SYLLABUS

2018-19 Onwards (MR-18)	MALLA REDDY ENGINEERING COLLEGE (Autonomous)	B.Tech. IV Semester		
Code: 80513	**FORMAL LANGUAGES AND AUTOMATA THEORY**	**L**	**T**	**P**
Credits: 3		**3**	**-**	**-**

Prerequisites: NIL

Course Objectives:

This course enable the students to define basic properties of formal languages, explain the Regular languages and grammars, inter conversion, Normalizing CFG , describe the context free grammars, minimization of CNF, GNF and PDA , designing Turing Machines and types of Turing Machines, church's hypothesis counter machines, LBA, P & NP problems and LR grammar.

MODULE I: Introduction [10 Periods]

Basics of Formal Languages - Strings, Alphabet, Language, Operations, Chomsky hierarchy of languages, Finite state machine Definitions, finite automation model, acceptance of strings and languages.

NFA and DFA - DFA and NFA, transition diagrams and language recognizers. NFA with ε transitions – Equivalence between NFA with and without ε transitions, NFA to DFA conversion, minimization FSM, equivalence between two FSM's, Output machines- Moore and Mealy machine.

MODULE II: Regular Languages [10 Periods]

Representation of Regular Expressions - Regular Sets, Regular Expressions, identity Rules, Constructing Finite automata for the given regular expressions, Conversion of Finite automata to regular expressions.

Pumping Lemma - Pumping lemma of regular sets, closure properties of regular sets (proofs not required). Regular Grammars – right linear and left linear grammars, equivalence between regular grammar and FA.

MODULE III: CNF and PDA [10 Periods]

A: Context Free Grammar - Derivation trees, sentential forms, right most and left most derivations of strings. Ambiguity in Context frees Grammars. Minimization of Context free grammars, CNF, GNF, Pumping Lemma for Context Free Languages. Enumeration properties of CFL (proofs not required).

B: Push Down Automata - Definition, model, acceptance of CFL, Acceptance by final state, acceptance by empty state and its equivalence, Equivalence of CFL and PDA (proofs not required), Introduction to DCFL and DPDA.

MODULE IV: Computable Functions [09 Periods]

Turing Machine - Definition, model, Design of TM, computable functions.

Recursive Enumerable Languages and Theorems - Recursively enumerable languages, Church's hypothesis, counter machine, types of Turing Machines (proofs not required)

MODULE V: Computability Theory [09 Periods]

Linear Bounded Automata - Linear Bounded Automata and context sensitive languages, LR (0) grammar, decidability of problems, Universal TM.
P and NP Problems - Undecidable problems about Turing Machine – Post's Correspondence Problem, The classes P and NP.

TEXT BOOKS:

1. H.E.Hopcroft, R.Motwani and J.D Ullman, **"Introduction to Automata Theory, Languages and Computations"**, Second Edition, Pearson Education, 2003.
2. KVN SUNITHA N Kalyani, "**Formal languages and Automata Theory",** Pearson Education

REFERENCES:

1. H.R.Lewis and C.H.Papadimitriou, **"Elements of The theory of Computation"**, Second Edition, Pearson Education/PHI, 2003
2. J.Martin, **"Introduction to Languages and the Theory of Computation"**, Third Edition, TMH, 2003.
3. Micheal Sipser, **"Introduction of the Theory and Computation"**, Thomson Brokecole, 1997.

E-RESOURCES:

1. https://books.google.co.in/books?isbn=8184313020
2. https://www.iitg.ernet.in/dgoswami/Flat-Notes.pdf
3. http://www.jalc.de/
4. https://arxiv.org/list/cs.FL/0906
5. http://freevideolectures.com/Course/3379/Formal-Languages-and-Automata-Theory
6. http://nptel.ac.in/courses/111103016/

Course Outcomes:

At the end of the course, students will be able to

1. **Define** the theory of automata types of automata and FA with outputs.
2. **Differentiate** regular languages and applying pumping lemma.
3. **Classify** grammars checking ambiguity able to apply pumping lemma for CFL various types of PDA.
4. **Illustrate** Turing machine concept and in turn the technique applied in computers.
5. **Analyze** P vs NP- Class problems and NP-Hard vs NP-complete problems, LBA, LR Grammar, Counter machines, Decidability of Problems.

CO- PO,PSO Mapping (3/2/1 indicates strength of correlation) 3-Strong, 2-Medium, 1-Weak															
COs	**Programme Outcomes(POs)**												**PSOs**		
	PO 1	**PO 2**	**PO 3**	**PO 4**	**PO 5**	**PO6**	**PO7**	**PO8**	**PO9**	**PO10**	**PO11**	**PO1 2**	**PSO 1**	**PSO 2**	**PSO3**
CO1	3	2	2									2	2	2	
CO2		2	2	2	2							2	2	2	
CO3		2	2	2	2							2	2	2	
CO4		2	2	2	2							2	2	2	
CO5		2	2	2	2							2	2	2	

FORMAL LANGUAGES AND AUTOMATA THEORY

This is an introductory course on formal languages, automata, computability and related matters. These topics form a major part of what is known as the theory of computation.

The **theory of computation** or **computer theory** is the branch of computer science and mathematics that deals with whether and how efficiently problems can be solved on a model of computation, using an algorithm. The field is divided into two major branches: computability theory and complexity theory, but both branches deal with formal models of computation.

The purpose of this course is to acquaint the student with an overview of the theoretical foundations of computer science from the perspective of formal languages.

- Classify machines by their power to recognize languages.
- Employ finite state machines to solve problems in computing.
- Explain deterministic and non-deterministic machines.
- Comprehend the hierarchy of problems arising in the computer sciences.

MOTIVATION

- Automata = abstract computing devices.
- Turing studied Turing Machines (=computers) before there were any real computers.
- We will also look at simpler devices than Turing machines (Finite State Automata, Push-down Automata, . . .), and specification means, such as grammars and regular expressions.
- NP-hardness = what cannot be efficiently computed

COURSE DESCRIPTION

This course will provide a foundation to the "Theory of Computation". The student will realize that the sometimes chaotic technology oriented world of computers has a very elegant mathematical basis to it. This basis is deeply rooted in mathematics developed before the days of modern computers. Our study will lead to some interesting implications concerning the theoretical limits of computing. On the practical side, this course is a background for a course on compilers. Topics covered in this course include: mathematical prerequisites, finite state machines (automata), concept of a language and grammars, deterministic and non-deterministic accepters, regular expressions and languages, context-free languages, normal/canonical forms, pushdown automata, Turing machines, context sensitive languages, recursive and recursively enumerable languages. Each of the language classes has two points of view: a class of automata defining the language, and a class of grammars defining the language. This dual approach to defining languages, will finally lead to the Chomsky hierarchy of languages. We shall observe that the Turing Machine not only serves to define a language class, but also a mathematical model for computation itself and defines the theoretical limits of computation.

Prerequisites

- Set theory:
  - Sets and operations on sets
  - Relations and classification of relations
  - Equivalence relations and partitions
  - Functions operations of functions
  - Fundamentals of logic
- Graph theory
- Algorithms and data structures at the level of an introductory programming sequence.
- Mathematical induction and its applications

Instructional Learning Outcomes

S.No.	Unit	Contents	Outcomes
1.	**I**	**Fundamentals :** Strings, Alphabet, Language, Operations, Finite state machine, definitions, finite automaton model, acceptance of strings, and languages, deterministic finite automaton and non deterministic finite automaton, transition diagrams and Language recognizers.	At the end of the chapter the student will be • Able to manipulate strings on a given alphabet by applying the operations there on. • Able to visualize languages and finite state machines and their equivalence. • Able to tell languages by the FSMs. • Able to differentiate Deterministic and Non-Deterministic automata. • Able to know the importance of finite automata in compiler design.

		Finite Automata: NFA with null transitions - Significance, acceptance of languages. Conversions and Equivalence: Equivalence between NFA with and without null transitions, NFA to DFA conversion, minimization of FSM, equivalence between two FSM's, Finite Automata with output- Moore and Mealy machines.	At the end of the chapter the sudent will be • Able to design NFA with null transitions for a given language. • Able to convert and prove equivalence between NFA and NFA without null transitions. • Able to minimize FSMs. • Able to design finite automata with outputs and prove their equivalence.

2	II	Regular Languages: Regular sets, regular	At the end of the chapter student will be • Able to know the importance of regular sets & expressions • Able to construct FAs for REs and vice versa. • Able to use pumping lemma for show that a language is not reguar.
		expressions, identity	
		rules, Constructing	
		finite Automata for a	
		given regular expressions,	

		Conversion of Finite	
		Automata to Regular	
		expressions. Pumping	
		lemma of regular sets,	
		closure properties of	
		regular sets	

		Grammar Formalism : Regular grammars-right linear and left linear grammars, equivalence between regular linear grammar and FA, inter conversion, Context free grammar, derivation trees, and sentential forms. Rightmost and leftmost derivation of strings.	At the end of the chapter the student will be able to • Write regular grammar for regular language and be able to differentiate between left linear & right linear grammars. • Prove the equivalence between regular linear grammar and FA • Define CFG. • Derive (L&R) of strings for given CFG.
3	III	Context Free Grammars: Ambiguity in context free grammars. Minimization of Context Free Grammars. Chomsky normal form, Greibach normal form, Pumping Lemma for	At the end of the chapter the student will be able to • Know the cause of ambiguity in CFG & minimize CFG. • Write CFG in the normal forms. • Use pumping lemma to
		Context Free Languages. Enumeration of properties of CFL	prove that a language is not a CFL.

		Push Down Automata: Push down automata, definition, model, acceptance of CFL, Acceptance by final state and acceptance by empty state and its equivalence. Equivalence of CFL and PDA, interconversion. Introduction to DCFL and DPDA.	At the end of the chapter the student will be able to • Define and design a PDA for a given CFL. • Prove the equivalence of CFL and PDA and their inter-conversions. • Differentiate DCFL and DPDA
4	**IV**	**Turing Machine :** Turing Machine, definition, model, design of TM, Computable functions, recursively enumerable languages. Church's hypothesis, counter machine, types of Turing machines. , linear bounded automata and context sensitive language. .	At the end of the chapter the student will be able to • Define and design TM for a given computation, a total function, or a language. • Convert algorithms into Turing Machines. • Arrange the machines in the hierarchy with respect to their capabilities.

5	**V**	**Computability Theory:** Chomsky hierarchy of languages, decidability of problems, Universal Turing machine, undecidability of posts correspondence problem, Turing reducibility, Definition of P and NP Problems, NP complete and NP hard problems.	At the end of the chapter the student will be able to • Know the hierarchy of languages and grammars. • Know decidability of problems. • Genralize Turing Machines into universal TMs • Classify P and NP (complete & hard) Problems.

UNIT I:

Fundamentals

- **Symbol** – An atomic unit, such as a digit, character, lower-case letter, etc. Sometimes a word. *[Formal language does not deal with the "meaning" of the symbols.]*

- **Alphabet** – A finite set of symbols, usually denoted by Σ.
  Σ = {0, 1}   Σ = {0, a, 9, 4}   Σ = {a, b, c, d}

- **String** – A finite length sequence of symbols, presumably from some alphabet.
  w = 0110   y = 0aa   x = aabcaa   z = 111

  Special string: ε (also denoted by λ)
  Concatenation: wz = 0110111
  Length: |w| = 4 |ε| = 0 |x| = 6
  Reversal: y^R = aa0

- Some special sets of strings:
  Σ^* All strings of symbols from Σ
  Σ^+ Σ^* - {ε}

- Example:
  Σ = {0, 1}
  Σ^* = {ε, 0, 1, 00, 01, 10, 11, 000, 001,…}
  Σ^+ = {0, 1, 00, 01, 10, 11, 000, 001,…}

- A **language** is:
  1) A set of strings from some alphabet (finite or infinite). In other words,
  2) Any subset L of Σ^*

- Some special languages:
  {}   The empty set/language, containing no string.
  {ε}   A language containing one string, the empty string.

- Examples:
  Σ = {0, 1}
  L = {x | x is in Σ^* and x contains an even number of 0's}

  Σ = {0, 1, 2,…, 9, .}
  L = {x | x is in Σ^* and x forms a finite length real number}
  = {0, 1.5, 9.326,…}

  Σ = {a, b, c,…, z, A, B,…, Z}
  L = {x | x is in Σ^* and x is a Pascal reserved word}

= {BEGIN, END, IF,…}

Σ = {Pascal reserved words} U { (,), ., :, ;,…} U {Legal Pascal identifiers}
L = {x | x is in Σ^* and x is a syntactically correct Pascal program}

Σ = {English words}
L = {x | x is in Σ^* and x is a syntactically correct English sentence}

Finite State Machines

- A finite state machine has a set of states and two functions called the next-state function and the output function
  - The set of states correspond to all the possible combinations of the internal storage
    - If there are n bits of storage, there are 2^n possible states
  - The next state function is a combinational logic function that given the inputs and the current state, determines the next state of the system
- The output function produces a set of outputs from the current state and the inputs
  - There are two types of finite state machines
  - In a Moore machine, the output only depends on the current state
  - While in a Mealy machine, the output depends both the current state and the current input
  - We are only going to deal with the Moore machine.
  - These two types are equivalent in capabilities
- A Finite State Machine consists of:

K states: $S = \{s_1, s_2, \dots, s_k\}$, s_1 is initial
state N inputs: $I = \{i_1, i_2, \dots, i_n\}$
M outputs: $O = \{o_1, o_2, \dots, o_m\}$
Next-state function T(S, I) mapping each current state and input to next state
Output Function P(S) specifies output

Finite Automata

- Two types – both describe what are called regular languages
  - – Deterministic (DFA) – There is a fixed number of states and we can only be in one state at a time
  - – Nondeterministic (NFA) –There is a fixed number of states but we can be in multiple states at one time
- While NFA's are more expressive than DFA's, we will see that adding nondeterminism does not let us define any language that cannot be defined by a DFA.
- One way to think of this is we might write a program using a NFA, but then when it is "compiled" we turn the NFA into an equivalent DFA.

Formal Definition of a Finite Automaton

1. Finite set of states, typically Q.
2. Alphabet of input symbols, typically ∑
3. One state is the start/initial state, typically q0 // $q0 \in Q$
4. Zero or more final/accepting states; the set is typically F. // $F \subseteq Q$
5. A transition function, typically δ.
   This function
   - Takes a state and input symbol as arguments.

Deterministic Finite Automata (DFA)

- A DFA is a five-tuple: $M = (Q, \Sigma, \delta, q0, F)$

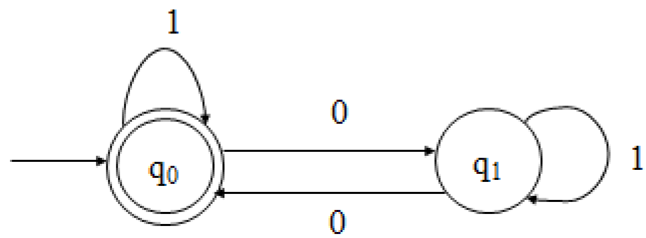
Q	A <u>finite</u> set of states
Σ	A <u>finite</u> input alphabet
q0	The initial/starting state, q0 is in Q
F	A set of final/accepting states, which is a subset of Q
δ	A transition function, which is a total function from Q x Σ to Q

$\delta: (Q \times \Sigma) \to Q$ δ is defined for any q in Q and s in Σ, and
$\delta(q,s) = q'$ is equal to another state q' in Q.

Intuitively, δ(q,s) is the state entered by M after reading symbol s while in state q.

- For Example #1:

$Q = \{q_0, q_1\}$
$\Sigma = \{0, 1\}$
Start state is q_0
$F = \{q_0\}$



δ:

	0	1
q_0	q_1	q_0
q_1	q_0	q_1

- Let $M = (Q, \Sigma, \delta, q_0, F)$ be a DFA and let w be in Σ^*. Then w is *accepted* by M iff $\delta(q_0,w) = p$ for some state p in F.

- Let $M = (Q, \Sigma, \delta, q_0, F)$ be a DFA. Then the *language accepted* by M is the set:

  $$L(M) = \{w \mid w \text{ is in } \Sigma^* \text{ and } \delta(q_0,w) \text{ is in } F\}$$

- Another equivalent definition:

  $$L(M) = \{w \mid w \text{ is in } \Sigma^* \text{ and } w \text{ is accepted by } M\}$$

- Let L be a language. Then L is a *regular language* iff there exists a DFA M such that L = L(M).

- Let M = (Q , Σ , δ , q , F) and M = (Q , Σ , δ , p , F) be DFAs. Then M and M are

1 1 1 1 0 1 2 2 2 2 0 2 1 2

equivalent iff $L(M_1) = L(M_2)$.

- Notes:
  - A DFA $M = (Q, \Sigma, \delta, q_0, F)$ partitions the set Σ^* into two sets: $L(M)$ and $\Sigma^* - L(M)$.
  - If $L = L(M)$ then L is a subset of $L(M)$ and $L(M)$ is a subset of L.
  - Similarly, if $L(M_1) = L(M_2)$ then $L(M_1)$ is a subset of $L(M_2)$ and $L(M_2)$ is a subset of $L(M_1)$.
  - Some languages are regular, others are not.

    For example, if

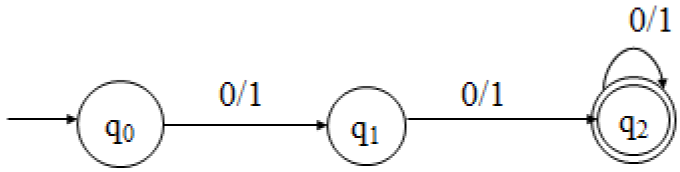
    $L_1 = \{x \mid x$ is a string of 0's and 1's containing an even number of 1's$\}$ and

    $L_2 = \{x \mid x = 0^n1^n$ for some $n >= 0\}$

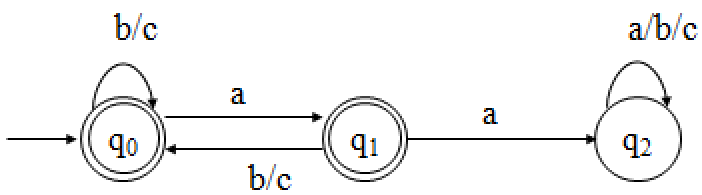
    then L_1 is regular but L_2 is not.

- Give a DFA M such that:

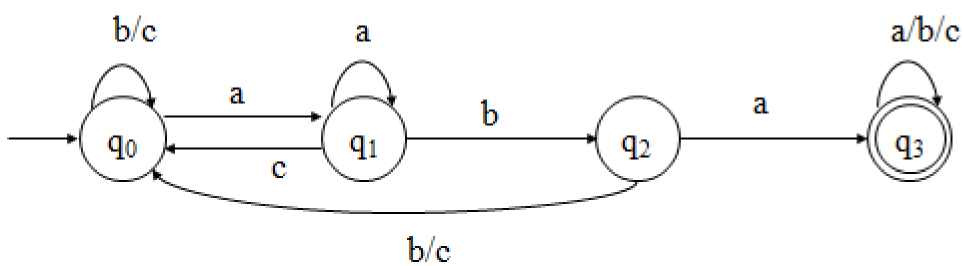
L(M) = {x | x is a string of 0's and 1's and |x| >= 2}



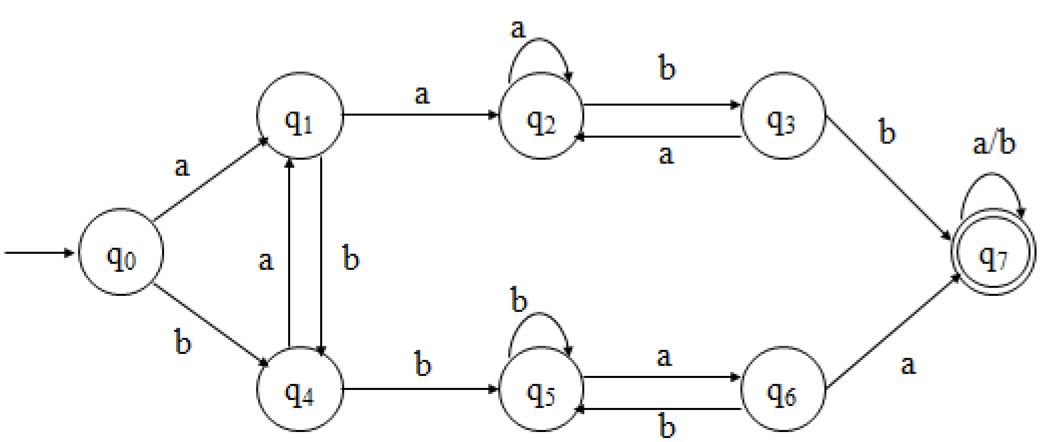
L(M) = {x | x is a string of (zero or more) a's, b's and c's such that x does not contain the substring *aa*}



L(M) = {x | x is a string of a's, b's and c's such that x contains the substring *aba*}

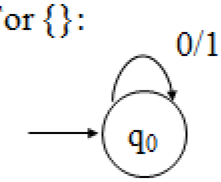


L(M) = {x | x is a string of a's and b's such that x contains both *aa* and *bb*}

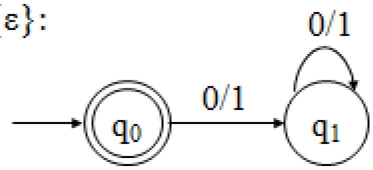


- Let $\Sigma = \{0, 1\}$. Give DFAs for $\{\}$, $\{\varepsilon\}$, Σ^*, and Σ^+.

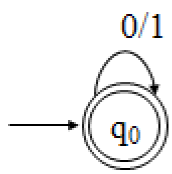
For $\{\}$:



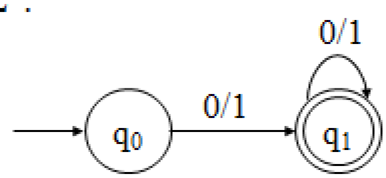
For $\{\varepsilon\}$:



For Σ^*:



For Σ^+:



<u>**Nondeterministic Finite Automata (NFA)**</u>

- An NFA is a five-tuple: $M = (Q, \Sigma, \delta, q0, F)$

Q	A <u>finite</u> set of states
Σ	A <u>finite</u> input alphabet
q0	The initial/starting state, q0 is in Q
F	A set of final/accepting states, which is a subset of Q
δ	A transition function, which is a total function from Q x Σ to 2^Q

δ: (Q x Σ) -> $\mathbf{2^Q}$ -2^Q is the power set of Q, the set of all subsets of Q

$\delta(q,s)$ -The set of all states p such that there is a transition labeled s from q to p

$\delta(q,s)$ is a function from Q x S to 2^Q (but not to Q)

- Let $M = (Q, \Sigma, \delta, q0, F)$ be an NFA and let w be in Σ^*. Then w is *accepted* by M iff $\delta(\{q0\}, w)$ contains at least one state in F.

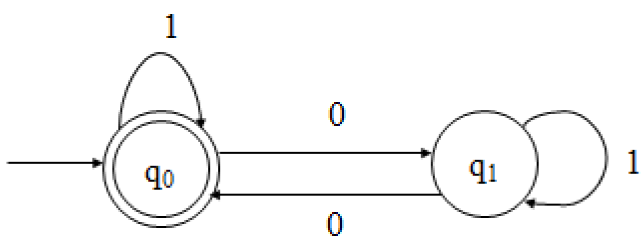
- Let M = (Q, Σ, δ,q0,F) be an NFA. Then the *language accepted* by M is the set:
  L(M) = {w | w is in Σ* and δ({q0},w) contains at least one state in F}

- Another equivalent definition:
  L(M) = {w | w is in Σ* and w is accepted by M}

TRANSITION DIAGRAMS

- The finite control can be described by a transition diagram:

  Example #1:

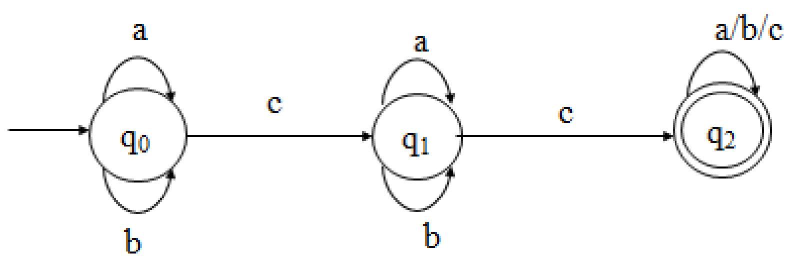


1 0 0 1 1

q_0 q_0 q_1 q_0 q_0 q_0

- One state is final/accepting, all others are rejecting.
- The above DFA accepts those strings that contain an even number of 0's

Example #2:



a c c c b accepted

q_0 q_0 q_1 q_2 q_2 q_2

a a c rejected

q_0 q_0 q_0 q_1

- Accepts those strings that contain at least two c's

<u>NFAs with ε Moves</u>

- An NFA-ε is a five-tuple: M = (Q, Σ, δ, q0, F)

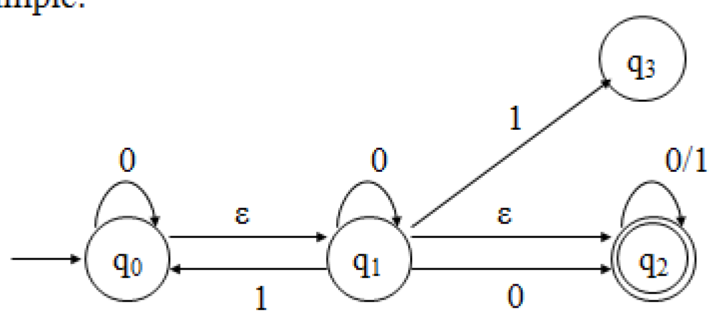
Q	A <u>finite</u> set of states
Σ	A <u>finite</u> input alphabet
q0	The initial/starting state, q0 is in Q
F	A set of final/accepting states, which is a subset of Q
δ	A transition function, which is a total function from Q x Σ U {ε} to 2^Q

δ: (Q x (Σ U {ε})) –> 2^Q

δ(q,s) -The set of all states p such that there is a transition labeled a from q to p, where a is in Σ U {ε}

- Sometimes referred to as an NFA-ε other times, simply as an NFA.

