c) Describe any three types of Turing Machines [6 Marks]

i. Deterministic Turing Machine (DTM):

- Has a single, unique next state for each combination of current state and input symbol.

- Follows a deterministic set of rules for transitions.

- Simulates deterministic computation.

ii. Non-Deterministic Turing Machine (NDTM):

- Can have multiple possible next states for a given combination of current state and input symbol.

- Allows for non-deterministic choices during computation.

- Used for theoretical analysis and understanding the power of non-determinism.

iii. Universal Turing Machine (UTM):

- A special Turing machine that can simulate the behavior of any other Turing machine.

- Takes as input the description of another Turing machine and its input.

- Demonstrates the universality of the Turing machine model and forms the basis for the Church-Turing thesis.

iv. Probabilistic Turing Machine (PTM):

A probabilistic Turing machine is designed to incorporate randomness in its computation. It uses transition probabilities to make decisions instead of deterministic rules. At each step, it can choose among multiple possible actions based on assigned probabilities. PTMs are employed in areas like cryptography, machine learning, and artificial intelligence. By introducing randomness, PTMs can model uncertain scenarios and perform tasks such as random sampling and stochastically optimizing algorithms.

QUESTION FIVE [20 MARKS]

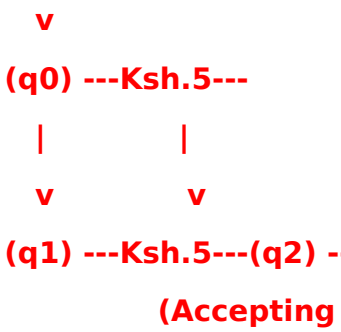
a) A toll gate to a park opens after a person inserts a coin denomination of Ksh. 20 or more. Assuming that a certain visitor inserts three-coin denominations in a sequence of Ksh. 5, Ksh. 5, and Ksh. 10 and assuming that no change is given back:

i. Draw the diagram of the DFA machine [5 Marks]

Here is a simple diagram for the DFA (Deterministic Finite Automaton) that models the toll gate scenario:

[Start]

|



The states are represented by q0, q1, and q2. The transitions are labeled with the coin denominations, and the accepting state is q2.

ii. Formally define the machine [5 Marks]

- Set of States: $Q = \{q0, q1, q2\}$
- Alphabet: $\Sigma = \{Ksh.5, Ksh.10\}$
- Transition Function: $\delta : Q \times \Sigma \to Q$
  - $\delta(q0, Ksh.5) = q1$
  - $\delta(q1, Ksh.5) = q2$
  - $\delta(q2, Ksh.10) = q2$
- Start State: $q0$
- Accepting State: $q2$

b) Making reference to the concept of algorithms and running time of algorithms, discuss the Theory of Complexity, a branch of the Theory of Computation [6 Marks]

B) Theory of Complexity:

The Theory of Complexity is a branch of the Theory of Computation that focuses on analyzing the efficiency of algorithms. It deals with understanding the resources (such as time and space) required by an algorithm to solve a particular problem. Here are key points related to the Theory of Complexity:

Algorithms:

An algorithm is a step-by-step procedure or set of rules for solving a specific problem.

Complexity theory aims to classify algorithms based on their efficiency and resource requirements.

Running Time of Algorithms:

The running time of an algorithm is the amount of time it takes to execute as a function of the input size.

It is often expressed using Big O notation, providing an upper bound on the growth rate of the running time.

Complexity Classes:

Complexity classes categorize problems based on their inherent difficulty and the resources required to solve them.

Classes like P (Polynomial Time), NP (Non-deterministic Polynomial Time), and EXP (Exponential Time) are examples.

P vs. NP Problem:

The P vs. NP problem is one of the most famous open problems in computer science and complexity theory.

It asks whether every problem that can be verified quickly (in polynomial time) can also be solved quickly (in polynomial time).

c) Let Σ be an alphabet and let A∈ Σ* be a language over the alphabet. Language A is decidable. Explain why Language A is said to be decidable [4 Marks]

c) Decidable Language:

If a language A is decidable, it means that there exists a Turing machine that can determine whether any given input string belongs to the language A or not. In other words, the machine will halt and accept if the input is in A, and it will halt and reject if the input is not in A.

Explanation:

Halting: The Turing machine must always halt after a finite number of steps.

Accept/Reject: After halting, the machine should enter an accepting state if the input string is in the language A, and a rejecting state if it is not.

Decision Procedure: There is a well-defined procedure that the Turing machine follows to make the decision, ensuring determinism.

In summary, a language is decidable if there exists a Turing machine that decides membership in that language, providing a clear and unambiguous answer for any input string.

PAPER 2

QUESTION ONE [30 MARKS]

a) Identify and explain any three areas in Computer Science that benefit from Context-Free Grammar [6 Marks]

1. Programming Languages: Context-free grammars play a vital role in defining the syntax of programming languages. They provide a set of production rules that define the valid structure and arrangement of program elements. For example, in languages like C, Java, and Python, context-free grammars are used to specify the rules for defining variables, functions, and control structures like loops and conditionals.

2. Compiler Design: Context-free grammars are essential in designing parsers and compilers. They are used to formally define the syntax of programming languages, allowing parsers to recognize and analyze the input source code. Context-free grammars facilitate the generation of abstract syntax trees, which represent the hierarchical structure of a

program and serve as an intermediate representation for subsequent compilation phases.

3. Natural Language Processing: Context-free grammars are employed to model the syntactic structure of natural languages. They allow for the representation of grammar rules that determine the arrangement and combination of words in a sentence. By using context-free grammars, it becomes easier to develop language parsers and generators capable of understanding and generating coherent sentences.

4. Syntax Analysis and Parsing: Context-free grammars are used extensively in syntax analysis and parsing algorithms. Parser generators like Yacc and Bison generate parsers based on a given context-free grammar, which then perform syntax analysis to verify if a given sequence of tokens conform to the grammar rules. Context-free grammars enable efficient parsing algorithms such as LR, LL, and CYK, which are widely used in compiler construction and syntax checking.

5. Natural Language Processing Applications: Context-free grammars are employed in various natural language processing applications such as part-of-speech tagging, named entity recognition, and syntactic parsing. By defining grammar rules using context-free grammars, it becomes easier to extract meaningful information from natural language text and perform tasks like sentiment analysis, question answering, and machine translation.

b) Describe the seven components that are used to formally describe a Turing Machine [3 marks]

1. Input Alphabet: This component consists of a set of symbols that are allowed as input to the Turing Machine

2. Tape Alphabet: This component is a finite set of symbols that can be written on the tape. It includes the input symbols as well as additional symbols such as blanks or separators.

3. State Set: This component defines the finite set of states that the Turing Machine can be in.

4. Transition Function: This component describes how the Turing Machine moves from one state to another based on the current state and the tape symbol it encounters.

5. Initial State: This component designates the initial state of the Turing Machine

6. Accepting States: This component specifies a subset of states from the state set that are considered accepting states. If the Turing Machine enters one of these states, it indicates that the input has been accepted.

7. Rejecting States: This component designates a subset of states from the state set that are considered rejecting states. If the Turing Machine enters one of these states, it indicates that the input has been rejected.

c) Giving an example for each differentiate between a set and a tuple [4 Marks]

A set is an unordered collection of unique elements, while a tuple is an ordered collection of elements, which may be duplicated.
Example for a set: Set A = {1, 2, 3, 4}
Example for a tuple: Tuple B = (1, 2, 2, 3, 4)

d) Present the symbol of the Start and Final States of DFAs. Explain the role of the start and final states of a DFA [6 Marks]

Start and Final States of DFAs:
Start State Symbol: Usually denoted by an arrow pointing towards the initial state.

Final State Symbol: Typically represented by a double circle or an additional state with a double circle indicating an accepting state.

Role of Start and Final States:
Start State: Represents the beginning of the computation.
Final State (Accepting State): Indicates successful completion of the computation or acceptance of the input string.

e) Discuss the Church-Turing thesis [5 Marks]

The Church-Turing thesis is a hypothesis in the theory of computation that suggests that any function that is algorithmically computable can be computed by a Turing machine (or an equivalent formalism). In other words:

Statement: Anything computable is computable by a Turing machine.
Equivalent Statement: A function is algorithmically computable if and only if it is computable by a Turing machine.
Discussion:

The Church-Turing thesis is a foundational concept in theoretical computer science.
It asserts the universality of the Turing machine as a model of computation.
While the thesis is widely accepted, it remains a hypothesis, as its proof is based on intuitive reasoning rather than formal proof.

QUESTION TWO [20 MARKS]

a) Assume we have two regular languages L (A) = {boy, girl} and L (B) = {good, bad}. Show the results of the regular operations below in the two languages:

i. Conjunction of Language L(A) and Language L(B) [2 Marks]
To find the conjunction of two languages, we take all possible combinations of strings from both languages.

L(A) = {boy, girl}
L(B) = {good, bad}

Conjunction of L(A) and L(B):
{boy, boygood, boybad, girl, girlgood, girlbad}

ii. Star of Language L(B) [2 Marks]
To find the star of a language, we concatenate any number of strings (including zero times) from the language.

L(B) = {good, bad}

Star of L(B):
ε (empty string), good, bad, goodgood, goodbad, badgood, badbad, goodgoodgood, goodgoodbad, goodbadgood, goodbadbad, ...
(Note: ε represents the empty string)

b) Describe the relationship between a computer virus and the theory of computability [7 Marks]

1) Turing's theory of computability is based on the concept of a Turing machine, a hypothetical computational device that can simulate any algorithmic computation. Similarly, computer viruses operate by executing code within the host computer, manipulating its resources and behavior.

