**CS 3452 - THEORY OF COMPUTATION**

## UNIT I AUTOMATA AND REGULAR EXPRESSIONS

Need for automata theory - Introduction to formal proof – Finite Automata (FA) – Deterministic Finite Automata (DFA) – Non-deterministic Finite Automata (NFA) – Equivalence between NFA and DFA – Finite Automata with Epsilon transitions – Equivalence of NFA and DFA- Equivalence of NFAs with and without ε-moves- Conversion of NFA into DFA – Minimization of DFAs.

## PART-A

1. **Define hypothesis. R**

The formal proof can be using deductive proof and inductive proof. The deductive proof consists of sequence of statements given with logical reasoning in order to prove the first or initial statement. The initial statement is called hypothesis.

## What is structural induction? R May/June 2011

To prove a property of the elements of a recursively defined set, we use structural induction.

**Basis Step:** Show that the result holds for all elements specified in the basis step of the recursive definition.

**Inductive Step:** Assume that the property holds for the elements currently in the recursively defined set.

Show that it is true for each of the rules used to construct new elements in the recursive step of the definition.

## Define Set, Infinite and Finite Set. R

**Set** is Collection of various objects. These objects are called the elements of the set.

## Eg : A = { a, e, i, o, u }

**Infinite Set** is a collection of all elements which are infinite in number.

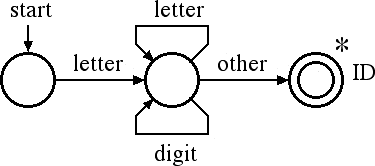
## Eg: A = {a | a is always even number}

**Finite Set** is a collection of finite number of elements. **Eg : A = { a, e, i, o, u }**

## Write any three applications of Automata Theory. U

* 1. It is base for the formal languages and these formal languages are useful of the programming languages.
  2. It plays an important role in complier design.
  3. To prove the correctness of the program automata theory is used.
  4. In switching theory and design and analysis of digital circuits automata theory is applied.
  5. It deals with the design finite state machines.

## Draw transition diagram for an identifier.

****

1. **Define Deterministic Finite Automata.**

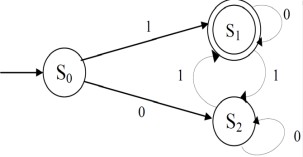
The finite automata are called DFA if there is **only one path for a specific input from current state to next state.**

A finite automata is a collection of 5 tuples (Q, Σ. δ, q0, F) where Q is a finite set of states, which is non-empty.

Σ is a input alphabet, indicates input set.

δ is a transition function or a function defined for going to next state.

q0 is an initial state (q0 in Q) F is a set of final states.



## Define Non-Deterministic Finite Automata. R Nov/Dec 2013

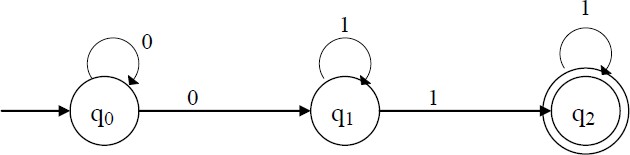
The finite automata are called NFA when there exists **many paths for a specific input from current state to next state.**

A finite automata is a collection of 5 tuples (Q, Σ. δ, q0, F ) where Q is a finite set of states, which is non-empty.

Σ is a input alphabet, indicates input set.

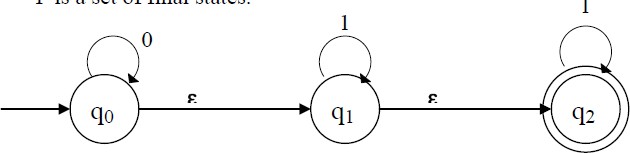
δ is a transition function or a function defined for going to next state. q0 is an initial state (q0 in Q)

F is a set of final states.



## Define NFA with € transition.

The € **is a character** used to indicate null string. i.e the string which is used simply for transition from **one state to other state without any input.**

****

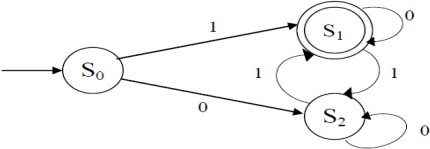
A Non Deterministic finite automata is a collection of 5 tuples (Q, Σ. δ, q0, F ) where Q is a finite set of states, which is non-empty.