- Example:



δ:

	0	1	ε
q_0	$\{q_0\}$	{ }	$\{q_1\}$
q_1	$\{q_1, q_2\}$	$\{q_0, q_3\}$	$\{q_2\}$
q_2	$\{q_2\}$	$\{q_2\}$	{ }
q_3	{ }	{ }	{ }

- A string $w = w_1w_2\ldots w_n$ is processed as $w = \varepsilon^*w_1\varepsilon^*w_2\varepsilon^* \ldots \varepsilon^*w_n\varepsilon^*$

- Example: all computations on 00:

$$\begin{array}{cccc} & 0 & \varepsilon & 0 \\ q_0 & q_0 & q_1 & q_2 \end{array}$$

:

- Let M = (Q, Σ, δ,q0,F) be an NFA-ε and let w be in Σ^*. Then w is *accepted* by M iff $\delta^\wedge$({q0}, w) contains at least one state in F.

- Let M = (Q, Σ, δ,q0,F) be an NFA-ε. Then the *language accepted* by M is the set: L(M) = {w | w is in Σ^* and $\delta^\wedge$({q0},w) contains at least one state in F}

- Another equivalent definition:
  L(M) = {w | w is in Σ^* and w is accepted by M}

Equivalence of NFA and NFA-ε

- Do NFAs and NFA-ε machines accept the same *class* of languages?
  - Is there a language L that is accepted by a NFA, but not by any NFA-ε?
  - Is there a language L that is accepted by an NFA-ε, but not by any DFA?

- Observation: Every NFA is an NFA-ε.

- Therefore, if L is a regular language then there exists an NFA-ε M such that L = L(M).

- It follows that NFA-ε machines accept all regular languages.

- But do NFA-ε machines accept more?

- **Lemma 1:** Let M be an NFA. Then there exists a NFA-ε M' such that L(M) = L(M').

- **Proof:** Every NFA is an NFA-ε. Hence, if we let M' = M, then it follows that L(M') = L(M).

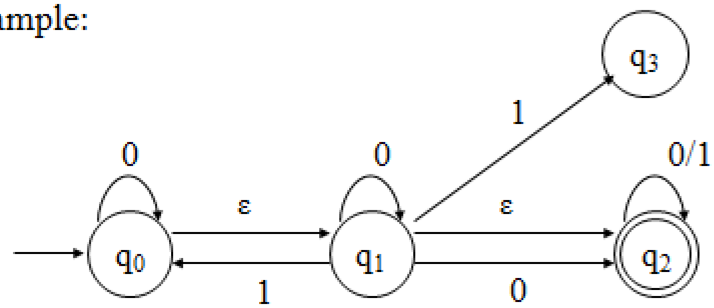
- **Lemma 2:** Let M be an NFA-ε. Then there exists a NFA M' such that L(M) = L(M').

- **Proof:**
  Let M = (Q, Σ, δ,q0,F) be an NFA-ε.
  Define an NFA M' = (Q, Σ, δ',q0,F')
  as:

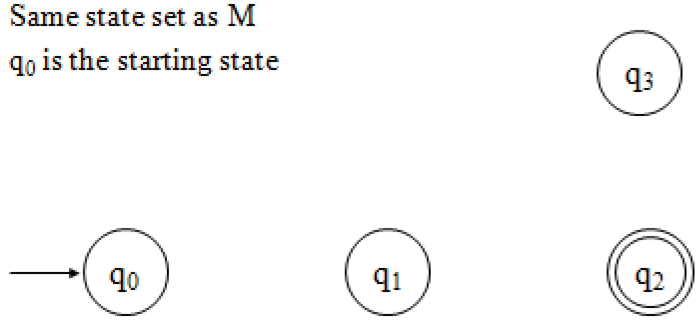
  F' = F U {q0} if ε-closure(q0) contains at least one state from
  F F' = F otherwise
  $\delta'(q, a) = \delta^{\wedge}(q, a)$ - for all q in Q and a in Σ

- Notes:
  - δ': (Q x Σ) –> 2^Q is a function
  - M' has the same state set, the same alphabet, and the same start state as M
  - M' has no ε transitions

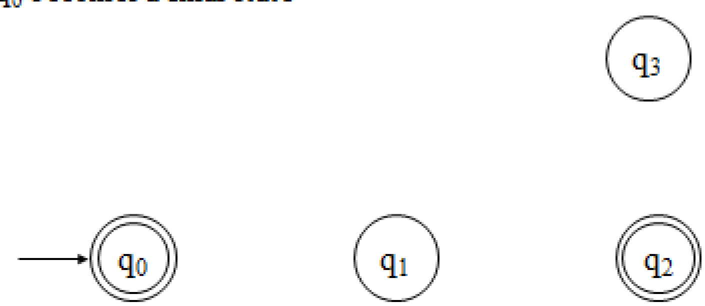
- Example:



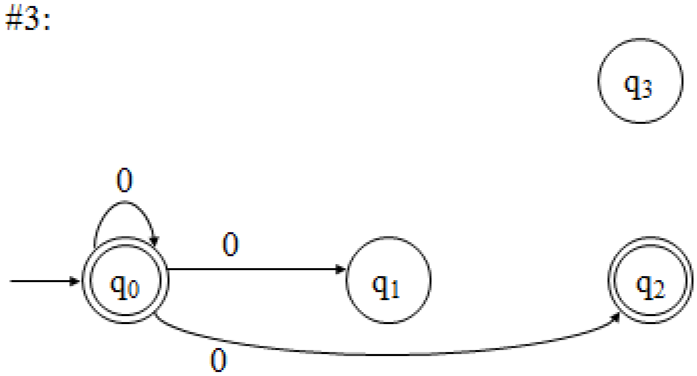
- Step #1:
  - Same state set as M
  - q_0 is the starting state



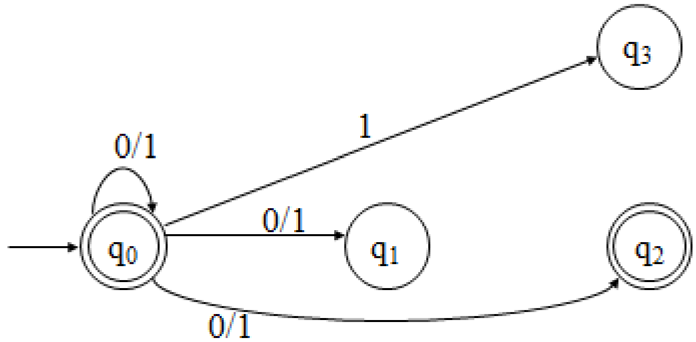
- Step #2:
  - q_0 becomes a final state



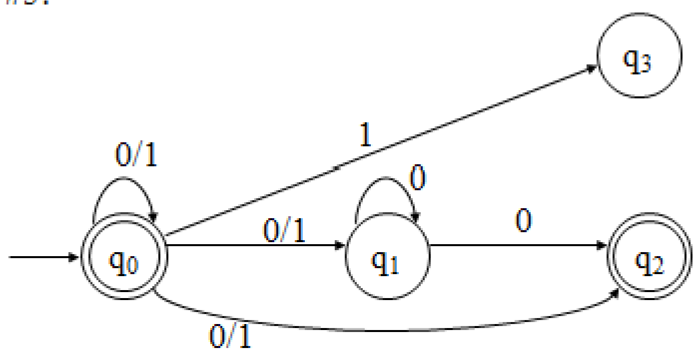
- Step #3:



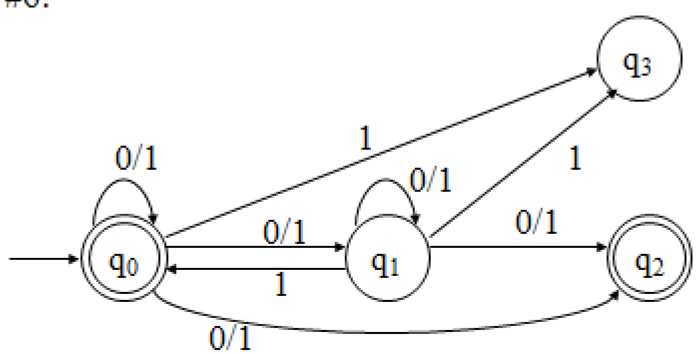
- Step #4:



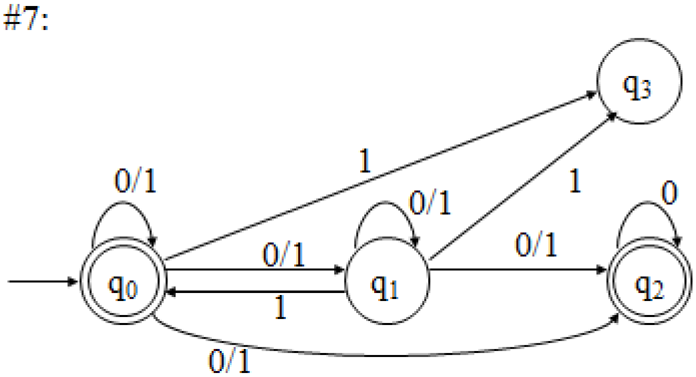
- Step #5:



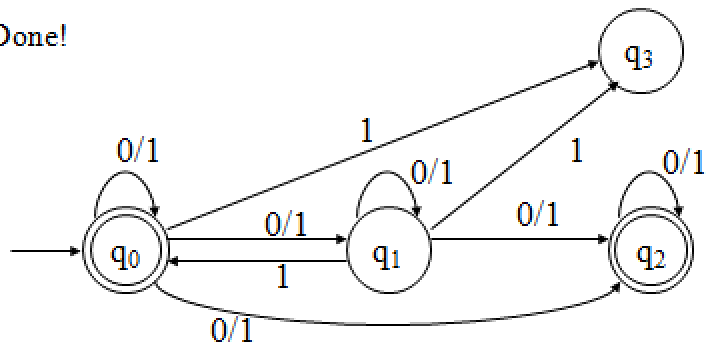
- Step #6:



- Step #7:



- Step #8:
  - Done!



- **Theorem:** Let L be a language. Then there exists an NFA M such that L= L(M) iff there exists an NFA-ε M' such that L = L(M').

- **Proof:**
  (if) Suppose there exists an NFA-ε M' such that L = L(M'). Then by Lemma 2 there exists an NFA M such that L = L(M).

  (only if) Suppose there exists an NFA M such that L = L(M). Then by Lemma 1 there exists an NFA-ε M' such that L = L(M').

- **Corollary:** The NFA-ε machines define the regular languages.

Equivalence of DFAs and NFAs

- Do DFAs and NFAs accept the same *class* of languages?
  - Is there a language L that is accepted by a DFA, but not by any NFA?
  - Is there a language L that is accepted by an NFA, but not by any DFA?

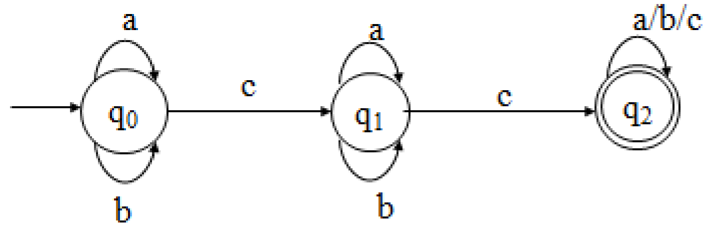
- Observation: Every DFA is an NFA.

- Therefore, if L is a regular language then there exists an NFA M such that L = L(M).

- It follows that NFAs accept all regular languages. But do NFAs accept all?

- Consider the following DFA: 2 or more c's

$Q = \{q_0, q_1, q_2\}$
$\Sigma = \{a, b, c\}$
Start state is q_0
$F = \{q_2\}$

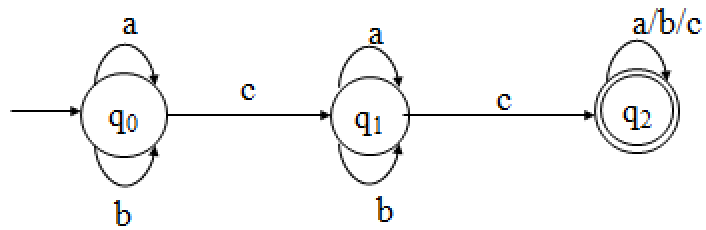


δ:

	a	b	c
q_0	$\{q_0\}$	$\{q_0\}$	$\{q_1\}$
q_1	$\{q_1\}$	$\{q_1\}$	$\{q_2\}$
q_2	$\{q_2\}$	$\{q_2\}$	$\{q_2\}$

- An Equivalent NFA:

$Q = \{q_0, q_1, q_2\}$
$\Sigma = \{a, b, c\}$
Start state is q_0
$F = \{q_2\}$



δ:

	a	b	c
q_0	$\{q_0\}$	$\{q_0\}$	$\{q_1\}$
q_1	$\{q_1\}$	$\{q_1\}$	$\{q_2\}$
q_2	$\{q_2\}$	$\{q_2\}$	$\{q_2\}$

- **Lemma 1:** Let M be an DFA. Then there exists a NFA M' such that L(M) = L(M').

- **Proof:** Every DFA is an NFA. Hence, if we let M' = M, then it follows that L(M') = L(M).
The above is just a formal statement of the observation from the above example.

- **Lemma 2:** Let M be an NFA. Then there exists a DFA M' such that L(M) = L(M').

- **Proof:** (sketch)

Let M = (Q, Σ, δ,q0,F).

Define a DFA M' = (Q', Σ, δ',q'_0,F') as:

$Q' = 2^Q$ Each state in M' corresponds to a

= {Q0, Q1,…,} subset of states from M

where Qu = [qi0, qi1,…qij]

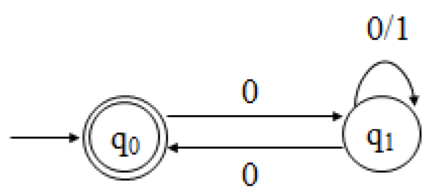
F' = {Qu | Qu contains at least one state in F}

q'_0 = [q_0]

δ'(Qu, a) = Qv iff δ(Qu, a) = Qv

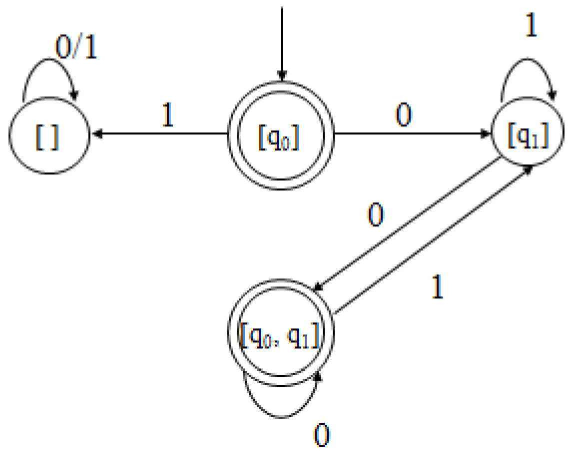
- Example: empty string or start and end with 0

$Q = \{q_0, q_1\}$
$\Sigma = \{0, 1\}$
Start state is q_0
$F = \{q_1\}$



δ:	0	1
q_0	$\{q_1\}$	$\{\}$
q_1	$\{q_0, q_1\}$	$\{q_1\}$

- Construct DFA M' as follows:



$\delta(\{q_0\}, 0) = \{q_1\}$	=>	$\delta'([q_0], 0) = [q_1]$
$\delta(\{q_0\}, 1) = \{\}$	=>	$\delta'([q_0], 1) = [\,]$
$\delta(\{q_1\}, 0) = \{q_0, q_1\}$	=>	$\delta'([q_1], 0) = [q_0, q_1]$
$\delta(\{q_1\}, 1) = \{q_1\}$	=>	$\delta'([q_1], 1) = [q_1]$
$\delta(\{q_0, q_1\}, 0) = \{q_0, q_1\}$	=>	$\delta'([q_0, q_1], 0) = [q_0, q_1]$
$\delta(\{q_0, q_1\}, 1) = \{q_1\}$	=>	$\delta'([q_0, q_1], 1) = [q_1]$
$\delta(\{\}, 0) = \{\}$	=>	$\delta'([\,], 0) = [\,]$
$\delta(\{\}, 1) = \{\}$	=>	$\delta'([\,], 1) = [\,]$

- **Theorem:** Let L be a language. Then there exists an DFA M such that L = L(M) iff there exists an NFA M' such that L = L(M').
- **Proof:**
  (if) Suppose there exists an NFA M' such that L = L(M'). Then by Lemma 2 there exists an DFA M such that L = L(M).

  (only if) Suppose there exists an DFA M such that L = L(M). Then by Lemma 1 there exists an NFA M' such that L = L(M').

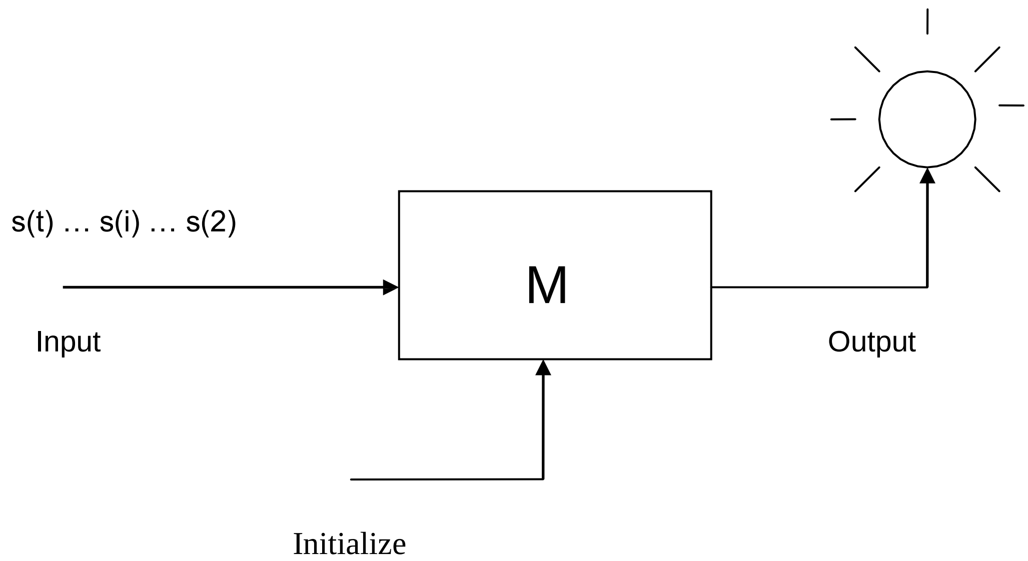
  Corollary: The NFAs define the regular languages.

<u>Finite Automata with Output</u>

- **Acceptor:**
  The symbols of the sequence

  <u>s(1) s(2) … s(i) … s(t)</u>

  are presented sequentially to a machine M. M responds with a binary signal to each input. If the string scanned so far is accepted, then the light goes on, else the light is off.



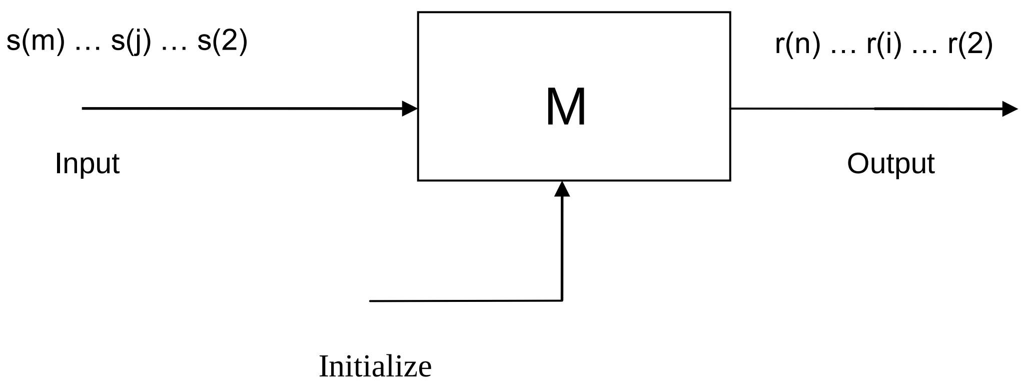
A language acceptor

- **Transducer**
  Abstract machines that operate as *transducers* are of interest in connection with the translation of languages. The following transducer produces a sentence

  r(1) r(2) … r(n)

  in response to the input sentence

  s(1) s(2) … s(m)



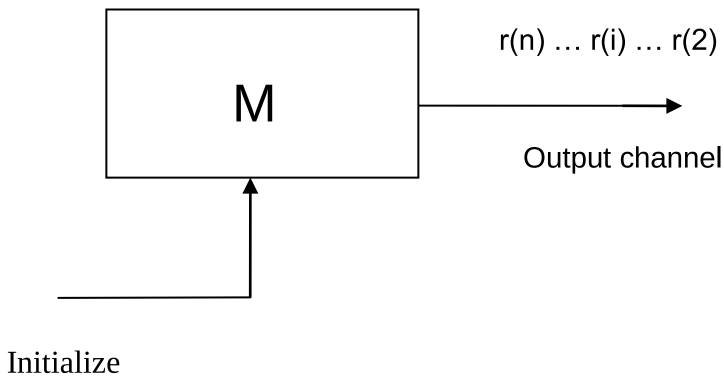
If this machine is *deterministic*, then each sentence of an *input language* is translated into a specific sentence of an *output language.*

Generator

When M is started from its initial state, it emits a sequence of symbols

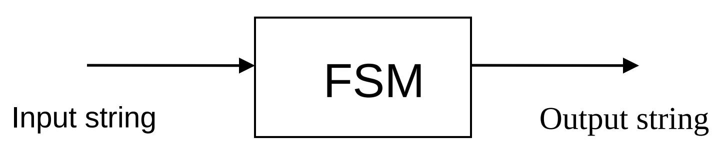
r(1) r(2) … r(i) … r(t)

from a set known as its *output alphabet*.



We will begin our study with the *transducer model* of *abstract machine* (or *automaton*). We often refer to such a device as a *Finite State Machine* (*FSM*) or as an *automaton with output.*

Finite State Machine (FSM)



The FSM model arises naturally from physical settings in which information-denoting signals are processed. Physical reality dictates that such systems are **finite.**

Only a finite number of operations may be performed in a finite amount of time. Such systems are necessarily **discrete.**

Problems are quite naturally decomposed into sequences of steps – hence our model is **sequential.**

We require that our machine not be subject to uncertainty, hence its behavior is **deterministic.**

There are two finite state machine models :

1) **Mealy model** – in which outputs occur during transitions.
2) **Moore model** – outputs are produced upon arrival at a new state.

Mealy Model of FSM

Mealy model – transition assigned output, M_t = <Q, S, R, f, g, q_I>

Where,

Q = finite set of states // the machine's memory

S = input alphabet // set of stimuli

R = output alphabet // set of responses

q_I = the machine's initial state

f : state transition function (or next state function)

f : Q * S Q

g : output function

g : Q * S R

- **Example#1:**

Design a FSM (Mealy model) which takes in binary inputs and produces a '1' as output whenever the parity of the input string (*so far*) is even.

S = R = {0, 1}

When designing such models, we should ask ourselves *"What is the state set of the machine?"*.

The state set Q corresponds to what we need to remember about input strings. We note that the number of possible input strings corresponds to |S*| which is *countably infinite.*

We observe, however, that a string may have only one of two possible parities.

even parity – if $n_1(w)$ is even.

odd parity – if $n_1(w)$ is odd.

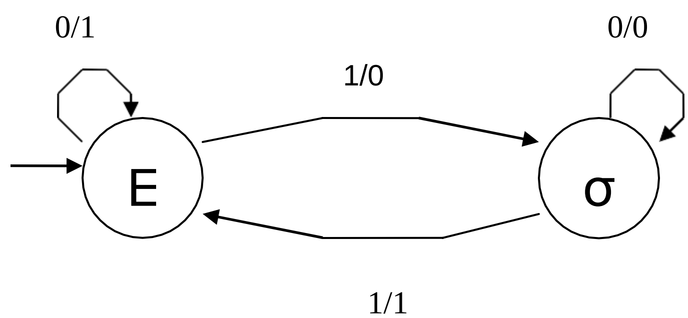
And this is all that our machine must remember about a string scanned so far.

Hence |Q| = 2 where Q = {E, σ} with q_I = E indicating the string has *even parity* and if M_t

is in state σ, then the string has *odd parity*.

- And finally, of course, we must specify *the output function g* for this Mealy machine.
- According to this machine's specifications, it is supposed to produce an output of '1' whenever the parity of the input string so far is even. Hence, *all arcs leading into state E should be labeled with a '1' output.*

<u>Parity Checker (Mealy machine)</u>



Observe our notation that **g(σ, 1) = 1** is indicated by the arc from state σ to state E with a '1' after a slash.

The output of our machine is 0 when the current string (*so far*) has odd parity.

state table	present state	input = 0 next state, output	input = 1 next state, output
for this parity machine	E	E, 1	σ, 0
	σ	σ, 0	E, 1

Observe for the input 10100011 our machine produces the output sequence 00111101

$$E \xrightarrow{1/0} \sigma \xrightarrow{0/0} \sigma \xrightarrow{1/1} E \xrightarrow{0/1} E \xrightarrow{0/1} E \xrightarrow{0/1} E \xrightarrow{1/0} \sigma \xrightarrow{1/1} E$$

the corresponding *admissible state sequence*

- **Example#2:**
  Construct a Mealy model of an FSM that behaves as a two-unit delay. i.e.
  r(t) = {s(t - 2), t > 2
  { 0 , otherwise
  A sample input/output session is given below :

time	1	2	3	4	5	6	7	8	9
stimulus	0	0	0	1	1	0	1	0	0
response	0	0	0	0	0	1	1	0	1

Observe that r(1) = r(2) = 0
r(6) = 1 which equals s(4) and so on

We know that S = R = {0, 1}.

