I.C.E.™

Inspiring Creativity & Endeavour

IES | GATE | PSU

THEORY OF COMPUTATION

For

COMPUTER SCIENCE

I.C.E.
Inspiring Creativity & Endeavour
IES | GATE | PSU

THEORY OF COMPUTATION

Syllabus

Regular expressions and finite automata. Context-free grammars and push-down automata. Regular and contex-free languages, pumping lemma. Turing machines and undecidability.

ANALYSIS OF GATE PAPERS

Exam Year	1 Mark Ques.	2 Mark Ques.	Total
2003	3	6	15
2004	1	4	9
2005	-	7	14
2006	2	5	12
2007	2	5	12
2008	3	5	13
2009	4	3	11
2010	1	3	7
2011	3	3	9
2012	4	1	6
2013	2	3	8
2014 Set-1	2	2	6
2014 Set-2	2	2	6
2014 Set-3	2	2	6
2015 Set-1	1	2	5
2015 Set-2	1	3	7
2015 Set-3	1	1	3
2016 Set-1	3	3	9
2016 Set-2	3	4	11
2017 Set-1	2	4	10
2017 Set-2	3	4	11

CONTENTS

1 REGULAR LANGUAGE & FINITE AUTOMATA

1.1 ALPHABETS, STRINGS & LANGUAGES

1.1.1 LANGUAGES:

The notion of natural languages like English, Hindi, etc. is familiar to us.
Informally, language can be defined as a system suitable for expression of certain ideas, facts, or concepts, which includes a set of symbols and rules to manipulate these.
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. (symbols are the atoms of the world of languages)
A symbol is any single object such as ♠, a, 0, 1, #, begin, or do.

1.1.2 ALPHABETS:

An alphabet is a finite, nonempty set of symbols. The alphabet of a language is normally denoted byΣ. When more than one alphabets are considered for discussion, then subscripts may be used (e.g. Σ_1, Σ_2 etc) or sometimes other symbol like G may also be introduced.

1.1.2.1 STRINGS OR WORDS OVER ALPHABET

A string or word over an alphabet Σ is a finite sequence of concatenated symbols of Σ.

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.

1.1.2.2 LENGTH OF A STRING:

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

Example:
|011|=4, |11|=2, |b|=1
It is convenient to introduce a notation e for the empty string, which contains no symbols at all. The length of the empty string ε is zero, i.e., $|\varepsilon| = 0$.

1.1.2.3 SOME STRING OPERATIONS

Let $x = a_1a_2a_3 \in a_n$ and $y = b_1b_2b_3 \in b_m$ be two strings.
Concatenation: The concatenation of x and y denoted by xy, is the string $a_1a_2a_3\cdots a_n\ b_1b_2b_3\cdot\cdot\ b_m$ That is, the concatenation of x and y denoted by xy is the string that has a copy of x followed by a copy of y without any intervening space between them.

Example
Concatenation of the strings 0110 and 11 is 011011 and concatenation of the strings good and boy is good boy.

Note :
for any string w, $w\varepsilon = \varepsilon w = w$.
It is also obvious that if $|x| = n$ and $|y| = m$, then $|x + y| = n + m$.
Prefix : u is a prefix of v if v = ux for some string x.
Suffix : u is a suffix of v if v = xu for some string x.

Substring : u is a substring of v if v = xuy for some strings x and y.

Example
Consider the string 011 over the binary alphabet. All the prefixes, suffixes and substrings of this string are listed below.
Prefixes: ε, 0, 01, 011.
Suffixes: ε, 1, 11, 011.
Substrings: ε, 0, 1, 01, 11, 011.
Note that x is a prefix (suffix or substring) to x, for any string x and ε 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.

1.1.2.3 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:
$x^n = e$, if $n = 0$; otherwise $x^n = xx^{n-1}$

Example
If x = 011, then x^3= 011011011, x^1= 011 and $x^0 = e$

1.1.2.4 POWERS OF ALPHABETS:

We write $\sum^k$ (for some integer k) to denote the set of strings of length k with symbols from $\sum$. In other words, $\sum^k$ ={w| w is a string over $\sum$ and |w|=k}. Hence, for any alphabet, $\sum^0$ denotes the set of all strings of length zero. That is, $\sum^0 = \{ \varepsilon \}$. For the binary alphabet {0, 1} we have the following.
$\sum^0 = \{\varepsilon\}$
$\sum^1 = \{0,1\}$
$\sum^2 = \{00, 01, 10, 11\}$
$\sum^3 = \{000,001,010,011, 100, 101, 110, 111\}$
The set of all strings over an alphabet $\sum$ is denoted by $\sum^*$. That is,
$\sum^* = \sum^0 \cup \sum^1 \cup \sum^2 \cup \text{---} \sum^n \cup \text{---} = \cup \sum^k$
The set $\sum^*$ contains all the strings that can be generated by iteratively concatenating symbols from $\sum$ any number of times.