2) A computer virus can be seen as a program or algorithm that infects other computer systems, propagating itself and potentially causing damage. This can be seen as a parallel to the idea of Turing machines being able to solve any computable problem, as viruses manipulate systems to perform tasks that may not have been intended by the original user.

3) Computability theory focuses on what can be solved algorithmically, and whether certain problems have solutions or are undecidable. Similarly, computer viruses often exploit vulnerabilities in software or operating systems, highlighting the existence of undecidable problems in the realm of computer security.

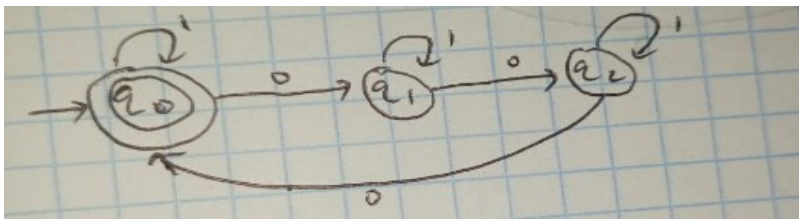
4) In computability theory, the halting problem is a fundamental concept that deals with determining whether a given program will eventually halt or continue running indefinitely. Computer viruses can leverage similar ideas, embedding code that may cause a system to enter an endless loop or exhibit unpredictable behavior.

5) Computability theory explores the limits of what can be computed, including problems that are impossible to solve. Similarly, computer viruses can exploit weaknesses in software, evading detection or removal, and highlighting that complete protection against all threats may be impossible to achieve.

6) Computability theory addresses the existence of undecidable problems, which are those for which no algorithm exists to provide a definite yes or no answer. Likewise, computer viruses often employ obfuscation techniques to make it difficult for anti-virus software to determine whether a file or program is malicious or benign.

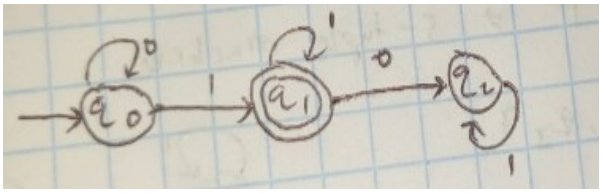
7) The impact of computer viruses on a computing system can be seen as equivalent to the impact of undecidable problems in computability theory. Both can lead to unpredictable behavior, potential loss of data or functionality, and overall disruption of normal operations.

c) For each of the following languages, construct a DFA that accepts the language. In all cases, the alphabet is {0, 1}.
   i. {w | the length of w is divisible by three} [3 Marks]



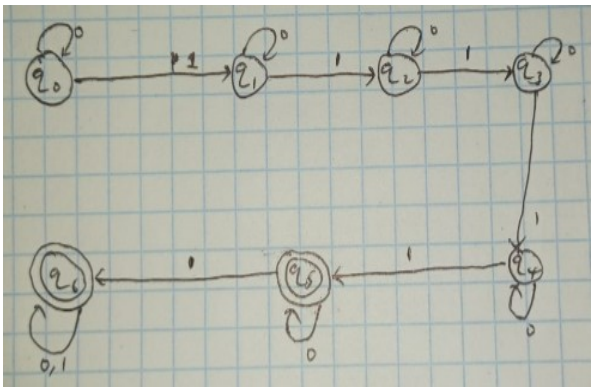
ii. {w | 110 is not a substring of w} [3 Marks]

	0	1
q0	q0	q1
q1	q2	q1
q2	q0	q2



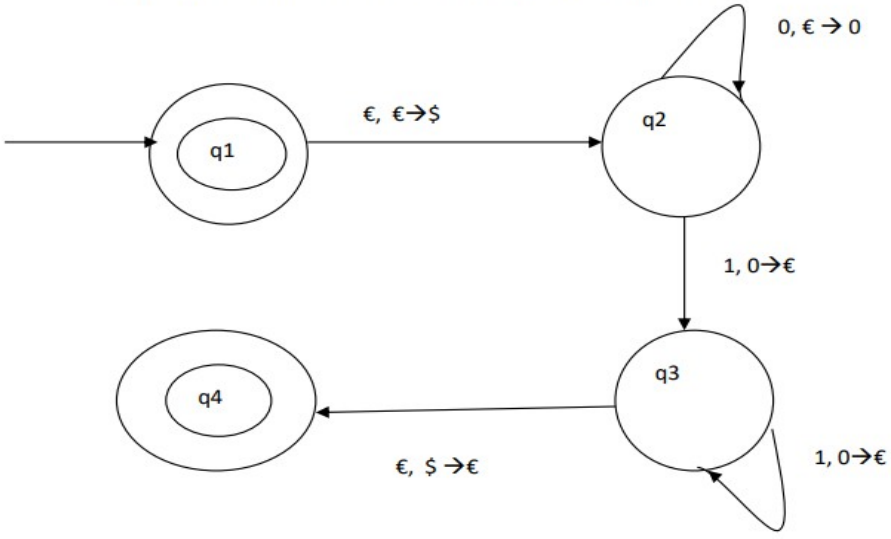
iii. {w | w contains at least five 1s} [3 Marks]

State	Input 0	Input 1
q0	q0	q1
q1	q1	q2
q2	q2	q3
q3	q3	q4
q4	q4	q5
q5	q5	q6
q6	q6	q6



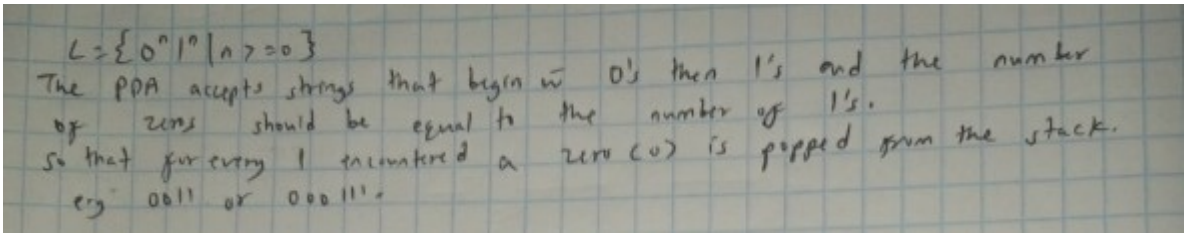
QUESTION THREE [20 MARKS]

a) A pushdown Automata PDA P is presented as follows:

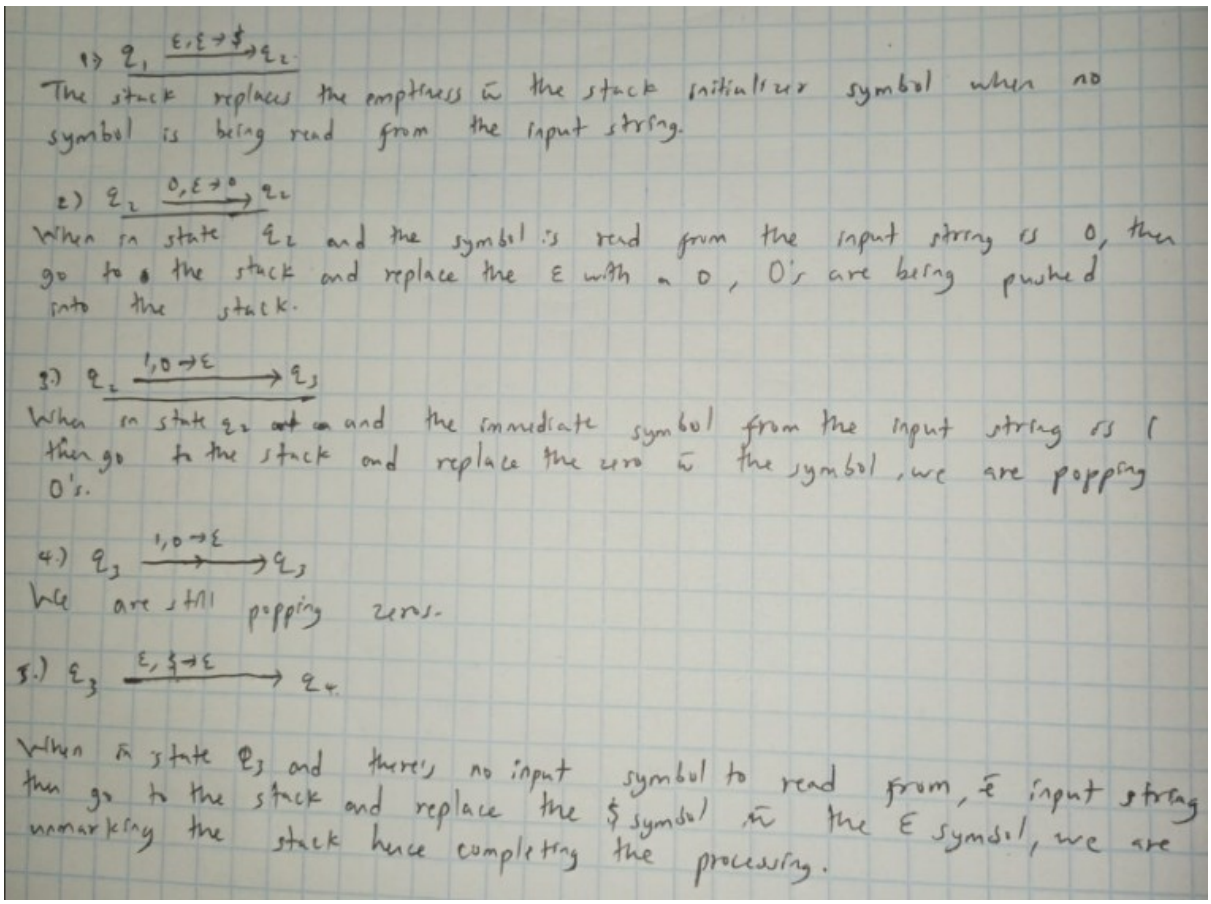


Making reference to the Push Down Automaton above:

b) Define the language accepted by PDA P [4 Marks]



c) Discuss the computation of PDA P [10 Marks]



d) Let B be the set of all infinite sequences over {0, 1}. Show that B is uncountable, using a proof by diagonalization. [6 Marks]

To prove that the set B of all infinite sequences over {0, 1} is uncountable using diagonalization, we will assume, for contradiction, that B is countable.