Moore model of FSM

Moore model of FSM – the output function assigns an output symbol to each state.

M_S = <Q, S, R, f, h, q_I>

Q = finite set of internal states

S = finite input alphabet

R = finite output alphabet

f : state transition function

f : Q * S □ Q

h : output function

h : Q → R

q_I = Є Q is the initial state

- **Example#1:**

Design a Moore machine that will analyze input sequences in the binary alphabet S = {0, 1}.
Let w = s(1) s(2) … s(t) be an input string

N_0(w) = number of 0's in

w $N_1(w)$ = number of 1's
in w

then we have that $|w| = N_0(w) + N_1(w) = t$.

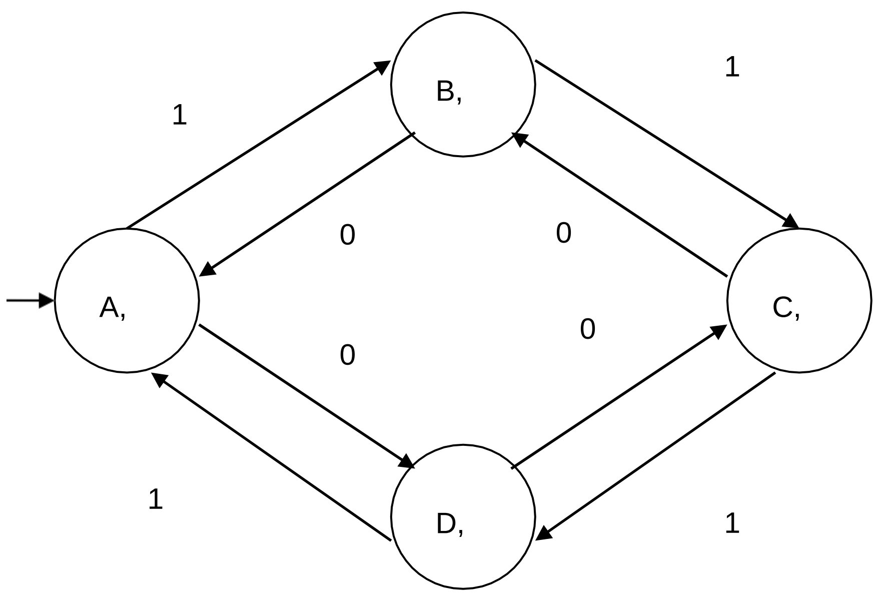
The last output of M_S should equal : $r(t) = [N_1(w) – N_0(w)] \bmod 4$.
So naturally, *the output alphabet R = {0, 1, 2, 3}*

A sample stimulus/response is given below :

stimulus	1 1 0 1 1 1 0 0
response	0 1 2 1 2 3 0 3 2

Observe that the length of the output sequence is one longer than the input sequence. Why is this so?
Btw : This will always be the case.

- The corresponding Moore machine :



State diagram

	0	1	
A	D	B	0
B	A	C	1
C	B	D	2
D	C	A	3

State table

This machine is referred to as an *up-down counter*.

For the previous input sequence : 11011100 the *state sequence is :*

$$(A, 0) \xrightarrow{1} (B, 1) \xrightarrow{1} (C, 2) \xrightarrow{0} (B, 1) \xrightarrow{1} (C, 2)$$

$$\xrightarrow{1} (D, 3) \xrightarrow{1} (A, 0) \xrightarrow{0} (D, 3) \xrightarrow{0} (C, 2)$$

- **Example#2:**

Design a Moore machine that functions as a *pattern recognizer* for “1011”. Your machine should output a ‘1’ whenever this pattern matches the last four inputs, and there has been no overlap, otherwise output a ‘0’.

Hence S = R = {0, 1}.

Here is a sample input/output sequence for this machine :

t = 1 2 3 4 5 6 7 8 9 10 11 12
S = 0 1 0 1 1 0 1 1 0 1 1 0
R = 0 0 0 0 1 0 0 0 0 0 0 1 0

We observe that r(5) = 1 because s(2) s(3) s(4) s(5) = 1011

however r(8) = 0 because there has been overlap

r(11) = 1 since s(8) s(9) s(10) s(11) = 1011

<u>Machine Identification Problem</u>

The following input-output behavior was exhibited by a transition-assigned machine (<u>Mealy machine</u>) M_t known to contain three states. Find an appropriate state table for M. Is the table unique?

time 1 2 3 4 5 6 7 8 9 10 11 12 13 14
input 0 0 0 0 1 0 0 0 1 0 0 0 1 0
output 0 1 0 1 0 0 0 0 1 0 1 0 0 1

This problem is useful in fault detection and fault location experiments with sequential circuits (i.e. *digital circuits with memory*).

One designs a computer circuit. Six months (or six years) later, how does one know that the circuit is working correctly?

The procedure to solve this problem is helpful in fault diagnosis of digital circuits.

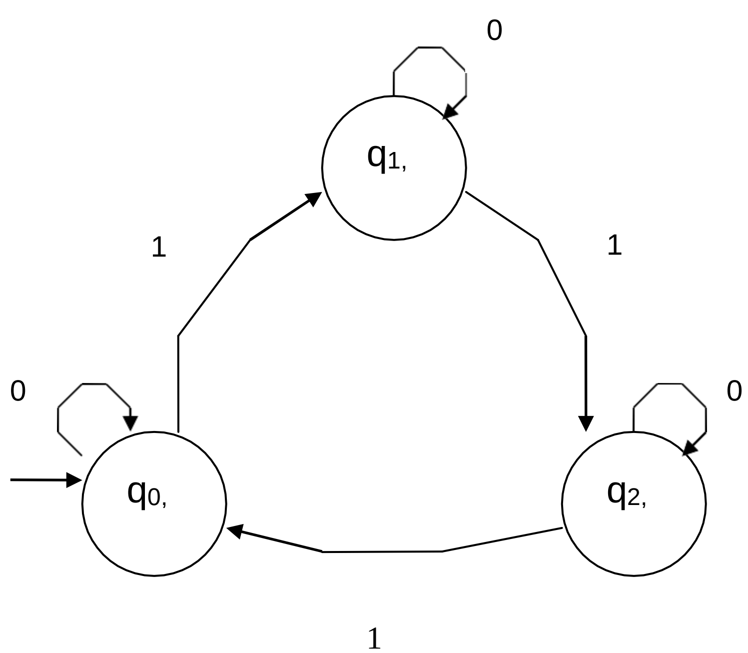
Equivalence of Mealy and Moore Models

The Mealy and Moore models of finite state machines are equivalent (actually similar). i.e. $M_t \approx M_S$

What does this mean ?

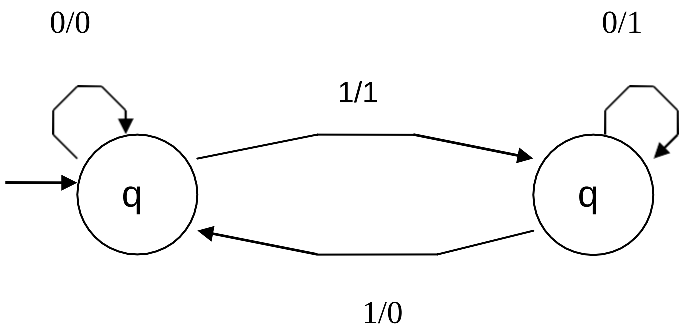
And how would be prove it ?

We will employ the following machines in our proof.

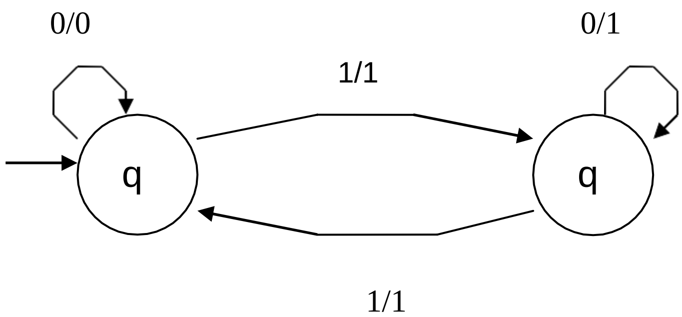


M_S : A mod 3 counter

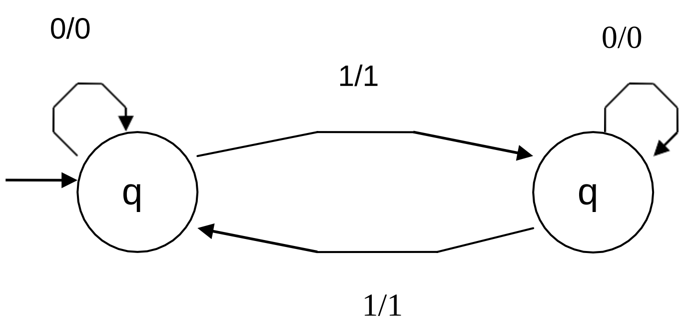
M1 :



M2 :



M3 :



Three helpful Mealy machines

UNIT II:

Regular Expressions

- A regular expression is used to specify a language, and it does so precisely.

- Regular expressions are very intuitive.
- Regular expressions are very useful in a variety of contexts.
- Given a regular expression, an NFA-ε can be constructed from it automatically.
- Thus, so can an NFA, a DFA, and a corresponding program, all automatically!

Definition:

- Let Σ be an alphabet. The regular expressions over Σ are:

–	Ø	Represents the empty set { }
–	ε	Represents the set {ε}
–	a	Represents the set {a}, for any symbol a in Σ

Let r and s be regular expressions that represent the sets R and S, respectively.

–	r+s	Represents the set R U S	(precedence 3)
–	rs	Represents the set RS	(precedence 2)
–	r^*	Represents the set R*	(highest precedence)
–	(r)	Represents the set R	(not an op, provides precedence)

- If r is a regular expression, then L(r) is used to denote the corresponding language.

- **Examples:** Let Σ = {0, 1}

(0 + 1)*	All strings of 0's and 1's
0(0 + 1)*	All strings of 0's and 1's, beginning with a 0
(0 + 1)*1	All strings of 0's and 1's, ending with a 1
(0 + 1)*0(0 + 1)*	All strings of 0's and 1's containing at least one 0
(0 + 1)*0(0 + 1)*0(0 + 1)*	All strings of 0's and 1's containing at least two 0's
(0 + 1)*01*01*	All strings of 0's and 1's containing at least two 0's
(1 + 01*0)*	All strings of 0's and 1's containing an even number of 0's
1*(01*01*)*	All strings of 0's and 1's containing an even number of 0's
(1*01*0)*1*	All strings of 0's and 1's containing an even number of 0's

Identities:

1. Øu = uØ = Ø Multiply by 0
2. εu = uε = u Multiply by 1
3. Ø* = ε
4. ε* = ε
5. u+v = v+u

6. $u + \emptyset = u$

7. $u + u = u$

8. $u^* = (u^*)^*$

9. $u(v+w) = uv+uw$

10. $(u+v)w = uw+vw$

11. $(uv)^*u = u(vu)^*$

12. $(u+v)^* = (u^*+v)^*$

$= u^*(u+v)^*$

$= (u+vu^*)^*$

$= (u^*v^*)^*$

$= u^*(vu^*)^*$

$= (u^*v)^*u^*$

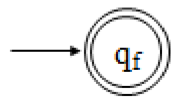
Equivalence of Regular Expressions and NFA-ε

- **Note:** Throughout the following, keep in mind that a string is accepted by an NFA-ε if there exists a path from the start state to a final state.
- **Lemma 1:** Let r be a regular expression. Then there exists an NFA-ε M such that L(M) = L(r). Furthermore, M has exactly one final state with no transitions out of it.
- **Proof:** (by induction on the number of operators, denoted by OP(r), in r).
- **Basis:** OP(r) = 0

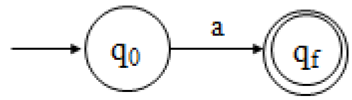
Then r is either Ø, ε, or **a**, for some symbol **a** in Σ

For Ø:

For ε:



For **a**:



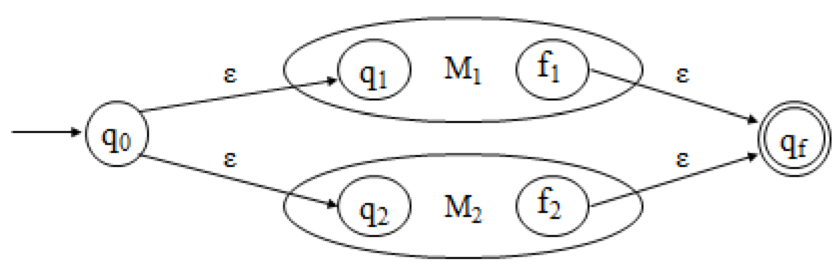
- **Inductive Hypothesis:** Suppose there exists a k □ 0 such that for any regular expression r where 0 □ OP(r) □ k, there exists an NFA-ε such that L(M) = L(r). Furthermore, suppose that M has exactly one final state.

- **Inductive Step:** Let r be a regular expression with k + 1 operators (OP(r) = k + 1), where k + 1 >= 1.

  Case 1) r = r1 + r2

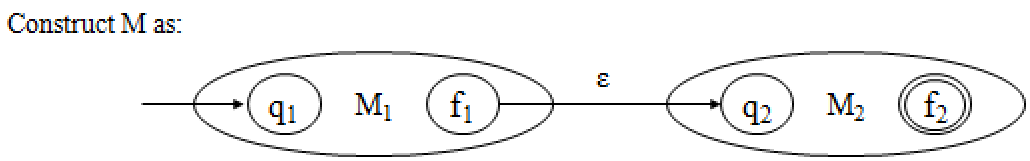
  Since OP(r) = k +1, it follows that 0<= OP(r1), OP(r2) <= k. By the inductive hypothesis there exist NFA-ε machines M1 and M2 such that L(M1) = L(r1) and L(M2) = L(r2). Furthermore, both M1 and M2 have exactly one final state.

  Construct M as:



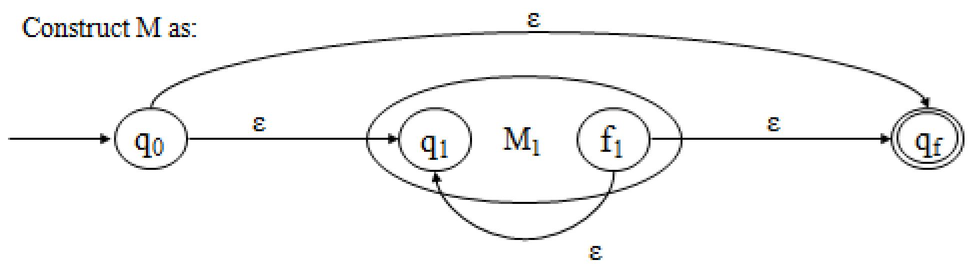
  Case 2) r = r1r2

  Since OP(r) = k+1, it follows that 0<= OP(r1), OP(r2) <= k. By the inductive hypothesis there exist NFA-ε machines M1 and M2 such that L(M1) = L(r1) and L(M2) = L(r2). Furthermore, both M1 and M2 have exactly one final state.



Case 3) r = r1*

Since OP(r) = k+1, it follows that 0<= OP(r1) <= k. By the inductive hypothesis there exists an NFA-ε machine M1 such that L(M1) = L(r1). Furthermore, M1 has exactly one final state.

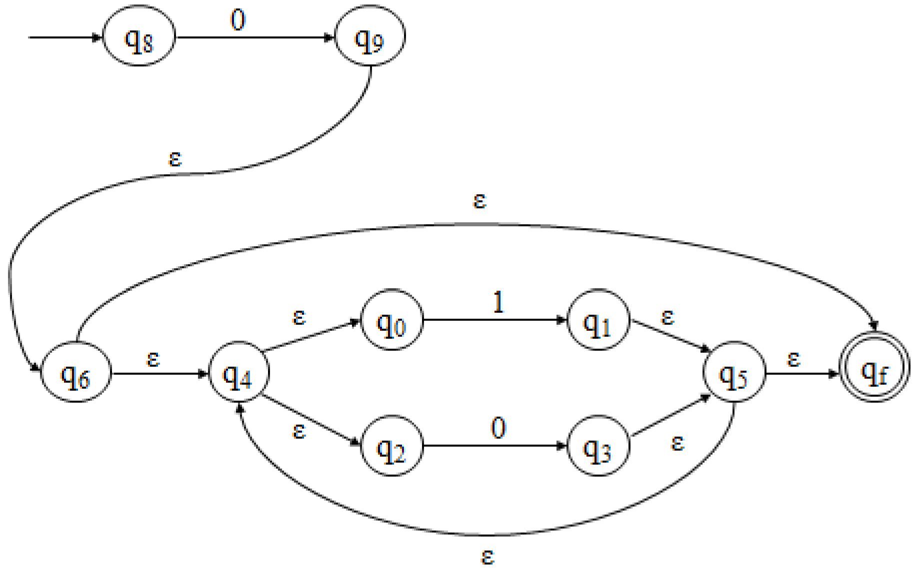


- **Example:**

Problem: Construct FA equivalent to RE, r = 0(0+1)*

Solution:
r = r1r2
r1 = 0
r2 = (0+1)*
r2 = r3*
r3 = 0+1
r3 = r4 + r5
r4 = 0
r5 = 1

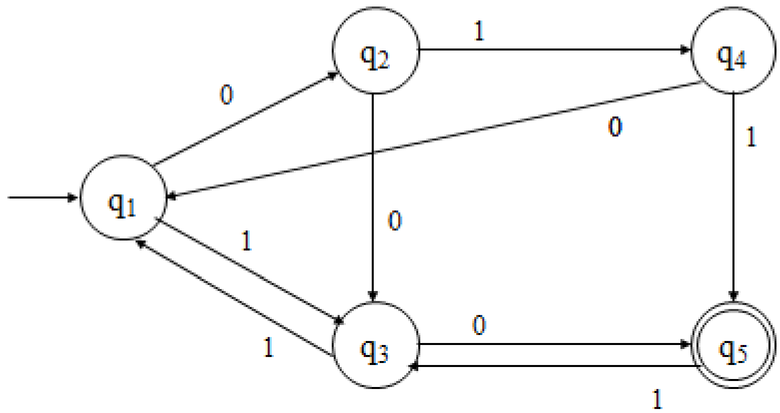
Transition graph:



Definitions Required to Convert a DFA to a Regular Expression

- Let M = (Q, Σ, δ, q1, F) be a DFA with state set Q = {q1, q2, …, qn}, and define: $R_{i,j}$ = { x | x is in Σ* and δ(q_i,x) = q_j}
  $R_{i,j}$ is the set of all strings that define a path in M from q_i to q_j.

- Note that states have been numbered starting at 1!
- Example:



$R_{2,3}$ = {0, 001, 00101, 011, …}
$R_{1,4}$ = {01, 00101, …}
$R_{3,3}$ = {11, 100, …}

- Observations:

1) $R^n_{i,j} = R_{i,j}$

2) $R^{k-1}_{i,j}$ is a subset of $R^k_{i,j}$

3) $L(M) = \bigcup_{q \in F} R^n_{1,q} = \bigcup_{q \in F} R_{1,q}$

4) $R^0_{i,j} = \begin{cases} \{a \mid \delta(q_i, a) = q_j\} & i \neq j \\ \{a \mid \delta(q_i, a) = q_j\} \bigcup \{\varepsilon\} & i = j \end{cases}$ Easily computed from the DFA!

5) $R^k_{i,j} = R^{k-1}_{i,k} (R^{k-1}_{k,k})^* R^{k-1}_{k,j} \cup R^{k-1}_{i,j}$

- **Lemma 2:** Let M = (Q, Σ, δ, q1, F) be a DFA. Then there exists a regular expression r such that L(M) = L(r).

- **Proof:**

  First we will show (by induction on k) that for all i,j, and k, where 1 □ i,j □ n And 0 □ k □ n, that there exists a regular expression r such that L(r) = R^ki,j .

  Basis: k=0

  R^0i,j contains single symbols, one for each transition from qi to qj, and possibly ε if i=j.

Case 1) No transitions from q_i to q_j and $i \neq j$

$$r^0_{i,j} = \emptyset$$

Case 2) At least one ($m \geq 1$) transition from q_i to q_j and $i \neq j$

$$r^0_{i,j} = a_1 + a_2 + a_3 + \dots + a_m \qquad \text{where } \delta(q_i, a_p) = q_j, \text{ for all } 1 \leq p \leq m$$

Case 3) No transitions from q_i to q_j and $i = j$

$$r^0_{i,j} = \varepsilon$$

Case 4) At least one ($m \geq 1$) transition from q_i to q_j and $i = j$

$$r^0_{i,j} = a_1 + a_2 + a_3 + \dots + a_m + \varepsilon \qquad \text{where } \delta(q_i, a_p) = q_j \text{ for all } 1 \leq p \leq m$$

- **Inductive Hypothesis:**
  Suppose that $R^{k-1}_{i,j}$ can be represented by the regular expression $r^{k-1}_{i,j}$ for all $1 \leq i,j \leq n$, and some $k \geq 1$.

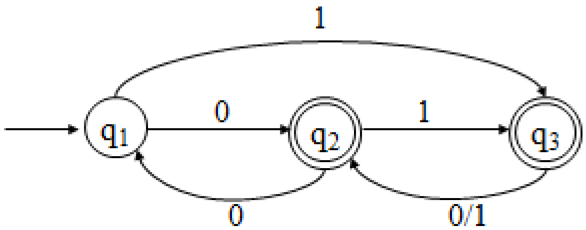
- **Inductive Step:**
  Consider $R^k_{i,j} = R^{k-1}_{i,k} (R^{k-1}_{k,k})^* R^{k-1}_{k,j} \cup R^{k-1}_{i,j}$. By the inductive hypothesis there exist regular expressions $r^{k-1}_{i,k}$, $r^{k-1}_{k,k}$, $r^{k-1}_{k,j}$, and $r^{k-1}_{i,j}$ generating $R^{k-1}_{i,k}$, $R^{k-1}_{k,k}$, $R^{k-1}_{k,j}$, and $R^{k-1}_{i,j}$, respectively. Thus, if we let

$$r^k_{i,j} = r^{k-1}_{i,k} (r^{k-1}_{k,k})^* r^{k-1}_{k,j} + r^{k-1}_{i,j}$$

  then $r^k_{i,j}$ is a regular expression generating $R^k_{i,j}$, i.e., $L(r^k_{i,j}) = R^k_{i,j}$.

- Finally, if $F = \{q_{j1}, q_{j2}, \dots, q_{jr}\}$, then
  $$r^n_{1,j1} + r^n_{1,j2} + \dots + r^n_{1,jr}$$
  is a regular expression generating $L(M)$.•

- **Example:**



First table column is computed from the DFA.

	k = 0	k = 1	k = 2
$r^k_{1,1}$	ε		
$r^k_{1,2}$	0		
$r^k_{1,3}$	1		
$r^k_{2,1}$	0		
$r^k_{2,2}$	ε		
$r^k_{2,3}$	1		
$r^k_{3,1}$	Ø		
$r^k_{3,2}$	0 + 1		
$r^k_{3,3}$	ε		

- All remaining columns are computed from the previous column using the formula.

$$\begin{aligned} r^1_{2,3} &= r^0_{2,1}\,(r^0_{1,1})^*\, r^0_{1,3} + r^0_{2,3} \\ &= 0\,(\varepsilon)^*\,1 + 1 \\ &= 01 + 1 \end{aligned}$$

	k = 0	k = 1	k = 2
$r^k_{1,1}$	ε	ε	
$r^k_{1,2}$	0	0	
$r^k_{1,3}$	1	1	
$r^k_{2,1}$	0	0	
$r^k_{2,2}$	ε	ε + 00	
$r^k_{2,3}$	1	**1 + 01**	
$r^k_{3,1}$	Ø	Ø	
$r^k_{3,2}$	0 + 1	0 + 1	
$r^k_{3,3}$	ε	ε	

$$
\begin{aligned}
r^2_{1,3} &= r^1_{1,2}\,(r^1_{2,2})^*\,r^1_{2,3} + r^1_{1,3} \\
&= 0\,(\varepsilon + 00)^*\,(1 + 01) + 1 \\
&= 0^*1
\end{aligned}
$$

	k = 0	k = 1	k = 2
$r^k_{1,1}$	ε	ε	(00)*
$r^k_{1,2}$	0	0	0(00)*
$r^k_{1,3}$	1	1	**0*1**
$r^k_{2,1}$	0	0	0(00)*
$r^k_{2,2}$	ε	ε + 00	(00)*
$r^k_{2,3}$	1	1 + 01	0*1
$r^k_{3,1}$	Ø	Ø	(0 + 1)(00)*0
$r^k_{3,2}$	0 + 1	0 + 1	(0 + 1)(00)*
$r^k_{3,3}$	ε	ε	ε + (0 + 1)0*1

- To complete the regular expression, we compute:

$r^3_{1,2} + r^3_{1,3}$

	k = 0	k = 1	k = 2
$r^k_{1,1}$	ε	ε	(00)*
$r^k_{1,2}$	0	0	0(00)*
$r^k_{1,3}$	1	1	0*1
$r^k_{2,1}$	0	0	0(00)*
$r^k_{2,2}$	ε	ε + 00	(00)*
$r^k_{2,3}$	1	1 + 01	0*1
$r^k_{3,1}$	Ø	Ø	(0 + 1)(00)*0
$r^k_{3,2}$	0 + 1	0 + 1	(0 + 1)(00)*
$r^k_{3,3}$	ε	ε	ε + (0 + 1)0*1

Pumping Lemma for Regular Languages

- Pumping Lemma relates the size of string accepted with the number of states in a DFA

- What is the largest string accepted by a DFA with *n* states?