Σ is a input alphabet, indicates input set.

δ is a transition function or a function defined for going to next state. q0 is an initial state (q0 in Q)

F is a set of final states.

1. **Design FA which accepts odd number of 1’s and any number of 0’s. C**

****

1. **State the difference between NFA & DFA. AN May/June 2011& May/June 2014**

**Nov/Dec 2018**

|  |  |  |
| --- | --- | --- |
| **S.NO** | **DFA** | **NFA** |
| 1 | For each input symbol there is exactly one transition out of each state. | For each input symbol there is one or more transition from a state on the same input  symbol. |
| 2 | It doesn‘t allow ξ moves | It allows ξ moves |
| 3 | 1. δ(q,ξ) =q 2. δ(q,wa) = δ(δ`(q,w),a) | 1. δ(q,ξ) =q 2. δ(q,wa) = δ(δ`(q,w),a) 3. δ(p,x) = U δ(q,w)   q in p |
| 4 | Every DFA can simulate as NFA | NFA can‘t simulate as DFA |
| 5 | Transition function mapping from Q×∑ to Q. | Transition function mapping from Q×∑ to 2Q. |

## UNIT II REGULAR EXPRESSIONS AND LANGUAGES

Regular expression – Regular Languages- Equivalence of Finite Automata and regular expressions – Proving languages to be not regular (Pumping Lemma) – Closure properties of regular languages.

**PART A**

1. **Give the regular expression for set of all strings ending in 00.**

R.E= (0+1)\*00

## State pumping lemma for regular language.

## Let L be regular language then there exist a constant n (Number of states that accept the language L) such that if W is the word or set of input string in the language L then,

* 1. Z = UVW
  2. |UV| ≤ n
  3. |V| >= 1
  4. UViW Є L For all i ≥ 0 5.

## Give the regular expression for the following C Nov/Dec 2012 L1= set of all strings of 0 and 1 ending in 00

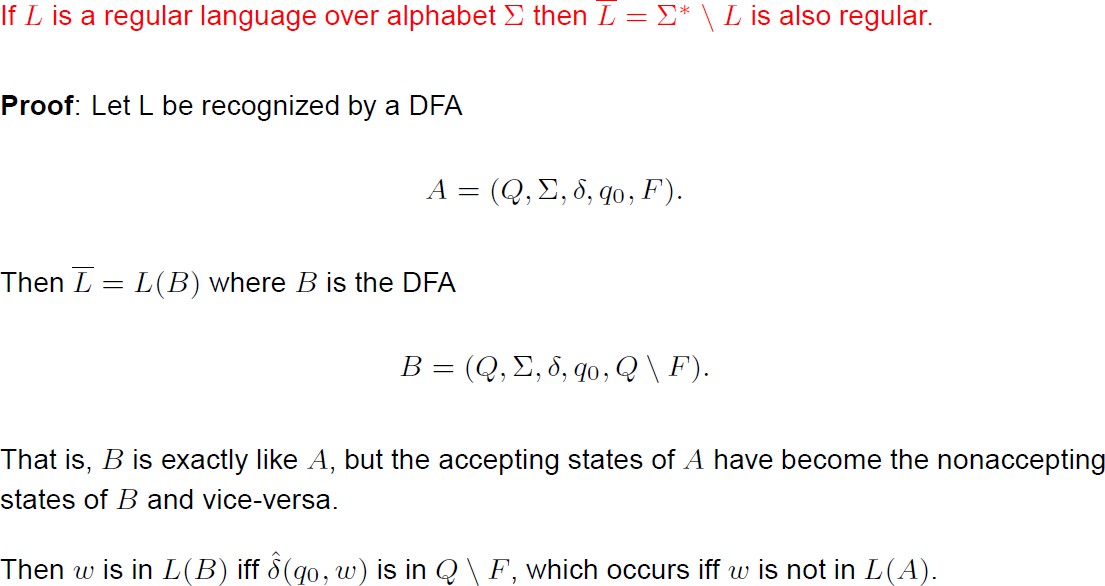
L2= set of all string 0 and 1 beginning with 0 and ending with 1

R1= (0+1)\*00 R2=0(0+1)\*1

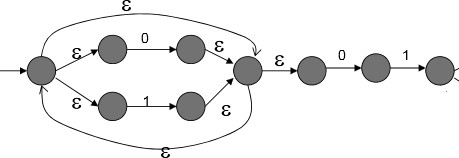
## Name any four CFG. U May/June 2013& May/June 2014 Nov-Dec 2016

* Union of two regular language is regular.
* Concatenation of regular language is regular.
* Closure of regular language is regular.
* Complement of regular language is regular.
* Intersection of regular language is regular.
* Difference of regular language is regular.
* Reversal of regular language is regular.
* Homomorphism of regular language is regular.

