What is TOC?

In theoretical computer science, the theory of computation is the branch that deals with whether and how efficiently problems can be solved on a model of computation, using an algorithm. The field is divided into three major branches: automata theory, computability theory and computational complexity theory.

In order to perform a rigorous study of computation, computer scientists work with a mathematical abstraction of computers called a model of computation. There are several models in use, but the most commonly examined is the Turing machine.

Automata theory

In theoretical computer science, automata theory is the study of abstract machines (or more appropriately, abstract 'mathematical' machines or systems) and the computational problems that can be solved using these machines. These abstract machines are called automata. This automaton consists of

- states (represented in the figure by circles),
- transitions (represented by arrows).

As the automaton sees a symbol of input, it makes a transition (or jump) to another state, according to its transition function (which takes the current state and the recent symbol as its inputs).

Uses of Automata: compiler design and parsing.

Figure 1.2: A finite automaton modeling recognition of then

Introduction to formal proof:
Basic Symbols used:

U – Union
∩- Conjunction ε
- Empty String
Φ – NULL set
7- negation
‘ – compliment
=> implies
Additive inverse: \(a + (-a) = 0\)

Multiplicative inverse: \(a * \frac{1}{a} = 1\)

Universal set \(U = \{1, 2, 3, 4, 5\}\)

Subset \(A = \{1, 3\}\)

\(A' = \{2, 4, 5\}\)

Absorption law: \(AU(A \cap B) = A, A \cap (AUB) = A\)

De Morgan’s Law:

\((A \cup B)' = A' \cap B'\)

\((A \cap B)' = A' \cup B'\)

Double compliment

\((A')' = A\)

\(A \cap A' = \emptyset\)

Logic relations:

\(a \land b = \rightarrow 7a \cup b\)

\(7(a \cap b) = 7a \cup 7b\)

Relations:

Let \(a\) and \(b\) be two sets a relation \(R\) contains \(aXb\).

Relations used in TOC:

Reflexive: \(a = a\)

Symmetric: \(aRb \rightarrow bRa\)

Transition: \(aRb, bRc \rightarrow aRc\)

If a given relation is reflexive, symmetric and transitive then the relation is called equivalence relation.

Deductive proof: Consists of sequence of statements whose truth lead us from some initial statement called the hypothesis or the give statement to a conclusion statement.

The theorem that is proved when we go from a hypothesis \(H\) to a conclusion \(C\) is the statement “if \(H\) then \(C\).” We say that \(C\) is deduced from \(H\).

Additional forms of proof:

Proof of sets

Proof by contradiction

Proof by counter example

Direct proof (AKA) Constructive proof:

If \(p\) is true then \(q\) is true

Eg: if \(a\) and \(b\) are odd numbers then product is also an odd number. Odd number can be represented as \(2n+1\)

\(a = 2x + 1, b = 2y + 1\)

Product of \(a\) \& \(b\) = \((2x+1) \times (2y+1) = 2(2xy+x+y)+1 = 2z+1\) (odd number)
Proof by contrapositive:

The contrapositive of the statement “if \( H \) then \( C \)” is “if not \( C \) then not \( H \).” A statement and its contrapositive are either both true or both false, so we can prove either to prove the other.

**Theorem 1.10:** \( R \cup (S \cap T) = (R \cup S) \cap (R \cup T) \).

<table>
<thead>
<tr>
<th>Statement</th>
<th>Justification</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. ( x ) is in ( R \cup (S \cap T) )</td>
<td>Given</td>
</tr>
<tr>
<td>2. ( x ) is in ( R ) or ( x ) is in ( S \cap T )</td>
<td>(1) and definition of union</td>
</tr>
<tr>
<td>3. ( x ) is in ( R ) or ( x ) is in both ( S ) and ( T )</td>
<td>(2) and definition of intersection</td>
</tr>
<tr>
<td>4. ( x ) is in ( R \cup S )</td>
<td>(3) and definition of union</td>
</tr>
<tr>
<td>5. ( x ) is in ( R \cup T )</td>
<td>(3) and definition of union</td>
</tr>
<tr>
<td>6. ( x ) is in ( (R \cup S) \cap (R \cup T) )</td>
<td>(4), (5), and definition of intersection</td>
</tr>
</tbody>
</table>

**Figure 1.5:** Steps in the “if” part of Theorem 1.10

<table>
<thead>
<tr>
<th>Statement</th>
<th>Justification</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. ( x ) is in ( (R \cup S) \cap (R \cup T) )</td>
<td>Given</td>
</tr>
<tr>
<td>2. ( x ) is in ( R \cup S )</td>
<td>(1) and definition of intersection</td>
</tr>
<tr>
<td>3. ( x ) is in ( R \cup T )</td>
<td>(1) and definition of intersection</td>
</tr>
<tr>
<td>4. ( x ) is in ( R ) or ( x ) is in both ( S ) and ( T )</td>
<td>(2), (3), and reasoning about unions</td>
</tr>
<tr>
<td>5. ( x ) is in ( R ) or ( x ) is in ( S \cap T )</td>
<td>(4) and definition of intersection</td>
</tr>
<tr>
<td>6. ( x ) is in ( R \cup (S \cap T) )</td>
<td>(5) and definition of union</td>
</tr>
</tbody>
</table>

**Figure 1.6:** Steps in the “only-if” part of Theorem 1.10

To see why “if \( H \) then \( C \)” and “if not \( C \) then not \( H \)” are logically equivalent, first observe that there are four cases to consider:
1. \( H \) and \( C \) both true.

2. \( H \) true and \( C \) false.

3. \( C \) true and \( H \) false.

4. \( H \) and \( C \) both false.

**Proof by Contradiction:**

\( H \) and not \( C \) implies falsehood.

That is, start by assuming both the hypothesis \( H \) and the negation of the conclusion \( C \). Complete the proof by showing that something known to be false follows logically from \( H \) and not \( C \). This form of proof is called *proof by contradiction*.

It often is easier to prove that a statement is not a theorem than to prove it *is* a theorem. As we mentioned, if \( S \) is any statement, then the statement “\( S \) is not a theorem” is itself a statement without parameters, and thus can be regarded as an observation than a theorem.

**Proof by mathematical Induction:**

Suppose we are given a statement \( S(n) \), about an integer \( n \), to prove. One common approach is to prove two things:

1. The *basis*, where we show \( S(i) \) for a particular integer \( i \). Usually, \( i = 0 \) or \( i = 1 \), but there are examples where we want to start at some higher \( i \), perhaps because the statement \( S \) is false for a few small integers.

2. The *inductive step*, where we assume \( n \geq i \), where \( i \) is the basis integer, and we show that “if \( S(n) \) then \( S(n+1) \).”

- *The Induction Principle*: If we prove \( S(i) \) and we prove that for all \( n \geq i \), \( S(n) \) implies \( S(n+1) \), then we may conclude \( S(n) \) for all \( n \geq i \).

**Languages :**

The languages we consider for our discussion is an abstraction of natural languages. That
is, our focus here is on formal languages that need precise and formal definitions. Programming languages belong to this category.

Symbols:

Symbols are indivisible objects or entity that cannot be defined. That is, symbols are the atoms of the world of languages. A symbol is any single object such as, $a$, 0, 1, #, begin, or do.

Alphabets:

An alphabet is a finite, nonempty set of symbols. The alphabet of a language is normally denoted by $\Sigma$. When more than one alphabets are considered for discussion, then subscripts may be used (e.g. $\Sigma_1, \Sigma_2$ etc) or sometimes other symbol like $G$ may also be introduced.

Strings or Words over Alphabet:

A string or word over an alphabet $\Sigma$ is a finite sequence of concatenated symbols of $\Sigma$.

Example: 0110, 11, 001 are three strings over the binary alphabet \{0, 1\}.

$aab, abcb, b, cc$ are four strings over the alphabet \{a, b, c\}.

It is not the case that a string over some alphabet should contain all the symbols from the alphabet. For example, the string $cc$ over the alphabet \{a, b, c\} does not contain the symbols $a$ and $b$. Hence, it is true that a string over an alphabet is also a string over any superset of that alphabet.

Length of a string:

The number of symbols in a string $w$ is called its length, denoted by $|w|$.

Example: $|011| = 4, |1| = 2, |b| = 1$

Convention: We will use small case letters towards the beginning of the English alphabet to denote symbols of an alphabet and small case letters towards the end to denote strings over an alphabet. That is, $a, b, c \in \Sigma$ (symbols) $u, v, w, x, y, z$ and are strings.

Example: Consider the string 011 over the binary alphabet. All the prefixes, suffixes and substrings of this string are listed below.

Prefixes: $\epsilon, 0, 01, 011$.
Suffixes: $\epsilon, 1, 11, 011$.
Substrings: $\epsilon, 0, 1, 01, 11, 011$. 
Note that $x$ is a prefix (suffix or substring) to $x$, for any string $x$ and $\varepsilon$ is a prefix (suffix or substring) to any string.

A string $x$ is a proper prefix (suffix) of string $y$ if $x$ is a prefix (suffix) of $y$ and $x \neq y$. In the above example, all prefixes except 011 are proper prefixes.

**Powers of Strings**: For any string $x$ and integer $n \geq 0$, we use $x^n$ to denote the string formed by sequentially concatenating $n$ copies of $x$. We can also give an inductive definition of $x^n$ as follows: $x^0 = \varepsilon$, if $n = 0$; otherwise $x^n = x^{n-1}$.