- Suppose there is no loop?
  Now, if there is a loop, what type of strings are accepted *via* the loop(s)?

- **Lemma:** (the pumping lemma)

  Let M be a DFA with |Q| = n states. If there exists a string x in L(M), such that |x| □ n, then there exists a way to write it as x = uvw, where u,v, and w are all in Σ* and:

  – 1□ |uv| □n
  – |v| □ 1
  – such that, the strings uv^iw are also in L(M), for all i □ 0

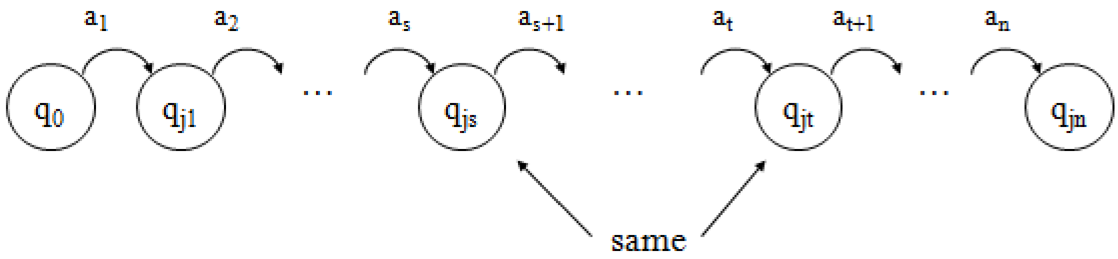
- **Proof:**
  Let $x = a_1a_2 \ldots a_m$ where $m \geq n$, x is in L(M), and $\delta(q_0, a_1a_2 \ldots a_p) = q_{jp}$

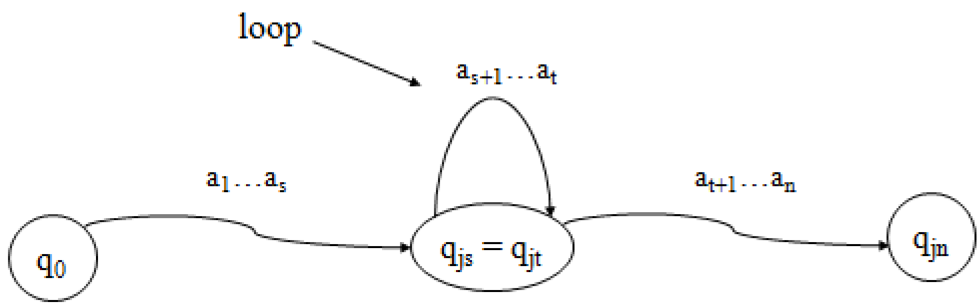
  $$\begin{array}{ccccccc} & a_1 & & a_2 & & a_3 \ \ldots & & a_m & \\ q_{j0} & & q_{j1} & & q_{j2} & & q_{j3} \cdots & & q_{jm} \end{array} \qquad m \geq n \qquad \text{and} \qquad q_{j0} \text{ is } q_0$$

  Consider the first n symbols, and first n+1 states on the above path:

  $$\begin{array}{ccccccc} & a_1 & & a_2 & & a_3 \ \ldots & & a_n & \\ q_{j0} & & q_{j1} & & q_{j2} & & q_{j3} \cdots & & q_{jn} \end{array}$$

  Since |Q| = n, it follows from the pigeon-hole principle that $j_s = j_t$ for some $0 \leq s < t \leq n$, i.e., some state appears on this path twice (perhaps many states appear more than once, but at least one does).





- Let:
  - $u = a_1 \ldots a_s$
  - $v = a_{s+1} \ldots a_t$

- Since $0 \le s < t \le n$ and $uv = a_1 \ldots a_t$ it follows that:
  - $1 \le |v|$ and therefore $1 \le |uv|$
  - $|uv| \le n$ and therefore $1 \le |uv| \le n$

- In addition, let:
  - $w = a_{t+1} \ldots a_m$

- It follows that $uv^iw = a_1 \ldots a_s(a_{s+1} \ldots a_t)^i a_{t+1} \ldots a_m$ is in L(M), for all $i \ge 0$.

  In other words, when processing the accepted string x, the loop was traversed once, but could have been traversed as many times as desired, and the resulting string would still be accepted.

Closure Properties of Regular Languages

- Consider various operations on languages:

$$\overline{L} = \{x \mid x \text{ is in } \Sigma^* \text{ and } x \text{ is not in } L\}$$
$$L_1 \cup L_2 = \{x \mid x \text{ is in } L_1 \text{ or } L_2\}$$
$$L_1 \cap L_2 = \{x \mid x \text{ is in } L_1 \text{ and } L_2\}$$
$$L_1 - L_2 = \{x \mid x \text{ is in } L_1 \text{ but not in } L_2\}$$
$$L_1L_2 = \{xy \mid x \text{ is in } L_1 \text{ and } y \text{ is in } L_2\}$$
$$L^* = \bigcup_{i=0}^{\infty} L^i = L^0 \cup L^1 \cup L^2 \cup \ldots$$
$$L^+ = \bigcup_{i=1}^{\infty} L^i = L^1 \cup L^2 \cup \ldots$$

1. Closure Under Union

- If L and M are regular languages, so is L ∪ M.
- Proof: Let L and M be the languages of regular expressions R and S, respectively.
- Then R+S is a regular expression whose language is L ∪ M.

2. Closure Under Concatenation and Kleene

- Closure RS is a regular expression whose language
- is LM. R* is a regular expression whose language is L*.

3. Closure Under Intersection

- If L and M are regular languages, then so is L ∩ M.
- Proof: Let A and B be DFA's whose languages are L and M, respectively.

4. Closure Under Difference

- If L and M are regular languages, then so is L – M = strings in L but not M.
- Proof: Let A and B be DFA's whose languages are L and M, respectively.

5. Closure Under Complementation

- The complement of language L (w.r.t. an alphabet Σ such that Σ* contains L) is Σ* – L.
- Since Σ* is surely regular, the complement of a regular language is always regular.

6. Closure Under Homomorphism

- If L is a regular language, and h is a homomorphism on its alphabet, then h(L) = {h(w) | w is in L} is also a regular language.

Grammar

- **Definition:** A grammar G is defined as a 4-tuple, G = (V, T, S, P)

  Where,

  - V is a finite set of objects called variables,
  - T is a finite set of objects called terminal symbols,
  - $S \in V$ is a special symbol called start variable,
  - P is a finite set of productions.

  Assume that V and T are non-empty and disjoint.

- **Example:**

  Consider the grammar G = ({S}, {a, b}, S, P) with P given by

  S □ aSb,   S □ ε_.

  For instance, we have $S \Rightarrow aSb \Rightarrow aaSbb \Rightarrow aabb$.

  It is not hard to conjecture that $L(G) = \{a^n b^n \mid n \geq 0\}$.

Right, Left-Linear Grammar

- **Right-linear Grammar:** A grammar G = (V, T, S, P) is said to be right-linear if all productions are of the form:

  A □ xB,

  A □ x,

  Where $A, B \in V$ and $x \in T^*$.

  - **Example#1:**

    S* → *abS* | *a is an example of a right-linear grammar.

    - Can you figure out what language it generates?
    - ***L* = {*w* ∈ {*a*,*b*}* | *w***

      Contains alternating ***a***'s and ***b***'s , begins with an ***a***, and ends with a ***b*}**

      ∪ **{*a*}**

    - ***L*((*ab*)**a*)**

- **Left-linear Grammar:** A grammar G = (V, T, S, P) is said to be left-linear if all productions are of the form:

  A □ Bx,

  A □ x,

  Where $A, B \in V$ and $x \in T^*$.

  - **Example#2:**

$S \rightarrow Aab$
$A \rightarrow Aab \mid aB$
$B \rightarrow a$
is an example of a left-linear grammar.

- Can you figure out what language it generates?
- ***L* = {*w* Î {*a*,*b*}* | *w*** is ***aa*** followed by at least one set of alternating ***ab***'s**}**
- ***L*(*aaab*(*ab*)*)**

- **Example#3:**

  Consider the grammar
  $S \rightarrow A$
  $A \rightarrow aB \mid \lambda$
  $B \rightarrow Ab$
  This grammar is NOT regular.

  - No "mixing and matching" left- and right-recursive productions.

Regular Grammar

- A linear grammar is a grammar in which at most one variable can occur on the right side of any production without restriction on the position of this variable.

- An example of linear grammar is G = ({S, S1, S2}, {a, b}, S, P) with
  S S1ab,
  S1 S1ab | S2,
  S2 a.

- A **regular grammar** is one that is either right-linear or left-liner.

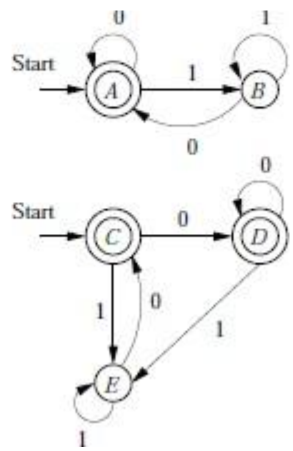
Testing Equivalence of Regular Languages

- Let L and M be reg langs (each given in some form).

  To test if L = M

  1. Convert both L and M to DFA's.
  2. Imagine the DFA that is the union of the two DFA's (never mind there are two start states)
  3. If TF-algo says that the two start states are distinguishable, then L 6= M, otherwise, L = M.

Example:



We can "see" that both DFA accept L(ε+(0+1)*0). The result of the TF-algo is

	A	B	C	D
B	x			
C		x		
D		x		
E	x		x	x

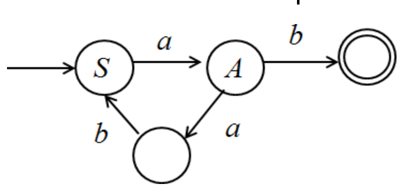
Therefore the two automata are equivalent.

<u>Regular Grammars and NFA's</u>

- It's not hard to show that regular grammars generate and nfa's accept the same class of languages: the regular languages!
- It's a long proof, where we must show that
  - Any finite automaton has a corresponding left- or right-linear grammar,
  - And any regular grammar has a corresponding nfa.
- **Example:**

  o We get a feel for this by example.

  Let $\boldsymbol{S \rightarrow aA} \quad \boldsymbol{A \rightarrow abS \mid b}$



CONTEXT FREE-GRAMMAR

- **Definition:** Context-Free Grammar (CFG) has 4-tuple: G = (V, T, P, S)

Where,

V	-	A finite set of variables or *non-terminals*
T	-	A finite set of *terminals* (V and T do not intersect)
P	-	A finite set of *productions*, each of the form $A \rightarrow \alpha$, Where A is in V and α is in $(V \cup T)^*$ Note: that α may be ε.
S	-	A starting non-terminal (S is in V)

- **Example#1 CFG:**

G = ({S}, {0, 1}, P, S)
P:

(1) $S \rightarrow 0S1$ or just simply $S \rightarrow 0S1 \mid \varepsilon$
(2) $S \rightarrow \varepsilon$

- **Example Derivations:**

S	=> 0S1	(1)
S	=> ε	(2)
	=> 01	(2)
S	=> 0S1	(1)
	=> 00S11	(1)
	=> 000S111	(1)
	=> 000111	(2)

- Note that G "generates" the language $\{0^k1^k \mid k>=0\}$

Derivation (or Parse) Tree

- **Definition:** Let G = (V, T, P, S) be a CFG. A tree is a derivation (or parse) tree if:
  - Every vertex has a label from $V \cup T \cup \{\varepsilon\}$
  - The label of the root is S
  - If a vertex with label A has children with labels $X_1, X_2, \ldots, X_n$, from left to right, then

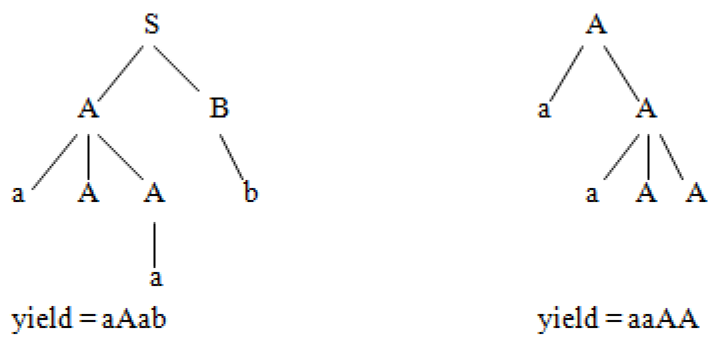
    $$A \rightarrow X_1, X_2, \ldots, X_n$$

    must be a production in P
- If a vertex has label ε, then that vertex is a leaf and the only child of its' parent

- More Generally, a derivation tree can be defined with any non-terminal as the root.

- **Example:**

S –> AB
A –> aAA
A –> aA
A –> a
B –> bB
B –> b



- **Notes:**
  - Root can be any non-terminal
  - Leaf nodes can be terminals or non-terminals
  - A derivation tree with root S shows the productions used to obtain a sentential form.

Sentential Form

- **Definition:** A sentence that contains variables and terminals.

$$S \Rightarrow aSb \Rightarrow aaSbb \Rightarrow aaaSbbb \Rightarrow aaabbb$$

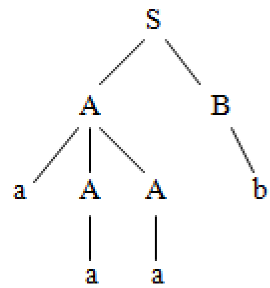
Sentential Forms sentence

Leftmost and Rightmost Derivation

Definition: A derivation is *leftmost (rightmost)* if at each step in the derivation a production is applied to the leftmost (rightmost) non-terminal in the sentential form.

- **Observation:** Every derivation corresponds to one derivation tree.

$$
\begin{aligned}
S &\Rightarrow AB \\
&\Rightarrow aAAB \\
&\Rightarrow aaAB \\
&\Rightarrow aaaB \\
&\Rightarrow aaab
\end{aligned}
$$



- **Observation:** Every derivation tree corresponds to one or more derivations.

$S \Rightarrow AB$	$S \Rightarrow AB$	$S \Rightarrow AB$
$\Rightarrow aAAB$	$\Rightarrow Ab$	$\Rightarrow Ab$
$\Rightarrow aaAB$	$\Rightarrow aAAb$	$\Rightarrow aAAb$
$\Rightarrow aaaB$	$\Rightarrow aAab$	$\Rightarrow aaAb$
$\Rightarrow aaab$	$\Rightarrow aaab$	$\Rightarrow aaab$

- The first derivation above is **leftmost**, second is **rightmost** and the third is neither.

Ambiguity in Context Free Grammar

- **Definition:** Let G be a CFG. Then G is said to be ambiguous if there exists an x in L(G) with >1 leftmost derivations. Equivalently, G is said to be ambiguous if there exists an x in L(G) with >1 parse trees, or >1 rightmost derivations.

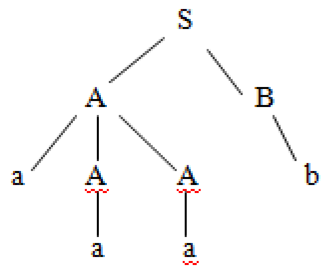
- Note: Given a CFL L, there may be more than one CFG G with L = L(G). Some ambiguous and some not.

- Definition: Let L be a CFL. If every CFG G with L = L(G) is ambiguous, then L is inherently ambiguous.

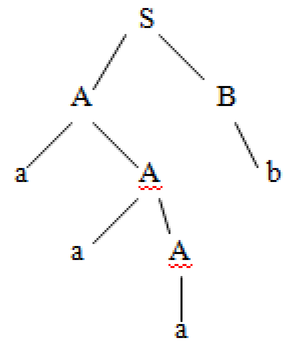
- **Example:** Consider the string aaab and the preceding grammar.

S-> AB
A-> aAA
A-> aA
A-> a
B-> bB
B-> b

S => AB
=> aAAB
=> aaAB
=> aaaB
=> aaab



S => AB
=> aAB
=> aaAB
=> aaaB
=> aaab



- The string has two left-most derivations, and therefore has two distinct parse trees and is ambiguous .

Eliminations of Useless Symbols

- **Definition:**
  Let $G = (V, T, S, P)$ be a context-free grammar. A variable $A \square V$ is said to be useful if and only if there is at least one $w \square L(G)$ such that

  $$S \rightarrow^{\square} xAy \rightarrow^{\square} w$$
  with $x, y \square (V \square T)^{\square}$.

  In words, a variable is useful if and only if it occurs in at least on derivation. A variable that is not useful is called useless. A production is useless if it involves any useless variable

- For a grammar with productions
  $$S \rightarrow aSb \mid \rightarrow \mid$$
  $$A \rightarrow aA$$

  A is useless variable and the production $S \square A$ plays no role since A cannot be eventually transformed into a terminal string; while A can appear in a sentential form derived from S, this sentential form can never lead to sentence!

Hence, removing $S \rightarrow A$ (and $A \rightarrow aA$) does not change the language, but does simplify the grammar.

- For a grammar with productions

$$S \rightarrow A$$
$$A \rightarrow aA \mid €$$
$$B \rightarrow bA$$

B is useless so is the production $B \rightarrow bA$! Observe that, even though a terminal string can be derived from B, there is no way to get to B from S, i.e. cannot achieve

$S \stackrel{*}{\Rightarrow} xBy$.

- **Example:**

Eliminate useless symbols and productions from $G = (V, T, S, P)$, where
$V = \{S, A, B, C\}$, $T = \{a, b\}$ and
P consists of

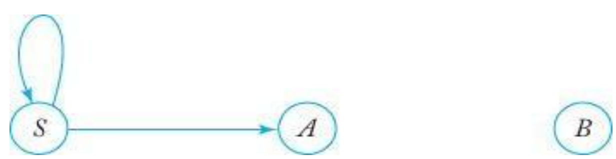
$$S \rightarrow aS \mid A \mid C$$
$$A \rightarrow a$$
$$B \rightarrow aa$$
$$C \rightarrow aCb$$

First, note that the variable C cannot lead to any terminal string, we can then remove C and its associated productions, we get $G1$ with $V1 = \{S, A, B\}$, $T1 = \{a\}$ and $P1$ consisting of

$$S \rightarrow aS \mid A$$
$$A \rightarrow a$$
$$B \rightarrow aa$$

Next, we identify variables that cannot be reached from the start variable. We can create a dependency graph for $V1$. For a context-free grammar, a dependency graph has its vertices labeled with variables with an edge between any two vertices I and J if there is a production of the form

$$I \rightarrow xJy$$



Consequently, the variable B is shown to be useless and can be removed together with its associated production.

The resulting grammar $G' = (V', T', S, P')$ is with $V' = \{S, A\}$, $T' = \{a\}$ and P' consisting of

$$S \rightarrow aS \mid A$$
$$A \rightarrow a$$

Eliminations of □-Production

- **Definition :**
  a) Any production of a context-free grammar of the form
  $$A \rightarrow € □$$
  is called a €-production.

  b) Any variable A for which the derivation
  $$A \rightarrow €$$
  is possible is called nullable.

- If a grammar contains some □-productions or nullable variables but does not generate the language that contains an empty string, the €-productions can be removed!

- **Example:**
  Consider the grammar, G with productions
  $$S \rightarrow aS1b$$
  $$S1 \rightarrow aS1b \mid €$$
  $L(G) = \{a^n b^n \mid n □ \ 1\}$ which is a€-free language. The €-production can be removed after adding new productions obtained by substituting € for $S1$ on the right hand side.

  We get an equivalent G' with productions
  $$S \rightarrow aS1b \mid ab$$
  $$S1 \rightarrow aS1b \mid ab$$

- **Theorem:**
  Let G be any context-free grammar with € $L(G)$. There exists an equivalent grammar G' without €-productions.

  Proof :
  Find the set VN of all nullable variables of G
  1. For all productions $A \rightarrow €$, put A in VN
  2. Repeat the following step until no further variables are added to
  VN: For all productions
  $$B \rightarrow A1A2\ldots An$$

  where $A1, A2, \ldots, An$ are in VN, put B in VN.

  With the resulting VN, P' can be constructed by looking at all productions in P of the form
  $$A \rightarrow x1x2\ldots xm,\ m \ □$$

  1 where each $xi \rightarrow V\ U\ T$.

For each such production of P, we put in P' the production plus all productions generated by replacing nullable variables with € in all possible combinations. However, if all x_i are nullable, the resulting production $A \rightarrow €$ is not put in P'.

- **Example:**

For the grammar G with

$S \rightarrow ABaC$
$A \rightarrow BC$
$B \rightarrow b \mid €$

$C \rightarrow D$
$D \rightarrow d$

the nullable variables are A, B, and C.

The equivalent grammar G' without □-productions has P' containing
$S \rightarrow ABaC \mid BaC \mid AaC \mid ABa \mid aC \mid Ba \mid Aa \mid a$
$A \rightarrow BC \mid C \mid B$
$B \rightarrow b$
$C \rightarrow D$
$D \rightarrow d$

Eliminations of Unit-Production

- **Definition:**
Any production of a context-free grammar of the form
$A \rightarrow B$
where A, B □ V is called a unit-production.

- **Theorem:**
Let $G = (V, T, S, P)$ be any context-free grammar without □-productions. There exists a context-free grammar $G' = (V', T', S, P')$ that does not have any unit-productions and that is equivalent to G.

Proof:
First of all, Any unit-production of the form $A \rightarrow A$ can be removed without any effect. We then need to consider productions of the form $A \rightarrow B$ where A and B are different variables.

Straightforward replacement of B (with $x_1 = x_2 =$ □) runs into a problem when we have
$A \rightarrow B$
$B \rightarrow A$
We need to find for each A, all variables B such that
$A \rightarrow^{□} B$
This can be done via a dependency graph with an edge (I, J) whenever the grammar G has a unit-production $I \rightarrow J$; $A \rightarrow B$ whenever there is a walk from A to B in the graph.

The new grammar G' is generated by first putting in P' all non-unit-productions of P. Then, for all A and B with $A \rightarrow B$, we add to P'

$$A \rightarrow y1 \mid y2 \mid \ldots \mid yn$$

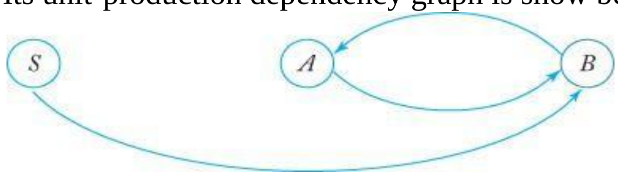
where $B \rightarrow y1 \mid y2 \mid \ldots \mid yn$ is the set of all rules in P' with B on the left. Not that the rules are taken from P', therefore, none of y_i can be a single variable! Consequently, no unit-productions are created by this step.

- **Example:**

  Consider a grammar G with

  $$S \rightarrow Aa \mid B$$
  $$A \rightarrow a \mid bc \mid B$$
  $$B \rightarrow A \mid bb$$

  Its unit-production dependency graph is show below

  We have $S \rightarrow A$, $S \rightarrow B$, $A \rightarrow B$ and $B \rightarrow A$.

  First, for the set of original non-unit-productions, we have

  $$S \rightarrow Aa$$
  $$A \rightarrow a \mid bc$$
  $$B \rightarrow bb$$

  We then add the new rules

  $$S \rightarrow a \mid bc \mid bb$$
  $$A \rightarrow bb$$
  $$B \rightarrow a \mid bc$$

  We finally obtain the equivalent grammar G' with P' consisting of

  $$S \rightarrow Aa \mid a \mid bc \mid bb$$
  $$A \rightarrow a \mid bc \mid bb$$
  $$B \rightarrow bb \mid a \mid bc$$

  Notice that B and its associate production become useless.

Minimization of Context Free Grammar

- **Theorem:**
  Let L be a context-free language that does not contain □ . There exists a context-free grammar that generates L and that does not have any useless productions, □-productions or unit-productions.

  Proof:
  We need to remove the undesirable productions using the following sequence of steps.
  1. Remove □-productions
  2. Remove unit-productions
  3. Remove useless productions

Chomsky Normal Form

- **Definition:**
  A context-free grammar is in Chomsky normal form if all productions are of the form
  $$A \rightarrow BC$$
  or
  $$A \rightarrow a$$
  where A, B, C $\rightarrow V$, and *a is in T*.

  Note: that the number of symbols on the right side of productions is strictly limited; not more than two symbols.

- **Example:**
  The following grammar is in Chomsky normal form.
  $$S \rightarrow AS \mid a$$
  $$A \rightarrow SA \mid b$$

  On the other hand, the grammar below is not.
  $$S \rightarrow AS \mid AAS$$
  $$A \rightarrow SA \mid aa$$

- **Theorem:**
  Any context-free grammar $G = (V, T, S, P)$ with € □ $L(G)$ has an equivalent grammar G' $= (V', T', S, P')$ in Chomsky normal form.

  Proof:
  First we assume (based on previous Theorem) without loss of generality that G has no □-productions and no unit-productions. Then, we show how to construct G' in two steps.

  Step 1:
  Construct a grammar $G_1 = (V_1, T, S, P_1)$ from G by considering all productions in P of the form
  $$A \rightarrow x_1x_2\ldots x_n$$
  Where each x_i is a symbol either in V or in T.

Note that if $n = 1$, x_1 must be a terminal because there is no unit-productions in G. In this case, put the production into P_1.

If n □ 2, introduce new variables B_a for each a T. Then, for each production of the form $A \rightarrow x_1x_2\ldots x_n$, we shall remove all terminals from productions whose right side has length greater than one

This is done by putting into P_1 a production

$$A \rightarrow C_1C_2\ldots C_n$$

Where

$$C_i = x_i \text{ if } x_i \ □ \ V$$

And

$$C_i = B_a \text{ if } x_i = a$$

And, for every B_a, we also put into P_1 a production

$$B_a \rightarrow a$$

As a consequence of Theorem 6.1, it can be claimed that

$$L(G_1) = L(G)$$

<u>Step 2</u>:

The length of right side of productions is reduced by means of additional variables wherever necessary. First of all, all productions with a single terminal or two variables ($n = 2$) are put into P'. Then, for any production with n □ 2,new variables D_1, D_2, … are introduced and the following productions are put into P'.

$$A \rightarrow C_1D_1$$
$$D_1 \rightarrow C_2D_2$$
$$\ldots$$
$$D_{n-2} \rightarrow C_{n-1}C_n$$

G' is clearly in Chomsky normal form.