1. **Is regular set is closed under complement? Justify. U May/June 2012**



1. **Construct NFA for the regular expression (0+1)01 C Nov/Dec 2013**

****

**7.Prove or disprove that (r+s)\*=r\*+s\*. A Nov/Dec 2014**

Replace r by *{a}* and s by *{b}*. The left side becomes all strings of *a*'s and *b*'s (mixed), while the right side consists only of strings of *a*'s (alone) and strings of *b*'s (alone). A string like *ab* is in the language of the left side but not the right.

## 8.Give English description of the following language (0+10)\*1\*. C April/May 2015

Set of all strings of 0‘s and 1‘s including ξ

## 9.Write RE for the set of strings over {0,1} that have atleast one

(0+1)\*1(0+1)\*

## 10.Show whether a language L=(0n12n/n>0} is regular or not using pumping Lemma.

Suppose *L* is regular. We then have some *p*>0 and some |*m*|>*p apb*2*p*

*m*=*uvw*

and |*uv*|≤*p* and *uviw*∈*L* for all *i*>0 As |*uv*|≤*p*

then it follows that *v*=*al*. However, as *uviw*≡*ap+lb*2*p* it shows that as *p*+*l*≠2*p* therefore *L* is not regular.

## UNIT III CONTEXT FREE GRAMMAR AND PUSH DOWN AUTOMATA

Types of Grammar - Chomsky‗s hierarchy of languages -Context-Free Grammar (CFG) and Languages – Derivations and Parse trees – Ambiguity in grammars and languages – Push Down Automata (PDA): Definition – Moves - Instantaneous descriptions -Languages of pushdown automata – Equivalence of pushdown automata and CFG-CFG to PDA-PDA to CFG – Deterministic Pushdown Automata

## Define CFG .Give an example.

This is the way of describing language by recursive rules called production. It consists of set of variables, set of terminal symbols, and a starting variable as well as the production. G = (V,T,P,S) Where V = variables, T = Terminals, P = productions, S = starting variable.

Eg 1: 𝐺 = (𝑉, 𝑇, 𝑃, 𝑆) 𝑉 = {𝐸} 𝑇 = {+,\*, 𝑖𝑑} 𝑆 = {𝐸}

E => E+E E => E\*E E => id

## What is CFL?

If grammar G = (V,T,P,S) be a context free grammar then the language L(G) is a set of terminal strings that have derivation from the starting symbol.

L(G) = { w in T / S =>w }

\*

𝐺

* + The language generated by the CFG is called Context Free Language.

**Ex:** Find L(G) for the following grammar. a)

S => aSb / ab

S=> aSb

=> aaSbb . . S=>aSb

.

=> aaaSbbb

=> aaaaSbbbb . . S=>ab

.

. S \* aaaaaSbbbbb .

L(G) = { 𝑎𝑛𝑏𝑛 / n > 1}

. . => . .

𝐺

## What is derivation?

It is defined as α \* β where β is derived from the symbol ‗α‘ with the grammar ‗G‘.

=>

𝐺

Here, we use the production from head to body (i.e.) from starting root node expanding until it reaches the given input string.

(i.e.) R.N => w

Eg:

w=01C10

## What are the 2 types of derivation? R Left most derivation:

If at each step in derivation, a production is applied to the left most variable (or) left most non-terminal then the derivation method is called left most derivation.

Eg:

W= id+id\*id

E => E+E

E => E\*E

E => id E => E+E

E => id+E . . E=>id

.

E => id+E\*E . . E=>E\*E

.

E => id+id\*E . . E=>id

E \* id+id\*id

.

## Right most derivation:

A derivation in which the right most variable is replaced at each step then, the derivation method is called right most derivation.