**Reversal**: For any string $\mathcal{w} = a_1a_2a_3 \cdots a_n$, the reversal of the string is $\mathcal{w}^R = a_n a_{n-1} \cdots a_3 a_2 a_1$.

An inductive definition of reversal can be given as follows:

Languages
- **Defn**: A language is a set of strings over an alphabet.
  - A more restricted definition requires some forms of restrictions on the strings, i.e., strings that satisfy certain properties
  - **Defn**: The syntax of a language restricts the set of strings that satisfy certain properties.

- **Defn**: A string over an alphabet $X$, denoted $\Sigma$, is a finite sequence of elements from $X$, which are indivisible objects.
  - e.g., Strings can be words in English.
  - The set of strings over an alphabet is defined recursively (as given below).

- **Example**: Given $\Sigma = \{a, b\}$, $\Sigma^*$ includes $\lambda$, $a$, $b$, $aa$, $ab$, $ba$, $bb$, $aaa$, …

- **Defn 2.1.2**: A language over an alphabet $\Sigma$ is a subset of $\Sigma^*$.

- **Defn 2.1.3**: Concatenation, is the fundamental binary operation in the generation of strings, which is associative, but not commutative, is defined as
  - **i. Basis**: If length($v$) = 0, then $v = \lambda$ and $uv = u$.
  - **ii. Recursion**: Let $v$ be a string with length($v$) = $n$ ($n > 0$). Then $v = wa$, for string $w$ with length $n-1$ and $a \in \Sigma$, and $uv = (uw)a$.

**Convention**: Capital letters $A$, $B$, $C$, $L$, etc. with or without subscripts are normally used to denote languages.

**Set operations on languages**: Since languages are set of strings we can apply set operations to languages. Here are some simple examples (though there is nothing new in it).

<table>
<thead>
<tr>
<th>Grammar</th>
<th>(Generates)</th>
<th>Language</th>
<th>(Recognizes)</th>
<th>Automata</th>
</tr>
</thead>
</table>

CS6503 Theory of computation       STUDENTSFOCUS.COM       Unit I
Automata: A algorithm or program that automatically recognizes if a particular string belongs to the language or not, by checking the grammar of the string.

An automata is an abstract computing device (or machine). There are different varities of such abstract machines (also called models of computation) which can be defined mathematically.

Every Automaton fulfills the three basic requirements.

- Every automaton consists of some essential features as in real computers. It has a mechanism for reading input. The input is assumed to be a sequence of symbols over a given alphabet and is placed on an input tape(or written on an input file). The simpler automata can only read the input one symbol at a time from left to right but not change. Powerful versions can both read (from left to right or right to left) and change the input. The automaton can produce output of some form. If the output in response to an input string is binary (say, accept or reject), then it is called an accepter. If it produces an output sequence in response to an input sequence, then it is called a transducer(or automaton with output).
- The automaton may have a temporary storage, consisting of an unlimited number of cells, each capable of holding a symbol from an alphabet ( which may be different from the input alphabet). The automaton can both read and change the contents of the storage cells in the temporary storage. The accusing capability of this storage varies depending on the type of the storage.

![Diagram of Automation](image)

Figure 1: The figure above shows a diagrammatic representation of a generic automation.

Operation of the automation is defined as follows.
At any point of time the automaton is in some integral state and is reading a particular symbol from the input tape by using the mechanism for reading input. In the next time step the automaton then moves to some other integral (or remain in the same state) as defined by the transition function. The transition function is based on the current state, input symbol read, and the content of the temporary storage. At the same time the content of the storage may be changed and the input read may be modified. The automation may also produce some output during this transition. The internal state, input and the content
of storage at any point defines the configuration of the automaton at that point. The transition from one configuration to the next (as defined by the transition function) is called a move. Finite state machine or Finite Automation is the simplest type of abstract machine we consider. Any system that is at any point of time in one of a finite number of interval state and moves among these states in a defined manner in response to some input, can be modeled by a finite automaton. It doesnot have any temporary storage and hence a restricted model of computation.

**Finite Automata**

Automata (singular: automation) are a particularly simple, but useful, model of computation. They were initially proposed as a simple model for the behavior of neurons.

**States, Transitions and Finite-State Transition System:**

Let us first give some intuitive idea about a state of a system and state transitions before describing finite automata.

Informally, a state of a system is an instantaneous description of that system which gives all relevant information necessary to determine how the system can evolve from that point on.

Transitions are changes of states that can occur spontaneously or in response to inputs to the states. Though transitions usually take time, we assume that state transitions are instantaneous (which is an abstraction).

Some examples of state transition systems are: digital systems, vending machines, etc.

A system containing only a finite number of states and transitions among them is called a finite-state transition system.

Finite-state transition systems can be modeled abstractly by a mathematical model called finite automaton.

**Deterministic Finite (-state) Automata**

Informally, a DFA (Deterministic Finite State Automaton) is a simple machine that reads an input string -- one symbol at a time -- and then, after the input has been completely read, decides whether to accept or reject the input. As the symbols are read from the tape, the automaton can change its state, to reflect how it reacts to what it has seen so far. A machine for which a deterministic code can be formulated, and if there is only one unique way to formulate the code, then the machine is called deterministic finite automata.

Thus, a DFA conceptually consists of 3 parts:
1. A tape to hold the input string. The tape is divided into a finite number of cells. Each cell holds a symbol from $\Sigma$.
2. A tape head for reading symbols from the tape.
3. A control, which itself consists of 3 things:
   - finite number of states that the machine is allowed to be in (zero or more states are designated as accept or final states),
   - a current state, initially set to a start state, a state transition function for changing the current state.

An automaton processes a string on the tape by repeating the following actions until the tape head has traversed the entire string:

1. The tape head reads the current tape cell and sends the symbol $s$ found there to the control. Then the tape head moves to the next cell.
2. The control takes $s$ and the current state and consults the state transition function to get the next state, which becomes the new current state.

Once the entire string has been processed, the state in which the automation enters is examined. If it is an accept state, the input string is accepted; otherwise, the string is rejected. Summarizing all the above we can formulate the following formal definition:

DFA (Deterministic Finite Automaton) is a quintuple $M = (Q, \Sigma, \delta, q_0, F)$, where
1) $Q$ is a finite set of states
2) $\Sigma$ is a finite set of (machine) alphabet
3) $\delta$ is a transitive function from $Q \times \Sigma$ to $Q$, i.e.,
4) $\delta: Q \times \Sigma \rightarrow Q$
5) $q_0 \in Q$, is the start state
6) $F \subseteq Q$, is the set of final (accepting) states

**Example 1:**

$M = (Q, \Sigma, \delta, q_0, F)$

$Q = \{q_0, q_1\}$

$q_0$ is the start state

$F = \{q_1\}$

$\delta(q_0, 0) = q_0 \quad \delta(q_1, 0) = q_1$

$\delta(q_0, 1) = q_1 \quad \delta(q_1, 1) = q_1$
It is a formal description of a DFA. But it is hard to comprehend. For ex. The language of
the DFA is any string over \{ 0, 1\} having at least one 1

We can describe the same DFA by transition table or state transition diagram as
following:

**Transition Table :**

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>→ 0</td>
<td>q₀</td>
<td>q₁</td>
</tr>
<tr>
<td>* q₁</td>
<td>q₁</td>
<td>q₁</td>
</tr>
</tbody>
</table>

It is easy to comprehend the transition diagram.

**Explanation :** We cannot reach find state \( q₁ \) w/o or in the i/p string. There can be any
no. of 0's at the beginning. (The self-loop at \( q₀ \) on label 0 indicates it.). Similarly there
can be any no. of 0's & 1's in any order at the end of the string.

**Transition table :**

It is basically a tabular representation of the transition function that takes two arguments
(a state and a symbol) and returns a value (the “next state”).

- Rows correspond to states,
- Columns correspond to input symbols,
- Entries correspond to next states
- The start state is marked with an arrow
- The accept states are marked with a star (\( * \)).
A state transition diagram or simply a transition diagram is a directed graph which can be constructed as follows:

1. For each state in $Q$ there is a node.
2. There is a directed edge from node $q$ to node $p$ labeled $a$ iff $\delta(q, a) = p$. (If there are several input symbols that cause a transition, the edge is labeled by the list of these symbols.)
3. There is an arrow with no source into the start state.
4. Accepting states are indicated by double circle.
5. Here is an informal description how a DFA operates. An input to a DFA can be any string $w \in \Sigma^*$. Put a pointer to the start state $q$. Read the input string $w$ from left to right, one symbol at a time, moving the pointer according to the transition function, $\delta$. If the next symbol of $w$ is $a$ and the pointer is on state $p$, move the pointer to $\delta(p, a)$. When the end of the input string $w$ is encountered, the pointer is on some state, $r$. The string is said to be accepted by the DFA if $r \in F$ and rejected if $r \not\in F$. Note that there is no formal mechanism for moving the pointer.
6. A language $L \subseteq \Sigma^*$ is said to be regular if $L = L(M)$ for some DFA $M$. 

(State) Transition diagram:

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>$q_0$</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>$q_1$</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>
Regular Expression:

A method of representing language as expression is known as regular expression.

We the languages accepted by finite state automata are easily described by simple expressions.

Definition.