- **Example:**

Convert to Chomsky normal form the following grammar G with productions.

$$S \rightarrow ABa$$
$$A \rightarrow aab$$
$$B \rightarrow Ac$$

<u>Solution</u>:

Step 1:

New variables B_a, B_b, B_c are introduced and a new grammar G_1 is obtained.

$$S \rightarrow ABB_a$$
$$A \rightarrow aB_aB_b$$
$$B \rightarrow AB_c$$
$$B_a \rightarrow a$$

$$B_b \rightarrow b$$

$B_C \rightarrow c$

Step 2:

Additional variables are introduced to reduce the length of the first two productions making them into the normal form, we finally obtain G'.

$S \rightarrow AD_1$
$D_1 \rightarrow BB_a$
$A \rightarrow B_aD_2$
$D_2 \rightarrow B_aB_b$
$B \rightarrow AB_c$
$B_a \rightarrow a$
$B_b \rightarrow b$
$B_c \rightarrow c$

Greibach normal form

- **Definition:**
  A context-free grammar is said to be in Greibach normal form if all productions have the form

  $$A \rightarrow ax$$

  where a *is in* T and $x \rightarrow V^*$

  Note that the restriction here is not on the number of symbols on the right side, but rather on the positions of the terminals and variables.

- **Example:**
  The following grammar is not in Greibach normal form.

  $S \rightarrow AB$
  $A \rightarrow aA \mid bB \mid b$
  $B \rightarrow b$

  It can, however, be converted to the following equivalent grammar in Greibach normal form.

  $S \rightarrow aAB \mid bBB \mid bB$
  $A \rightarrow aA \mid bB \mid b$
  $B \rightarrow b$

- **Theorem:**
  For every context-free grammar G with □ □$L(G)$, there exists an equivalent grammar G' in Greibach normal form.

Conversion

- Convert from Chomsky to Greibach in two steps:
  1. From Chomsky to intermediate grammar
     a) Eliminate direct left recursion
     b) Use $A \rightarrow uBv$ rules transformations to improve references (explained later)

2. From intermediate grammar into Greibach

1.a) Eliminate direct left recursion

Step1:

- Before

  $A \rightarrow A\underline{a} \mid \mathbf{b}$

- After

  $A \rightarrow \mathbf{b}Z \mid \mathbf{b}$

  $Z \rightarrow \underline{a}Z \mid \underline{a}$

- Remove the rule with direct left recursion, and create a new one with recursion on the right

Step2:

- Before

  $A \rightarrow A\underline{a} \mid A\underline{b} \mid \mathbf{b} \mid \mathbf{c}$

- After

  $A \rightarrow \mathbf{b}\underline{Z} \mid \mathbf{c}\underline{Z} \mid \mathbf{b} \mid \mathbf{c}$

  $Z \rightarrow \underline{a}Z \mid \underline{b}Z \mid \underline{a} \mid \underline{b}$

- Remove the rules with direct left recursion, and create new ones with recursion on the right

Step3:

- Before

  $A \rightarrow A\underline{B} \mid \mathbf{BA} \mid \mathbf{a}$ $\rightarrow$

  $B \rightarrow b \mid c$

- After

  $A \rightarrow \mathbf{BA}\underline{Z} \mid \mathbf{a}\underline{Z} \mid \mathbf{BA} \mid \mathbf{a}$

  $Z \rightarrow \underline{B}Z \mid \underline{B}$

  $B \rightarrow b \mid c$

1.b) Transform $A \rightarrow uBv$ rules

- Before

  $A \rightarrow uBb$

  $B \rightarrow w1 \mid w1 \mid \ldots \mid wn$

- After

  Add $A \rightarrow uw1b \mid uw1b \mid \ldots \mid uwnb$

  Delete $A \rightarrow uBb$

<u>Background Information for the Pumping Lemma for Context-Free Languages</u>

- **Definition:** Let G = (V, T, P, S) be a CFL. If every production in P is of the form

A –> BC

or A –> a

where A, B and C are all in V and a is in T, then G is in Chomsky Normal Form (CNF).

- **Example:**

  S –> AB | BA |
  aSb A –> a
  B –> b

- **Theorem:** Let L be a CFL. Then L – {ε} is a CFL.

- **Theorem:** Let L be a CFL not containing {ε}. Then there exists a CNF grammar G such that L = L(G).

- **Definition:** Let T be a tree. Then the height of T, denoted h(T), is defined as follows:
  - If T consists of a single vertex then h(T) = 0
  - If T consists of a root r and subtrees T_1, T_2, … T_k, then h(T) = $\max_i\{h(T_i)\} + 1$

- **Lemma:** Let G be a CFG in CNF. In addition, let w be a string of terminals where A=>*w and w has a derivation tree T. If T has height h(T)□ 1, then |w| □$2^{h(T)-1}$.

- **Proof:** By induction on h(T) (exercise).

- **Corollary:** Let G be a CFG in CNF, and let w be a string in L(G). If |w| □ 2^k, where k □ 0, then any derivation tree for w using G has height at least k+1.

- **Proof:** Follows from the lemma.

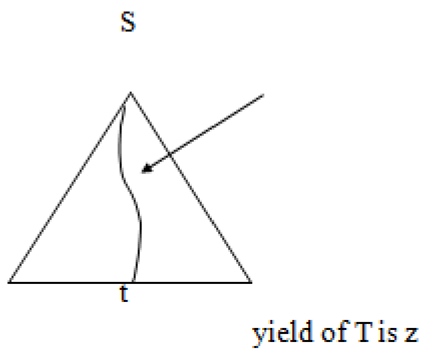
Pumping Lemma for Context-Free Languages

- **Lemma:**
  Let G = (V, T, P, S) be a CFG in CNF, and let n = $2^{|V|}$. If z is a string in L(G) and |z| □ n, then there exist strings u, v, w, x and y in T* such that z=uvwxy and:
  - |vx| □ 1 (i.e., |v| + |x| □ 1)
  - |vwx| □ n
  - uv^iwx^iy is in L(G), for all i □ 0

- **Proof:**
  Since |z| □ n = 2^k, where k = |V|, it follows from the corollary that any derivation tree for z has height at least k+1.

  By definition such a tree contains a path of length at least k+1.

  Consider the longest such path in the tree:

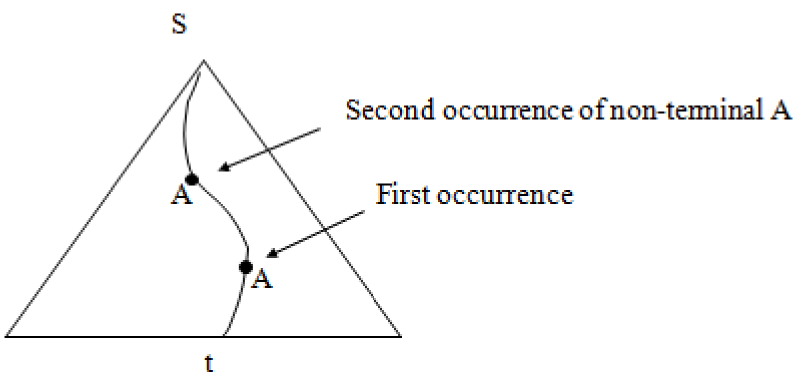


Such a path has:

- Length □ k+1 (i.e., number of edges in the path is □ k+1)
- At least k+2 nodes
- 1 terminal

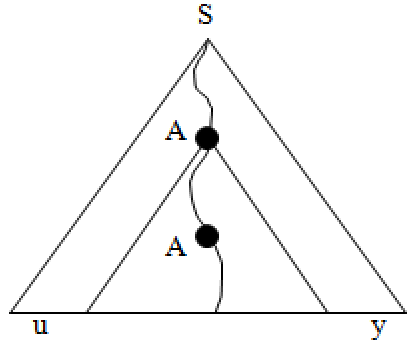
At least k+1 non-terminals

- Since there are only k non-terminals in the grammar, and since k+1 appear on this long path, it follows that some non-terminal (and perhaps many) appears at least twice on this path.
- Consider the first non-terminal that is repeated, when traversing the path from the leaf to the root.

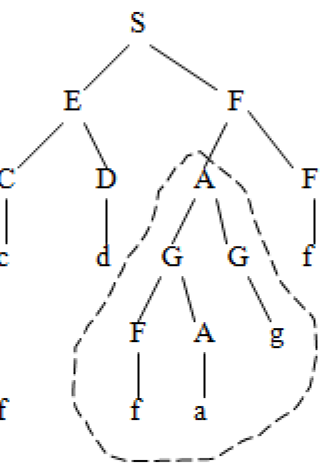


This path, and the non-terminal A will be used to break up the string z.

- **Generic Description:**

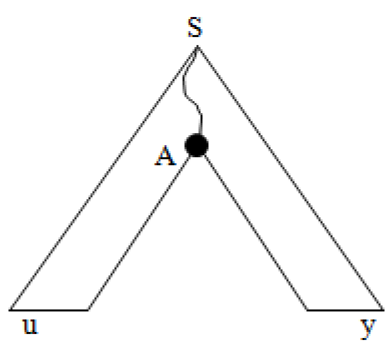


- **Example:**



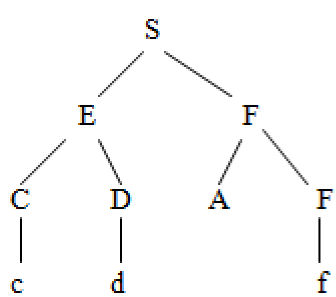
In this case u = cd and y =f

- **Cut out the subtree rooted at A:**



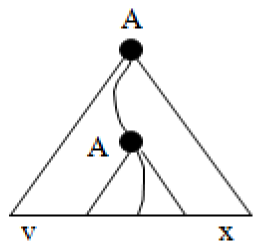
S =>* uAy (1)

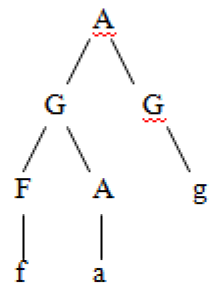
- **Example:**



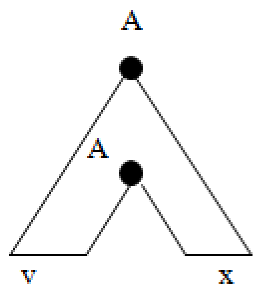
S =>* cdAf

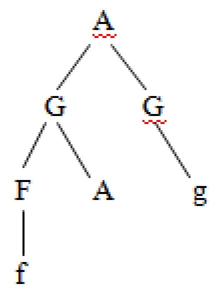
- **Consider the subtree rooted at A:**





- **Cut out the subtree rooted at the first occurrence of A:**

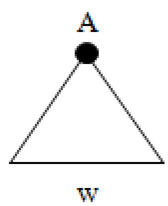




A =>* vAx (2)

A =>* fAg

- **Consider the smallest subtree rooted at A:**



A =>* w (3)

A =>* a

- **Collectively (1), (2) and (3) give us:**

S =>* uAy	(1)
=>* uvAxy	(2)
=>* uvwxy	(3)
=>* z	since z=uvwxy

- **In addition, (2) also tells us:**

S =>* uAy	(1)
=>* uvAxy	(2)

=>* uv^2Ax^2y (2)
=>* uv^2wx^2y (3)

- **More generally:**
  S =>* uv^iwx^iy for all i>=1

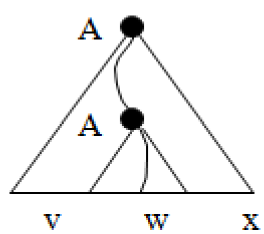
- **And also:**
  S =>* uAy (1)
  =>* uwy (3)

- **Hence:**
  S =>* uv^iwx^iy for all i>=0

- **Consider the statement of the Pumping Lemma:**

  – *What is n?*
  $n = 2^k$, where k is the number of non-terminals in the grammar.

  – *Why is |v| + |x| □ 1?*

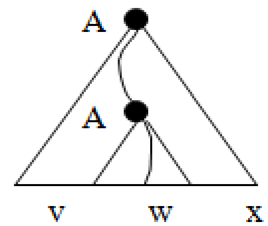


Since the height of this subtree is □ 2, the first production is A->V1V2. Since no non-terminal derives the empty string (in CNF), either V1 or V2 must derive a non-empty v or x. More specifically, if w is generated by V1, then x contains at least one symbol, and if w is generated by V2, then v contains at least one symbol.

– *Why is |vwx| □ n?*

Observations:

- The repeated variable was the first repeated variable on the path from the bottom, and therefore (by the pigeon-hole principle) the path from the leaf to the second occurrence of the non-terminal has length at most k+1.
- Since the path was the largest in the entire tree, this path is the longest in the subtree rooted at the second occurrence of the non-terminal. Therefore the subtree has height □ k+1. From the lemma, the yield of the subtree has length □ 2^k=n.



<u>CFL Closure Properties</u>

- **Theorem#1:**
  The context-free languages are closed under concatenation, union, and Kleene closure.

- **Proof:**
  Start with 2 CFL $L(H1)$ and $L(H2)$ generated by $H1 = (N1,T1,R1,s1)$ and $H2 = (N2,T2,R2,s2)$.
  Assume that the alphabets and rules are disjoint.

  <u>*Concatenation:*</u>
  Formed by $L(H1)\cdot L(H2)$ or a string in $L(H1)$ followed by a string in $L(H2)$ which can be generated by $L(H3)$ generated by $H3 = (N3,T3,R3,s3)$. $N3 = N1 \cup N2$, $T3 = T1 \cup T2$, $R3 = R1 \cup R2 \cup \{s3 \text{-->} s1s2\}$ where $s3 \rightarrow s1s2$ is a new rule introduced. The new rule generates a string of L(H1) then a string of $L(H2)$. Then $L(H1) \cdot L(H2)$ is context-free.

  <u>*Union*</u>*:*
  Formed by $L(H1) \cup L(H2)$ or a string in $L(H1)$ or a string in $L(H2)$. It is generated by $L(H3)$ generated by $H4 = (N4,T4,R4,s4)$ where $N4 = N1 \cup N2$, $T4 = T1 \cup T2$, and $R4 = R1 \cup R2 \cup \{s4 \text{-->} s1,\ s4 \rightarrow s2\}$, the new rules added will create a string of $L(H1)$ or $L(H2)$. Then $L(H1) \cup L(H2)$ is context-free.

  <u>*Kleene*</u>*:*
  Formed by $L(H1)^*$ is generated by the grammar $L(H5)$ generated by $H5 = (N1,T1,R5,s1)$ with $R5 = R1 \cup \{s1 \rightarrow e,\ s1 \rightarrow s1s1\}$. $L(H5)$ includes e, every string in $L(H1)$, and through i-1 applications of $s1 \rightarrow s1s1$, every string in $L(H1)i$. Then $L(H1)^*$ is generated by $H5$ and is context-free.

- **Theorem#2:**
  The set of context-free languages is not closed under complementation or intersection.

- **Proof:**
  Intersections of two languages $L1 \cap L2$ can be defined in terms of the Complement and Union operations as follows:
  $L1 \cap L2 = \Sigma^* - (\Sigma^* - L1) \cup (\Sigma^* - L2)$

  Therefore if CFL are closed under intersection then it is closed under compliment and if closed under compliment then it is closed under intersection.

The proof is just showing two context-free languages that their intersection is not a context-free language.

Choose *L1* = {*anbncm* | *m,n* □□0} is generated by grammar *H1* = {*N1,T1,R1,s1*}, where

N1 = {*s*, *A*, *B*}
T1 = {*a*, *b*, *c*}
R1 = {*s*□*AB*,
A□*aAb*,
A□*e*,
B□*Bc*,
B□*e*}.

Choose *L2* = {*ambncn* | *m,n* □□0} is generated by grammar *H2* = {*N2,T2,R2,s2*}, where

N1 = {*s*, *A*, *B*}
T1 = {*a*, *b*, *c*}
R2 = {*s*□ *AB*,
A□*aA*,
A□*e*,
B□*bBc*,
B□*e*}.

Thus *L1* and *L2* are both context-free.

The intersection of the two languages is *L3* = {*anbncn* | *n* □□0}. This language has already been proven earlier in this paper to be not context-free. Therefore CFL are not closed under intersections, which also means that it is not closed under complementation.

Pushdown Automata (PDA)

• **Informally:**

– A PDA is an NFA-ε with a stack.

–Transitions are modified to accommodate stack operations.

• **Questions:**

–What is a stack?

–How does a stack help?

• A DFA can “remember” only a finite amount of information, whereas a PDA can “remember” an infinite amount of (certain types of) information.

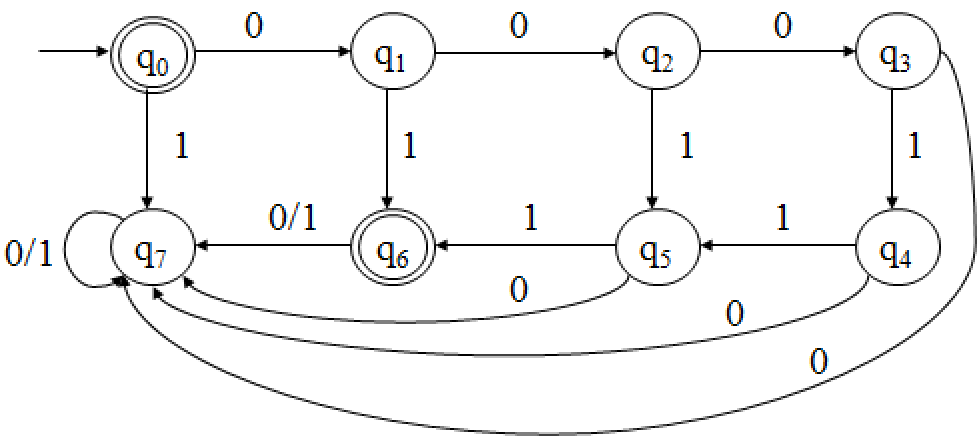
• **Example:**

$\{0^n1^n \mid 0 =< n\}$ Is *not* regular.

{0^n1^n | 0□ n□ k, for some fixed k}   Is regular, for any fixed k.

• **For k=3:**

L = {ε, 01, 0011, 000111}



• In a DFA, each state remembers a finite amount of information.

• To get {0^n1^n | 0□ n} with a DFA would require an infinite number of states using the preceding technique.

• An infinite stack solves the problem for {0^n1^n | 0□ n} asfollows:

–Read all 0's and place them on a stack

–Read all 1's and match with the corresponding 0's on the stack

• Only need two states to do this in a PDA

• Similarly for {$0^n1^m0^{n+m}$ | n,m□ 0}

<u>Formal Definition of a PDA</u>

• A <u>pushdown automaton (PDA)</u> is a seven-tuple:

M = (Q, Σ, Γ, δ, q0, z0, F)

Q	A <u>finite</u> set of states
Σ	A <u>finite</u> input alphabet
Γ	A <u>finite</u> stack alphabet
q0	The initial/starting state, q0 is in
Q z0	A starting stack symbol, is in Γ
F	A set of final/accepting states, which is a subset of Q
δ	A transition function, where

δ: Q x (Σ U {ε}) x Γ → finite subsets of Q x Γ^*

• Consider the various parts of δ:

Q x (Σ U {ε}) x Γ → finite subsets of Q x Γ^*

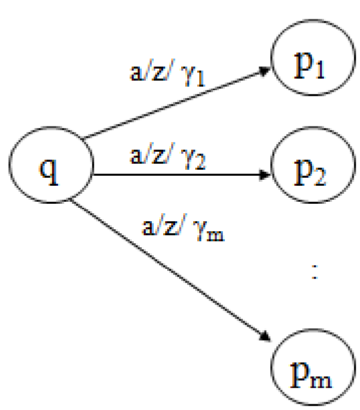
–Q on the LHS means that at each step in a computation, a PDA must consider its' current state.
–Γ on the LHS means that at each step in a computation, a PDA must consider the symbol on top of its' stack.
–Σ U {ε} on the LHS means that at each step in a computation, a PDA may or may not consider the current input symbol, i.e., it may have epsilon transitions.

–"Finite subsets" on the RHS means that at each step in a computation, a PDA will have several options.
–Q on the RHS means that each option specifies a new state.
–Γ^* on the RHS means that each option specifies zero or more stack symbols that will replace the top stack symbol.

• **Two types of PDA transitions #1:**

$\delta(q, a, z) = \{(p1,\gamma1), (p2,\gamma2),\ldots, (pm,\gamma m)\}$

–Current state is q
–Current input symbol is a
–Symbol currently on top of the stack z
–Move to state p_i from q
–Replace z with γ_i on the stack (leftmost symbol on top)
–Move the input head to the next input symbol



• **Two types of PDA transitions #2:**

$\delta(q, \varepsilon, z) = \{(p1,\gamma1), (p2,\gamma2),\ldots, (pm,\gamma m)\}$

–Current state is q

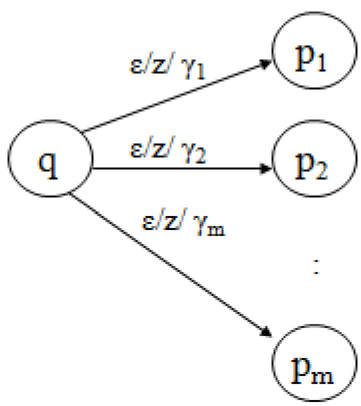
–Current input symbol is not considered

–Symbol currently on top of the stack z

–Move to state p_i from q

–Replace z with γ_i on the stack (leftmost symbol on top)

–**No input symbol is read**



• **Example:** (balanced parentheses)

M = ({q1}, {"(", ")"}, {L, #}, δ, q1, #, Ø)

δ:

(1)	δ(q1, (, #) = {(q1, L#)}
(2)	δ(q1,), #) = Ø
(3)	δ(q1, (, L) = {(q1, LL)}
(4)	δ(q1,), L) = {(q1, ε)}
(5)	δ(q1, ε, #) = {(q1, ε)}
(6)	δ(q1, ε, L) = Ø

• **Goal:** (acceptance)

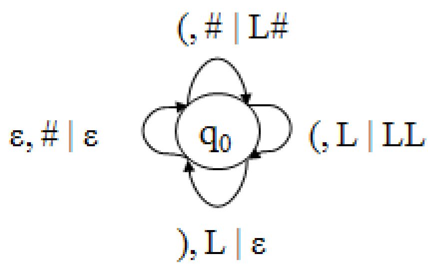
–Terminate in a non-null state

–Read the entire input string

–Terminate with an empty stack

• Informally, a string is accepted if there exists a computation that uses up all the input and leaves the stack empty.

• Transition Diagram:



• Example Computation:

Current Input	Stack	Transition	
(())	#		
())	L#	(1)	- Could have applied rule
))	LL#	(3)	(5), but it would have
)	L#	(4)	done no good
ε	#	(4)	
ε	-	(5)	

• **Example PDA #1:** For the language $\{x \mid x = wcw^r$ and w in $\{0,1\}^*\}$

M = ({q1, q2}, {0, 1, c}, {R, B, G}, δ, q1, R, Ø)

δ:

(1)	δ(q1, 0, R) = {(q1, BR)}	(9)	δ(q1, 1, R) = {(q1, GR)}
(2)	δ(q1, 0, B) = {(q1, BB)}	(10)	δ(q1, 1, B) = {(q1, GB)}
(3)	δ(q1, 0, G) = {(q1, BG)}	(11)	δ(q1, 1, G) = {(q1, GG)}
(4)	δ(q1, c, R) = {(q2, R)}		
(5)	δ(q1, c, B) = {(q2, B)}		
(6)	δ(q1, c, G) = {(q2, G)}		
(7)	δ(q2, 0, B) = {(q2, ε)}	(12)	δ(q2, 1, G) = {(q2, ε)}
(8)	δ(q2, ε, R) = {(q2, ε)}		

• **Notes:**

–Only rule #8 is non-deterministic.

–Rule #8 is used to pop the final stack symbol off at the end of a computation.

- **Example Computation:**

(1) δ(q1, 0, R) = {(q1, BR)}
(2) δ(q1, 0, B) = {(q1, BB)}
(3) δ(q1, 0, G) = {(q1, BG)}
GG)} (4)
(5) δ(q1, c, B) = {(q2, B)}
(6) δ(q1, c, G) = {(q2, G)}
(7) δ(q2, 0, B) = {(q2, ε)}
ε)} (8) δ(q2, ε, R) = {(q2, ε)}

(9) δ(q1, 1, R) = {(q1, GR)}
(10) δ(q1, 1, B) = {(q1, GB)}
(11) δ(q1, 1, G) = {(q1,
δ(q1, c, R) = {(q2, R)}
(12) δ(q2, 1, G) = {(q2,

State	Input	Stack	Rule Applied	Rules Applicable
q1	**0**1c10	R	-	(1)
q1	**1**c10	BR	(1)	(10)
q1	**c**10	GBR	(10)	(6)
q2	**1**0	GBR	(6)	(12)
q2	**0**	BR	(12)	(7)
q2	**ε**	R	(7)	(8)
q2	ε	ε	(8)	-

- **Example Computation:**

(1) δ(q1, 0, R) = {(q1, BR)}
(2) δ(q1, 0, B) = {(q1, BB)}
(3) δ(q1, 0, G) = {(q1, BG)}
GG)} (4)
(5) δ(q1, c, B) = {(q2, B)}
(6) δ(q1, c, G) = {(q2, G)}
(7) δ(q2, 0, B) = {(q2, ε)}
ε)} (8) δ(q2, ε, R) = {(q2, ε)}

(9) δ(q1, 1, R) = {(q1, GR)}
(10) δ(q1, 1, B) = {(q1, GB)}
(11) δ(q1, 1, G) = {(q1,
δ(q1, c, R) = {(q2, R)}
(12) δ(q2, 1, G) = {(q2,

State	Input	Stack	Rule Applied
q1	**1**c1	R	
q1	**c**1	GR	(9)
q2	**1**	GR	(6)
q2	**ε**	R	(12)
q2	ε	ε	(8)

- **Definition:** |—* is the reflexive and transitive closure of |—.