Assume B is countable, which means we can list all the sequences in B as S1, S2, S3, ... for some enumeration.
Each sequence in B can be written as S = (s1, s2, s3, ...), where s1, s2, s3, ... are the elements of {0, 1}.
Now, construct a new sequence D = (d1, d2, d3, ...) as follows:
- Choose d1 to be the complement of s1 in the first sequence S1.
- Choose d2 to be the complement of s2 in the second sequence S2.
- Choose d3 to be the complement of s3 in the third sequence S3.
- Continue this process for all positive integers, choosing di to be the complement of si in the sequence Si.

The sequence D we obtained differs from each sequence in B at every position, as d1 is different from s1, d2 is different from s2, and so on.

Therefore, the sequence D cannot be part of the enumeration of sequences in B, since it differs from every sequence in B at least at one position.

This contradicts our assumption that B is countable, as we have found a sequence, D, which does not appear in the enumeration.

Hence, B must be uncountable.
This completes the proof by diagonalization that the set B of all infinite sequences over {0, 1} is uncountable.

QUESTION FOUR [20 MARKS]

a) Explain the relationship between cryptography and the theory of complexity [4 Marks]

1. Security and Efficiency: Cryptography involves designing and analyzing algorithms that ensure secure communication and data protection. The theory of complexity provides tools to understand the computational resources required to break cryptographic schemes and evaluate their security. By analyzing the computational complexity of cryptographic algorithms, experts can assess their vulnerability to attacks and identify the level of protection they offer. Complexity theory helps ensure that cryptographic algorithms are efficient and provide adequate security within feasible computational limits.

2. One-way functions: One of the fundamental concepts in cryptography is the use of one-way functions, which are computationally easy to calculate in one direction but challenging to invert. Complexity theory studies the existence and properties of such functions, known as one-way functions or trapdoor functions. These functions form the foundation for various cryptographic primitives, such as public-key cryptography, digital signatures, and key exchange protocols. Complexity theory contributes to the understanding of the properties and limitations of one-way functions, aiding in the design and analysis of secure cryptographic systems.

3. Cryptographic assumptions: Cryptography relies on certain assumptions about computational hardness, such as the difficulty of factoring large numbers or computing discrete logarithms. These assumptions are deeply rooted in complexity theory, which investigates the computational complexity of various problems and the hardness of specific computational tasks. Complexity theory provides the basis for making these assumptions and assessing their validity, thereby ensuring the security of cryptographic protocols and schemes.

In summary, the relationship between cryptography and the theory of complexity encompasses security analysis, efficient algorithm design, the study of one-way functions, cryptographic assumptions, and the development of post-quantum cryptography. Complexity theory provides the essential tools and concepts to ensure the security and effectiveness of cryptographic systems.

b) Compare and contrast Push Down Automata to the following computation models:

i. DFA [2 Marks]

- A DFA is a computation model that recognizes regular languages.
- It is characterized by a finite set of states, an alphabet of input symbols, a transition function, a start state, and a set of accepting states.
- DFA operates by reading symbols from the input and transitioning between states based on the current state and the input symbol.
- It processes inputs deterministically, meaning that for every state and input symbol, there is exactly one transition to the next state.

- DFA cannot handle languages that require memory to recognize, such as those with context-free or context-sensitive properties.

ii. NFA [2 Marks]
- An NFA is a computation model that also recognizes regular languages.
- Like DFA, it has a set of states, an alphabet of input symbols, a transition function, a start state, and a set of accepting states.
- However, NFA differs from DFA in that it can have multiple transitions for a given state and input symbol combination.
- It allows the possibility of being in more than one state at a time and can transition to multiple states simultaneously.
- NFA can handle languages that require more complex recognition, including those with context-free or context-sensitive properties.

iii. Turing Machines [2 Marks]
- A Turing Machine consists of an infinite tape divided into discrete cells, a finite set of states, an alphabet of input symbols, a transition function, a start state, and a set of accepting states.
- It operates by reading symbols from the tape, transitioning between states based on the current state and the symbol under the machine's head, and modifying the tape by writing symbols to the current cell.
- Turing Machines are capable of solving any problem that can be solved algorithmically, making them a universal computational model.
- They have the ability to simulate any other computation model, including DFAs, NFAs, and even themselves.
- Turing Machines can handle languages with arbitrary complexity, including recursively enumerable languages.

b) Consider the context-free grammar G = (V, Σ, R, A), where V = {A, B}, Σ = {0, 1}, A is the start variable, and R consists of the rules

$A \rightarrow BAB|B|\in$

$B \rightarrow 00|\in$

Convert this grammar to Context-Free Grammar in Chomsky's Normal Form whose language is the same as that of G. [10 Marks]

To convert the given grammar G into Chomsky Normal Form (CNF), we need to eliminate the productions with more than two non-terminals and the productions with ε (empty string), and then replace the remaining productions with two non-terminals on the right-hand side.

The steps to convert G into CNF are as follows:

Step 1: Remove the ε-productions
The production $A \rightarrow \in$ is removed.
Step 2: Remove unit-productions
The production $B \rightarrow \in$ is removed.
Step 3: Introduce new variables for every terminal
Introduce new variables for each terminal in the grammar:
$S_0 \rightarrow A$
$A_0 \rightarrow 0$
$A_1 \rightarrow 1$
$B_0 \rightarrow 00$
Step 4: Eliminate productions with more than two non-terminals

In the given grammar, the only production with more than two non-terminals is A → BAB. To eliminate this, we'll introduce a new variable, say C:

C → AB

A → BC

B → A_0B_0

Step 5: Replace non-terminals with terminals (CNF form)

Substitute the single terminal symbols with their respective variables:

S_0 → A

A_0 → 0

A_1 → 1

B_0 → 00

C → AB

A → BC

B → A_0B_0

The resulting Chomsky Normal Form (CNF) grammar is:

S_0 → A

A_0 → 0

A_1 → 1

B_0 → 00

C → AB

A → BC

B → A_0B_0

This CNF grammar generates the same language as the original grammar G.

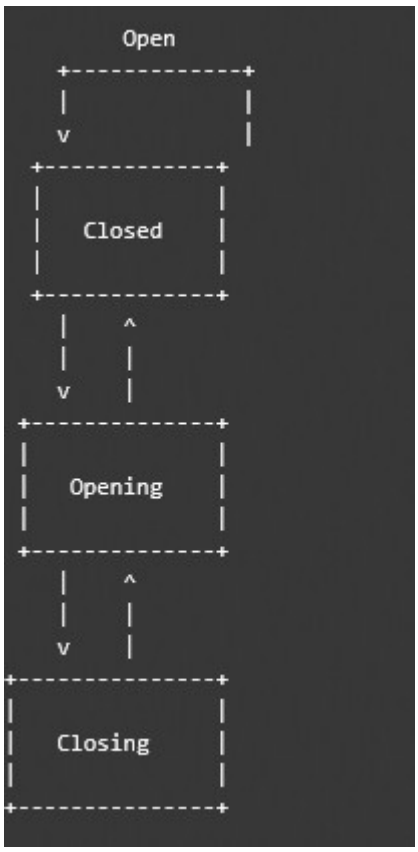
QUESTION FIVE [20 MARKS]

a) An Automatic door is one real-life implementation of Finite Automaton computational model.

i. Explain the workings of an Automatic door [2 Marks]

An automatic door works by using sensors to detect the presence of a person approaching the door. When someone is detected, the door opens automatically to allow them to pass through. The sensors may include infrared sensors, motion detectors, or pressure sensors. Once the person has passed through, the door will close automatically, either after a set amount of time or when another sensor detects that the person has moved away from the door.

ii. Present the State diagram of such an Automatic door [3 Marks]



iii. Formally define the State diagram of the Automatic door [5 Marks]

Let S be the set of states: S = {Closed, Opening, Open, Closing}
Let Σ be the set of input signals: Σ = {Open, Close, Opened}
Let δ be the transition function: δ: S × Σ → S

Note: The transition function δ represents the change of state based on the input signal received. For example, if the door is currently in the "Closed" state and an "Open" signal is received, it transitions to the "Opening" state.

b) Explain how you would apply knowledge in the following to your computing profession:

i. Regular Expressions [3 Marks]

Regular expressions are a powerful tool used to describe patterns in strings. In the computing profession, knowledge of regular expressions can be applied in various ways. For example, when developing software, regular expressions can be used to validate user input, such as email addresses, phone numbers, or passwords. They can also be used for data extraction from text files or for parsing and manipulating data.

ii. Finite Automaton [2 Marks]

A finite automaton is a mathematical model used to describe the behavior of systems that can be in a finite number of states. In the computing profession, knowledge of finite automaton theory can be applied in several areas. One application is in the design and implementation of compilers, where finite automata are used to recognize and analyze programming language syntax. They can also be used in network protocols, security systems, and pattern recognition tasks.

iii. Pumping Marks Lemma [2]

The Pumping Lemma is a fundamental concept in formal language theory that helps determine if a given language is regular. While it may not be directly applicable to everyday computing tasks, knowledge of the Pumping Lemma can be useful when analyzing and designing formal languages. When developing compilers or parsing algorithms, understanding the Pumping Lemma can help ensure the correct handling of regular languages.

iv. Kleene's theorem [3 Marks]
Kleene's theorem is a fundamental result in automata theory that relates regular expressions to finite automata. It states that for any regular expression, there exists an equivalent finite automaton and vice versa. In the computing profession, this theorem can be applied in various ways. It can be used to prove the equivalence of different representations of regular languages, such as regular expressions and finite automata. It also provides a theoretical basis for implementing regular expression engines used in text processing, searching, and pattern matching tasks.