The regular expression over \( \Sigma \) can be defined as follows:

1) \( \emptyset \) is a regular expression which denotes the empty set.
2) \( \Sigma \) is a regular expression which denotes the set \( \{ \Sigma \} \).
3) For each \( \alpha \) in \( \Sigma \), \( \alpha \) is a regular expression and denotes the set \( \{ \alpha \} \).
4) If \( r \) and \( s \) are regular expression denoting the language \( L_1 \) and \( L_2 \) respectively then:

\[ r + s \text{ is equivalent to } L_1 \cup L_2 \text{ (union)} \]
\[ rs \text{ is equivalent to } L_1 L_2 \text{ (concatenation)} \]
\[ r^* \text{ is equivalent to } L_1^* \text{ (closure)} \]

Example:

\( \Sigma_1 = \{ a \} \) and we have regular expression \( R = a^* \)

Then \( R \) is a set denoted by

\[ R = \{ \epsilon, a, aa, aaa, aaaa \ldots \} \]
Closure Properties of Regular Languages.

A set is closed under an operation if wherever the operation is applied to members of the set, the result is also a member of the set.

The principal closure properties of regular languages are as follows:

1. The union of two regular languages is regular.
2. The concatenation of two regular languages is regular.
3. The closure of a regular language is regular.
4. The complement of a regular language is regular.
5. The intersection of two regular languages is regular.
6. The difference of two regular languages is regular.
7. The reverse of a regular language is regular.
8. A homomorphism (substitution of strings for symbols) of a regular language is regular.
9. An inverse homomorphism of a regular language is regular.
Example

Design regular expression for the language containing all the strings containing any number of "a"s and "b"s.

Solution:

\[ L = \{ \epsilon, a, \epsilon a, b, b \epsilon, b \epsilon a, b \epsilon b a, b \epsilon b a b, b \epsilon b a b b, \ldots \} \]

\[ R.E = (a + b)^* \]

Finite Automata and Regular Expression

Equivalence of NFA and Regular Expression

Let \( x \) be a regular expression, then there exists a NFA with \( \epsilon \) transitions that accepts \( L(x) \).

This theorem can be proved by induction method.

The basis of induction will be by considering \( x \) has zero operators.

Basis (Zero operators) — Now, since \( x \) has zero operators, means \( x \) can be either \( \epsilon \) or \( \{\} \) or \( \{a\} \) for some \( a \) in input set \( \Sigma \).
This theorem can be true for a number of operators, means \( n \) can be either greater than or equal to 1. The regular expression contains equal to a more than one operator in any type of regular expression there are only three cases possible. They are

- Union
- Concatenation
- Closure

**Union Case:**

Let \( r = r_1 + r_2 \) where \( r_1 \) and \( r_2 \) are regular expressions.

There exists two NFAs

\[ N_1 = \{ Q_1, \Sigma, S_1, F_1 \} \quad \text{and} \quad N_2 = \{ Q_2, \Sigma, S_2, F_2 \}. \]

- \( Q_1 \rightarrow \) set of all states in Machine \( N_1 \).
- \( Q_2 \rightarrow \) set of all states in Machine \( N_2 \).
- \( q_0 \rightarrow \) initial state.
- \( q_0 \rightarrow \) final state.
Example:

Concatenation Case:
Let $\gamma = \gamma_1 \gamma_2$, where $\gamma_1$ and $\gamma_2$ are two regular expressions. Then $M_1$ and $M_2$ denote the two machines such that $L(M_1) = L(\gamma_1)$ and $L(M_2) = L(\gamma_2)$.

Example:

Closure Case:
Let $\gamma = \gamma_1^*$ where $\gamma_1$ be a regular expression.

Example:
Construct FA for regular Expression:

\[(A+B) \cdot a + b\]

Solution:

a) "a"

![Diagram](start-i-a-f)

b) "b"

![Diagram](start-i-b-f)

c) "a + b"

![Diagram](start-i-a-e-f)

Conversion of FA to RE:

Theorem:
The regular expression can also be represented by its equivalent DFA.

Proof:
Let \( L \) be the set of the languages accepted by the DFA. The DFA can be denoted by

\[ M = \{ q_1, q_2, \ldots, q_n, \varepsilon, s, q_F \} \]
Let \( \gamma_{yz} \) denote the set of strings \( x \) such that \((q_i, x) = q_j\). The \( q_i \) and \( q_j \) indicate source state to target state.

The inputs are going through the states of finite automata means that with some input entering into the states and coming out of it. The value of \( k \) is always less than \( i \) or \( j \).

The \( \gamma_{yz} \) is denoted by,

\[
\gamma_{yz} = \gamma_{yz}^x \cup (\gamma_{yz}^{x-1})^* \gamma_{yz}^{x-1} + \gamma_{yz}^{x-1}
\]

\[
\gamma_{yz} = \{ x \in \Sigma^* | S(q_i, a) = q_j, \text{ if } i \neq j \}
\]

Each \( i, j, k \) three exists a regular expression \( \gamma_{yz}^x \). We will put the induction on \( k \) basis \( (k=0) \).

The \( \gamma_{yz} \) is a set of strings which is denoted by the language \( \gamma_{yz}^x \). It is based on either \( \varepsilon \) or single symbol. It is based on \( S(q_i, a) = q_j \).

This denotes the set of such \( a's \), if there is no such \( a \) then \( a's \) will be taken \( a's \).

If \( i = j \) the all the \( a's \) and \( \varepsilon \) will be the set.
Induction:

The formula is

\[ R_{ij}^k = R_{ij}^{(k-1)} + R_{ik}^{(k-1)} \cdot R_{kk}^{(k-1)} \cdot R_{kj}^{(k-1)} \]

To get the final regular expression, we have to simply get the language \( r_{ij}^* \), where 
\( i \) indicates start state, \( j \) indicates final state and \( n \) will be no of items.

If there are \( p \) no of paths, leading to final states, then

\[ r_{ij}^n = r_{ij}^1 + r_{ij}^2 + \ldots + r_{ij}^p \]

\( f \) is a set of final states.

\[ f = \{ r_{ji}^1, r_{ji}^2, r_{ji}^3 \} \]

Example:

Consider the following finite automata.

\[ \begin{array}{c}
\rightarrow \quad a, b \quad \rightarrow \quad \end{array} \]

\[ (q_1, q_2) \]

Find the regular expression.

Proof:

\[ R_{ij}^k = R_{ij}^{(k-1)} + R_{ik}^{(k-1)} \cdot R_{kk}^{(k-1)} \cdot R_{kj}^{(k-1)} \]

\[ R_{12}^2 = R_{12}^1 + R_{12}^1 \cdot R_{22}^1 \cdot R_{22}^2 \]
\[ \gamma_{12} = \gamma_{12}^0 + \gamma_{12}^1 (\gamma_{11}^0) \ \gamma_{12}^0 \]
\[ \gamma_{22} = \gamma_{22}^0 + \gamma_{22}^1 (\gamma_{11}^0) \ \gamma_{12}^0 \]
\[ \gamma_{11}^0 = \varepsilon + c \]
\[ \gamma_{12}^0 = a + b \]
\[ \gamma_{22}^0 = \varepsilon \]
\[ \gamma_{12}^1 = (a + b) (\varepsilon + c) (\varepsilon + c)^* (a + b) \]
\[ = (a + b) \left[ \varepsilon + (\varepsilon + c) (\varepsilon + c)^* \right] \]
\[ \therefore (\varepsilon + \varepsilon) = (a + b) \]
\[ \gamma_{22}^1 = \varepsilon + \phi (\varepsilon + c)^* (a + b) \]
\[ = \varepsilon + \phi = \varepsilon \]
\[ \gamma_{22}^1 = \varepsilon \]
\[ \gamma_{12}^2 = (a + b) c^* + (a + b) c^* (\varepsilon)^* \]
\[ = (a + b) c^* + (a + b) c^* c \varepsilon \]
\[ = (a + b) c^* + (a + b) c^* \varepsilon \]
\[ = (a + b) c^* \]
\[ \gamma_{12}^3 = (a + b) c^* \]
Proving Languages not to be regular

It is used for checking whether given string is accepted by regular expression or no, lemme theory whether the given language is regular or not.

Theorem:
"Let L be a regular set. Then there is a constant n such that if z is any word in L and |z| ≥ n we can write z = uvw such that |uvw| ≤ n, |v| ≥ 1 for all i ≥ 0 uv^i w is in L. Then n should not be greater than the number of states".

Proof:
If the language L is regular its accepted by a DFA.

Number of states = n.
Consider the input can be a_1, a_2, ..., a_m, m ≥ n. The mapping function S can be written as:

\[ S(q_0, q_1, \ldots, q_i) = q_j \]

\[ q_0 \xrightarrow{a_i \ldots a_j} q_i = q_j \xrightarrow{a_{k+1} \ldots a_m} q_m \]
If \( q_m \in \mathcal{F} \), then \( q_1, q_2, \ldots, q_m \in L(\mathcal{M}) \).

Then \( a_1, a_2, \ldots, a_j, a_{k+1}, a_{k+2}, \ldots, q_m \) is also in \( L(\mathcal{M}) \).

Since there is a path from \( q_0 \) to \( q_m \) that goes through \( q_j \) but not around the loop labelled \( a_{j+1}, \ldots, a_i \),

\[
S(q_0, a_1, a_j, a_{k+1}, \ldots, a_m) = S(S(q_0, q_1, \ldots, q_j), q_{k+1}, \ldots, q_m)
\]

\[
= S(q_j, q_{k+1}, \ldots, q_m)
\]

\[
= q_m.
\]

Hence we proved that given any long string, it can be accepted by \( \mathcal{F}A \). This is used to check whether the given language is regular or not.