–I |—* I for each instantaneous description I

–If I |— J and J |—* K then I |—* K

• Intuitively, if I and J are instantaneous descriptions, then I $|$—* J means that J follows from I by zero or more transitions.

• **Definition:** Let $M = (Q, \Sigma, \Gamma, \delta, q_0, z_0, F)$ be a PDA. The *language accepted by empty stack*, denoted $L_E(M)$, is the set

$$\{w \mid (q_0, w, z_0) \vdash^* (p, \varepsilon, \varepsilon) \text{ for some } p \text{ in } Q\}$$

• **Definition:** Let $M = (Q, \Sigma, \Gamma, \delta, q_0, z_0, F)$ be a PDA. The *language accepted by final state,* denoted $L_F(M)$, is the set

$$\{w \mid (q_0, w, z_0) \vdash^* (p, \varepsilon, \gamma) \text{ for some } p \text{ in } F \text{ and } \gamma \text{ in } \Gamma^*\}$$

• **Definition:** Let $M = (Q, \Sigma, \Gamma, \delta, q_0, z_0, F)$ be a PDA. The *language accepted by empty stack and final state*, denoted $L(M)$, is the set

$$\{w \mid (q_0, w, z_0) \vdash^* (p, \varepsilon, \varepsilon) \text{ for some } p \text{ in } F\}$$

• **Lemma 1:** Let $L = L_E(M_1)$ for some PDA M_1. Then there exits a PDA M_2 such that $L = L_F(M_2)$.

• **Lemma 2:** Let $L = L_F(M_1)$ for some PDA M_1. Then there exits a PDA M_2 such that $L = L_E(M_2)$.

• **Theorem:** Let L be a language. Then there exits a PDA M_1 such that $L = L_F(M_1)$ if and only if there exists a PDA M_2 such that $L = L_E(M_2)$.

• **Corollary:** The PDAs that accept by empty stack and the PDAs that accept by final state define the same class of languages.

• **Note:** Similar lemmas and theorems could be stated for PDAs that accept by both final state and empty stack.

Greibach Normal Form (GNF)

• **Definition:** Let $G = (V, T, P, S)$ be a CFL. If every production in P is of the form

$$A \rightarrow a\alpha$$

Where A is in V, a is in T, and α is in V^*, then G is said to be in Greibach Normal Form (GNF).

• **Example:**

$S \rightarrow aAB \mid bB$
$A \rightarrow aA \mid a$

B –> bB | c

• **Theorem:** Let L be a CFL. Then L – {ε} is a CFL.

• **Theorem:** Let L be a CFL not containing {ε}. Then there exists a GNF grammar G such that L = L(G).

• **Lemma 1:** Let L be a CFL. Then there exists a PDA M such that L = L_E(M).

• **Proof:** Assume without loss of generality that ε is not in L. The construction can be modified to include ε later.

Let G = (V, T, P, S) be a CFG, and assume without loss of generality that G is in GNF.
Construct M = (Q, Σ, Γ, δ, q, z, Ø) where:

Q = {q}
Σ = T
Γ = V
z = S

δ: for all a in Σ and A in Γ, δ(q, a, A) contains (q, γ) if A –> aγ is in P or rather:
δ(q, a, A) = {(q, γ) | A –> aγ is in P and γ is in Γ*}, for all a in Σ and A in Γ

• For a given string x in Σ* , M will attempt to simulate a leftmost derivation of x with G.

• **Example #1:** Consider the following CFG in GNF.

S → aS	G is in GNF
S → a	L(G) = a+

Construct M as:

Q = {q}
Σ = T = {a}
Γ = V = {S}
z = S

δ(q, a, S) = {(q, S), (q, ε)}
δ(q, ε, S) = Ø

• **Example #2:** Consider the following CFG in GNF.

(1)	S –> aA	
(2)	S –> aB	
(3)	A –> aA	G is in GNF
(4)	A –> aB	L(G) = a+b+

(5) B –> bB
(6) B –> b

Construct M as:

Q = {q}
Σ = T = {a, b}
Γ = V = {S, A, B}
z = S

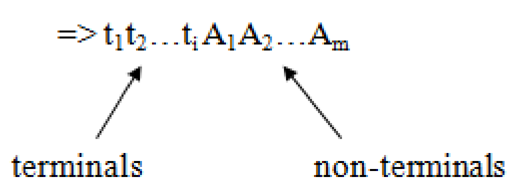
(1) δ(q, a, S) = {(q, A), (q, B)} From productions #1 and 2, S->aA, S->aB
(2) δ(q, a, A) = {(q, A), (q, B)} From productions #3 and 4, A->aA, A->aB
(3) δ(q, a, B) = Ø
(4) δ(q, b, S) = Ø
(5) δ(q, b, A) = Ø
(6) δ(q, b, B) = {(q, B), (q, ε)} From productions #5 and 6, B->bB, B->b
(7) δ(q, ε, S) = Ø
(8) δ(q, ε, A) = Ø
(9) δ(q, ε, B) = Ø Recall δ: Q x (Σ U {ε}) x Γ –> finite subsets of Q x Γ*

•For a string w in L(G) the PDA M will simulate a leftmost derivation of w.

–If w is in L(G) then (q, w, z0) |—* (q, ε, ε)

–If (q, w, z0) |—* (q, ε, ε) then w is in L(G)

•Consider generating a string using G. Since G is in GNF, each sentential form in a *leftmost* derivation has form:



•And each step in the derivation (i.e., each application of a production) adds a terminal and some non-terminals.

$$A_1 \to t_{i+1}\alpha$$

$$\Rightarrow t_1t_2\ldots t_i\, t_{i+1}\, \alpha A_1A_2\ldots A_m$$

•Each transition of the PDA simulates one derivation step. Thus, the ith step of the PDAs' computation corresponds to the ith step in a corresponding leftmost derivation.

•After the ith step of the computation of the PDA, $t_1t_2\ldots t_{i+1}$ are the symbols that have already been read by the PDA and $\alpha A_1A_2\ldots A_m$ are the stack contents.

• For each leftmost derivation of a string generated by the grammar, there is an equivalent accepting computation of that string by the PDA.

• Each sentential form in the leftmost derivation corresponds to an instantaneous description in the PDA's corresponding computation.

• For example, the PDA instantaneous description corresponding to the sentential form:

$$\Rightarrow t_1t_2\ldots t_i\ A_1A_2\ldots A_m$$

would be: $(q,\ t_{i+1}t_{i+2}\ldots t_n\ ,\ A_1A_2\ldots A_m)$

• **Example:** Using the grammar from example #2:

S => aA	(1)
=> aaA	(3)
=> aaaA	(3)
=> aaaaB	(4)
=> aaaabB	(5)
=> aaaabb	(6)

• The corresponding computation of the PDA:

•(q, aaaabb, S)	|— (q, aaabb, A)	(1)/1
	|— (q, aabb, A)	(2)/1
	|— (q, abb, A)	(2)/1
	|— (q, bb, B)	(2)/2
	|— (q, b, B)	(6)/1
	|— (q, ε, ε)	(6)/2

–String is read
–Stack is emptied
–Therefore the string is accepted by the PDA

•**Example #3:** Consider the following CFG in GNF.

(1)	S –> aABC	
(2)	A –> a	G is in GNF
(3)	B –> b	
(4)	C –> cAB	
(5)	C –> cC	

Construct M as:

Q = {q}
$\Sigma = T = \{a, b, c\}$
$\Gamma = V = \{S, A, B, C\}$
z = S

(1) $\delta(q, a, S) = \{(q, ABC)\}$ S->aABC (9) $\delta(q, c, S) = \varnothing$
(2) $\delta(q, a, A) = \{(q, \varepsilon)\}$ A->a (10) $\delta(q, c, A) = \varnothing$
(3) $\delta(q, a, B) = \varnothing$ (11) $\delta(q, c, B) = \varnothing$
(4) $\delta(q, a, C) = \varnothing$ C->cAB|cC (12) $\delta(q, c, C) = \{(q, AB), (q, C))$
(5) $\delta(q, b, S) = \varnothing$ (13) $\delta(q, \varepsilon, S) = \varnothing$
(6) $\delta(q, b, A) = \varnothing$ (14) $\delta(q, \varepsilon, A) = \varnothing$
(7) $\delta(q, b, B) = \{(q, \varepsilon)\}$ B->b (15) $\delta(q, \varepsilon, B) = \varnothing$
(8) $\delta(q, b, C) = \varnothing$ (16) $\delta(q, \varepsilon, C) = \varnothing$

• **Notes:**
–Recall that the grammar G was required to be in GNF before the construction could be applied.
–As a result, it was assumed at the start that ε was not in the context-free language L.

• **Suppose ε is in L:**

1) First, let L' = L – {ε}

Fact: If L is a CFL, then L' = L – {ε} is a CFL.

By an earlier theorem, there is GNF grammar G such that L' = L(G).

2) Construct a PDA M such that L' =

LE(M) How do we modify M to accept ε?

Add $\delta(q, \varepsilon, S) = \{(q, \varepsilon)\}$? No!

• **Counter Example:**

Consider L = {ε, b, ab, aab, aaab, …}
Then L' = {b, ab, aab, aaab, …}

• **The GNF CFG for L':**

(1) $S \rightarrow aS$
(2) $S \rightarrow b$

• **The PDA M Accepting L':**

$Q = \{q\}$
$\Sigma = T = \{a, b\}$
$\Gamma = V = \{S\}$
$z = S$

$\delta(q, a, S) = \{(q, S)\}$
$\delta(q, b, S) = \{(q, \varepsilon)\}$
$\delta(q, \varepsilon, S) = \emptyset$

• If $\delta(q, \varepsilon, S) = \{(q, \varepsilon)\}$ is added then:

$L(M) = \{\varepsilon, a, aa, aaa, \ldots, b, ab, aab, aaab, \ldots\}$

3) Instead, add a new *start* state q' with transitions:

$\delta(q', \varepsilon, S) = \{(q', \varepsilon), (q, S)\}$

• **Lemma 1:** Let L be a CFL. Then there exists a PDA M such that $L = L_E(M)$.

• **Lemma 2:** Let M be a PDA. Then there exists a CFG grammar G such that $L_E(M) = L(G)$.

• **Theorem:** Let L be a language. Then there exists a CFG G such that $L = L(G)$ iff there exists a PDA M such that $L = L_E(M)$.

• **Corollary:** The PDAs define the CFLs.

Equivalence of CFG to PDAs

- **Example:** Consider the grammar for arithmetic expressions we introduced earlier. It is reproduced below for convenience. G = ({E, T, F}, {n, v, +, *, (,)}, P, E), where

  E = { 1: E □ E + T,
  2: E □ T,
  3: T □ T□ F,
  4: T □ F,
  5: F □ n,
  6: F □ v,
  7: F □ (E),
  }

  Suppose the input to our parser is the expression, n*(v+n*v). Since G is unambiguous this expression has only one leftmost derivation, p = 2345712463456. We describe the behavior of the PDA in general, and then step through its moves using this derivation to guide the computation.

- **PDA Simulator:**
  - Step 1: Initialize the stack with the start symbol (E in this case). The start symbol will serve as the bottom of stack marker (Z_0).
  - Step 2: Ignoring the input, check the top symbol of the stack.
    - Case (a) Top of stack is a nonterminal, “X”: non-deterministically decide which
      X-rule to use as the next step of the derivation. After selecting a rule, replace X in the stack with the rightpart of that rule. If the stack is non-empty, repeat step 2. Otherwise, halt (input may or may not be empty.)
    - Case(b) Top of stack is a terminal, “a”: Read the next input. If the input matches a, then pop the stack and repeat step 2.
      Otherwise, halt (without popping “a” from the stack.)
  - This parsing algorithm by showing the sequence of configurations the parser would assume in an accepting computation for the input, n*(v+n*v).
    Assume “q0” is the one and only state of this PDA.
  - p (leftmost derivation in G) = 2345712463456

    (q0, n*(v+n*v), E)

2□ M (q0, n*(v+n*v), T)

3□ M (q0, n*(v+n*v),T*F)

4□ M (q0, n*(v+n*v),F*F)

5□ M (q0, n*(v+n*v), n*F) read□ M (q0, *(v+n*v), *F)

read□ M (q0, (v+n*v), F)

7□ M (q0, (v+n*v), (E)) read□ M (q0, v+n*v), E))

1□ M (q0, v+n*v),E+T))

2□ M (q0, v+n*v), T+T))

4□ M (q0, v+n*v), F+T)) read□ M (q0, +n*v), +T))

6□ M (q0, v+n*v), v+T))

read□ M (q0, n*v), T))

3□ M (q0, n*v), T*F))

4□ M (q0, n*v), F*F))

5□ M (q0, n*v), n*F)) read□ M (q0, *v), *F))

read□ M (q0, v), F))

6□ M (q0, v),v)) read□ M (q0,),)) read□ M

(q0, l, l) accept!

Deterministic PDAs and DCFLs

- **Definition:** A *Deterministic Pushdown Automaton* (DPDA) is a 7-tuple,

M = (Q, □ , □ , □ , q0, Z0, A),
where
Q = finite set of states,
□ = input alphabet,
□ = stack alphabet,

q0 □ Q = the initial state,
Z0 □ □ = bottom of stack marker (or initial stack symbol), and

□ : Q □ (□ □ {L}) □ □ Q □ □ * = the transition function (not necessarily total). Specifically,

[1] if d(q, a, Z) is defined for some a □ □ and Z □ □ , then d(q, L, Z) = □ and
□ d(q, a, Z)□ = 1.

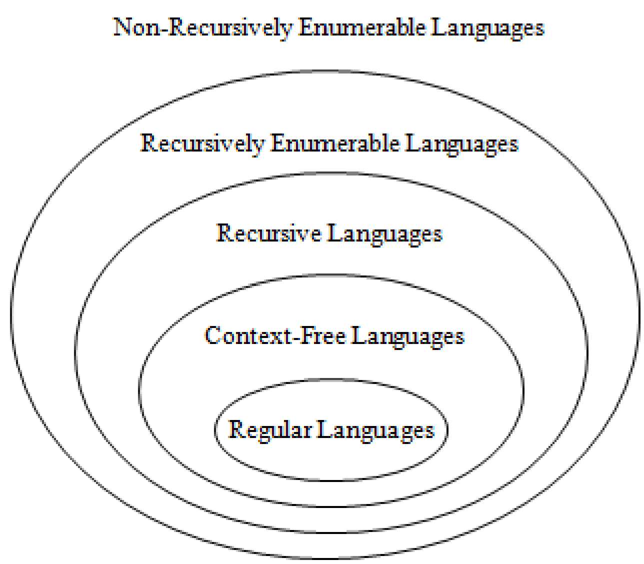
[2] Conversely, if d(q, L, Z) □ □ ,for some Z, then d(q, a, Z) □ □ ,for all a □ □ , and □ d(q, L, Z)□ 1.

- **NOTE:** DPDAs can accept their input either by final state or by empty stack – just as for the non-deterministic model. We therefore define *Dstk* and *Dste*, respectively, as the corresponding families of Deterministic Context-free Languages accepted by a DPDA by empty stack and final state.

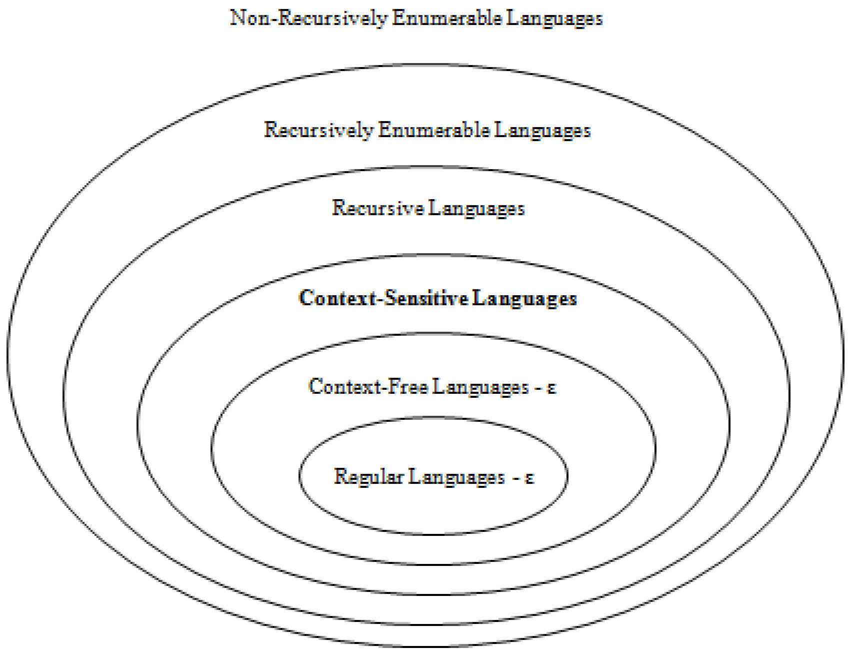
UNIT IV:

Turing Machines (TM)

- **Generalize the class of CFLs:**



- **Another Part of the Hierarchy:**

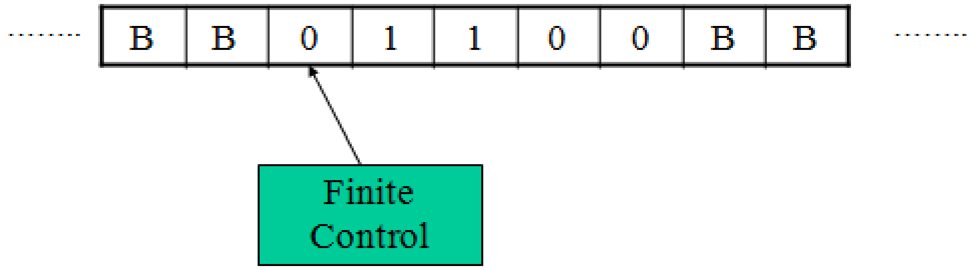


- Recursively enumerable languages are also known as *type 0* languages.
- Context-sensitive languages are also known as *type 1* languages.
- Context-free languages are also known as *type 2* languages.
- Regular languages are also known as *type 3* languages.

- TMs model the computing capability of a general purpose computer, which informally can be described as:
  - Effective procedure
    - Finitely describable
    - Well defined, discrete, “mechanical” steps
    - Always terminates
  - Computable function
    - A function computable by an effective procedure

- TMs formalize the above notion.

Deterministic Turing Machine (DTM)



- Two-way, infinite tape, broken into cells, each containing one symbol.
- Two-way, read/write tape head.
- Finite control, i.e., a program, containing the position of the read head, current symbol being scanned, and the current state.
- An input string is placed on the tape, padded to the left and right infinitely with blanks, read/write head is positioned at the left end of input string.
- In one move, depending on the current state and the current symbol being scanned, the TM 1) changes state, 2) prints a symbol over the cell being scanned, and 3) moves its' tape head one cell left or right.
- Many modifications possible.

Formal Definition of a DTM

– A DTM is a seven-tuple:

M = (Q, Σ, Γ, δ, q0, B,

F)

Q	A finite set of states
Γ	A finite tape alphabet
B	A distinguished blank symbol, which is in Γ
Σ	A finite input alphabet, which is a subset of Γ– {B}
q0	The initial/starting state, q0 is in Q
F	A set of final/accepting states, which is a subset of Q
δ	A next-move function, which is a *mapping* from Q x Γ –> Q x Γ x {L,R}

Intuitively, δ(q,s) specifies the next state, symbol to be written and the direction of tape head movement by M after reading symbol s while in
state q.

- **Example #1:** $\{0^n1^n \mid n >= 1\}$

	0	1	X	Y	B
q0	(q1, X, R)	-	-	(q3, Y, R)	-
q1	(q1, 0, R)	(q2, Y, L)	-	(q1, Y, R)	-
q2	(q2, 0, L)	-	(q0, X, R)	(q2, Y, L)	-
q3	-	-	-	(q3, Y, R)	(q4, B, R)
q4	-	-	-	-	-

- **Sample Computation:** (on 0011)

$$
\begin{aligned}
q_0 0011 &\vdash Xq_1 011 \\
&\vdash X0q_1 11 \\
&\vdash Xq_2 0Y1 \\
&\vdash q_2 X0Y1 \\
&\vdash Xq_0 0Y1 \\
&\vdash XXq_1 Y1 \\
&\vdash XXYq_1 1 \\
&\vdash XXq_2 YY \\
&\vdash Xq_2 XYY \\
&\vdash XXq_0 YY \\
&\vdash XXYq_3 Y \\
&\vdash XXYYq_3 \\
&\vdash XXYYBq_4
\end{aligned}
$$

Making a TM for $\{0^n1^n \mid n >= 1\}$

Try n=1 first.

- q0 is on B expecting to see 0, sees it
- q1 sees next 0
- q1 hits a 1
- q2 sees a 0, continues
- q2 sees X, loops step 1 through 5
- finished, q0 sees Y (replacement of first 1)
- q3 sees Y
- q3 sees B, done
- blank line for final state q4

Now try for n=2

- q1 hits Y
- q2 sees Y
- complete the unfinished entries verifying "crashes" as it should be

– **Example #1:** $\{0^n1^n \mid n >= 1\}$

	0	1	X	Y	B
q0	(q1, X, R)	-	-	(q3, Y, R)	-
q1	(q1, 0, R)	(q2, Y, L)	-	(q1, Y, R)	-
q2	(q2, 0, L)	-	(q0, X, R)	(q2, Y, L)	-
q3	-	-	-	(q3, Y, R)	(q4, B, R)
q4	-	-	-	-	-

- The TM basically matches up 0's and 1's
- q1 is the "scan right" state
- q2 is the "scan left" state
- q4 is the final state

– **Example #2:** {w | w is in {0,1}* and w ends with a 0}

0
00
10
10110
Not ε

Q = {q0, q1, q2}
Γ = {0, 1, B}
Σ = {0, 1}

F = {q2}

	0	1	B
q0	(q0, 0, R)	(q0, 1, R)	(q1, B, L)
q1	(q2, 0, R)	-	-
q2	-	-	-

- q0 is the "scan right" state
- q1 is the verify 0 state

- **Definition:** Let M = (Q, Σ, Γ, δ, q0, B, F) be a TM, and let w be a string in Σ*. Then w is *accepted* by M iff

  $$q0w \mid\!\!-\!\!-^* \alpha 1 p \alpha 2$$

  Where p is in F and α1 and α2 are in Γ*

- **Definition:** Let M = (Q, Σ, Γ, δ, q0, B, F) be a TM. The *language accepted by M*, denoted L(M), is the set

  $$\{w \mid w \text{ is in } \Sigma^* \text{ and } w \text{ is accepted by } M\}$$

- **Notes:**
  - In contrast to FA and PDAs, if a TM simply *passes through* a final state then the string is accepted.
  - Given the above definition, no final state of an TM need have any exiting transitions. *Henceforth, this is our assumption*.
  - **If x is not in L(M) then M may enter an infinite loop, or halt in a non-final state**.
  - Some TMs halt on all inputs, while others may not. In either case the language defined by TM is still well defined.
- **Definition:** Let L be a language. Then L is *recursively enumerable* if <u>there exists</u> a TM M such that L = L(M).

  - If L is r.e. then L = L(M) for some TM M, and
    - •If x is in L then M halts in a final (accepting) state.
    - •If x is not in L then M may halt in a non-final (non-accepting) state, or loop forever.

- **Definition:** Let L be a language. Then L is *recursive* if <u>there exists</u> a TM M such that L = L(M) and M halts on all inputs.

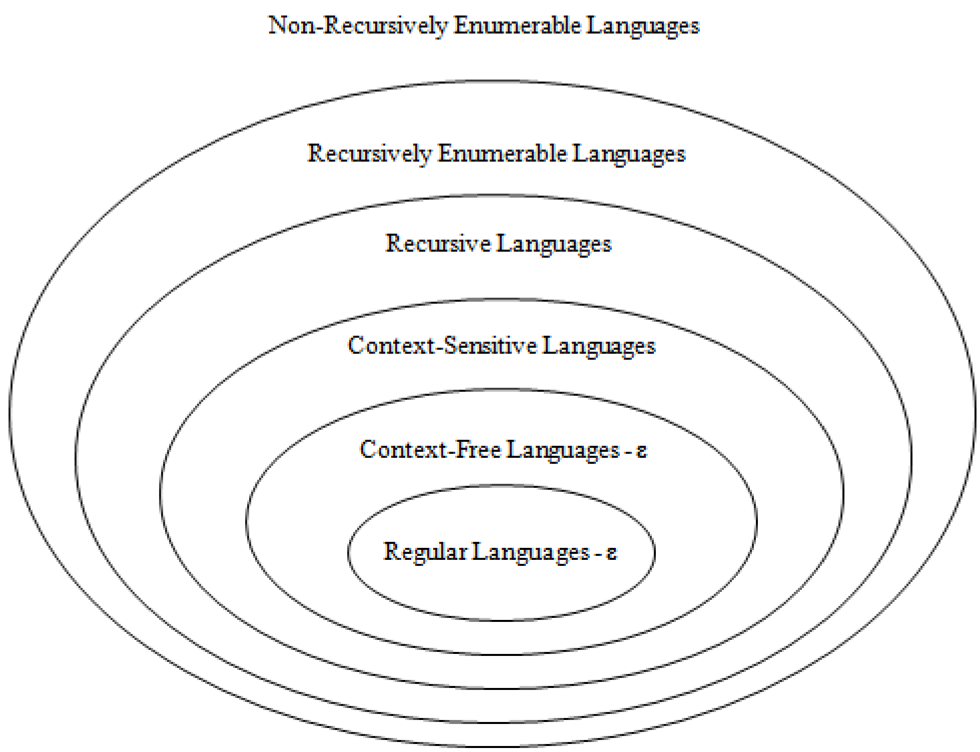
  - If L is recursive then L = L(M) for some TM M, and
    - •If x is in L then M halts in a final (accepting) state.
    - •If x is not in L then M halts a non-final (non-accepting) state.