Eg:

E => E+E

E => E+E\*E . . E=>E\*E

.

E => E+E\*id . . E=>id

.

E => E+id\*id

. .

. E \* id+id\*id

=>

𝐺

## What is parse tree (or) derivation tree? R

Parse tree is a pictorial representation of derivation, where the interior nodes are labeled by variables (or) non-terminals and leaf nodes are labeled by terminals symbols.

Eg:

w = 01C10

E => 0E0 E => 1E1 E => C

## Derivation:

E => 0E0

=>01E10 . . E => 1E1

.

=> 01C10 . . E => C

. . E ⇒ 01C10

.

\*

.

## Parse tree (or) derivation tree:

*E*

0

E

E

1

C

1. **What is ambiguous grammar? Or When do you say grammar is ambiguous?**

0

1

A grammar that produces more than one parse tree (or) derivation tree for some sentence, then the grammar is said to be an ambiguous grammar. An ambiguous grammar produces more than one left most derivation (or) more than 1 RMD then, the given grammar is set to be an ambiguous grammar.

1. **For the grammar defined by the productions recognize the string** 1001 **and also construct the parse tree**

S => A,B

A =>0A/ξ

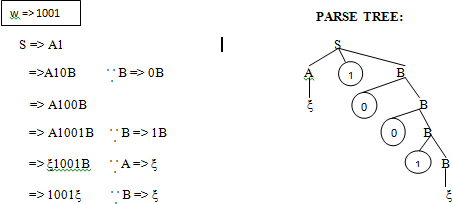
B => 0B/1B/ξ

1. **Consider the alphabet Σ = { a,b,(,),+,\*,-, . ,ξ }. Construct a CFG that generate all the strings in Σ\* that are regular expression on the alphabet, Σ. C**

**Nov/Dec 2007**

E => E+E E => E\*E E => (E) E => E.E E => E-E

E => a / b / ξ



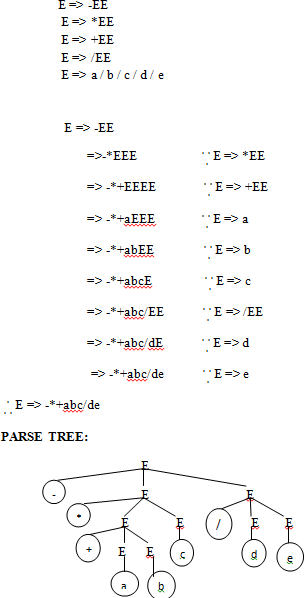
## Find LMD & RMD, parse tree for the following grammar. A May/June 2007

w = 00110101

S => 0B / 1A

A => 0/0S/1AA

B => 1/1S/0BB



## Give the general forms of CNF. (Or) State CNF. R Nov/Dec 2014, Nov-Dec 2016

Every CFL is generated by a CFG in which all productions are of the form A->BC

(or) A->a

Where A,B,C – variables a – terminals

This form of CFG is called as Chomsky Normal Form

In order to find CNF,we need to perform the following operations.

1. Eliminate useless symbols i.e.,symbols or terminals which do not appear in any derivation of a terminal string from start symbol.
2. Eliminate €-productions which are of the form A->€ form some variable A. 3.Eliminate unit production which are of the form A->B for variables A and B.

## Construct a PDA that accepts the language generated by the grammer S → aSbb

**S → aab AN**

***Solution:***

The PDA is given by

A = ({q},{a,b},{S,A,B,Z,a,b},δ,q,S}

Where δ is given by

δ(q,z,S)={(q,aABB)}

δ(q,z,A)={(q,aB),{q,a}}

δ(q,z,B)={(q,bA),(a,b)}

δ(q,a,a)={(q,ε)} δ(q,b,b)={(q,ε)}

## When pushdown automata is said to be deterministic?

If, in every situation, at most one such transition action is possible, then the automaton is called a deterministic pushdown automaton (DPDA). In general, if several actions are possible, then the automaton is called a general, or nondeterministic, PDA.

## UNIT IV

**NORMAL FORMS AND TURING MACHINES**

Normal forms for CFG – Simplification of CFG- Chomsky Normal Form (CNF) and Greibach Normal Form (GNF) – Pumping lemma for CFL – Closure properties of Context Free Languages –Turing Machine : Basic model – definition and representation – Instantaneous Description – Language acceptance by TM – TM as Computer of Integer functions – Programming techniques for Turing machines (subroutines).