Example
If $\sum = \{ a, b \}$, then $\sum^* = \{ \varepsilon, a, b, aa, ab, ba, bb, aaa, aab, aba, abb, baa, \ldots\}$.
Please note that if $\sum$ = F, then $\sum^*$ that is $\emptyset=\{\varepsilon\}$. It may look odd that one can proceed from the empty set to a non-empty set by iterated concatenation. But there is a reason for this and we accept this convention. The set of all nonempty strings over an alphabet $\sum$ is denoted by $\sum^*$. That is,
$\sum^+ = \sum^0 \cup \sum^1 \cup \sum^2 \cup \text{---} \sum^* \cup \text{---} = \cup \sum^k$
Note that is infinite. It contains no infinite strings but strings of arbitrary lengths.

Reversal
For any string $w = a_1a_2a_3 \cdots a_n$ the reversal of the string is $w^R = a_n a_{n-1} \cdots a_3 a_2 a_1$ .An inductive definition of reversal can be given as follows:

1.1.3 Languages

A language over an alphabet is a set of strings over that alphabet. Therefore, a language L is any subset of $\sum^*$. That is, any $L \subseteq \sum^*$ is a language.

Example
1. F is the empty language.
2. $\sum^*$is a language for any$\sum$.
3. { ε } is a language for any $\sum$.
   Note that, $\Phi \neq \{ \varepsilon\}$. Because the language F does not contain any string but { ε } contains one string of length zero.
4. The set of all strings over {0, 1 } containing equal number of 0's and 1's.
5. The set of all strings over {a,b,c} that starts with a.

1.1.3.1 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).

Union

A string $x \in L_1 \cup L_2$ iff $x \in L_1$ or $x \in L_2$

Example

$\{ 0, 11, 01, 011 \} \cup \{ 1, 01, 110 \} = \{ 0, 11, 01, 011, 111 \}$

Intersection

A string $x \in L_1 \cap L_2$ iff $x \in L_1$ and $x \in L_2$

Example

$\{ 0, 11, 01, 011 \} \cap \{ 1, 01, 110 \} = \{ 01 \}$

Complement

Usually, $\sum^*$ is the universe that a complement is taken with respect to. Thus for a language L, the complement is L(bar) $= \{ x \in \sum^* \mid x \notin\in L \}$.

Example

Let L = {x | |x| is even}. Then its complement is the language $\{ x \in \sum^* \mid |x|$ is odd}. Similarly we can define other usual set operations on languages like relative complement, symmetric difference, etc.

1.1.3.2 REVERSAL OF A LANGUAGE

The reversal of a language L, denoted as L^R, is defined as: $L^R = \{ W^R \mid W \in L \}$.

Example

1. Let $L = \{ 0, 11, 01, 011 \}$. Then $L^R = \{ 0, 11, 10, 110 \}$.
2. Let $L = \{ 1^n0^n \mid n$ is an integer $\}$. Then $L^R = \{ 1^n0^n \mid n$ is an integer $\}$.

Language concatenation

The concatenation of languages L_1 and L_2 is defined as $L_1 L_2 = \{ xy \mid x \in L_1$ and $y \in L_2 \}$.

Example

$\{ a, ab \}\{ b, ba \} = \{ ab, aba, abb, abba \}$.

Note that,

1. $L_1 L_2 \neq L_2 L_1$ in general.
2. $L\Phi = \Phi$
3. $L\{ \varepsilon \} = L + \{ \varepsilon \}$

1.1.3.3 TERATED CONCATENATION OF LANGUAGES

Since we can concatenate two languages, we also repeat this to concatenate any number of languages. Or we can concatenate a language with itself any number of times. The operation L^n denotes the concatenation of L with itself n times. This is defined formally as follows:

$L_0 = \{ \varepsilon \}$

$L^n = LL^{n-1}$

Example

Let L = {a,ab}.