Notes:

- The set of all recursive languages is a subset of the set of all recursively enumerable languages

- Terminology is easy to confuse: A *TM* is not recursive or recursively enumerable, rather a *language* is recursive or recursively enumerable.

- **Recall the Hierarchy:**



- **Observation:** Let L be an r.e. language. Then there is an infinite list M_0, M_1, … of TMs such that L = L(M_i).

- **Question:** Let L be a recursive language, and M_0, M_1, … a list of all TMs such that L = L(M_i), and choose any i>=0. Does M_i always halt?

  Answer: Maybe, maybe not, but *at least one in the list does*.

- **Question:** Let L be a recursive enumerable language, and M_0, M_1, … a list of all TMs such that L = L(M_i), and choose any i>=0. Does M_i always halt?

  Answer: Maybe, maybe not. Depending on L, none might halt or some may halt.

  - If L is also recursive then L is recursively enumerable.

- **Question:** Let L be a recursive enumerable language that is not recursive (L is in r.e. – r), and M_0, M_1, … a list of all TMs such that L = L(M_i), and choose any i>=0. Does M_i always halt?
  Answer: No! If it did, then L would not be in r.e. – r, it would be recursive.

- **Let M be a TM.**

  - Question: Is L(M) r.e.?
    Answer: Yes! By definition it is!

  - Question: Is L(M) recursive?
    Answer: Don't know, we don't have enough information.

  - Question: Is L(M) in r.e – r?
    Answer: Don't know, we don't have enough information.

- **Let M be a TM that halts on all inputs:**

  - Question: Is L(M) recursively enumerable?
    Answer: Yes! By definition it is!

  - Question: Is L(M) recursive?
    Answer: Yes! By definition it is!

  - Question: Is L(M) in r.e – r?
    Answer: No! It can't be. Since M always halts, L(M) is recursive.

- **Let M be a TM.**

  - As noted previously, L(M) is recursively enumerable, but may or may not be recursive.

  - Question: Suppose that L(M) is recursive. Does that mean that M always halts?
    Answer: Not necessarily. However, some TM M' must exist such that L(M') = L(M) and M' always halts.

  - Question: Suppose that L(M) is in r.e. – r. Does M always halt?
    Answer: No! If it did then L(M) would be recursive and therefore not in r.e. – r.

- **Let M be a TM, and suppose that M loops forever on some string x.**

  - Question: Is L(M) recursively enumerable?
    Answer: Yes! By definition it is.

  - Question: Is L(M) recursive?
    Answer: Don't know. Although M doesn't always halt, some other TM M' may exist

such that L(M’) = L(M) and M’ always halts.

- Question: Is L(M) in r.e. – r?
  Answer: Don’t know.

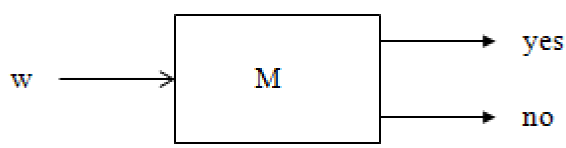
<u>Closure Properties for Recursive and Recursively Enumerable Languages</u>

- **TMs Model General Purpose Computers:**
  - If a TM can do it, so can a GP computer
  - If a GP computer can do it, then so can a TM

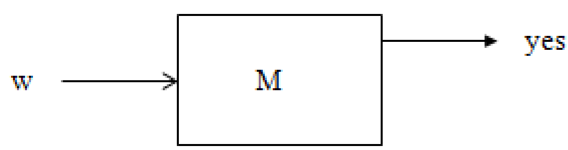
If you want to know if a TM can do X, then some equivalent question are:
- *Can a general purpose computer do X?*
- *Can a C/C++/Java/etc. program be written to do X?*

For example, is a language L recursive?
- *Can a C/C++/Java/etc. program be written that always halts and accepts L?*

- **TM Block Diagrams:**
  - If L is a recursive language, then a TM M that accepts L and always halts can be pictorially represented by a “chip” that has one input and two outputs.



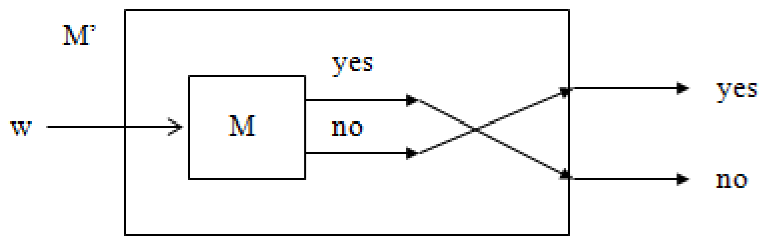
  - If L is a recursively enumerable language, then a TM M that accepts L can be pictorially represented by a “chip” that has one output.



  - Conceivably, M could be provided with an output for “no,” but this output cannot be counted on. Consequently, we simply ignore it.

– **Theorem:** The recursive languages are closed with respect to complementation, i.e., if L is a recursive language, then so is

Proof: Let M be a TM such that L = L(M) and M always halts. Construct TM M’ as follows:

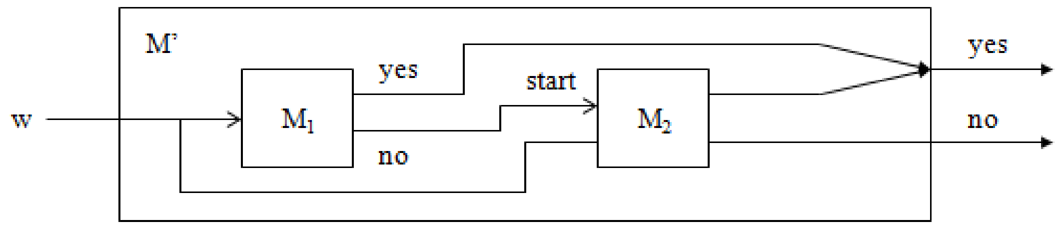


– **Note That:**
  – M' accepts iff M does not
  – M' always halts since M always halts

From this it follows that the complement of L is recursive. •

- **Theorem:** The recursive languages are closed with respect to union, i.e., if L1 and L2 are recursive languages, then so is

  Proof: Let M1 and M2 be TMs such that L1 = L(M1) and L2 = L(M2) and M1 and M2 always halts. Construct TM M' as follows:



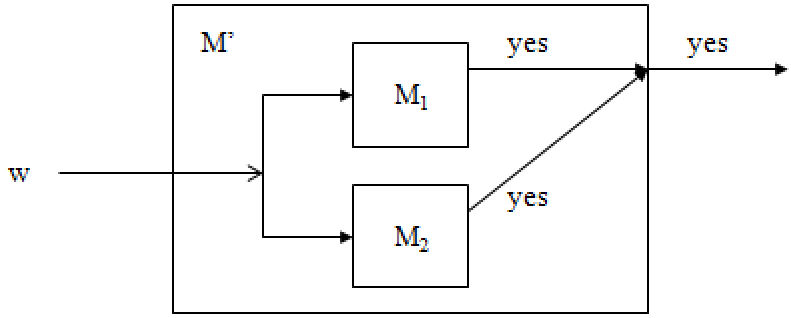
- **Note That:**
  - L(M') = L(M1) U L(M2)
    - •L(M') is a subset of L(M1) U L(M2)
    - •L(M1) U L(M2) is a subset of L(M')
  - M' always halts since M1 and M2 always

  halt It follows from this that L3 = L1 U L2 is

  recursive.

- **Theorem:** The recursive enumerable languages are closed with respect to union, i.e., if L1 and L2 are recursively enumerable languages, then so is L3 = L1 U L2

Proof: Let M_1 and M_2 be TMs such that $L_1 = L(M_1)$ and $L_2 = L(M_2)$. Construct M' as follows:



- **Note That:**
  – L(M') = L(M1) U L(M2)
    - •L(M') is a subset of L(M1) U L(M2)
    - •L(M1) U L(M2) is a subset of L(M')
  – M' halts and accepts iff M1 or M2 halts and accepts

It follows from this that   is recursively enumerable.

The Halting Problem – Background

- **Definition:** A decision problem is a problem having a yes/no answer (that one presumably wants to solve with a computer). Typically, there is a list of parameters on which the problem is based.
  – Given a list of numbers, is that list sorted?
  – Given a number x, is x even?
  – Given a C program, does that C program contain any syntax errors?
  – Given a TM (or C program), does that TM contain an infinite loop?

From a practical perspective, many decision problems do not seem all that interesting. However, from a theoretical perspective they are for the following two reasons:
  – Decision problems are more convenient/easier to work with when proving complexity results.
  – Non-decision counter-parts are typically at least as difficult to solve.

- **Notes:**
  – The following terms and phrases are analogous:

Algorithm	-	A halting TM program
Decision Problem	-	A language
(un)Decidable	-	(non)Recursive

Statement of the Halting Problem

- **Practical Form:** (P1)
  Input: Program P and input I.
  Question: Does P terminate on input I?

- **Theoretical Form:** (P2)
  nput: Turing machine M with input alphabet Σ and string w in Σ*.
  Question: Does M halt on w?

- **A Related Problem We Will Consider First:** (P3)
  Input: Turing machine M with input alphabet Σ and one final state, and string w in Σ*.
  Question: Is w in L(M)?

- **Analogy:**
  Input: DFA M with input alphabet Σ and string w in Σ*.
  Question: Is w in L(M)?

  Is this problem decidable? Yes!

- **Over-All Approach:**

  - We will show that a language L_d is not recursively enumerable
  - From this it will follow that is not recursive
  - Using this we will show that a language L_u is not recursive
  - From this it will follow that the halting problem is undecidable.

The Universal Language

- Define the language L_u as follows:

  L_u = {x | x is in {0, 1}* and x = <M,w> where M is a TM encoding and w is in L(M)}

- Let x be in {0, 1}*. Then either:

1. x doesn't have a TM prefix, in which case x is **not** in L_u

2. x has a TM prefix, i.e., x = <M,w> and either:

   a) w is not in L(M), in which case x is **not** in L_u

   b) w is in L(M), in which case x is in L_u

- **Compare P3 and Lu:**

  (P3):
  Input: Turing machine M with input alphabet Σ and one final state, and string w in Σ*.

- **Notes:**
  - Lu is P3 expressed as a language
  - Asking if Lu is recursive is the same as asking if P3 is decidable.
  - We will show that Lu is not recursive, and from this it will follow that P3 is un- decidable.
  - From this we can further show that the halting problem is un-decidable.
  - Note that Lu is recursive if M is a DFA.

Church-Turing Thesis

- There is an effective procedure for solving a problem if and only if there is a TM that halts for all inputs and solves the problem.

- There are many other computing models, but all are equivalent to or subsumed by TMs. *There is no more powerful machine* (Technically cannot be proved).

- DFAs and PDAs do not model all effective procedures or computable functions, but only a subset.

- If something can be “computed” it can be computed by a Turing machine.

- Note that this is called a ***Thesis***, not a theorem.

- It can’t be proved, because the term “can be computed” is too vague.

- But it is universally accepted as a true statement.

- Given the ***Church-Turing Thesis***:

  - What does this say about "computability"?

  - Are there things even a Turing machine can't do?

  - If there are, then there are things that simply can't be "computed."

    - Not with a Turing machine

        - Not with your laptop
        - Not with a supercomputer
    - There ARE things that a Turing machine can't do!!!
- The ***Church-Turing Thesis:***
    - In other words, there is no problem for which we can describe an algorithm that can’t be done by a Turing machine.

<u>The Universal Turing machine</u>

- If Tm’s are so damned powerful, can’t we build one that simulates the behavior of any Tm on any tape that it is given?
- Yes. This machine is called the ***Universal Turing machine***.
- How would we build a Universal Turing machine?
    - We place an encoding of any Turing machine on the input tape of the Universal Tm.
    - The tape consists entirely of zeros and ones (and, of course, blanks)
    - Any Tm is represented by zeros and ones, using unary notation for elements and zeros as separators.
- Every Tm instruction consists of four parts, each a represented as a series of **1**'s and separated by **0**'s.
- Instructions are separated by **00**.
- We use unary notation to represent components of an instruction, with
    - 0 = **1**,
    - 1 = **11**,
    - 2 = **111**,

- 3 = **1111**,
- ***n*** = **111...111** (***n***+1 1's).

- We encode q_n as <u>$n + 1$</u> 1's
- We encode symbol a_n as <u>$n + 1$</u> 1's
- We encode move left as 1, and move right as 11

1111011101111101110100101101101101100

q3, a2, q4, a2, L q0, a1, q1, a1, R

- Any Turing machine can be encoded as a unique long string of zeros and ones, beginning with a **1**.
- Let T_n be the Turing machine whose encoding is the number ***n***.

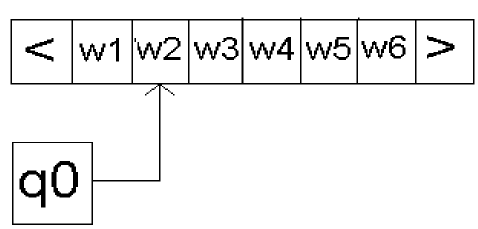
<u>Linear Bounded Automata</u>

- A Turing machine that has the length of its tape limited to the length of the input string is called a linear-bounded automaton (LBA).
- A linear bounded automaton is a 7-tuple *nondeterministic* Turing machine M = (Q, S, G, d, q_0,q_{accept}, q_{reject}) except that:

1. There are two extra tape symbols < and >, which are not elements of G.

2. The TM begins in the configuration (q_0<u><</u>*x*>), with its tape head scanning the symbol < in cell 0. The > symbol is in the cell immediately to the right of the input string *x*.

3. The TM cannot replace < or > with anything else, nor move the tape head left of < or right of >.



Context-Sensitivity

- *Context-sensitive production* any production □ □ □ satisfying |□□ | □|.
- *Context-sensitive grammar* any generative grammar *G* = □□ □, □, □ , □ 〉 such that every production in □ context-sensitive.
- No empty productions.

Context-Sensitive Language

- Language *L context-sensitive* if there exists context-sensitive grammar *G* such that either
  L = *L*(*G*) or *L* = *L*(*G*) □ {□}.

- **Example**:
  The language L = $\{a^nb^nc^n : n$ □ $1\}$ is a C.S.L. the grammar

  is S □ abc/ aAbc,

  Ab □ bA,

  AC □ Bbcc,

  bB □ Bb,

  aB □ aa/ aaA

  The derivation tree of $a^3b^3c^3$ is looking to be as following

  S ⇒aAbc

  ⇒abAc

  ⇒abBbcc

  ⇒aBbbcc ⇒aaAbbcc

  ⇒aabAbcc

  ⇒aabbAcc ⇒aabbBbccc

  ⇒aabBbbccc

  ⇒aaBbbbccc

$\Rightarrow$ aaabbbccc

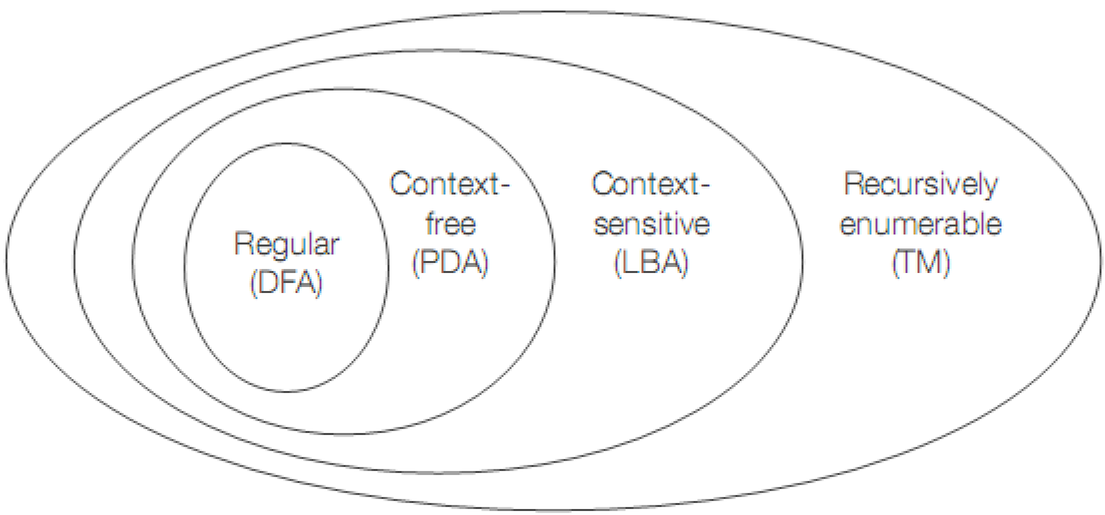
CSG = LBA

- A language is accepted by an LBA iff it is generated by a CSG.
- Just like equivalence between CFG and PDA
- Given an x □ CSG G, you can intuitively see that and LBA can start with S, and nondeterministically choose all derivations from S and see if they are equal to the input string x. Because CSL's are non-contracting, the LBA only needs to generate derivations of length □ |x|. This is because if it generates a derivation longer than |x|, it will never be able to shrink to the size of |x|.

UNIT V

Chomsky Hierarchy of Languages

- A containment hierarchy (strictly nested sets) of classes of formal grammars



The Hierarchy

Class	Grammars	Languages	Automaton
Type-0	Unrestricted	Recursively enumerable (Turing-recognizable)	Turing machine
	none	Recursive	Decider

		(Turing-decidable)	
Type-1	Context-sensitive	Context-sensitive	Linear-bounded
Type-2	Context-free	Context-free	Pushdown
Type-3	Regular	Regular	Finite

<u>Type 0 Unrestricted</u>:

Languages defined by Type-0 grammars are accepted by Turing machines .

Rules are of the form: $\alpha \rightarrow \beta$, where α and β are arbitrary strings over a vocabulary V and $\alpha \neq \varepsilon$

<u>Type 1 Context-sensitive</u>:

Languages defined by Type-1 grammars are accepted by linear-bounded automata.

Syntax of some natural languages (Germanic)

Rules are of the form:

$\alpha A\beta \rightarrow \alpha B\beta$

$S \rightarrow \varepsilon$

where

$A, S \in N$

$\alpha, \beta, B \in (N \cup \Sigma)*$

$B \neq \varepsilon$

<u>Type 2 Context-free</u>:

Languages defined by Type-2 grammars are accepted by push-down automata.

Natural language is almost entirely definable by type-2 tree structures

Rules are of the form:

$A \rightarrow \alpha$

Where

$A \in N$

$\alpha \in (N \cup \Sigma)*$

Type 3 Regular:

Languages defined by Type-3 grammars are accepted by finite state automata

Most syntax of some informal spoken dialog

Rules are of the form:

$A \rightarrow \varepsilon$

$A \rightarrow \alpha$

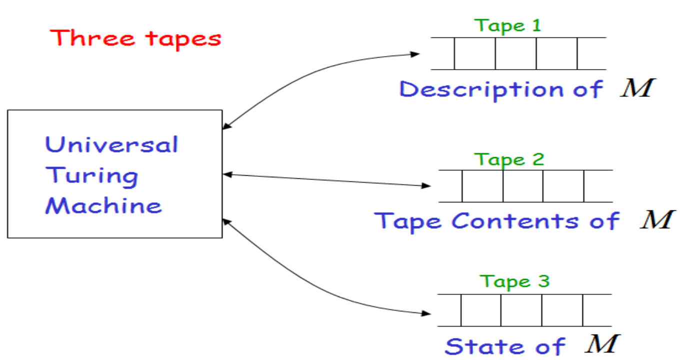
$A \rightarrow \alpha B$

where

$A, B \in N$ and $\alpha \in \Sigma$

The Universal Turing Machine

- If Tm's are so damned powerful, can't we build one that simulates the behavior of any Tm on any tape that it is given?



- Yes. This machine is called the ***Universal Turing machine***.
- How would we build a Universal Turing machine?
    - We place an encoding of any Turing machine on the input tape of the Universal Tm.

- The tape consists entirely of zeros and ones (and, of course, blanks)

- Any Tm is represented by zeros and ones, using unary notation for elements and zeros as separators.

- Every Tm instruction consists of four parts, each a represented as a series of **1**'s and separated by **0**'s.
- Instructions are separated by **00**.
- We use unary notation to represent components of an instruction, with
  - 0 = **1**,
  - 1 = **11**,
  - 2 = **111**,
  - 3 = **1111**,
  - ***n*** = **111...111** (***n***+1 1's).
- We encode q_n as $\underline{n + 1}$ 1's
- We encode symbol a_n as $\underline{n + 1}$ 1's
- We encode move left as 1, and move right as 11

1111011101111101110100101101101101100

q_3, a_2, q_4, a_2, L $\quad\quad\quad$ q_0, a_1, q_1, a_1, R

- Any Turing machine can be encoded as a unique long string of zeros and ones, beginning with a **1**.
- Let T_n be the Turing machine whose encoding is the number ***n***.

Turing Reducibility

- A language A is Turing reducible to a language B, written A $\leq_T$ B, if A is decidable relative to B
- Below it is shown that E_{TM} is Turing reducible to EQ_{TM}
- Whenever A is mapping reducible to B, then A is Turing reducible to B

  - The function in the mapping reducibility could be replaced by an oracle

- An oracle Turing machine with an oracle for EQ_{TM} can decide E_{TM}

$T^{EQ\text{-}TM}$ = "On input <M>

1. Create TM M_1 such that $L(M_1)$ = □

   M_1 has a transition from start state to reject state for every element of □

1. Call the EQ_{TM} oracle on input <M,M_2>

2. If it accepts, accept; if it rejects, reject"

- $T^{EQ\text{-}TM}$ decides E_{TM}
- E_{TM} is decidable relative to EQ_{TM}
- **Applications**
    - If A □$_T$ B and B is decidable, then A is decidable
    - If A □$_T$ B and A is undecidable, then B is undecidable
    - If A □$_T$ B and B is Turing-recognizable, then A is Turing-recognizable
    - If A □$_T$ B and A is non-Turing-recognizable, then B is non-Turing-recognizable

The class P

A decision problem *D* is *solvable in polynomial time* or *in the class P,* if there exists an algorithm *A* such that

- *A Takes instances* of *D* as inputs.
- *A* always outputs the correct answer "Yes" or "No".
- There exists a polynomial *p* such that the execution of *A* on inputs of size *n* always terminates in *p(n)* or fewer steps.
- **EXAMPLE**: The Minimum Spanning Tree Problem is in the class P.

  The class *P* is often considered as synonymous with the class of computationally feasible problems, although in practice this is somewhat unrealistic.

The class NP

A decision problem is *nondeterministically polynomial-time solvable* or *in the class NP* if there exists an algorithm *A* such that

- *A* takes as inputs potential witnesses for "yes" answers to problem *D.*
- *A* correctly distinguishes true witnesses from false witnesses.

- There exists a polynomial *p* such that for each potential witnesses of each instance of size *n* of *D,* the execution of the algorithm *A* takes at most *p(n)* steps.
- Think of a non-deterministic computer as a computer that magically "guesses" a solution, then has to verify that it is correct
    - If a solution exists, computer always guesses it
    - One way to imagine it: a parallel computer that can freely spawn an infinite number of processes
        - Have one processor work on each possible solution
        - All processors attempt to verify that their solution works
        - If a processor finds it has a working solution
    - So: **NP** = problems *verifiable* in polynomial time
    - Unknown whether **P = NP** (most suspect not)

NP-Complete Problems

- We will see that NP-Complete problems are the "hardest" problems in NP:
    - If any *one* NP-Complete problem can be solved in polynomial time.
    - Then *every* NP-Complete problem can be solved in polynomial time.
    - And in fact *every* problem in **NP** can be solved in polynomial time (which would show **P = NP**)
    - Thus: solve hamiltonian-cycle in O(n^{100}) time, you've proved that **P = NP**. Retire rich & famous.
- The crux of NP-Completeness is *reducibility*
    - Informally, a problem P can be reduced to another problem Q if *any* instance of P can be "easily rephrased" as an instance of Q, the solution to which provides a solution to the instance of P
        - *What do you suppose "easily" means?*
        - This rephrasing is called *transformation*
    - Intuitively: If P reduces to Q, P is "no harder to solve" than Q
- An example:
    - P: Given a set of Booleans, is at least one TRUE?
    - Q: Given a set of integers, is their sum positive?

- Transformation: ($x_1, x_2, \ldots, x_n$) = ($y_1, y_2, \ldots, y_n$) where y_i = 1 if x_i = TRUE, y_i = 0 if x_i = FALSE

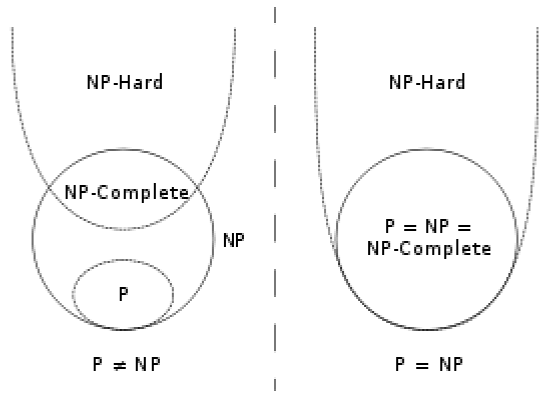
- Another example:
    - Solving linear equations is reducible to solving quadratic equations
        - *How can we easily use a quadratic-equation solver to solve linear equations?*

- Given one NP-Complete problem, we can prove many interesting problems NP-Complete
    - Graph coloring (= register allocation)
    - Hamiltonian cycle
    - Hamiltonian path
    - Knapsack problem
    - Traveling salesman
    - Job scheduling with penalties, etc.