## PART A

1. **What are the three ways to simplify a context free grammar?**

\_ By removing the useless symbols from the set of productions.

\_ By eliminating the empty productions.

\_ By eliminating the unit productions.

## What are the closure properties of CFG?

**Union :** If L1 and If L2 are two context free languages, their union L1 ∪ L2 will also be context free.

**Concatenation :** If L1 and If L2 are two context free languages, their concatenation L1.L2 will also be context free.

**Kleene Closure :** If L1 is context free, its Kleene closure L1\* will also be context free. **Intersection and complementation :** If L1 and If L2 are two context free languages, their intersection L1 ∩ L2 need not be context free.

## State the pumping lemma for CFL.R

**May/June 2012, Nov/Dec 2012, May/June 2014, April/May2015**

Let L be a CFL then there exist a constant M such that if Z is any word in language L and |Z| ≥ n then we may write the above statements.

By pumping lemma,

Z = UVWXY |Z| >= n

|VWX| ≤ n

|VX| >= 1

UViWXi Y Є L For all i ≥ 0

## Give the steps to eliminate useless symbols.

* 1. Find the non-generating variables and delete them, along with all productions involving non-generating variables.
  2. Find the non-reachable variables in the resulting grammar and delete them, along with all productions involving non-reachable variables.

## Show that CFLs are closed under substitutions

If L is a Context – free language over alphabet £ ,and S is a substitution on £ such that S(a) is a CFL for each a in £ ,then S(L) is a CFL.

## Proof:

The idea here is that for a CFG,replce each terminal a by the start symbol for language S(a).The result is a single CFG that generates S(L).

Let G =(V,£,P,S) be a grammer for L.

and Ga = (Va , Ta , Pa ,Sa) be a grammer for each a in £.

Construct a new grammer G1 = (V1,T1,P1,S) for S(L). Where

* V1 is the union of V and Va .[for all a in £]
* T1 is the union of all Ta.
* P1 is given by
* Pa for a in £.
* P where each terminal a is replaced by Sa.

Thus all parse trees in grammer G1 sart out with parse trees in G but all nodes have labels that are Sa for some a in £.Then the generation of each such node produces a parsertree of Ga whose yield belongs to S(a).

s

sa1

sa2

san

x1 x2 xn

## (Parse tree of G1 due to substitution)

|  |  |  |  |
| --- | --- | --- | --- |
| **7. List the closure properties CFL.** | |  | |
|  |
| * Substitutions * Union * Concatenation * Closure and Positive Closure * Homomorphism * Reversal * Intersection * Inverse Homomorphism | |  |

## Define Turing Machine.

The Turing machine is denoted by M=(Q,Σ,├,δ,q0,B,F) Where Q –finite set of states Σ -finite set of allowable tape symbols

a symbol of ├, a blank Σ- set of input symbols q0ЄQ- start state

F- set of final state

δ-Transition function mapping Q x ├ Q x ├ x{L,R}

Where L,R –Directions

## Define multitape TM. R Nov/Dec 2014, Nov/Dec 2015

A **Multi-tape Turing machine** is like an ordinary [Turing machine](http://en.wikipedia.org/wiki/Turing_machine) with several tapes.

Each tape has its own head for reading and writing. Initially the input appears on tape 1, and the others start out blank.

A k-tape Turing machine can be described as a 6-tuple M=\langle Q, \Gamma, s, b, F, \delta \rangle where:





is a finite set of states

is a finite set of the tape alphabet is the initial state

is the blank symbol

is the set of final or accepting states









* + is a partial function called the transition

function, where k is the number of tapes, L is left shift, R is right shift and S is no shift.