Then according to the definition, we have

$L_0 = \{ \varepsilon \}$

$L_1 = L\{ e \} = L = \{ a, ab \}$

$L_2 = LL_1 = \{a, ab \} \{a, ab\} = \{ aa, aab, aba, abab \}$

$L_3 = LL_2 = \{a, ab\} \{aa, aab, aba, abab\}$

$= \{$ aaa, aaab, aaba, aabab, abaa, abaab, ababa, ababab $\}$ and so on.

Kleene's Star operation

The Kleene star operation on a language L, denoted as L^* is defined as follows:

$L^* = ($ Union n in N $) L^n$

$= L^0 \cup L^1 \cup L^2 \cup \cdots$

$= \{ x \mid x$ is the concatenation of zero or more strings from L $\}$

Thus L^* is the set of all strings derivable by any number of concatenations of strings in L. It is also useful to define $L^+ = L L^*$, i.e., all strings derivable by one or more concatenations of strings in L. That is $L^+ =$ (Union n in N and n >0) L^n

$= L^1 \cup L^2 \cup L^3 \cup \cdots$

Example

Let $L = \{ a, ab \}$. Then we have,

$L^* = L^0 \cup L^1 \cup L^2 \cup \cdots$

$= \{e\} \cup \{a, ab\} \cup \{aa, aab, aba, abab\} \cup \ldots$

$L^+ = L^1 \cup L^2 \cup L^3 \cup \cdots$

$= \{a, ab\} \cup \{aa, aab, aba, abab\} \cup \ldots$

Note:

ε is in L^*, for every language L, including .

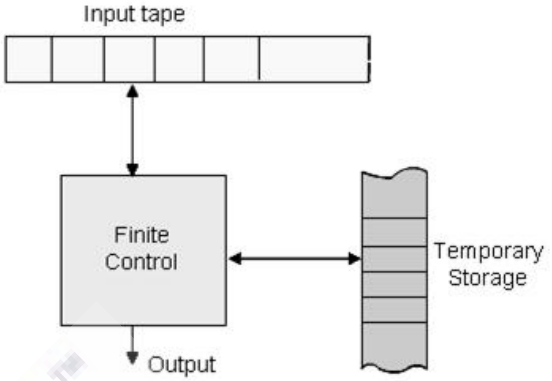
The previously introduced definition $\sum^*$ of is an instance of Kleene star.

1.2 AUTOMATA AND GRAMMARS

1.2.1 AUTOMATA

An automata is an abstract computing device (or machine). There are different verities of such abstract machines (also called models of computation) which can be defined mathematically. Some of them are as powerful in principle as today's real computers, while the simpler ones are less powerful. (Some models are considered even more powerful than any real computers as they have infinite memory and are not subject to physical constraints on memory unlike in real computers). Studying the simpler machines are still worth as it is easier to introduce some formalisms used in theory.

- 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 tape 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
- The most important feature of the automaton is its control unit, which can be in any one of a finite number of interval states at any point. It can change state in some defined manner determined by a transition function



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 does not have any temporary storage and hence a restricted model of computation.

1.2.2 GRAMMAR

A grammar is a mechanism used for describing languages. This is one of the most simple but yet powerful mechanism. There are other notions to do the same, of course.

In everyday language, like English, we have a set of symbols (alphabet), a set of words constructed from these symbols, and a set of rules using which we can group the words to construct meaningful sentences. The grammar for English tells us what are the words in it and the rules to construct sentences. It also tells us whether a particular sentence is well-formed (as per the grammar) or not. But even if one follows the rules of the English grammar it may lead to some sentences which are not meaningful at all, because of impreciseness and ambiguities involved in the language. In English grammar we use many other higher level constructs like noun-phrase, verb-phrase, article, noun, predicate, verb etc.

A typical rule can be defined as < sentence >→< noun-phrase > <predicate>

meaning that "a sentence can be constructed using a 'noun-phrase' followed by a predicate".

Some more rules are as follows:

< noun-phrase >→< article >< noun >

< predicate >→< verb >

with similar kind of interpretation given above.

If we take {a, an, the} to be <article>; cow, bird, boy, Ram, pen to be examples of <noun>; and eats, runs, swims, walks, are associated with <verb>, then we can construct the sentence- a cow runs, the boy eats, an pen walks- using the above rules. Even though all sentences are well-formed, the last one is not meaningful. We observe that we start with the higher level construct <sentence> and then reduce it to <noun-phrase>, <article>, <noun>, <verb> successively, eventually leading to a group of words associated with these constructs.

These concepts are generalized in formal language leading to formal grammars. The word 'formal' here refers to the fact that the specified rules for the language are explicitly stated in terms of what strings or symbols can occur. There can be no ambiguity in it.