Prove that \( L = \{a^p \mid p \text{ is prime} \} \) is not regular.

Let \( L \) is regular and \( p \) is prime number \( L = a^p \).

\[
|L| = |uv^i w| = 1.
\]

\[
L = uv^i w \text{ where } i = 2.
\]

\[
v = uv^i w.
\]

If we add 1 to \( p \) where \( p \leq |uv^i w| \).

\[
|p| < p + 1.
\]

\[
p + 1 \text{ is not prime number. Contradiction!}
\]

\( L \) is not regular.
closure by definition

1. closure under union, concatenation and closure (*)

Regular language are closed under
1. Union \( L(r_1) \cup L(r_2) \)
2. Concatenation \( L(r_1) \cdot L(r_2) \)
3. Star-closure \( L(r_1^*) \)

Proof

If \( L_1 \) is a regular language, then there exists some regular expression \( r_1 \) which describes it.
If \( L_2 \) is a regular language, then there exists some regular expression \( r_2 \) which describes it.

Then \( L_1 \cup L_2 = L(r_1) \cup L(r_2) = r_1 + r_2 \), \( r_1 + r_2 \) is a regular expression and therefore describes a regular language.

\( L_1 \cdot L_2 = L(r_1) \cdot L(r_2) = r_1 \cdot r_2 \), \( r_1 \cdot r_2 \) is a regular expression and therefore describes a regular language.

\( L_1^* = L(r_1^*) = r_1^* \), \( r_1^* \) is a regular expression and therefore describes a regular language.
Closure under complementation

Theorem

If $L$ is a regular language over alphabet $\Sigma$, then $\Sigma^* - L$ is also a regular language.

Proof

Let $L = L(A)$ for the DFA $A = (Q, \Sigma, \delta, q_0, F)$.

Then $\Sigma^* - L = L(B)$ where $B$ is a DFA $G = (Q, \Sigma, \delta, q_0, F)$.

$B$ is similar to $A$ but the accepting states of $A$ have become the non-accepting states of $B$ and the accepting states of $B$ have become the non-accepting states of $A$.

Then $w$ is in $L(B)$ if and only if $\delta(q_0, w)$ is in $F$ which occurs if and only if $w$ is not in $L(A)$.

The steps for converting a regular exprn to its complement are as follows:

1. Convert the regular exprn to an $\varepsilon$-NFA
2. Convert the $\varepsilon$-NFA to a DFA
3. Complement the accepting states of that DFA.
   * Change accepting states to non-accepting
   * Change non-accepting states to accepting
4. Convert the complement DFA back into regular exprn.
Example:

Find the complement of \((0+1)^*01\).

1. Convert the regular expression to NFA.

2. Convert the NFA to a DFA.

Starting state of DFA is \([q_0]\).

\[ S'(\{q_0, q_1\}) = S(q_0, 0) = \{q_0, q_1, q_2\} \]

\[ S'(\{q_0, q_1\}, 1) = S(q_0, 1) = \{q_0\} \]

\[ S'(\{q_0, q_1, q_2\}, 0) = S(q_0, 0) \cup S(q_1, 0) = \{q_0, q_1, q_2\} \]

\[ S'(\{q_0, q_1, q_2\}, 1) = S(q_0, 1) \cup S(q_1, 1) = \{q_0, q_1, q_2\} \]

\[ S'(\{q_0, q_1, q_2\}, 0) = S(q_0, 0) \cup S(q_1, 0) = \{q_0, q_1, q_2\} \]

\[ S'(\{q_0, q_1, q_2\}, 1) = S(q_0, 1) \cup S(q_1, 1) = \{q_0, q_1, q_2\} \]
Closure under Intersection

Theorem:
Let $L_1$ and $L_2$ be regular languages. Then $L_3 = L_1 \cap L_2$

Proof 1:
De Morgan's Law: $L_1 \cap L_2 = \overline{L_1} \cup \overline{L_2}$

$L_1$ and $L_2$ are regular.

So $\overline{L_1}$ and $\overline{L_2}$ are regular (closure under complementation).

So $\overline{L_1} \cup \overline{L_2}$ is regular (closure under union).

So $\overline{L_1} \cup \overline{L_2}$ is regular (closure under complementation).

So $L_1 \cap L_2$ is regular.

Proof 2:
Construction of DFA

Let $L_1 = L(M_1)$ for the DFA $M_1 = (Q_1, \Sigma, \delta_1, q_0, F_1)$

Where $Q_1 = \{q_0, v_1, \ldots, v_T\}$

Let $L_2 = L(M_2)$ for the DFA $M_2 = (Q_2, \Sigma, \delta_2, q_0, F_2)$

Where $Q_2 = \{q_0, v_1, \ldots, v_T\}$

To construct $M$ where:
$G = G_1 \times G_2$

$= \{ (q_0, v_0), (q_0, v_1), (q_0, v_2), \ldots \}$

Note that $M$ has a state for each pair of states in $M_1$ and $M_2$. 
3. Complement the accepting states of DFA.

4. Convert the DFA into regular expression.

\[ A = \varepsilon + \varepsilon A + \varepsilon C \]
\[ B = \emptyset A + \emptyset B + \emptyset C \]
\[ C = \varepsilon B \]

Substitute (3) in (2):
\[ B = CA + CB + 1B \]
\[ B = Ac + B(C + 1) \]
\[ B = A0(B + 1) \]

Substitute (4) in (3):
\[ C = A0(C + 1) \]

Substitute (5) in (4):
\[ A = \varepsilon + A1 + A0(C + 1) \]
\[ A = \varepsilon + A(1 + 0(C + 1)) \]
\[ A = (1 + 0(C + 1)) \]

\[ E = A + B = (1 + 0(C + 1)) + (1 + 0(C + 1)) \]
\[ S = \{ \text{accept with } M_1 \text{ and } M_2 \} \]

\[ F = F_1 \times F_2 \]

Start state: \([P_0, q_0]\)

\[ S([P_1, q_1], a) = \{ S([P_1, a]), S([q_1, a]) \} \]

Example:

\[ \begin{align*}
N_1 & \rightarrow \alpha \rightarrow \delta \\
N_2 & \rightarrow \beta \\
\end{align*} \]

The construct given:

\[ \begin{align*}
G &= G_1 \times G_2 \\
&= \{ [P_0, q_0], [P_0, q_1], [P_0, q_2], [P_0, q_3], [P_1, q_4], [P_1, q_5] \} \\
\end{align*} \]

\[ \begin{align*}
\Sigma &= \{ a, \beta \} \\
F &= \frac{1}{2} [P_1, q_5] \\
\end{align*} \]

Start state: \([P_0, q_0]\)

Some examples:

\[ \begin{align*}
S([P_0, q_1], 1) &= \{ S([P_0, 0]), S_2(q_2, 0) \} = [P_1, q_2] \\
S([P_1, q_2], 1) &= \{ S([P_1, 1]), S_2(q_1, 1) \} = [P_1, q_3] \\
S([P_0, q_2], 0) &= \{ S([P_0, 0]), S_2(q_0, 0) \} = [P_1, q_1] \\
S([P_0, q_3], 0) &= \{ S([P_0, 0]), S_2(q_0, 1) \} = [P_0, q_0] \\
S([P_0, q_4], 0) &= \{ S([P_0, 0]), S_2(q_0, 0) \} = [P_1, q_2] \\
S([P_0, q_5], 1) &= \{ S([P_0, 1]), S_2(q_1, 1) \} = [P_0, q_0] \\
\end{align*} \]
Final result:

Closure under difference

Theorem:
If L1 and L2 are regular language then
L₁ - L₂ exists regular.

Proof:
we know that L₁ - L₂ = H₁ \cap L₂

1) L₁ and L₂ are regular [by given]
2) \overline{L₂} is regular [closure under complementation]
3) L₁ \cap \overline{L₂} is regular [closure under intersection]
4) So L₁ - L₂ is regular.
Minimization of DFA

KR Chowdhary
Professor & Head
Email: kr.chowdhary@acm.org
webpage: www.krchowdhary.com

Department of Computer Science and Engineering
MBM Engineering College, Jodhpur

August 22, 2011

Introduction to Minimization

- Each DFA defines a unique language but reverse is not true.
- Larger number of states in FA require higher memory and computing power during implementation.
- An NFA of n states results to $2^n$ states in the equivalent DFA, therefore design of DFA is crucial.
- Minimization of a DFA refers to detecting those states whose presence or absence does not affect the language acceptability of FA.
- A reduced automata consumes lesser memory, complexity of implementation is reduced, results to faster execution time, easier to analyse.
Some definitions

Unreachable states: if $\delta^*(q_0, w) = q'$ is not true for any $w$, then $q'$ is unreachable/unaccessible state.

- **dead state**: $\forall a, a \in \Sigma$, $q$ is dead state if $\delta(q, a) = q$ and $q \in Q - F$.

- **reachability**: FA $M$ is accessible if $\exists w, w \in \Sigma^*$, and $(q_0, w) \vdash^* (q, \epsilon)$ for all $q \in Q$. $\vdash^*$ is called reachability relation.

- **Indistinguishable states**: two states are indistinguishable if their behaviour are indistinguishable with respect to each other. For example, $p, q$ are in distinguishable if $\delta^*(p, w) = \delta^*(q, w) = r \in Q$ for all $w \in \Sigma^*$.

- **k-equivalence**: $p, q$ are $k-$equivalence if:

  $\delta^*(q, w) \in F \iff \delta^*(p, w) \in F,$

  for all $w \in \Sigma^*$ and $|w| \leq k$; written as $p \sim_k q$.