PAPER 3

QUESTION ONE [30 MARKS]

a) Describe the features of a Turing Machine [3 marks]
A Turing Machine is a theoretical computing device that consists of the following features:

1. Tape: The machine has an infinite tape divided into discrete cells. Each cell can hold a symbol, which can be read, written, or erased by the machine.

2. Head: The machine has a movable head that scans the tape. The head can read the symbol on the current cell, write a symbol on the current cell, and move left or right along the tape.

3. State: The machine has a finite set of states, including an initial state and possibly multiple final states. The machine can transition from one state to another based on the current symbol read from the tape, the state it is currently in, and a set of predefined rules.

These features allow the Turing Machine to perform computations by manipulating symbols on the tape according to the specified rules. The theoretical model of a Turing Machine serves as a foundation for understanding the limits and capabilities of computation.

b) You are given the language {an b n |n>=1}
   i. Describe this language [2 Marks]
   The language {an b n | n ≥ 1} consists of all strings that start with one or more 'a's, followed by an equal number of 'b's. The number of 'a's and 'b's in each string is the same.
   ii. Is the language regular or irregular? [1 Mark]
   The language is irregular.
   iii. Justify your answer given in ii) above [3 Marks]
   To justify the answer, we can use the pumping lemma for regular languages. The pumping lemma states that if a language L is regular, then there exists a constant p (the pumping length)

such that all strings s in L with length at least p can be split into three parts: s = xyz, satisfying the following conditions:
1. For any i >= 0, the strings xyiz are also in L.
2. |y| > 0 (y is non-empty).
3. |xy| ≤ p.

iv. Identify and discuss the computational machine able to process the specified language [3 Marks]

The computational machine able to process this language is a Turing machine. A Turing machine is a theoretical computational model that can manipulate symbols on an infinite tape according to a set of rules. In this case, the Turing machine would scan the input string, making sure that the number of 'a's and 'b's are equal. If at any point the number of 'a's and 'b's does not match, the Turing machine would reject the string. If the number of 'a's and 'b's is equal throughout the entire string, the Turing machine would accept the string.

c) Giving an example for each differentiate between a set and a tuple [6 Marks]

A set is an unordered collection of unique elements, while a tuple is an ordered collection of elements, which may be duplicated.
Example for a set: Set A = {1, 2, 3, 4}
Example for a tuple: Tuple B = (1, 2, 2, 3, 4)

QUESTION FOUR [20 MARKS]

a) Differentiate between complexity classes P and NP. Discuss [6 Marks]

P: P is the class of decision problems that can be solved by a deterministic Turing machine in polynomial time. In other words, if a problem is in P, there exists an algorithm that can solve it efficiently in a time complexity bounded by a polynomial function of the input size.

NP: NP is the class of decision problems for which a solution can be checked/verified in polynomial time by a deterministic Turing machine. If a problem is in NP, there exists a "certificate" or "witness" that can be verified in polynomial time to confirm the correctness of a solution.

The key difference between P and NP is that P deals with the ability to find a solution efficiently, while NP deals with the ability to verify a solution efficiently. In other words, P focuses on finding an algorithm to solve a problem in polynomial time, while NP focuses on finding an algorithm to verify a solution in polynomial time.

QUESTION FIVE [20 MARKS]

a) Assume we have two regular languages L (A) = {boy, girl} and L (B) = {good, bad}. Show the results of the regular operations below in the two languages:

i. Conjunction of Language L(A) and Language L(B) [3 Marks]

The conjunction of L(A) and L(B) would be:
L(A) ∩ L(B) = {boygood, boybad, girlgood, girlbad}

ii. Star of Language L(B) [3 Marks]

The star of L(B) would be:
L(B)* = {ε, good, bad, goodgood, goodbad, badgood, badbad, goodgoodgood, ...}

iii. Union of Language L(A) to L(B) [3 Marks]

The union of L(A) and L(B) would be:

$L(A) \cup L(B) = \{boy, girl, good, bad\}$

b) Differentiate between Acceptable Languages and Recognizable Languages [4 marks]

1. Acceptable Languages: An acceptable language refers to a language that can be generated by a certain type of formal grammar or accepted by a specific type of automaton.An acceptable language may or may not halt for some inputs on a Turing machine, implying that it could be partially or entirely undecidable.

2. Recognizable Languages: A recognizable language is a language that can be accepted by a Turing machine. Recognizable languages may either halt and accept or run indefinitely for some inputs, but they cannot halt and reject.

c) Differentiate between Enumerators and Deciders as classes of Turing Machines [4 marks]

1. Enumerators: An enumerator is a specific class of Turing machines that generates the strings of a language in a step-by-step manner. It can list, or enumerate, all the strings that belong to a given language. These machines do not necessarily halt for non-member strings.

2. Deciders: Deciders, are a class of Turing machines that can determine whether an input string belongs to a particular language or not. These machines always halt and either accept or reject the input string in a finite amount of time.

d) List and explain any three areas where Context-Free Grammar is used [3 Marks]

1. Programming Languages: Context-free grammars are commonly used to define the syntax of programming languages. They provide a set of rules to generate valid statements and constructs in a programming language.

2. Natural Language Processing: Context-free grammars are utilized in various tasks of natural language processing, such as parsing, information extraction, and language generation.

3. Compiler Design: Context-free grammars play a critical role in developing the syntax analysis phase of a compiler. They are used to specify the grammar rules for the language being compiled.

4. Syntax Tree Generation: Context-free grammars help in generating parse trees or syntax trees, which represent the syntactic structure of sentences or programs. These trees are used in tasks like semantic analysis and code optimizations.

5. Formal Language Theory: Context-free grammars are a fundamental tool in formal language theory, where they are used to define and study various language classes, such as context-free languages, regular languages, etc.

PAPER 4

QUESTION ONE [30 MARKS]

a) Differentiate between regular and non-regular languages. [4 Marks]

Regular languages are a subset of formal languages that can be recognized by a finite automaton, which is a machine with a finite number

of states. Regular languages are characterized by the fact that they can be described using regular expressions.

On the other hand, non-regular languages are formal languages that cannot be recognized by a finite automaton. These languages may require more powerful computational models, such as pushdown automata or Turing machines, to be recognized.

b) Explain three differences between deterministic and non-deterministic finite automaton. [6 Marks]

1. Determinism vs. Non-determinism: A DFA is a machine where for each state and input symbol, there is exactly one transition defined. In contrast, an NFA can have multiple transitions defined for a state and input symbol combination, or it can even have epsilon (ε) transitions which are transitions that don't require any input.
2. Memory: DFAs do not have memory capabilities, meaning they can only recognize languages that can be described by a regular expression. NFAs, on the other hand, have memory capabilities in the form of multiple possible transitions or epsilon transitions. This allows them to recognize languages beyond the regular language class.
3. Acceptance: In DFAs, a string is accepted if it ends in a final state after being processed. In NFAs, a string is accepted if there is at least one path through the machine that ends in a final state after being processed.

c) Highlight any four types of Turing Machines. [4 Marks]

1. Standard Turing Machines.
2. Multi-Tape Turing
3. Non-deterministic Turing Machines
4. Universal Turing Machines.
5. Probabilistic Turing Machine

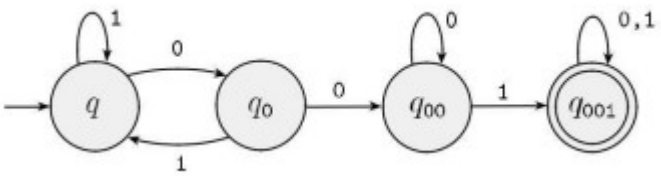
d) Given two strings such that string A is “Chuka” and string B is “University”, show the:

i. Union of string A and string B. [2 Marks]
"ChukaUniversity"

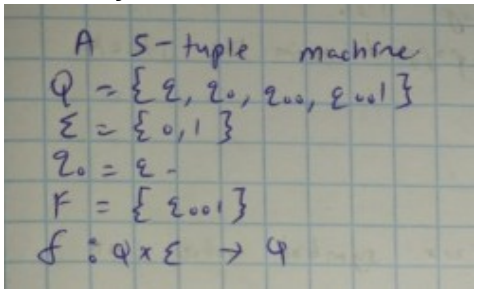
ii. The star of string A. [1 Mark]
"Chuka*"

iii. The concatenation of string B and string A. [2 Marks]
"UniversityChuka"

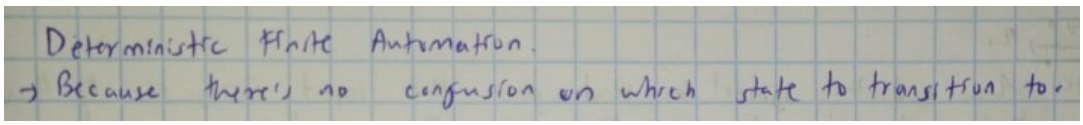
e) Given the following machine:



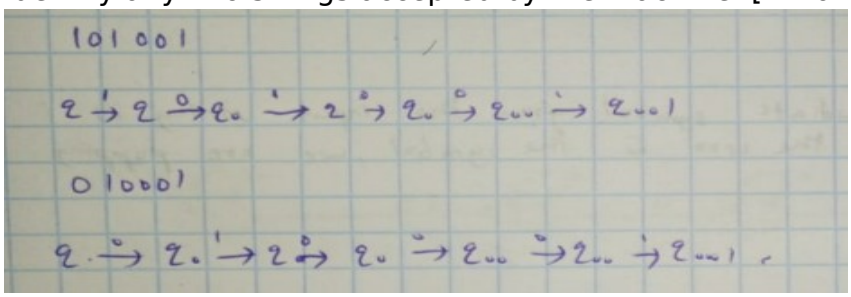
iv. Formally define the machine. [5 Marks]



v. Explain what type of machine it is. [2 Marks]



vi. Identify any two strings accepted by the machine. [2 Marks]



f) Describe the central question in the Theory of Complexity. [2 Marks]
The central question in the Theory of Complexity is how to measure and understand the inherent complexity of computational problems, such as the time and resources required to solve them. It aims to determine the inherent difficulty of problems and classify them into different complexity classes based on their computational complexity.