Formal definitions of a Grammar .

A grammar G is defined as a quadruple.

$G = (N, \sum , P, S)$

N is a non-empty finite set of non-terminals or variables, $\sum$ is a non-empty finite set of terminal symbols such that $N \cap \sum = \emptyset$]

$S \in N$, is a special non-terminal (or variable) called the start symbol, and

$P (N \cup \sum)^{+} X (N \cup \sum)^{*}$ is a finite set of production rules.

The binary relation defined by the set of production rules is denoted by →, i.e. $\alpha \rightarrow \beta$ if

($\alpha\beta \in P$. In other words, P is a finite set of production rules of the form $\alpha \rightarrow \beta$, where

$\alpha \in (N \cup \sum)^{+}$and $\beta \in (N \cup \sum)^{*}$

The production rules specify how the grammar transforms one string to another. Given a string δαγ, we say that the production rule $\alpha \rightarrow \beta$ is applicable to this string, since it is possible to use the rule $\alpha \rightarrow \beta$ to rewrite the α (in δαγ) to β obtaining a new string δβγ We say that δαγ derives δβγ and is denoted as $\delta\alpha\gamma \Rightarrow \delta\beta\gamma$

Successive strings are derived by applying the productions rules of the grammar in any arbitrary order. A particular rule can be used if it is applicable, and it can be applied as many times as described.

We write $\alpha \Rightarrow \beta$ if the string β can be derived from the string α in zero or more steps; $\alpha \Rightarrow \beta$ if β can be derived from α in one or more steps.

By applying the production rules in arbitrary order, any given grammar can generate many strings of terminal symbols starting with the special start symbol, S, of

GATE QUESTIONS

Topics **Page No**

1 FINITE AUTOMATA: REGULAR LANGUAGES

Q.1 Consider a DFA over $\sum = \{a, b\}$ accepting all strings which have number of a's divisible by 6 and number of b's divisible By 8. What is the minimum number of states that the DFA will have?

a) 8 b) 14
c) 15 d) 48

[GATE-2001]

Q.2 Given An arbitrary Non-deterministic Finite Automaton (NFA) with N states, the maximum number of states in an equivalent minimized DFA is at least

a) N^2 b) 2^N
c) 2N d) N!

[GATE-2001]

Q.3 Consider the following languages:

$L_1 = \{ww \mid w \in \{a, b\}^*\}$

$L_2 = \{ww^R \mid w \in \{a, b\}^*, w^R \text{is the reverse of } w\}$

$L_3 = \{0^{2i} \mid i \text{ is an integer}\}$

$L_4 = \{0^{i^2} \mid i \text{ is an integer}\}$

[GATE-2001]

Q.4 Consider the following two statements:

S1- $\{0^{2n} \mid n \geq 1\}$ is regular language.

S2 - $\{0^m 1^n 0^{m+n} \mid m \geq 1 \text{ and } n \geq 1\}$ is a regular language.

Which of the following statements is correct?

a) Only S_1 is correct
b) Only S_2 is correct
c) Both S_1 and S_2 are correct
d) None of these

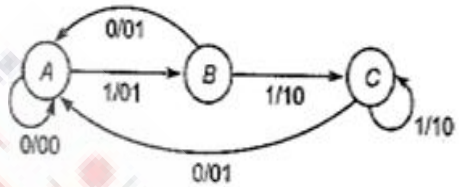
[GATE-2001]

Q.5 The smallest finite automaton which accepts the language {x | length of x is divisible by 3} has

a) 2 states b) 3 states
c) 4 states d) 5 states

[GATE-2002]

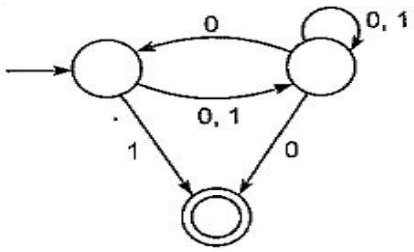
Q.6 The finite state machine described by the following state diagram with A as starting state, where an arc label is $\frac{x}{y}$ and stands for 1 -bit input and y stands for 2 bit output



a) Outputs the sum of the present and the previous bits of the input
b) Outputs 01 whenever the input sequence contains 11
c) Outputs 00 whenever the input sequence contains 10
d) None of the above

[GATE-2002]

Q.7 Consider the NFA, M, shown below:



Let the language accepted by M be L. Let L_1 be the language accepted by the NFA M_1 obtained by changing the accepting state of M to a non-accepting state and by changing the non-accepting states of M to accepting states.