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Ф | 1 | 0 | 1 | 1 | $ | B | B |  |  |
| B | B | B | B | 1 | 0 | B | B | B | B……… |
|  | B | B | 1 | 0 | 1 | 1 | B | B |  |
|  |  |  |  |  |  |  |  |  |  |

1. **What are the differences between a Finite automata and a Turing machine? A**

**May-June2103**

|  |  |
| --- | --- |
| **Finite Automata** | **Turing Machine** |
| Finite Automation is a 5-tuple (Q,  ∑,δ,q0,F) where Q be a finite set of states  ∑ be a finite set of symbols  δ be a transition function mapping from Q X ∑ to Q  q0 the initial state and F the set of final state | A Turing Machine M is a 7-Tuple M = (Q,∑,┬,δ,q0,B,F) Where  Q – finite set of states  ∑ - finite set of input symbols  ┬ - finite set of tape symbols.  δ – Transition function mapping the states of finite automaton and tape symbols to states,tape symbols and movement of the head. i.e., Q x ┬ -> Q x ┬ x {L,R}  q0 ∑ Q is the intial state  F ≤ Q is the set of final states. B ∑ ┬ is the blank symbol. |

## UNIT V UNDECIDABILITY

Unsolvable Problems and Computable Functions –PCP-MPCP- Recursive and recursively enumerable languages – Properties - Universal Turing machine -Tractable and Intractable problems - P and NP completeness – Kruskal‘s algorithm – Travelling Salesman Problem- 3- CNF SAT problems.

## PART A

1. **When a language is said to be recursively enumerable?**

A language is recursively enumerable if there exists a Turing machine that accepts every string of the language and does not accept strings that are not in the language.

## Define Non Recursive language.

If the languageL is not recursively enumerable, then there is no algorithm for listing the members of L. It might be possible to define L by specifying some property that all its members satisfy, but that property can't be computable.

## When a language is said to be recursive?

A language L is said to be recursive if there exists a Turing machine M that accepts L, and goes to halt state or else M rejects L.

## Define decidable problems.

A problem is said to be decidable if there exists a Turing machine which gives one ‗yes‘ or ‗no‘ answer for every input in the language.

## Define the class P and NP. R May/June 2013, 2014 & Nov-Dec 2019

P consists of all those languages or problems accepted by some Turing machine that runs in some polynomial amount of time, as function of its input length.

NP is the class of languages or problems that are accepted by nondeterministic TM‘s with a polynomial bound on the time taken along any sequence of non – deterministic choices.

## Define NP – Complete Problem. R Nov-Dec2016

A language L is NP – complete if the following statements are true.

* 1. L is in NP.
  2. For every language L1 in NP there is a polynomial – time reduction of L1 to L.

## Write the Significance of NP-Complete Problem. R Nov-Dec2022

NP-complete languages are significant because all NP-complete languages are thought of having similar hardness, in that process solving one implies that others are solved as well. If some NP-complete languages are proven to be in P, then all of NPs are proven to be in P.

1. **What are tractable problems? R Nov-Dec2017** The problems which are solvable by polynomial – time algorithm are called tractable problems. For Eg. The complexity of the Kruskal‘s algorithm is 0(e(e+m)where e ,the number of edges and m,the number of nodes.

## Define the class P and NP.

P consists of all those languages or problems accepted by some Turing machine that runs in some polynomial amount of time, as function of its input length.

NP is the class of languages or problems that are accepted by nondeterministic TM‘s with a polynomial bound on the time taken along any sequence of non – deterministic choices.

## Define NP – Complete Problem.

A language L is NP – complete if the following statements are true.

* 1. L is in NP.
  2. For every language L1 in NP there is a polynomial – time reduction of L1 to L.

## Write the Significance of NP-Complete Problem.

NP-complete languages are significant because all NP-complete languages are thought of having similar hardness, in that process solving one implies that others are solved as well. If some NP-complete languages are proven to be in P, then all of NPs are proven to be in P.

1. **What are tractable problems?**

The problems which are solvable by polynomial – time algorithm are called tractable problems. For Eg. The complexity of the Kruskal‘s algorithm is 0(e(e+m)where e ,the number of edges and m,the number of nodes.

1. **Mention the difference between P and NP problems.**

|  |  |
| --- | --- |
| **P problems** | **NP problems** |
| P consists of all those languages or problems accepted by some Turing machine that runs in some polynomial amount of time, as function of its input length. | NP is the class of languages or problems that are accepted by nondeterministic TM‘s with a polynomial bound on the time taken along any sequence of non – deterministic choices. |

## Give examples for NP – Complete Problem.

1. Complete sub graph problem is NP-complete.

The *k*-colorability problem is *NP*-complete

.