QUESTION TWO [20 MARKS]

a) A Turing machine can define either Turing acceptable or Turing recognizable languages. Demonstrating using a Turing Machine, differentiate between Turing Recognizable and Turing Decidable Languages. [8 Marks]
A Turing machine is a theoretical model of computation that operates on an infinite tape divided into cells. It consists of a control unit (a finite-state machine) and a head that can read, write, or move on the tape.

Turing Recognizable Languages:
A language L is considered Turing recognizable (also known as recursively enumerable) if and only if there exists a Turing machine that accepts every string belonging to L and either halts or enters an infinite loop for inputs not belonging to L. In other words, a Turing machine can recognize a language if it can eventually accept all the strings that belong to that language.

To demonstrate this, let's consider a Turing machine M_1 that recognizes the language L containing all strings of 0's and 1's that begin and end with the same symbol. Here's how the Turing machine M_1 operates:

1. Start in a designated start state.
2. If the current symbol is a 0 or 1, move right.
3. Repeat step 2 until reaching a blank symbol.
4. If the current symbol is the same as the initial symbol, go back to the beginning of the string.
5. If the current symbol is different from the initial symbol, reject.

This Turing machine will eventually accept all strings that begin and end with the same symbol. If the string does not meet the criteria, the machine will either halt if it finishes reading the whole string (rejecting it) or enter an infinite loop if the string is non-terminating.

Turing Decidable Languages:

A language L is considered Turing decidable (also known as recursive) if and only if there exists a Turing machine that accepts every string belonging to L and halts for inputs not belonging to L. In other words, a Turing machine can decide a language if it can both recognize and always halt on every input.

To demonstrate this, let's consider a Turing machine M_2 that decides the language L containing all strings of 0's and 1's that have an equal number of 0's and 1's. Here's how the Turing machine M_2 operates:

1. Start in a designated start state.
2. If the current symbol is a 0 or 1, move right and count the number of 0's and 1's.
3. Repeat step 2 until reaching a blank symbol.
4. If the number of 0's and 1's is different, reject.
5. If the number of 0's and 1's is equal, accept.

This Turing machine will eventually accept all strings that contain an equal number of 0's and 1's. If the string does not meet the criteria, the machine will halt and reject it.

In summary, a language is Turing recognizable if there exists a Turing machine that can accept all strings belonging to that language (and potentially enter an infinite loop for non-members), while a language is Turing decidable if there exists a Turing machine that can accept all strings belonging to that language and always halt for non-members.

b) The following is a description of a given Context Free Grammar:
S◊aSb◊SS◊ ε

i. Give a formal definition of this grammar. [4 Marks]
- The start symbol is S.
- The set of nonterminal symbols is {S}.
- The set of terminal symbols is {a, b}.
- The set of production rules is:
  1. S -> aSb
  2. S -> SS
  3. S -> ε (epsilon represents the empty string)

ii. Show how to generate the following languages from the grammar

i. abab [2 Marks]
- Start with the start symbol S.
- Apply the production rule S -> aSb, which gives the derivation: S -> aSb -> abSb.
- Apply the production rule S -> aSb again, which gives the derivation: S -> aSb -> abSb -> abaSbb.
- Apply the production rule S -> ε, which gives the final derivation: S -> aSb -> abSb -> abaSbb -> abab.

ii. aaabbb [2 Marks]
- Start with the start symbol S.
- Apply the production rule S -> aSb, which gives the derivation: S -> aSb -> aaSbb.
- Apply the production rule S -> aSb again, which gives the derivation: S -> aSb -> aaSbb -> aaSbbb.
- Apply the production rule S -> ε, which gives the final derivation: S -> aSb -> aaSbb -> aaSbbb -> aaabbb.

iii. aababb [2 Marks]

- Start with the start symbol S.
  - Apply the production rule S -> aSb, which gives the derivation: S -> aSb -> aaSbb.
  - Apply the production rule S -> aSb again, which gives the derivation: S -> aSb -> aaSbb -> aaaSbbb.
  - Apply the production rule S -> SS, which gives the derivation: S -> aSb -> aaSbb -> aaaSbbb -> aaaSbbbbb.
  - Apply the production rule S -> ε, which gives the final derivation: S -> aSb -> aaSbb -> aaaSbbb -> aaaSbbbbb -> aababb.

c) Describe any two areas where Context-Free Grammar is used in computer science. [2 Marks]

1. Programming languages: Context-free grammars are commonly used in programming languages to define the syntax of the language.

2. Natural language processing: Context-free grammars are used in natural language processing tasks, such as parsing and generating sentences. They can be used to define the grammatical structure of a language and to parse sentences into a syntax tree, which can then be used for various language understanding tasks.

3. Compiler design: Context-free grammars are an essential component of compiler design. They are used to define the syntax of a programming language, and parsers are used to check the syntax of source code against the grammar.

4. Syntax highlighting and code editors: Context-free grammars are used in syntax highlighting tools and code editors to recognize and format different syntactic elements of a programming language.

QUESTION THREE [20 MARKS]

a) The halting problem is a problem of interest in computer science.

i. Describe the halting problem. [2 Marks]

The halting problem is a famous problem in computer science that deals with determining whether a given Turing Machine will eventually halt (stop) or continue running indefinitely.

QUESTION FOUR [20 MARKS]

a) Giving a real example for each, describe the following class of problems found in computer science

i. NP-complete problems. [3 Marks]

An example of an NP-complete problem is the traveling salesman problem (TSP). In this problem, a salesman is given a list of cities and the distances between each pair of cities. The salesman wants to find the shortest possible route that visits each city exactly once and returns to the starting city. The TSP is NP-complete because it is easy to verify a solution (i.e., given a solution, it is easy to check if it satisfies the constraints), but it is computationally difficult to find the optimal solution.

ii. NP problems. [3 Marks]

An example of an NP problem is the subset sum problem. In this problem, we are given a set of integers and a target number. The task is to find a subset of the given set whose sum equals the target

number, if such a subset exists. It is easy to verify if a given subset does indeed sum up to the target number, but finding the subset itself can be computationally challenging.

iii. P problems. [3 Marks]
An example of a P problem is determining whether a number is prime or not. Given a number, the task is to check if it is divisible by any number other than 1 and itself. This problem can be solved using algorithms with a polynomial time complexity, as the number of divisions required is bounded by the square root of the given number. Therefore, it falls within the class of P problems.

b) Differentiate between the Finite automaton and Turing Machines in regards to space complexity [6 Marks]

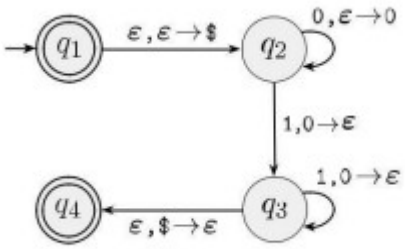
1. Finite Automaton (FA):
- FA has a constant space complexity since it only requires a fixed amount of memory to store the current state of the machine.
- The amount of memory used remains the same regardless of the input size.
- FA has a limited memory capacity, as it can only store a fixed number of states.

2. Turing Machine (TM):
- TM has an unbounded space complexity since it can potentially use an infinite amount of memory.
- TM has an infinite tape that can store an infinite amount of symbols.
- As the input size increases, the space required by the TM can also increase, potentially using an arbitrarily large amount of memory.

3. Space Complexity Comparison:
- Finite automaton is more limited in terms of space complexity compared to Turing machines.
- A TM can solve problems that require more memory by using an unlimited tape to store and manipulate data, whereas an FA's memory capacity is fixed.
- TM can solve problems that require unbounded space, such as recognizing recursively enumerable languages, that an FA cannot solve due to its limited memory.