  If they are equalet for all $k$, then $p \sim k$. $p \sim q$ and $p \sim_k q$ are equivalent relations.

Minimization Example

$q_6$ has no role, hence it can be removed.

$q_1, q_5$ are indistinguishable states because their behavior is identical for any string supplied at these states. These are called equivalent states, and can be merged. In merging, the state which remains will have in addition, all incoming transitions from the removed state.

Similarly, the states $q_0, q_4$ are also indistinguishable states, hence they can also be merged. $q_3$ is dead state.
Closure properties of Regular languages.

Language is formed with some operations they are union, concatenation then this also regular.

These properties are closure properties of regular languages.

Such languages represent the class of regular languages which is closed under the certain specific operations.

The properties of regular languages are:

1. The union of two languages is regular.
2. The intersection of two language is regular.
3. The complement of regular language is regular.
4. The difference of two regular language is regular.
5. The reversal of language is regular.
6. A homomorphism of language is regular.
7. The closure operation on a language is regular.
8. The inverse homomorphism of regular language is regular.
Equivalence and Minimization of Automata

Two states \( p \) and \( q \) of a DFA are equivalent if and only if \( S(p, w) \) and \( S(q, w) \) are final states for all \( w \in \Sigma^* \).

If \( S(p, w) \in F \) and \( S(q, w) \notin F \) then the states \( p \) and \( q \) are in distinguishable.

If \( S(p, w) \notin F \) and \( S(q, w) \in F \) then also there are distinguishable if there exist a string \( w \) such that one of \( S(p, w) \) and \( S(q, w) \) is final state and the other final states then these states are called distinguishable.

Minimization of DFA means reducing the number of states from given DFA.

While minimizing DFA we first find out which two states are equivalent we then replace the two states by one representative state.

The two states \( S_1 \) and \( S_2 \) are equivalent if and only if both the states are final states or both are non-final states.
Construct minimized DFA

Solution:

\[ \varepsilon\text{-closure } \{0,2,4,7\} \]
\[ \varepsilon\text{-closure } \{1,2,4\} \]
\[ \varepsilon\text{-closure } \{2,4\} \]
\[ \varepsilon\text{-closure } \{3,6,7,1,2,4\} \]
\[ \varepsilon\text{-closure } \{4,7\} \]
\[ \varepsilon\text{-closure } \{5,6,7,1,2,4\} \]
\[ \varepsilon\text{-closure } \{6,7,1,2,4\} \]
\[ \varepsilon\text{-closure } \{7,1,2,4\} \]
\[ \varepsilon\text{-closure } \{8,3\} \]
\[ \varepsilon\text{-closure } \{9,4\} \]
\[ \varepsilon\text{-closure } \{10,3\} \]

\[ S(0,a) = \varepsilon\text{-closure } (S(0,a)) \]
\[ = \varepsilon\text{-closure } (S(\varepsilon\text{-closure } (0),a)) \]
\[ = \varepsilon\text{-closure } (S(0,1,2,4,7,3,6,7,1,2,4)) \]
\[ = \{3,6,7,1,2,4\} \]

\[ S(0,b) = \varepsilon\text{-closure } (S(0,b)) \]
\[ = \varepsilon\text{-closure } (S(\varepsilon\text{-closure } (0),b)) \]
\[ = \{5,6,7,1,2,4\} \]
\[ S(1,a) = \varepsilon\text{-closure}(S(1,a)) \]
\[ = \{3, 6, 7, 1, 2, 4, 7\} \]
\[ S(1,b) = \varepsilon\text{-closure}(S(1,b)) \]
\[ = \{3, 6, 7, 1, 2, 4, 7\} \]
\[ S(2,a) = \varepsilon\text{-closure}(S(2,a)) \]
\[ = \{3, 6, 7, 1, 2, 4, 7\} \]
\[ S(2,b) = \varepsilon\text{-closure}(S(2,b)) \]
\[ = \phi \]
\[ S(3,a) = \varepsilon\text{-closure}(S(3,a)) \]
\[ = \{3, 6, 7, 1, 2, 4, 7\} \]
\[ S(3,b) = \varepsilon\text{-closure}(S(3,b)) \]
\[ = \{3, 6, 7, 1, 2, 4, 7\} \]
\[ S(4,a) = \varepsilon\text{-closure}(S(4,a)) \]
\[ = \varepsilon\text{-closure}(4,a) \]
\[ = \phi \]
\[ S(4,b) = \varepsilon\text{-closure}(S(4,b)) \]
\[ = \{3, 6, 7, 1, 2, 4, 7\} \]
\[ S(5,a) = \varepsilon\text{-closure}(S(5,a)) \]
\[ = \{3, 6, 7, 1, 2, 4, 8\} \]
\[ S(5,b) = \varepsilon\text{-closure}(S(5,b)) \]
\[ = \{3, 6, 7, 1, 2, 4\} \]
\[ S(6,a) = \{3, 6, 7, 1, 2, 4\} \]
$\delta(6, a) = \{5, 6, 7, 1, 2, 4, 8\} = A$

$\delta(7, a) = 8$

$\delta(7, b) = \emptyset$

$\delta(8, a) = 9$

$\delta(8, b) = 9$

$\delta(9, a) = 9$

$\delta(9, b) = 10$

$\delta(10, a) = \emptyset$

$\delta(10, b) = \emptyset$.

Minimized DFA:

$\delta(0, a) = \{3, 6, 7, 1, 2, 4, 8\} = A$

$\delta(0, b) = \{5, 6, 7, 1, 2, 4, 8\} = B$

$\delta(1, a) = \{3, 6, 7, 1, 2, 4, 8\} = A$

$\delta(1, b) = \{5, 6, 7, 1, 2, 4, 8\} = B$

$\delta(2, a) = \{3, 6, 7, 1, 2, 4, 8\} = A$

$\delta(2, b) = \{5, 6, 7, 1, 2, 4, 8\} = B$

$\delta(3, a) = \{3, 6, 7, 1, 2, 4, 8\} = A$

$\delta(3, b) = \{5, 6, 7, 1, 2, 4, 8\} = A$

$\delta(4, a) = \{3, 6, 7, 1, 2, 4, 8\} = A$

$\delta(4, b) = \{5, 6, 7, 1, 2, 4, 8\} = B$. 

CS6503 Theory of computation

STUDENTSFOCUS.COM
Transition Table:

<table>
<thead>
<tr>
<th>Input</th>
<th>A</th>
<th>B</th>
</tr>
</thead>
<tbody>
<tr>
<td>O</td>
<td>A</td>
<td>B</td>
</tr>
<tr>
<td>A</td>
<td>A</td>
<td>B</td>
</tr>
<tr>
<td>B</td>
<td>A</td>
<td>C</td>
</tr>
<tr>
<td>C</td>
<td>A</td>
<td>D</td>
</tr>
<tr>
<td>D</td>
<td>A</td>
<td>B</td>
</tr>
</tbody>
</table>
Important Rules:

1. \( \epsilon R = R \epsilon = R \)
2. \( \epsilon^* = \epsilon \)
3. \( \phi^* = \phi \)
4. \( \phi R = R \phi = \phi \)
5. \( \phi + R = R \)
6. \( \epsilon + R = R \)
7. \( R + R = R \)
8. \( R R^* = R^* R = R^+ \)
9. \( (R^*)^* = R^* \)
10. \( R^* (\epsilon + R) = (\epsilon + R) R^* = R^* \)
11. \( (R + \epsilon)^* = R^* \)
12. \( \epsilon + R^* = R^* \)
13. \( (R + G)^* = R^* G^* = (R^* + G^*)^* \)
14. \( (P + G)^* = P^* G^* = (P^* + G^*)^* \)
15. \( (PG)^* P = P (GP)^* \)
16. \( R^* R + R = R^* R \)
Obtain the regular expression that denotes the language accepted by the following DFA:

[Diagram of a DFA with states and transitions]

**Exercise:** Find \( R_{ij}^k \) for all the values of \( i, j, k \).

We can obtain the RE using the formula:

\[
R_{ij}^k = R_{ij}^{k-1} + R_{ik}^{k-1} (R_{kk}^{k-1})^* R_{kj}
\]

Let \( k = 0 \) (no intermediate state):

\[
R_{ij}^0 = \begin{cases} 
\varepsilon + a_1 + a_2 + \ldots + a_i & \text{if } i = j \\
2a_1 + a_2 + \ldots + a_i & \text{if } i \neq j 
\end{cases}
\]

(i) \( R_{ii}^0 = \varepsilon + 1 \)
(ii) \( R_{ij}^0 = 0 \)
(iii) \( R_{21}^0 = \emptyset \)
(iv) \( R_{22}^0 = \varepsilon + 0 + 1 \)

If \( k = 1 \):

\( R_{ii}^1 = R_{ii}^0 + R_{ii}^0 (R_{ii}^0)^* R_{ii}^0 \)

\[
= (\varepsilon + 1) + (\varepsilon + 1)(\varepsilon + 1)^* (\varepsilon + 1)
\]

\[
= \varepsilon + 1 \left[ \varepsilon + (\varepsilon + 1)(\varepsilon + 1)^* \right]
\]

\[
= (\varepsilon + 1)(\varepsilon + 1)^* 
\]

\[
= (\varepsilon + 1)^*
\]