Which of the following statements is true?

a) $L_1 = \{0, 1\}^* - L$ b) $L_1 = \{0, 1\}^*$
c) $L_1 \subseteq 1$ d) $L_1 = L$

[GATE-2003]

ASSIGNMENT QUESTIONS

Q.1 The word "formal" in formal languages
means
a) the symbols used have well-defined meaning
b) they are unnecessary, in reality
c) only the form of the string of symbols is significant
d) none of the above

Q.2 Let A = {0, 1} the number of possible strings of length "n" that can be formed by the elements of the set A is
a) n! b) n^2
c) n^n d) 2^n

Q.3 Choose the correct statements
a) Moore and Mealy machines are FSM's with output capability.
b) Any given Moore machine has an equivalent Mealy machine.
c) Any given Mealy machine has an equivalent Moore machine.
d) Moore machine is not an FSM.

Q.4 The major difference between a Moore and a Mealy machine is that
a) The output of the former depends on the present state and the current input
b) The output of the former depends only on the present state
c) The output of the former depends only on the current input
d) None of the above

Q.5 Choose the correct statements.
a) A Mealy machine generates no language as such
b) A Moore machine generates no language as such
c) A Mealy machine has no terminal state.
d) For a given input string, length of the output string generated by a Moore machine is one more than the length of the output string generated by that of a Mealy machine.

Q.6 The recognizing capability of NDFSM and DFSM
a) may be different
b) must be different
c) must be the same
d) none of the above

Q.7 FSM can recognize
a) any grammar
b) only CFG
c) any unambiguous grammar
d) only regular grammar

Q.8) Pumping lemma is generally used for proving
a) a given grammar is regular
b) a given grammar is not regular
c) whether two given regular expressions are equivalent
d) none of the above

Q.9 Which of the following are not regular?
a) string of 0's whose length is a perfect square
b) set of all palindromes made up of 0's and 1's
c) strings of 0's, whose length is a prime number
d) string of odd number of zeroes

Q.10 Which of the following pairs of regular expressions are equivalent?
a) 1(01) * and (10) * 1
b) x(xx) * and (xx) * x
c) (ab) * and a*b
d) x^+ and x^*x^+

Q.11 Choose the correct statements.

EXPLANATIONS

Q.6 (c)

DFSM is a special case of NDFSM. Corresponding to any given NDFSM, one can construct an equivalent DFSM. Corresponding to any given DFSM, one can construct an equivalent NDFSM. So they are equally powerful.

Q.9 (a, b, c)

Strings of odd number of zeroes can be generated by the regular expression
(0 0)*0. Pumping lemma can be used to prove the non-regularity of the other

Q.10 (a)

Two regular expression R1 and R2 are equivalent if any string that can be generated by R1 can be generated by R2 and vice-versa. In (c), (ab)* will generate abab, which is not of the form a^nb^n (because a's and b's should come together). All other s are correct (check it out)

Q.12 (a)

Pigeon-hole principle is that if 'n' balls are put in 'm' boxes, then at least one box will have more than one ball if n>m. Though this is obvious, still powerful.

Q.13 (a)

That's why it can't recognize strings of equal number of a's and b's, well-formed ness of nested parenthesis etc.

Q.17 (c)

Here the final state and start state are one and the same. No transition is there. But by definition, there is an (implicit) Ɛ-transition from any state to itself. So, the only string that could be accepted is Ɛ.

Q.22 (d)

In the second diagram, the final state is unreachable from the state. So not even Ɛ could be accepted.

Q.24 (c)

Let 011011 be the input to the FSM and let it be fed from the right (i.e., Least significant digit first). If we add 1 to 011011 we should get 011100. But did we obtain it? Whenever we add 1 to an 1, we make it 0 and carry 1 to the next stage (state) and repeat the process. If we add 1 to a 0, then first make it 1 and all the more significant digits will remain the same, i.g., a 0 will be 0 and an 1 will be 1.
That's what the given machine does. Hence the answer is (c).

Q.26 (c)

Here the initial and the final state are one and same. If you carefully examine the transition diagram, to move right you have to consume a 'b', to move left a 'b', to
go up an 'a' and to go down an 'a'. Whenever we move right, we have to move left at some stage or the other, to get back to the initial-cum-final state. This implies, a 'b' essentially has an associated another 'b'. Same is the case with 'a' (since any up (down) has a corresponding down (up)). So even number of a's and b's have to be present.

Q.29 (d)