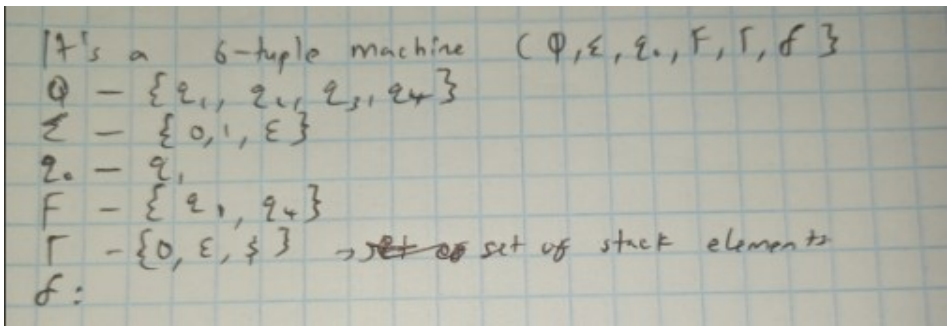
QUESTION FIVE [20 MARKS]

a) Given the following computation machine



i. Explain what type of machine this is. [3 Marks]
This is a Pushdown Automaton, because it consists of the stack initializer symbols together with other alphabet inputs and the empty symbol, it involves stack operation like pushing and popping symbols.

ii. Give a formal definition of the machine. [3 Marks]



iii. Explain what type of languages are recognized by this machine and justify why this is so. [2 Marks]
Regular Languages.
$L=\{0^n1^n \mid n\geq 0\}$, accepts all strings that have the same number of 0's and 1's.
Reason; It has the stack that has the ability to push and pop symbols hence handle recursiveness.

iv. Identify any two strings accepted by this machine [2 Marks]
01
0011

v. Application of PDA.
Compiler design.
Expression parsing.
DNA Sequence Analysis

b) Define the pumping lemma as used in the Theory of Computation. [2 Marks]
It is a tool used in theory of compytation to prove that a given language is not regular..

c) Giving an example, discuss how the pumping is used in a language defined as L= (0n 1 n |n>=0}.
In the language L = {(0^n 1^n | n ≥ 0)}, pumping is used as a technique to show that the language is not a regular language. A regular language can be recognized by a finite automaton, but L cannot be recognized using only a finite automaton.

To demonstrate this, let's consider the word w = 0^p 1^p, where p is a prime number. This word belongs to L because the number of 0s is equal to the number of 1s.

According to the pumping lemma for regular languages, if L is a regular language, then there exists a pumping length p, where any word w in L with a length ≥ p can be divided into three parts: w = xyz, satisfying the following conditions:
1. |xy| ≤ p
2. |y| > 0
3. For all i ≥ 0, the word xy^iz also belongs to L.

Let's assume that L is regular and follows the pumping lemma. According to condition 1, the string xy consists of only 0s. This means that y contains only 0s as well. Since |y| > 0 (condition 2), we can assume that y contains at least one 0.

Now let's consider string w' = xy^0z = xz. Since all the 0s in y have been removed, |x| < |xy| = p. Therefore, the first condition of the pumping lemma is not satisfied.

As a result, we can conclude that L = {(0^n 1^n | n ≥ 0)} is not a regular language, as it does not fulfill the conditions of the pumping lemma.

PAPER 5

QUESTION 1[30MKS]

a) Explain what theory of computation deals with (2 marks)
The theory of computation deals with the study of abstract models of computation and the algorithms used for solving computational problems. It explores concepts such as automata, formal languages, and computability.

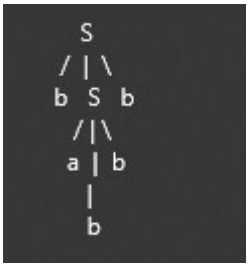
b) Define the following terms giving an example of each
   i. Alphabet(Σ) (2 marks)
   An alphabet is a non-empty set of symbols that is used to construct strings in a formal language. For example, an alphabet Σ can consist of the symbols {0, 1}, which can be used to form binary strings.
   ii. String over an alphabet Σ (2 marks)
   A string over an alphabet Σ is a finite sequence of symbols chosen from Σ. For example, if Σ consists of {a, b}, then the string "aab" is a string over Σ.
   iii. Empty strings(ε) (2 marks)
   The empty string ε is a string that contains no symbols. It has a length of zero. For example, in the alphabet {0, 1}, the empty string ε would be represented by "".
   iv. Power of an alphabet Σ* (3 marks)
   The power of an alphabet Σ* represents the set of all possible strings that can be constructed using the symbols from Σ, including the empty string ε. For example, if Σ consists of {a, b}, then Σ* would include strings like "a", "b", "aa", "ab", "ba", "bb", "aaa", and so on.
   v. Language over an alphabet (L) (3 marks)
   A language over an alphabet Σ is a set of strings that are formed using symbols from Σ. It can include any combination of strings, including the empty string ε. For example, if Σ consists of {0, 1}, a language L could be {ε, "0", "1", "01", "10"}.

c) What is an NP-complete problem? Give an example of an NP Problem. (4 marks)
An NP-complete problem is a computational problem in which the solution can be verified in polynomial time, but there is no known algorithm to solve it in polynomial time. In other words, it is a problem that is believed to be computationally difficult or "hard" to solve efficiently. An example of an NP-complete problem is the traveling salesman problem, which involves finding the shortest route that visits a given set of cities and returns to the original city.

d) Write the regular expression for the language accepting all combinations of a's, over the set ∑ = {a} (4 marks)
a*. This regular expression allows for any combination of zero or more occurrences of 'a'.

e) What is a derivation tree? Draw a derivation tree for the string "bab" from the CFG given by S → bSb | a | b (4 marks)
A derivation tree is a graphical representation of the production rules used to derive a string in a context-free grammar (CFG). It demonstrates how the non-terminal symbols in the grammar can be replaced with terminal symbols in order to generate a specific string.

Here is the derivation tree for the string "bab" from the given CFG:



In this derivation tree, we start with the starting symbol S and replace it with bSb according to the production rule S → bSb. Then, we replace the leftmost S with a according to the production rule S → a, resulting in the string "bab".

f) Construct the CFG for the language having any number of a's over the set ∑= {a}. (4 marks)

The CFG for the language having any number of 'a's over the set Σ = {a} can be constructed as follows:

1. Start symbol: S
2. Terminal symbol: a
3. Production rules:
   - S → ε
   - S → aS

Explanation:
- The start symbol S represents the language.
- The terminal symbol a represents the character 'a'.
- The production rule S → ε allows for the empty string to be generated.
- The production rule S → aS allows for the generation of any number of 'a's by recursively applying the rule.

This CFG generates strings such as ε (empty string), a, aa, aaa, etc., representing any number of 'a's.

Question 2 (20 marks)

a) What do you understand from the following terms as used in the theory of computation? (6 marks)

i. Regular expression

A regular expression is a sequence of characters that defines a search pattern. It is used to match and manipulate text in various applications, including searching for patterns in strings, validating input, and transforming text.

ii. Context-free grammar

Context-free grammar (CFG) is a formalism used to describe the syntax of a formal language. It consists of a set of production rules that define how strings of symbols can be generated in the language.

iii. Ambiguous grammar

An ambiguous grammar is a type of context-free grammar where a single string in the language can be derived by more than one different derivation trees or sequences of production rules. In other words, an ambiguous grammar allows for multiple interpretations or meanings of a given string.

b) Using a diagram if necessary, explain the FIVE operations of Turing machine. (5 marks)

1. Read: The Turing machine has a tape divided into cells, and each cell can hold a symbol. During the "read" operation, the Turing machine reads the current symbol on the tape.

2. Write: Based on the current state and current symbol being read, the Turing machine performs the "write" operation to replace the current symbol on the tape with a new symbol. This operation updates the tape with new information.

3. Move: The Turing machine is equipped with a tape head that can move left or right. The "move" operation determines the direction of the tape head movement after reading or writing a symbol. This operation allows the Turing machine to navigate along the tape.

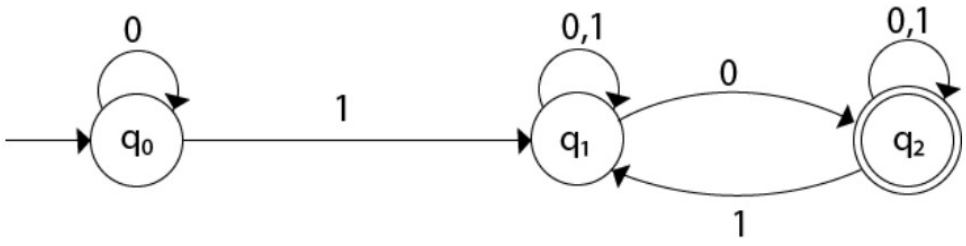
4. Transition: The "transition" operation defines the machine's behavior based on its current state, the symbol being read, and the specific transition function of the Turing machine. It essentially determines the next state of the machine and the next symbol to write on the tape.

5. Change State: The Turing machine has a set of states that it can be in at any given time. The "change state" operation involves updating the current state of the machine based on the transition function and the current state of the machine. This step allows the Turing machine to progress through various states

Question 3 (20 marks)

a) Give a formal definition of a Deterministic Finite Automata (DFA) (5 marks)
It is a 5-tuple machine (Q, Σ, δ, q0, F), where:
1. Q is a finite set of states.
2. Σ is the input alphabet, which is a finite set of symbols.
3. δ: Q × Σ → Q is the transition function.
4. q0 ∈ Q is the initial state.
5. F ⊆ Q is the set of final states.

b) Convert the given NFA to DFA. (10 marks)



Question 5 (20 marks)

a) Briefly describe the following terms as used in theory of computation. (10 marks)
   i. P Problem
   It is an algorithmic problem that can be solved efficiently, with the running time of the algorithm being a polynomial function of the problem size.
   ii. Non-deterministic Finite automata
   NFA is a computational model used to describe and recognize regular languages. It consists of a finite number of states and transitions between these states based on the input symbols. an

NFA can have multiple possible transitions for a given input. At any given time, an NFA can be in multiple states, giving it a non-deterministic behavior.

iii. Turing machine

Turing machine is a theoretical computational model that consists of an infinite tape divided into cells, a tape head that can read and write symbols on the tape, a finite control unit that guides the operations of the machine, and a set of states. It can simulate any algorithmic computation and is often used to study the concepts of computability and decidability.

iv. Push Down Automata

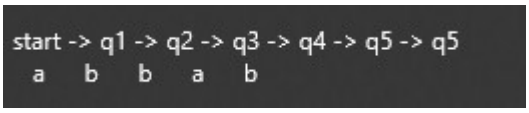
It is used to recognize and generate context-free languages. PDAs have the ability to read an input symbol, change the state of the machine, and manipulate the stack based on the current state and input symbol.

v. Deterministic Finite automata

DFA is a computational model used to recognize regular languages. It accepts an input if, after reading the entire input, the machine is in an accepting state.