Thus, if \( k = 1 \), there is a path from \( i \) to \( i \) through a single intermediate state.
\[(\text{ii}) \quad \gamma_{12} = \gamma_{12}^0 + \gamma_{11}^0 (\gamma_{11}^0)^* \gamma_{12}^0 \]
\[= 0 + (e+1)^* 0 \]
\[= 0 (e+1)^* \]
\[\gamma_{12}^1 = 0 (e+1)^* \]

\[(\text{iii}) \quad \gamma_{21} = \gamma_{21}^0 + \gamma_{11}^0 (\gamma_{11}^0)^* \gamma_{11}^0 \]
\[= \emptyset + \emptyset (e+1)^* (e+1) \]
\[= \emptyset + \emptyset \]
\[\gamma_{21}^1 = \emptyset \]

\[(\text{iv}) \quad \gamma_{22} = \gamma_{22}^0 + \gamma_{21}^0 (\gamma_{11}^0)^* \gamma_{12}^0 \]
\[= (e+0+1) + \emptyset (e+1)^* 1 \]
\[= e+0+1 \]

Now \(k=2\), we will calculate \(\gamma_{12}^2\) that gives us the required regular expression.
\[
\gamma_{11}^m = \gamma_{12}^2 + \gamma_{12}^1 (\gamma_{22}^1)^* \gamma_{22}^1 \gamma_{22}^1 \gamma_{22}^1 \gamma_{22}^1 \gamma_{22}^1 \]
\[= 01^* 01^* (e+0+1)^* (e+0+1) \]
\[= 01^* (e+0+1) \]
\[= 01^* (e+0+1)^* \]
\[\text{RE.} = 01^* (e+0+1)^* \]
Example: Convert the following to a regular expression:

```
1
```

Solution:

\[ R_0 = r_1^0 + r_2^0 \]

\[ r_1^0 = \epsilon \]
\[ r_2^0 = \epsilon \]
\[ r_1^0 = \emptyset \]
\[ r_2^0 = 0^+ \]
\[ r_1^0 = 1 \]
\[ r_2^0 = 1^+ \]

\[ R_1 = r_1^1 + r_2^1 \]
\[ r_1^1 = \epsilon \]
\[ r_2^1 = \epsilon + 0^+ \]
\[ r_1^1 = 1 \]
\[ r_2^1 = 1^+ \]

\[ R_2 = r_1^2 + r_2^2 \]
\[ r_1^2 = 0(00)^* \]
\[ r_2^2 = (00)^* \]
\[ r_1^2 = 0^+ \]
\[ r_2^2 = 0^+ \]

\[ R_3 = r_1^3 + r_2^3 \]
\[ r_1^3 = (00)^+ \]
\[ r_2^3 = (00)^+ \]

\[ R = R_1 + R_2 + R_3 \]

\[ R = r_1^1 + r_2^1 + r_1^2 + r_2^2 + r_1^3 + r_2^3 \]

\[ R = 0^+ (00)^+ (00)^+ (00)^+ (00)^+ (00)^+ (00)^+ \]

\[ R = 0^+ ((00)^+ (00)^+ (00)^+ (00)^+ (00)^+ (00)^+ ) \]

\[ R = 0^+ ((00)^+ (00)^+ (00)^+ (00)^+ (00)^+ (00)^+ ) \]

```

\[ R = (00)^+ (00)^+ (00)^+ (00)^+ (00)^+ (00)^+ \]

\[ R = 0^+ ((00)^+ (00)^+ (00)^+ (00)^+ (00)^+ (00)^+ ) \]

\[ R = 0^+ ((00)^+ (00)^+ (00)^+ (00)^+ (00)^+ (00)^+ ) \]
Arden’s Theorem:

Let P and Q be two regular expressions over E.

If P does not contain E, then the equation \( R = Q + RP \)
has a solution \( R = QP^* \).

Using this theorem, it is easy to find the RE.

The conditions to apply this theorem are:

1) Finite automata does not have \( E \)-moves.
2) It has only 1 start state.

Example: Construct regular expression from given DFA.

And:

```
State: q1 ----> q2 ----> q3
```

Let us write down the equation:

\( q_1 = q_1a + E \)

Since \( q_1 \) is a start state, \( E \) will be added and the input \( a \) is coming to \( q_1 \) from \( q_3 \).

State = Source state of \( 1/P * \) input coming to it.

\( q_1 = q_1a + E \) — (1)

\( q_2 = q_1b + q_2b \) — (2)

Simplify \( q_1 = q_1a \cdot \)

\( q_1 = E + q_1a \cdot \) \[ R = Q + RP \]

\( q_1 = E \cdot a^+ \) \[ R = QP^* \]

\( q_1 = a^* \) — (3)
Sub \( q_1 = a^* \) in eqn. (2):

\[
q_2 = q_1 + q_2 b
\]

\[
q_2 = a^* b + q_2 b
\]

\[
q_2 = a^* b . b^*
\]

\[
q_2 = a^* b^+ \]

From the given DFA, if we want to find out the regular expn, we normally calculate the expn for final state. Since in the given DFA \( q_2 \) is a final state & \( a^* b^+ \) is a final expn.

Find out the regular expn from given DFA:

\[
q_1 = q_1 b + q_2 \]  \( \text{--- (1)} \)
\[
q_2 = q_1 + q_2 b + q_3 \]  \( \text{--- (2)} \)
\[
q_3 = q_2 \]  \( \text{--- (3)} \)

Sub (3) in (2):

\[
q_2 = q_1 + q_2 b + q_2 b_1 \]
\[
q_2 = q_1 + q_2 (1+ b_1) \]
\[
q_2 = q_1 (1+1) \]

(4)
Sub 3 and 4 in eqn 1

\[ q_1 = q_1 + q_{20} + \varepsilon \]
\[ = q_1 + q_{200} + \varepsilon \]
\[ = q_1 + q_1 (1+0)^*00 + \varepsilon \]
\[ \eta_1 = q_1 + q_1 (0+1 (1+0)^*00) \]
\[ \eta_1 = \varepsilon \cdot [0+1 (1+0)^*00] \]  

Ex (3) Construct \( L \) for the given DFA.

![DFA Diagram]

1. \( q_1 = q_1 + \varepsilon \)
2. \( q_2 = q_1 + q_2 \)
3. \( q_2 = q_2 + q_2 (0+1) \)

Simplify eqn 1 using Arden's theorem.

\[ q_1 = q_1 + \varepsilon \]
\[ q_1 = \varepsilon + q_{10} \]
\[ q_1 = \cdot q_{0} \]
\[ q_1 = \varepsilon \]

Sub \( q_1 = 0^* \) in 2

\[ q_2 = q_1 + q_2 \]
\[ q_2 = 0^* + q_2 \]
The regular exprn $r$ given by 

$$r = q_1 + q_2 .$$

$$= 0^* + 1^* 1^* r^c \{0^* 1^* 1^* \}^c = 0^* + 01^*$$

$$\left[ RR^c = R^c \right]$$
Pumping Lemma

Any language \( L \) is not a regular language \( \text{if} \)
- for any integer \( n \geq 0 \),
- there is a string \( x \) in \( L \) such that \( |x| \geq n \),
- for any strings \( u, v \) and \( w \), such that \( x = uvw \), \( v \neq \varepsilon \), and \( |uv| \leq n \),
  -there is \( k \geq 0 \), \( uv^kw \) is not in \( L \)

Using Pumping Lemma

- Given a language \( L \).
- Let \( n \) be any integer \( \geq 0 \).
- Choose a string \( x \) in \( L \) that \( |x| \geq n \).
- Consider all possible ways to chop \( x \) into \( u, v \) and \( w \) such that \( v \neq \varepsilon \), and \( |uv| \leq n \).
- For all possible \( u, v \), and \( w \), show that there is \( k \geq 0 \) such that \( uv^kw \) is not in \( L \).
- Then, we can conclude that \( L \) is not regular.
Prove \( \{0^i 1^i \mid i \geq 0 \} \) is not regular

Let \( L = \{0^i 1^i \mid i \geq 0 \} \).
Let \( n \) be any integer \( \geq 0 \).
Let \( x = 0^n 1^n \).

Make sure that \( x \) is in \( L \) and \( |x| \geq n \).

The only possible way to chop \( x \) into \( u, v, \) and \( w \) such that
\( v \neq \epsilon \), and \( |u v| \leq n \) is:

\[ u = 0^p, \quad v = 0^q, \quad w = 0^{n-p-1} 1^n, \quad \text{where } 0 \leq q \leq n \]

Show that there is \( k \geq 0 \), \( u^k \) \( w \) is not in \( L \).

\[ u^k \ w = 0^n 0^{q(k-1)} 1^n = 0^n 1^n \text{ if } k = 1 \]

Then, \( L \) is not regular.

---

Prove \( \{0^i 1^i \mid i \geq 0 \} \) is not regular

Let \( L = \{0^i 1^i \mid i \geq 0 \} \).
Let \( n \) be any integer \( \geq 0 \), and \( m = \lfloor n/2 \rfloor \).
Let \( x = 0^m 1^m \).

Make sure that \( x \) is in \( L \) and \( |x| \geq n \).

Possible ways to chop \( x \) into \( u, v, \) and \( w \) such that \( v \neq \epsilon \), and
\( |u v| \leq n \) are:

- \( u = 0^p, \quad v = 0^q, \quad w = 0^{m-p-1} 1^m, \quad \text{where } 0 \leq q \leq m \)
- \( u = 0^p, \quad v = 0, \quad w = 0^{m-p} 1^m, \quad \text{where } 0 \leq p \leq m \)
- \( u = 0^m 1^p, \quad v = 0^q, \quad w = 1^{m-p} 1^q, \quad \text{where } 0 \leq q \leq m \)
- \( u = 0^m 1^p, \quad v = 0^q, \quad w = 1^{m-p} 0^q, \quad \text{where } 0 \leq q \leq m \)
Prove \( \{0^i1^i \mid i \geq 0 \} \) is not regular

Show that there is \( k \geq 0 \), \( u v^k w \) is not in \( L \).
- \( u = 0^p, v = 0^q, w = 0^{m-p-q} 1^m \), where where \( 0 \leq p < m \) and \( 0 < q \leq m \).
  \[ u v^k w = 0^p 0^k 0^{m-p-q} 1^m = 0^{m+q(k-1)} 1^m \text{ is not in } L \text{ if } k \neq 1. \]
- \( u = 0^p, v = 0^{m-p} 1^q, w = 1^{m-q} \), where where \( 0 \leq p < m \) and \( 0 < q \leq m \).
  \[ u v^k w = 0^p (0^{m-p} 1^q)^k 1^{m-q} \text{ is not in } L \text{ if } k \neq 1. \]
- \( u = 0^m 1^p, v = 1^q, w = 1^{m-p-q} \), where where \( 0 \leq p < m \) and \( 0 < q \leq m \).
  \[ u v^k w = 0^m 1^p 1^{qk} 1^{m-p-q} = 0^m 1^{m+q(k-1)} \text{ is not in } L \text{ if } k \neq 1. \]

Then, \( L \) is not regular.

Januloj Chongsitvatana 2201379

---

Prove \( \{1^i \mid i \text{ is prime} \} \) is not regular

Let \( L = \{1^i \mid i \text{ is prime} \} \).

Let \( n \) be any integer \( \geq 0 \).

Let \( p \) be a prime \( \geq n \), and \( w = 1^p \).

Only one possible way to chop \( w \) into \( x, y, \) and \( z \) such that \( y \neq e \), and \( |x, y| \leq n \) is:
- \( x = 1^n, y = 1^r, z = 1^{p-q} \), where \( 0 \leq q < n \) and \( 0 < r < n \).

Show that there is \( k \geq 0 \), \( x y^k z \) is not in \( L \).
- \( x y^k z = 1^n 1^k 1^{p-q} = 1^{q+r(k-1)} \).

If \( k = p+1 \), then \( p+r(k-1) = p(r+1) \), which is not a prime.

Then, \( x y^k z \) is not in \( L \).

Then, \( L \) is not regular.

Januloj Chongsitvatana 2201379
Formalism for minimization

- Identification and removal of unreachable states
- Identification and merging of indistinguishable states
- Identification and merging of dead states.
- First step: Find all reachable states \( S_R \), the non-reachable states are \( Q - S_R \).
  
  Let \( S_R = \{ s \} \); the start state
  
  while \( \exists p, p \in S_R \land \exists a, a \in \Sigma, \delta(p, a) \notin S_R \)
  
  do
  
  \( S_R = S_R \cup \delta(p, a) \)
  
  done
- A sequence is accepted if \( \delta^*(q, w) \in F \)

**Indistinguishability** is an equivalent relation. Let \( p, q, r \in Q \). Let

\( p \equiv q \), if they are indistinguishable.

\( p \equiv p \); reflexive

\( p \equiv q \iff q \equiv p \); symmetry

\( p \equiv q, q \equiv r \Rightarrow p \equiv r \); transitivity, \( \therefore \), indistinguishability is an equivalent relation.

---

Formalism for minimization

- Let \( \delta(p, a) = p' \) and \( \delta(q, a) = q' \), for \( a \in \Sigma \). If \( p', q' \) are distinguishable then so are \( p, q \)

**proof:** if \( p', q' \) are distinguishable by \( wa \) then \( p, q \) are distinguishable by string \( w \).

- Let \( x, y \in \Sigma^* \), then \( x \) and \( y \) are said to be equivalent with respect to \( L \) (i.e. \( x \approx_L y \)), if for some \( z \in \Sigma^* \), \( xy \in L \) iff \( yz \in L \).

\( \approx_L \) relation is reflexive, symmetric, and transitive, \( \therefore \), it is equivalence relation, which divides the language set \( L \) into equivalence classes.

- For a DFA \( M \); \( x, y \in \Sigma^* \) are equivalent with respect to \( M \), if \( x, y \) both derive \( M \) from a state \( q_0 \) to same state \( q' \),

\( \delta^*(q_0, x) = q' \) and \( \delta^*(q_0, y) = q' \).

\( \therefore x \approx_M y \)
minimization Algorithm

(1) Remove inaccessible/unreachable states:
   delete $Q - Q_R$, where $Q_R$ is set of accessible states.

(2) Marking unreachable states:
   i) Mark $p, q$ as distinguishable, where $p \in F, q \notin F$
   ii) For all marked pairs $p, q$ and $a \in \Sigma$
        if $\delta(p, a) = \delta(q, a)$ is already marked distinguishable then mark $p, q$ as distinguishable.

(3) construct reduced automata:
   i) Let the set of indistinguishable(equivalent) states be sets
      $[p], [q], \ldots$ such that $\forall i,j \ [p_i] \cap [q_j] = \emptyset$ and $[p_i] \cup [q_j] = Q_R$
      For each $\delta(p_i, a) = q_j$, add an edge from $[p_i]$ to $[q_j]$

(4) mark the start and final states:
    if $q_0 \in [p_i]$ then mark $[p_i]$ as start state,
    if $q_f \in F$ then mark $[q_f]$ as final state.

minimization Example #1

(1) There is no unreachable state
(2) Indistinguishable states
   $q_1, q_2$ are indistinguishable, and $q_0, q_3$ are distinguishable
(3) Reduced automata: The set of distinguishable states are:
    $[s_0] = \{q_0\}, [s_1] = \{q_1, q_2\}, [s_2] = \{q_3\}$
    Start and final states are $[s_0], [s_2]$. 
Table Filling Algorithm

- Let $M = (Q, \Sigma, \delta, s, F)$. Remove first all the non-reachable states.
- step 1. For $p \in F$ and $q \in Q - F$, put "x" in table at $(p, q)$. This shows that $p, q$ are distinguishable.
- step 2. If $\exists w$, and $\delta^*(p, w) \in F \land \delta^*(q, w) \notin F$ mark $(p, q)$ as distinguishable.
- step 3. Recursion rule:
  if $\delta^*(p, w) = r, \delta^*(q, w) = s$, and $(r, s)$ were earlier proved distinguishable, then $(p, q)$ are also distinguishable.

Table Filling minimize- example

- Consider that we have to minimize the FA shown above. The state $q_3$ is unreachable, so it can be dropped.
- Next, we mark the distinguishable states at begin as final and non-final states. and make their entries in table as $(q_2, q_0), (q_2, q_1), (q_4, q_2), (q_5, q_2), (q_6, q_2), (q_7, q_2)$ and indicate these by mark "x"
Table Filling minimize- example

<table>
<thead>
<tr>
<th>q1</th>
<th>x</th>
</tr>
</thead>
<tbody>
<tr>
<td>q2</td>
<td>x</td>
</tr>
<tr>
<td>q3</td>
<td>x</td>
</tr>
<tr>
<td>q4</td>
<td>x</td>
</tr>
<tr>
<td>q5</td>
<td>x</td>
</tr>
<tr>
<td>q6</td>
<td>x</td>
</tr>
<tr>
<td>q7</td>
<td>x</td>
</tr>
<tr>
<td>q0</td>
<td>q1</td>
</tr>
</tbody>
</table>

- Next we consider the case \( \delta(q_0, 1) = q_5, \delta(q_1, 1) = q_2 \). Since \( (q_5, q_2) \) are already marked distinguishable, therefore, \( (q_0, q_1) \) are also distinguishable.
- Like this we have filled the table shown above. unmarked are indistinguishable states.

Table Filling minimize- example

- Only states pairs which are not marked distinguishable are \( \{q_0, q_4\} \) and \( \{q_1, q_7\} \). The automata shown in figure above is reduced automata.
UNIT-I AUTOMATA

PART-A(2-MARKS)

1. List any four ways of theorem proving.
2. Define Alphabets.
3. Write short notes on Strings.
4. What is the need for finite automata?
5. What is a finite automaton? Give two examples.
6. Define DFA.
7. Explain how DFA process strings.
8. Define transition diagram.
10. Define the language of DFA.
11. Construct a finite automata that accepts \( \{0,1\}^+ \).
12. Give the DFA accepting the language over the alphabet 0,1 that have the set of all strings ending in 00.
13. Give the DFA accepting the language over the alphabet 0,1 that have the set of all strings with three consecutive 0’s.
14. Give the DFA accepting the language over the alphabet 0,1 that have the set of all strings with 011 as a substring.
15. Give the DFA accepting the language over the alphabet 0,1 that have the set of all strings whose \( n_{th} \) symbol from the right end is 1.
16. Give the DFA accepting the language over the alphabet 0,1 that have the set of all strings such that each block of 5 consecutive symbol contains at least two 0’s.
17. Give the DFA accepting the language over the alphabet 0,1 that have the set of all strings that either begins or end(or both) with 01.
18. Give the DFA accepting the language over the alphabet 0,1 that have the set of all strings such that the no of zero’s is divisible by 5 and the no of 1’s is divisible by 3.
19. Find the language accepted by the DFA given below.
20. Define NFA.
21. Define the language of NFA.
22. Is it true that the language accepted by any NFA is different from the regular language? Justify your Answer.
23. Define \( \epsilon \)-NFA.
24. Define \( \epsilon \) closure.
25. Find the \( \epsilon \) closure for each state from the following automata.
27. What are the operators of RE.
28. Write short notes on precedence of RE operators.
29. Write Regular Expression for the language that have the set of strings over \( \{a,b,c\} \) containing at least one a and at least one b.
30. Write Regular Expression for the language that have the set of all strings of 0’s
30 and 1’s whose 10th symbol from the right end is 1.
31 Write Regular Expression for the language that has the set of all strings of 0’s and 1’s with at most one pair of consecutive 1’s.
32 Write Regular Expression for the language that have the set of all strings of 0’s and 1’s such that every pair of adjacent 0’s appears before any pair of adjacent 1’s.
33 Write Regular Expression for the language that have the set of all strings of 0’s and 1’s whose no of 0’s is divisible by 5.
34 Write Regular Expression for the language that has the set of all strings of 0’s and 1’s not containing 101 as a substring.
35 Write Regular Expression for the language that have the set of all strings of 0’s and 1’s such that no prefix has two more 0’s than 1’s, not two more 1’s than 0’s.
36 Write Regular Expression for the language that have the set of all strings of 0’s and 1’s whose no of 0’s is divisible by 5 and no of 1’s is even.
37 Write Regular Expression for the language that have the set of all strings of 0’s and 1’s such that no prefix has two more 0’s than 1’s, not two more 1’s than 0’s.
38 Write Regular Expression for the language that have the set of all strings of 0’s and 1’s whose no of 0’s is divisible by 5 and no of 1’s is even.
39 Give English descriptions of the languages of the regular expression (1+ ε)(00*1)*0*.
40 Give English descriptions of the languages of the regular expression (0*1*)*000(0+1)*.
41 Give English descriptions of the languages of the regular expression (0+10)*1*.
42 Convert the following RE to ε-NFA.01*.
43 State the pumping lemma for Regular languages.
44 What are the application of pumping language?
45 State the closure properties of Regular language.
46 Prove that if L and M are regular languages then so is LUM.
47 What do you mean by Homomorphism?
48 Suppose H is the homomorphism from the alphabets {0,1,2} to the alphabets {a,b} defined by h(0)=a h(1)=ab h(2)=ba. What is h(0120) and h(21120).
49 Suppose H is the homomorphism from the alphabets {0,1,2} to the alphabets {a,b} defined by h(0)=a h(1)=ab h(2)=ba. If L is the language L(01*2) what is h(L).
50 Let R be any set of regular languages is U Ri regular?Prove it.
51 Show that the compliment of regular language is also regular.
52 What is meant by equivalent states in DFA.
Part B

1. a) If L is accepted by an NFA with ε-transition then show that L is accepted by an NFA without ε-transition.
   b) Construct a DFA equivalent to the NFA.

\[ M = (\{p,q,r\}, \{0,1\}, \delta, p, \{q,r\}) \]

Where \( \delta \) is defined in the following table.

<table>
<thead>
<tr>
<th>( \delta )</th>
<th>0</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>p</td>
<td>{q,r}</td>
<td>{q}</td>
</tr>
<tr>
<td>q</td>
<td>{r}</td>
<td>{q,r}</td>
</tr>
<tr>
<td>r</td>
<td>{s}</td>
<td>{p}</td>
</tr>
<tr>
<td>s</td>
<td>-</td>
<td>{p}</td>
</tr>
</tbody>
</table>

2. a) Show that the set \( L = \{a^n b^n / n \geq 1\} \) is not a regular. (6) b) Construct a DFA equivalent to the NFA given below: (10)

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>p</td>
<td>{p,q}</td>
<td>P</td>
</tr>
<tr>
<td>q</td>
<td>r</td>
<td>R</td>
</tr>
<tr>
<td>r</td>
<td>s</td>
<td>-</td>
</tr>
<tr>
<td>s</td>
<td>s</td>
<td>S</td>
</tr>
</tbody>
</table>

3. a) Check whether the language \( L = \{0^n 1^n / n \geq 1\} \) is regular or not? Justify your answer.
   b) Let \( L \) be a set accepted by a NFA then show that there exists a DFA that accepts \( L \).

4. Define NFA with ε-transition. Prove that if \( L \) is accepted by an NFA with ε-transition then \( L \) is also accepted by a NFA without ε-transition.

5. a) Construct a NDFA accepting all strings in \( \{a,b\}^+ \) with either two consecutive a’s or two consecutive b’s.
   b) Give the DFA accepting the following language: set of all strings beginning with a 1 that when interpreted as a binary integer is a multiple of 5.

6. Draw the NFA to accept the following languages.
   (i) Set of Strings over alphabet \( \{0,1,\ldots,9\} \) such that the final digit has appeared before. (8)
   (ii) Set of strings of 0’s and 1’s such that there are two 0’s separated by a number of positions that is a multiple of 4.
7.a) Let L be a set accepted by an NFA. Then prove that there exists a deterministic finite automaton that accepts L. Is the converse true? Justify your answer. (10)

b) Construct DFA equivalent to the NFA given below: (6)

8.a) Prove that a language L is accepted by some $\varepsilon$–NFA if and only if L is accepted by some DFA. (8)

b) Consider the following $\varepsilon$–NFA. Compute the $\varepsilon$–closure of each state and find it’s equivalent DFA. (8)

8
\begin{center}
\begin{tabular}{|c|c|c|c|}
\hline
 & $\varepsilon$ & A & b \ \\
\hline
 p & {q} & {p} & $\Phi$ & $\Phi$ \\
 q & {r} & $\Phi$ & {q} & $\Phi$ \\
 *r & $\Phi$ & $\Phi$ & {r} & \\
\hline
\end{tabular}
\end{center}

9.a) Prove that a language L is accepted by some DFA if L is accepted by some NFA.

b) Convert the following NFA to it’s equivalent DFA

\begin{center}
\begin{tabular}{|c|c|c|}
\hline
 & 0 & 1 \\
\hline
 p & {p,q} & {p} \\
 q & {r} & {r} \\
 r & {s} & $\Phi$ \\
 *s & {s} & {s} \\
\hline
\end{tabular}
\end{center}

10.a) Explain the construction of NFA with transition from any given regular expression.
b) Let \( A = (Q, \Sigma, \delta, q_0, \{q_f\}) \) be a DFA and suppose that for all \( a \) in \( \Sigma \) we have \( \delta(q_0, a) = \delta(q_f, a) \). Show that if \( x \) is a non-empty string in \( L(A) \), then for all \( k > 0 \), \( x^k \) is also in \( L(A) \).

**PART-B**

11.a) Construct an NFA equivalent to \( (0+1)^*(00+11) \)

12.a) Construct a Regular expression corresponding to the state diagram given in the following figure.

\[ \text{Diagram} \]

b) Show that the set \( E = \{0^i1^i | i \geq 1\} \) is not Regular. (6)

13.a) Construct an NFA equivalent to the regular expression \( (0+1)^*(00+11)(0+1)^* \).

b) Obtain the regular expression that denotes the language accepted by the following DFA.

\[ \text{Diagram} \]

14.a) Construct an NFA equivalent to the regular expression \( ((0+1)(0+1)(0+1))^* \)

b) Construct an NFA equivalent to \( 10+(0+11)0^*1 \)

15.a) Obtain the regular expression denoting the language accepted by the following DFA (8)

b) Obtain the regular expression denoting the language accepted by the following DFA by using the formula \( R_{ij} \)
16. a) Show that every set accepted by a DFA is denoted by a regular expression.
   b) Construct an NFA equivalent to the following regular expression: 01*+1.

17. a) Define a regular set using pumping lemma. Show that the language $L = \{0^i \text{ } / \text{ } i \text{ is an integer, } i \geq 1 \}$ is not regular.
   b) Construct an NFA equivalent to the regular expression: 10+((0+1)*01.)

18. a) Show that the set $L = \{0^n 2^n \text{ } / \text{ } n \text{ is an integer, } n \geq 1 \}$ is not regular.
   b) Construct an NFA equivalent to the following regular expression: $(10)+(0+1)^*$

19. Find whether the following languages are regular or not.
   i) $L = \{w \in \{a,b\} \mid w = w^R \}$
   ii) $L = \{0^k 1^2 2^m, n,m \geq 1 \}$
   iii) $L = \{1^k \mid k=n \text{ } . n \geq 1 \}$
   iv) $L_1/L_2 = \{x \mid \text{ for some } y \in L_2, xy \in L_1 \}$, where $L_1$ and $L_2$ are any two languages and $L_1/L_2$ is the quotient of $L_1$ and $L_2$.

20. a) Find the regular expression for the set of all strings denoted by $R^{13}$ from the
deterministic finite automata given below:

![Finite Automaton Diagram]

b) Verify whether the finite automata M1 and M2 given below are equivalent over \{a,b\}.

21.a) Construct transition diagram of a finite automaton corresponding to the regular expression \((ab+c^*)\) b.

22.a) Find the regular expression corresponding to the finite automaton given below.

![Finite Automaton Diagram]

b) Find the regular expression for the set of all strings denoted by \(R^2\) from the deterministic finite automata given below.

![Finite Automaton Diagram]

23.a) Find whether the languages \(\{ww, w \text{ is in } (1+0)^*\}\) and \(\{1^k | k=n^2, n \geq 1\}\) are regular or not.