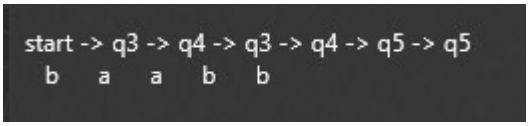
b) Draw a state diagram to show how the machine responds to the following input

a) abbab (5 marks)



In this state diagram, 'start' represents the initial state of the machine, while each qi represents a different state. The arrows between the states represent the transitions based on the input characters.

b) baabba (5 marks)



In this state diagram, 'start' represents the initial state of the machine, while each qi represents a different state. The arrows between the states represent the transitions based on the input characters.

PAPER 6

QUESTION ONE [30 MARKS]

a) Explain and give an example of each of the following terms used in Theory of Computation:

i. Alphabet. [2 Marks]

In the Theory of Computation, an alphabet refers to a finite set of symbols from which strings are formed.
Example: Let's consider an alphabet Σ = {0, 1}.

ii. String. [2 Marks]

In the Theory of Computation, a string is a finite sequence of symbols from an alphabet. The length of a string is the number of symbols it contains.

Example: Consider the alphabet Σ = {0, 1}. Some examples of strings over this alphabet are "010101", "1111", "000", and "10".

b) Describe the role of language in the theory of Computation. [3 Marks]
language is essential in the theory of computation as it allows us to formalize problems, express computations, and analyze computational complexity. It provides a framework for studying what can be computed and how efficiently it can be done, ultimately leading to the development of algorithms and computational models.

c) Let M be a Turing machine, and let w be an input string for M. Define the running time tM(w) of M on input w according to the Theory of Complexity. [4 Marks]
The running time tM(w) of a Turing machine M on input w is the number of steps or operations performed by M before it halts and produces the output for input w. It is a measure of the efficiency or complexity of the algorithm implemented by M. The Theory of Complexity aims to classify problems based on the resources required to solve them, such as time and space.

d) List and explain the four contributions that Alan Turing is famous for in Computer Science. [4 Marks]
1. Turing Machines: Turing introduced the concept of a theoretical machine, later known as a Turing machine, which served as the foundation for the study of computability and algorithms.
2. The Turing Test: A standard test for AI.
3. He designed CAPTCHA to know if the one logging into a web is a human or robot.
4. Theoretical Computer Science: Turing's work laid the groundwork for the development of theoretical computer science, including the study of formal languages, computability theory, and complexity theory.
5.In cryptography by decoding of encryption of german enigma machine in 2nd world war.

e) Describe the halting problem and give an example of its applications to nowadays computer professionals. [4 Marks]
The halting problem is an unsolvable problem in computer science. It asks whether it is possible to write a program that can determine, given any arbitrary program and input, whether that program will eventually halt (stop running) or will run forever. Alan Turing proved that it is not possible to solve the halting problem for all programs.

An example of its application to nowadays computer professionals is in program verification and bug detection. Many software engineers use specialized tools that analyze the code and try to find potential infinite loops or cases where the program may run indefinitely. These tools make intelligent guesses and heuristics but cannot guarantee a complete solution due to the fundamental undecidability of the halting problem.

f) Explain the importance of NP-complete problems to computer scientists. [4 Marks]
NP-complete problems are of great importance to computer scientists because they represent a class of problems that are believed to be among the most difficult to solve efficiently. An NP-complete problem is one for which a potential solution can be verified in polynomial time, but no efficient algorithm is currently known to solve all instances of the problem in polynomial time.

The importance of NP-complete problems lies in the fact that if an efficient algorithm is discovered for any one of them, it would imply that efficient

solutions exist for all problems in the class NP (nondeterministic polynomial time). Conversely, if it could be proven that no efficient algorithm exists for any NP-complete problem, it would indicate that the class NP is inherently difficult and would have profound implications for computational complexity theory.

g) Highlight in brief each of the branches of the Theory of Computation. [3 Marks]

1. Automata Theory: Study of abstract machines, including finite automata, pushdown automata, and Turing machines, to understand the fundamental capabilities and limitations of computation.

2. Computability Theory: Investigation of what problems can be solved algorithmically, exploring the concept of computability and the existence of unsolvable problems.

3. Complexity Theory: Deals with the analysis of the resources required to solve computational problems, such as time and space complexity. It classifies problems into various complexity classes, such as P, NP, and NP-complete.

4. Formal Languages and Grammar: Examines the structure and properties of formal languages, including regular languages, context-free languages, and formal grammars. It is used to define programming languages and study the syntax of natural languages.

QUESTION TWO [20 MARKS]

a) Let L be the language {0n 1 n |n>=0}. Explain why language L is not considered a regular language. [4 Marks]

The language L = {0^n 1^n | n >= 0} is not considered a regular language because it cannot be recognized or generated by a regular expression or a finite automaton. A regular language is one that can be recognized by a finite automaton, which has a finite number of states. In the case of L, there is no way to determine an equal number of 0s and 1s using a finite number of states. This is because for every n, there needs to be n 0s followed by n 1s. As n can be any non-negative integer, it is not possible to determine the number of 0s and 1s using a finite number of states, making L not regular.

b) Let ADFA= {(M, w): M is a Deterministic Finite Automata that accepts the string w}. ADFA is decidable. Explain and justify your answer through proof. [6 Marks]

ADFA = {(M, w): M is a Deterministic Finite Automaton that accepts the string w} is decidable. To prove this, we can construct a decider that will determine whether a given pair (M, w) is a member of ADFA.

Here is the high-level description of the decider algorithm:
1. Simulate the given DFA M on the input string w.
2. If the simulation ends in an accepting state, accept. Otherwise, reject.

To justify the correctness of this algorithm:
- If (M, w) is in ADFA, it means that M accepts the string w. The simulation of M on w will end in an accepting state, and the algorithm will correctly accept.

- If (M, w) is not in ADFA, it means that M does not accept the string w. The simulation of M on w will not end in an accepting state, and the algorithm will correctly reject.

Since the algorithm always halts and produces the correct output for any input (M, w), we can conclude that ADFA is decidable.

c) The language ATM is undecidable. Explain. [4 Marks]

The language ATM stands for the "Acceptance Turing Machine" language, which consists of the set of all encodings of Turing machines that accept at least one input. The language ATM is undecidable, meaning that there is no algorithm that can determine, for any given Turing machine, whether it accepts at least one input or not.

This undecidability of ATM can be proven by utilizing a technique called the diagonalization argument. The idea is to assume that there exists a decider for ATM and then construct a "contradictory" Turing machine that leads to a contradiction.

Assuming that there exists a decider for ATM, we can construct a Turing machine that takes its own encoding as input, simulates the behavior of the assumed decider on itself, and then does the opposite (i.e., accepts if the assumed decider would reject and rejects if it would accept). This leads to a contradiction because the output of this constructed Turing machine is opposite to the behavior predicted by the assumed decider.

Therefore, we can conclude that ATM is undecidable.

d) Identify any three modern applications that reference Finite Automata in their designs. [6 Marks]

1) Computer compilers: Finite automata are used in the lexical analysis phase of compiler design to recognize and categorize tokens in the source code.

2) Natural language processing (NLP): Finite automata can be used to design finite-state transducers, which are widely employed in applications like spelling correction, machine translation, and speech recognition.

3) Regular expression engines: Many modern regular expression engines utilize finite automata to efficiently match and search patterns in strings.

4) Network protocols: Finite automata are used to model and implement various network protocols, such as the Transmission Control Protocol (TCP) and the Border Gateway Protocol (BGP), to facilitate reliable and efficient communication.

QUESTION THREE [20 MARKS]

a) Demonstrating using a Turing Machine, differentiate between Turing Recognizable and Turing Decidable Languages. [4 Marks]

1. Turing Recognizable (also known as recursively enumerable) languages: A language L is Turing Recognizable if there exists a Turing machine that will halt and accept any input string belonging to L. However, if an input string does not belong to L, the Turing machine may either halt and reject, or run indefinitely without halting.

2. Turing Decidable (also known as recursive) languages: A language L is Turing Decidable if there exists a Turing machine that halts and accepts any input string belonging to L, and also halts and rejects any input string not belonging to L. In other words, a Turing Decidable language is always decidable, as the Turing machine will always halt and provide the correct answer.

To differentiate between Turing Recognizable and Turing Decidable languages using a Turing Machine, consider the following steps:

1. For a Turing Recognizable language, design a Turing machine that will accept any input string belonging to the language. This Turing machine may halt and reject or run indefinitely for input strings not belonging to the language.

2. For a Turing Decidable language, design a Turing machine that will halt and accept any input string belonging to the language, and halt and reject any input string not belonging to the language. This Turing machine will always provide the correct answer.

b) An important application of context-free grammars occurs in the specification and compilation of programming languages. Discuss. [6 Marks]

Context-free grammars (CFGs) play a crucial role in the specification and compilation of programming languages. Here are six reasons why:

1. Syntax Specification: CFGs are used to define the syntax of programming languages. They describe the structure and arrangement of valid program statements, expressions, and other language constructs. By defining the grammar rules, a CFG provides a formal representation of the language syntax, allowing developers to write code that adheres to the language rules.

2. Parsing: CFGs are essential in the parsing phase of the compilation process. During parsing, a compiler analyzes the sequence of tokens in the source code to determine if it follows the language grammar. By building a parse tree based on the CFG, the compiler can check for syntactic errors and ensure that the code is well-formed.

3. Ambiguity Resolution: CFGs help in resolving syntactic ambiguities that might occur in programming languages. Ambiguities arise when a statement or expression has multiple possible interpretations, leading to ambiguity in the language grammar. By carefully designing the CFG, ambiguities can be minimized or eliminated, ensuring unambiguous parsing and compilation.

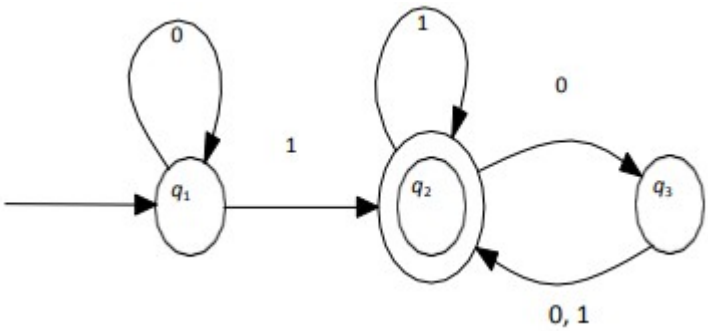
4. Error Detection: CFGs enable the detection of syntax errors in the source code. When the parser encounters code that violates the language grammar rules defined in the CFG, it can report the error to the programmer, providing helpful diagnostics. This helps developers to identify mistakes early and fix them efficiently.

5. Language Extensions: CFGs facilitate the ability to extend programming languages. By modifying the CFG, language designers can introduce new

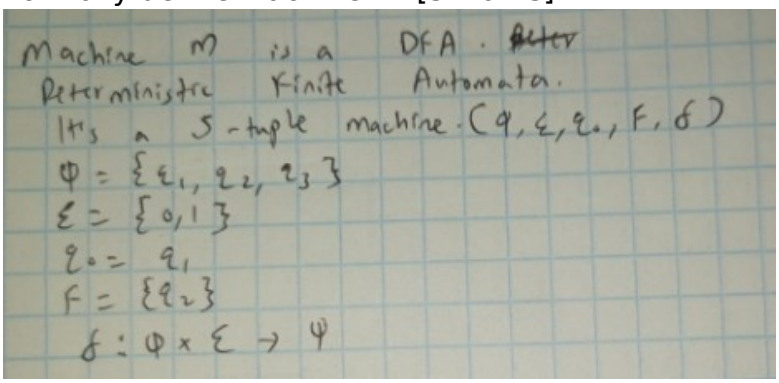
syntax and language features without breaking backward compatibility. This flexibility allows languages to evolve and adapt to changing programming paradigms and requirements.

6. Compiler Optimization: CFGs contribute to compiler optimization techniques. By analyzing the control flow of a program through its CFG, compilers can optimize the generated code. Techniques like loop optimization, dead code elimination, and code hoisting heavily rely on CFG analysis to identify opportunities for improvement and produce more efficient executable code.

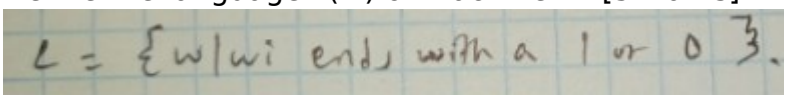
c) Machine M is given below:



i. Formally define Machine M. [5 Marks]



ii. Define the language L(M) of machine M. [5 Marks]



QUESTION FOUR [20 MARKS]

a) Identify the language used by each of the following computation models and explain your answer:

i. Finite Automaton. [2 Marks]
The language used by Finite Automaton is regular language. A finite automaton is a computation model with a finite set of states and transitions between those states, which can be used to recognize regular languages. It accepts or rejects strings based on whether it reaches a final state after processing the input.

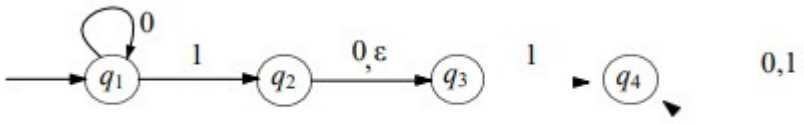
ii. Push Down Automata. [2 Marks]
The language used by Push Down Automata is context-free language. A pushdown automaton extends the capabilities of a finite automaton by introducing a stack. It can recognize context-free languages, which are more powerful than regular languages. It uses a stack to keep track of the input and can perform specific operations like push, pop, and peek on the stack.

iii. Context Free Grammar. [2 Marks]

The language used by Context-Free Grammar is a context-free language. Context-free grammars specify the syntax of programming languages and other formal languages. They consist of a set of production rules that define how symbols can be replaced by other symbols. Context-free languages are more expressive than regular languages but less powerful than recursively enumerable languages.

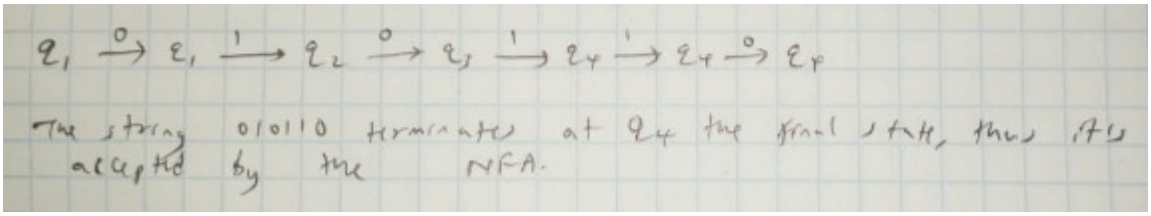
iv. Turing Machines. [2 Marks]
The language used by Turing Machines is recursively enumerable language. Turing Machines are the most powerful computation model and can simulate any algorithmic process. They consist of an infinite tape and a read/write head that can move along the tape. Turing Machines can accept or reject strings, halt, or enter an infinite loop. Recursively enumerable languages are a superset of all computable languages, including regular, context-free, and context-sensitive languages.

b) Study the machine given below:



i. Explain with justification what type of machine this is. [4 Marks]
It is a Non-Deterministic Finite Automata.
It has an empty symbol as a label on a transition.
The empty symbol as a member of the alphabet.

ii. Demonstrate the computation of an input 010110 on this machine. [8 Marks]



QUESTION FIVE [20 MARKS]

a) Let L (A) = {0, 01} and L (B) = {1, 10}. Find:

i. Union of L(A) and L(B). [2 Marks]
The union of L(A) and L(B) is the set of all elements that are in either L(A) or L(B), or both.
L(A) = {0, 01}
L(B) = {1, 10}
The union of L(A) and L(B) is {0, 01, 1, 10}.

ii. Concatenation of L(A) and L(B). [2 Marks]
The concatenation of L(A) and L(B) is the set of all possible combinations of concatenating an element from L(A) and an element from L(B).

L(A) = {0, 01}
L(B) = {1, 10}

The concatenation of L(A) and L(B) is {01, 010, 011, 0110}.

b) Consider the following components of Context Free Grammar:
S->AB

A->aA
B->bB

i. Define the Grammar. [4 Marks]
S -> AB
A -> aA | ε (epsilon)
B -> bB | ε (epsilon)

ii. Generate the string aaaabb from the grammar. [2 Marks]
S -> AB (start with S)
-> aAB (replace S with A)
-> aaAB (replace A with aA)
-> aaaAB (replace A with aA)
-> aaaaAB (replace A with ε)
-> aaaaB (replace B with bB)
-> aaaabB (replace B with ε)
-> aaaabb (replace B with ε)

iii. Draw the parse tree to generate the string aaaabb from the grammar. [2 Marks]

c) Explain how Hilbert's problems have influenced Computer Science. [4 Marks]

) Hilbert's problems have had a significant influence on Computer Science. One of the most notable impacts is in the area of mathematical logic and foundations of mathematics. Hilbert's tenth problem, which sought to find an algorithmic method to determine the solvability of Diophantine equations, led to the development of the field of computational number theory. This has helped in understanding the limitations of computer algorithms and the concept of undecidable problems.

Hilbert's problems also influenced the development of algorithmic complexity theory and the field of computational complexity. His questions regarding the consistency and completeness of formal systems inspired researchers to study the limits of what can be computed or proven within a given system. This led to the development of complexity classes, such as P, NP, and NP-complete, which are fundamental in understanding the efficiency and difficulty of solving computational problems.

Additionally, Hilbert's problems have influenced the field of artificial intelligence and automated theorem proving. Several of his problems were related to the mechanization of mathematics and the possibility of constructing a universal algorithm that could solve any mathematical problem. These ideas have paved the way for the development of

computer-based theorem proving systems and automated reasoning tools that can assist in mathematical discovery and proof verification.

d) State and explain the Church-Turing thesis highlighting its impact to the Theory of Computation. [4 Marks]

The Church-Turing thesis, proposed by Alonzo Church and Alan Turing in the 1930s, is a fundamental concept in the theory of computation. It suggests that any computable function can be effectively computed by a Turing machine, or equivalently, by any other computational model that can simulate a Turing machine.

According to the Church-Turing thesis, any algorithmic task that can be described precisely and effectively computable can be solved by "mechanical" means, either on paper or by a machine. This means that if there is a general process to compute a function, it can be executed by a Turing machine, given enough time and resources. The thesis doesn't state that all problems can be solved algorithmically, but rather that any problem that can be solved algorithmically can be solved by a Turing machine.

The impact of the Church-Turing thesis on the Theory of Computation is profound. It forms the basis for understanding the limitations of computation and helps define the boundaries of what can and cannot be computed. It allows us to reason about the complexity and efficiency of algorithms, as well as the overall capabilities of different computational systems.

The thesis has also served as an inspiration and guideline for the development of modern computers and programming languages. It provides a theoretical framework for designing and analyzing algorithms, as well as determining the computability of problems. In essence, the Church-Turing thesis is a cornerstone concept in the theory of computation, enabling us to understand and explore the limits and possibilities of computation.