NP Hard

- **Definition:** Optimization problems whose decision versions are NP- complete are called *NP-hard.*

- **Theorem:** If there exists a polynomial-time algorithm for finding the optimum in any *NP*-hard problem, then *P = NP*.
  Proof: Let *E* be an *NP*-hard optimization (let us say minimization) problem, and let *A* be a polynomial-time algorithm for solving it. Now an instance *J* of the corresponding decision problem *D* is of the form *(I, c),* where *I* is an instance of *E,* and *c* is a number. Then the answer to *D* for instance *J* can be obtained by running *A* on *I* and checking whether the cost of the optimal solution exceeds *c*. Thus there exists a polynomial-time algorithm for *D*, and *NP*-completeness of the latter implies *P= NP*.


NP-Hard
NP-Complete
NP
P
P ≠ NP
NP-Hard
P = NP =
NP-Complete
P = NP

Additional Topics

- Two Way Finite Automata
- Proof of Closure properties of Regular Languages
- Two Stack Pushdown Automata
- CYK Algorithm for CFL
- Cooks's Theorem

University Question Papers

Code No: R09220504 **R09** **Set No. 1**

II B.Tech II Semester Examinations,APRIL 2011
FORMAL LANGUAGES AND AUTOMATA THEORY
Computer Science And Engineering

Time: 3 hours Max Marks: 75

Answer any FIVE Questions
All Questions carry equal marks

⋆⋆⋆⋆⋆

1. Describe the following sets by regular expressions
   (a) {101}
   (b) {abba}
   (c) {01,10}
   (d) {a, ab} [15]

2. (a) Draw the transition diagram for a NFA which accepts all strings with either two consecutive 0's or two consecutive 1's.
   (b) differentiate NFA and DFA.
   (c) Construct DFA accepting the set of all strings with atmost one pair of consecutive 0's and atmost one pair of consecutive 1's. [6+4+5]

3. State and explain about closure properties of Context Free Languages. [15]

4. Obtain Chomsky Normal form for following Context Free Grammar
   $S \rightarrow \sim S \mid [S \supset S] \mid p \mid q$. [15]

5. (a) Construct a NFA accepting {ab, ba} and use it to find a deterministic automaton accepting the same set.
   (b) M = ({q1, q2, q3}, {0, 1}, δ, q1, {q3}) is a NFA where δ is given by
   δ (q1, 0) = {q2, q3}, δ (q1, 1) = {q1}
   δ (q2, 0) = {q1, q2}, δ (q2, 1) = $\emptyset$
   δ (q3, 0) = {q2}, δ (q3, 1) = {q1, q2}
   construct an equivalent DFA. [7+8]

6. (a) Design Turing Machine over {0,1}, L = {w | |w| is a multiple of 3}.
   (b) Draw the transition diagram for above language. [11+4]

7. (a) Find the language generated by the grammar. $S \rightarrow 0A \mid 1S \mid 0 \mid 1$, $A \rightarrow 1A \mid 1S \mid 1$
   (b) Construct context-free grammars to generate the set $\{a^l b^m c^n \mid$ one of l, m, n equals 1 and the remaining two are equal$\}$. [7+8]

8. Construct LR(0) items for the grammar given find it's equivalent DFA.
   S' → S
   S → AS | a
   A → aA | b [15]

Code No: R09220504 **R09** **Set No. 2**

II B.Tech II Semester Examinations,APRIL 2011
FORMAL LANGUAGES AND AUTOMATA THEORY
Computer Science And Engineering

Time: 3 hours Max Marks: 75

Answer any FIVE Questions
All Questions carry equal marks

⋆⋆⋆⋆⋆

1. (a) Construct DFA and NFA accepting the set of all strings not containing 101 as a substring.
   (b) Draw the transition diagram of a FA which accepts all strings of 1's and 0's in which both the number of 0's and 1's are even.
   (c) Define NFA with an example. [6+5+4]

2. Discuss about
   (a) Context Free Grammar
   (b) Left most derivation
   (c) Right most derivation
   (d) Derivation tree. [15]

3. (a) If G=({S}, {0, 1}, {S→0S1, S→ ε}, S), find L(G).
   (b) If G=({S}, {a}, {S→SS}, S) find the language generated by G. [7+8]

4. (a) What is unrestricted grammar? Give an Example.
   (b) Explain the language generated by unrestricted grammar.
   (c) Write about the machine corresponding to unrestricted grammar. [5+5+5]

5. (a) Construct a DFA with reduced states equivalent to the regular expression 10 + (0 + 11)0* 1.
   (b) Prove (a + b)* = a*(ba*)* [7+8]

6. (a) Construct a Mealy machine which can output EVEN, ODD according as the total number of 1's encountered is even or odd. The input symbols are 0 and 1.
   (b) Construct Moore machine equivalent to Mealy machine described in (a).[8+7]

7. (a) Convert the following Push Down Automata to Context Free Grammar
   M= ({q0,q1},{a,b}{z0,za},δ,q0,z0,φ)
   δ is given by
   δ (q0,a,z0)=(q0,za z0)
   δ (q0,a,za) =(q0,za za)
   δ (q0,b,za) =(q1,ε)
   δ (q1,b,za) =(q1,ε)
   δ (q1, ε,z0) =(q1,ε)

(b) Write the corresponding language for above Push Down Automata. [13+2]

8. Design Turing Machine to increment the value of any binary number by one. The out put should also be a binary number with value one more the number given. [15]

⋆⋆⋆⋆⋆

Code No: R09220504 **R09** **Set No. 3**

II B.Tech II Semester Examinations,APRIL 2011
FORMAL LANGUAGES AND AUTOMATA THEORY
Computer Science And Engineering

Time: 3 hours Max Marks: 75

Answer any FIVE Questions
All Questions carry equal marks
⋆⋆⋆⋆⋆

1. Find regular expressions representing the following sets
   (a) the set of all stings over {0, 1} having at most one pair of 0's or atmost of one pair of 1's
   (b) the set of all strings over {a, b} in which the number of occurrences of a is devisible by 3
   (c) the set of all strings over {a, b} in which there are at least two occurrences of b between any two occurrences of a.
   (d) the set of all strings over {a, b} with three consecutive b's.

   [15]

2. (a) What is generating variable? Give example.
   (b) Reduce the following Context Free Grammar
   S→ aAa
   A→ sb / bCC / DaA
   C→abb / DD
   E→aC
   D→aDA [4+11]

3. Construct
   (a) A context-free but not regular grammar.
   (b) A regular grammar to generate {a^n | n>=1}. [15]

4. (a) Construct a transition system which can accept strings over the alphabet a, b, ... containing either cat or rat.
   (b) Show that there exist no finite automaton accepting all palindromes over $\{a, b\}$. [7+8]

5. Design Push Down Automata for the language L={wcw^R | w ε (0+1)*}. [15]

6. Consider the grammar given below
   S → Aa
   A → AB | ε
   B → aB | b
   (a) Find the CLOSURE (S'→ .S)
   (b) GOTO({ A → .AB], [B → . aB] }, A) [7+8]

7. (a) Draw the transition diagram and transition table of FA which accept the set of all strings over the alphabet $\{0, 1\}$ with equal number of 0's and 1's such that each prefix has atmost one more 0 than 1's and atmost one more 1 than 0's.

   (b) Draw transition diagram and transition table of NFA which accepts the set of all strings over an alphabet $\{0, 1\}$, beginning with a '1' which, interpreted as the binary representation of an integer is congruent to 0 modulo 5. And construct an equivalent DFA. [6+9]

8. Design Turing Machine to find 2's complement of a given binary number. [15]

⋆⋆⋆⋆⋆

II B.Tech II Semester Examinations.APRIL 2011
FORhIAL LANGUAGES AND AUTO hIATA THEORY
Computer Science And Engineering

Time: 3 hours Atax Marks: 75

Answer any FIVE Questions All Questions carry equal marks

1. (a) Define NF.A with s moves.
   (b) differentiat.e âloore and Healy machines.
   (c) \¥'rite the st.eps in minimization of FA. [J+5+6]
2. (a) \Srite and explain t.he properties of transition funct.ion.
   (b) Prove t.hot for any transition function h and for anv tw•o input strings x and
   › (q ›) — ((q) ›)
   (c) Define Finit.e .but.omata and Transition diagram. 6+5+4]
3. Describe, in the English language, the sets represented by the following regular expressions:
   (a) a(a+b)*ab
   (b) a*b + b*a [15]
4. (a) \¥'1iat is type I grammar? Give an Example.
   (b) Explain the language generated by t.ype1 grammar.
   (c) \¥'rite about the machine corresponding t.o typel grammar. [5+5+5]
5. Design Turing machine for L } a" b" c" n 1 }. [15]
6. (a) Let C be the grammar. So aS | asbs | c. Prove that L(G) x | such t.hat each prefix of x has atleast as many a's as b's}
   (b) Show that abc, bra. cab } can be generated b\ a regular grammar whose terminal set is (a. b, c}
   8+7]
7. (a) Show that the grammar is ambiguous
   5 a | 5a | bss | S5b SbS.
   (b) Find Context. Free Grammar for L }a' W c' | j=i or j=k}.
   [7+8]
8. \S'liicli of the follow'ing are CFL's? explain
   (a) $\{a^i\ b^j \mid i \neq j \text{ and } i \neq 2j\}$
   (b) $\{a^i\ b^j \mid i \geq 1 \text{ and } j \geq 1\}$
   (c) $\{(a+b)^* - \{a^n\ b^n \mid n \geq 1\}\}$
   (d) $\{a^n\ b^n\ c^m \quad n \leq m \leq 2n\}$. [15]

Code No: R09220504

R09

JAWAHARLAL NEHRU TECHNOLOGICAL UNIVERSITY, HYDERABAD

B. Tech II Year - II Semester Examinations, November/December, 2012

FORMAL LANGUAGES AND AUTOMATA THEORY

(COMPUTER SCIENCE AND ENGINEERINR)

Time: 3 hours **Max. Marks: 75**

Answer any five questions

All questions carry equal marks

1.a) Design FA to accept string with 'a' and 'b' such that the number of a's are divisible by 3.

b) Design a FA accepting a binary string ending with last two characters are same over $\Sigma=\{0,1\}$. [15]

2.a) Convert the Moore machine to determine residue mod 3 into Mealy machine.

b) Write the steps in minimization of FA. [15]

3.a) Give regular expression for representing the set L of strings in which every 0 is immediately followed by at least two 1's.

b) Is $L=\{a^{2n} \mid n \geq 1\}$ regular? [15]

4.a) Find CFG for the language $L=\{ a^i b^j c^k \mid i=j\}$

b) If $G=(\{S\},\{0,1\},\{S \to 0S1, S \to \varepsilon\},S)$, find L(G). [15]

5.a) Convert the following Context Free Grammar to Chomsky Normal Form

$S \to AaB \mid aaB$
$A \to \varepsilon$
$B \to bbA \mid \varepsilon$

b) Convert the following grammar to Greibach Normal Form [15]

$S \to ABA \mid AB \mid BA \mid AA \mid B$
$A \to aA \mid a$
$B \to bB \mid b$

6.a) Design Push Down Automata for $L = \{a^{2n}b^n \mid n \geq 1\}$

b) Convert the following Context Free Grammar to Push Down Automata [15]

$S \to aSbb \mid aab$

7.a) Construct Turing Machine to compute the function $\log_2{}^n$.

b) Design a Turing Machine to recognize the language $L=\{a^n b^n a^n \mid n \geq 1\}$. [15]

8. Write about the following:

a) Linear Bound Automata

b) Decidable and Undecidable problem

c) Modified PCP. [15]

****

Code No: R09220504 | R09 | SET-1

B.Tech II Year - II Semester Examinations, April-May, 2012
FORMAL LANGUAGES AND AUTOMATA THEORY
(Computer Science and Engineering)

Time: 3 hours **Max. Marks: 75**

Answer any five questions
All questions carry equal marks

- - -

1.a) What is Automata? Discuss why study automata.
b) Define DFA and Design the DFA for the following languages on $\Sigma = \{a, b\}$
i) The set of all strings that either begins or ends or both with substring 'ab'.
ii) The set of all strings that ends with substring 'abb'. [15]

2.a) Design an NFA that accepts the language $(aa^*(a+b)^*)$.
b) Consider the following NFA – ε

	ε	a	b	c
→p	Φ	{p}	{q}	{r}
q	{p}	{q}	{r}	Φ
r	{q}	{r}	Φ	{p}

i) Compute the ε-closure of each state.
ii) Give all the strings of length 3 or less accepted by the automation.
iii) Convert the automation to DFA. [15]

3.a) Prove that every language defined by a Regular expression is also defined by Finite automata.
b) State and prove pumping lemma for regular languages. Apply pumping lemma for following language and prove that it is not regular $L=\{a^n / n \text{ is prime}\}$.
c) If L_1 and L_2 are regular languages then prove that family of regular language is closed under L_1-L_2. [15]

4.a) Define CFG. Obtain CFG for the following languages
i) $L=\{ww^R \mid W \text{ is in } (a,b)^*,\ w^R \text{ is the reversal of } W\}$
ii) L=(W | W has a substring}
b) What is an ambiguous grammar? Show that the following grammar is ambiguous
$E \to E+E|E-E|E^*E|E/E|(E)|a$
where E is the start symbol. Find the unambiguous grammar. [15]

5.a) Define PDA and construct a PDA that accepts the following languages
L={W | W is in (a+b)* and number of a's equal to number of b's} write the instantaneous description for the string ' aababb'.
b) For the following grammar construct a PDA
$S \to aABB|aAA$
$A \to aBB|a$
$B \to bBB|A$
$C \to a$. [15]

6.a) State and prove pumping lemma for context free languages.
b) What are CNF and GNF for context free grammar? Give examples.
c) Using CFL pumping lemma show that the following language is not context free $L=\{a^ib^jc^k | i<j<k\}$. [15]

7.a) What is Turing Machine and Multi tape Turing Machine? Show that the language accepted by these machines are same.

b) Design Turing Machine for the language to accept the set of strings with equal number of 0's and 1's and also give the instantaneous description for the input '110100'. [15]

8. Write short notes on

a) Homomorphism

b) Recursive Languages

c) Post's correspondence problem. [15]

****

Code No: R22055 **R10** SET - 1

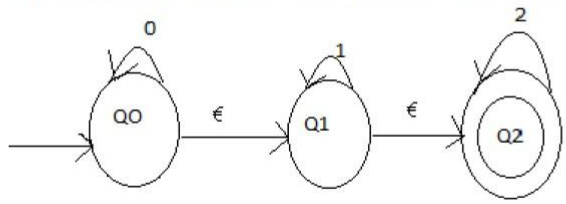
II B.Tech II Semester, Regular Examinations, April/May – 2012
FORMAL LANGUAGES AND AUTOMATA THEORY
(Computer Science and Engineering)

Time: 3 hours Max. Marks: 75

Answer any **FIVE** Questions
All Questions carry **Equal** Marks
~~~~~~~~~~~~~~~~~~~~~~~~

1. a) Explain NFA. Construct NFA for accepting the set of all strings with either two consecutive 0's or two consecutive 1's.
   b) What is a relation? Explain properties of a relation?
   c) What is a language? Explain different operations on languages? (7M+5M+3M)

2. a) State Myhill-Nerode theorem.
   b) Explain equivalence between two DFA's with an example.
   c) Find an equivalent NFA without €-transitions for NFA with €-transitions (3M+5M+7M)



3. a) Construct finite automaton to accept the regular expression (0+1)*(00+11)(0+1)*.
   b) Construct NFA with €-moves for regular expression (0+1)*.
   c) State and explain Arden's theorem. (7M+5M+3M)

4. Let G be the grammar S→0B|1A, A→0|0S|1AA, B→1|1S|0BB. For the string 00110101, find
   i) Leftmost derivation
   ii) Rightmost derivation
   iii) Derivation tree
   iv) Sentential form. (15M)

5. a) Discuss ambiguity, left recursion and factoring in context free grammars. Explain how to eliminate each one.
   b) Discuss closure and decision properties of context free languages. (8M+7M)

6. Explain equivalence of CFG and PDA. (15M)

7. a) Explain the properties of recursive enumerable languages.
   b) Explain counter machine in detail. (8M+7M)

8. Define P and NP problems. Also write notes on NP-complete and NP-hard problems. (15M)
~~~~~~~~~~~~~~~~~~~~~~~~

Code No: R22055 **R10** SET - 3

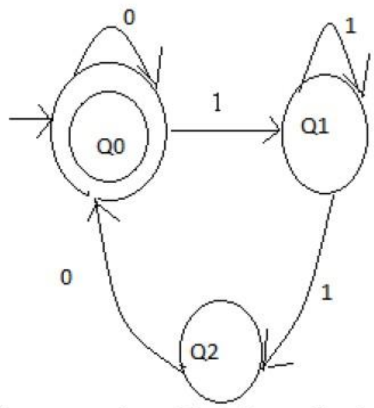
II B.Tech II Semester, Regular Examinations, April/May – 2012
FORMAL LANGUAGES AND AUTOMATA THEORY
(Computer Science and Engineering)

Time: 3 hours Max. Marks: 75

Answer any **FIVE** Questions
All Questions carry **Equal** Marks
~~~~~~~~~~~~~~~~~~~~~~~~

1. a) Explain DFA and NFA with an example.
   b) Define set, relation, graph and tree with examples. (8M+7M)

2. Define NFA mathematically. Explain its significance and function. Convert the given finite automaton into its DFG. Explain method used. Take suitable example and prove both accept the same string. (15M)



3. a) Define regular sets and regular expressions. Explain applications of regular expressions.
   b) Explain pumping lemma for regular sets. (8M+7M)

4. a) Define the following and give examples:
   i) Context Free Grammar ii) Derivation tree
   iii) Sentential form iv) Leftmost and rightmost derivation of strings.
   b) Obtain a right linear grammar for the language $L=\{a^n b^m / n>=2, m>=3\}$. (8M+7M)

5. a) Reduce the grammar S→ aAa, A→ SB|bcc|DaA, C→ abb|DD, E→ ac, D→ aDA.
   b) What is left recursion? How to eliminate it. (8M+7M)

6. a) Explain the terms: PDA and CFL.
   b) Explain equivalence of acceptance by final state and empty stack. (8M+7M)

7. a) Explain Church's hypothesis.
   b) Explain counter machine in detail. (8M+7M)

8. a) Explain different decision problems of DCFL and Turing machine halting problem.
   b) Explain universal Turing machine. (8M+7M)
~~~~~~~~~~~~~~~~~~~~~~~~

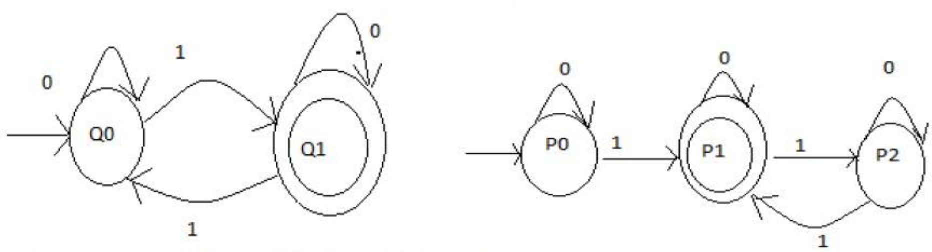
II B.Tech II Semester, Regular Examinations, April/May – 2012
FORMAL LANGUAGES AND AUTOMATA THEORY
(Computer Science and Engineering)

Time: 3 hours Max. Marks: 75

Answer any **FIVE** Questions
All Questions carry **Equal** Marks
~~~~~~~~~~~~~~~~~~~~~~

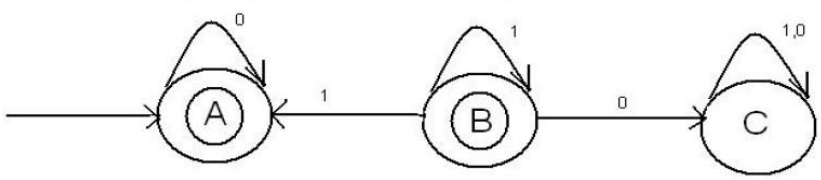
1. a) Explain principle of mathematical induction.
Prove that $1^2+2^2+3^2+\ldots\ldots+n^2=n(n+1)(2n+1)/6$ by using mathematical induction.
b) Explain DFA. Construct DFA accepting the set of all strings with an even no. of a's and even no. of b's over an alphabet {a,b}. (7M+8M)

2. a) Prove with the help of algorithm that "Every NFA will have an equivalent DFA".
b) Show that the following finite automata are equivalent: (8M+7M)



3. a) Explain equivalence of NFA and regular expression.
b) Design FA for regular expression 10+(0+11)0*1. (9M+6M)

4. a) Obtain a regular grammar for the following finite automata



b) What is the language of a grammar? Explain different types of grammars. (6M+9M)

5. What is GNF. Explain in detail. Convert the following grammar to GNF:
a) $A_1 \rightarrow A_1A_3$ b) $A_2 \rightarrow A_3A_1|b$ c) $A_3 \rightarrow A_1A_2|a$. (15M)

6. a) Explain acceptance of language by PDA.
b) Design a PDA that accepts the language L={w/w has equal no. of a's and b's} over an alphabet {a,b}. (7M+8M)

7. a) How a Turing machine accepts a language? Compare Turing machine and push down automata.
b) Define Turing machine. Explain the significance of movements of R/W head. (8M+7M)

8. a) Explain universal Turing machine.
b) Write about decidability of PCP.
c) Define P and NP problems. (6M+5M+4M)

1 of 1
~~~~~~~~~~~~~~~~~~~~~~

Code No: R22055 **R10** SET - 4

II B.Tech II Semester, Regular Examinations, April/May – 2012
FORMAL LANGUAGES AND AUTOMATA THEORY
(Computer Science and Engineering)

Time: 3 hours Max. Marks: 75

Answer any **FIVE** Questions
All Questions carry **Equal** Marks
~~~~~~~~~~~~~~~~~~~~~~

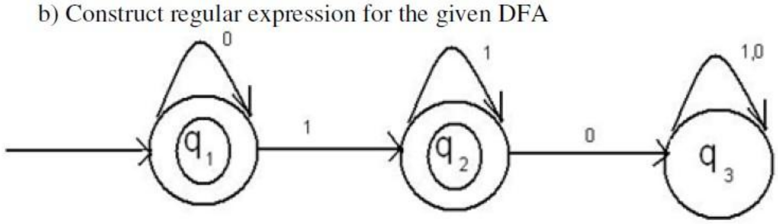
1. a) Design DFA for accepting set of all strings having
   i) odd no. of a's and odd no. of b's
   ii) even no. of a's and even no. of b's over an alphabet {a,b}.
   b) Define set, relation, graph and tree with examples. (8M+7M)

2. a) Construct a minimum state automaton equivalent to a given automaton M whose transition table is

State/input	0	1
→ q_0	q_1	q_3
q_1	q_2	q_4
q_2	q_1	q_4
q_3	q_2	q_4
q_4	q_4	q_4

   b) Discuss finite automata with outputs in detail. (9M+6M)

3. a) Draw NFA with ∈-moves recognizing regular expression 01*0+0(01+10)*11 over {0, 1}.
   b) Construct regular expression for the given DFA



(8M+7M)

4. a) Explain Chomsky classification of languages.
   b) Construct RLG and LLG for the regular expression (0+1)*00(0+1)*. (8M+7M)

5. a) Convert the following grammar to GNF:-
   i) $A_1 \rightarrow A_1A_3$ ii) $A_2 \rightarrow A_3A_1|b$ iii) $A_3 \rightarrow A_1A_2|a$.
   b) Explain the concept of ambiguity in context free grammars. How to eliminate it. (9M+6M)

6. a) Convert the following Context Free Grammar to Push down Automata
   i) $S \rightarrow aA|bB$ ii) $A \rightarrow aB|a$ iii) $B \rightarrow b$. Verify the string aab is accepted by equivalent PDA.
   b) Explain instantaneous description for PDA. (10M+5M)

7. a) Define Turing machine. Explain the significance of movements of R/W head.
   b) Design a Turing machine to recognize the language $L=\{a^nb^n/n>=1\}$. (6M+9M)

8. a) Write about LR(0) grammars.
   b) Explain halting problem of a Turing machine. (8M+7M)
~~~~~~~~~~~~~~~~~~~~~~

ode : R0 220 04 R09 SET-2

B.Tech II Year - II Semester Examinations, .Ypril-Islay, 2012

FORXIAL L.YNGt1AGES AND AUTOAIAT.4 THEORY

(Computer Science and Engineering)

Time: 3 hours **flax. Marks:** 7s

Answer and five questions

All questions carry equal marks

1.a) Defhie the following terms.

i) Alphabets ii) Strings

iii) Power of an alphabet iv) Language.

b) Define DFA. Design a DFA to accept the binary iiiunbers which are divisible by 5. [15]

2.a) Consider the transition table *ot* DFA given below:

	0	I
A	B	A
B	A	C
C	D	B
D	D	A
E	D	F
F	G	E
G	F	G
H	G	D

i) Draw the table *ot* distinguish abilities of tliis automaton.

ii) Constrict the niininnini state equivalent DFA.

b) Desipn an NFA that accepts the language (0+1)*1(0+1)*. [15]

S.a) Define a regular expression. Find the regular expression for the Language L= (a^2°b'* | n/0, m>0}.

b) State pumping lemma for regular lan rapes. R'ove that the following language {a'b° n>l} is not re ilar.

c) Convert the refill' expression (01+1)* to an NFA — e. [15]

4.a) Detme Context free gi auiuiar and write context fi ee griiiiiuira for the languages i) L={a'1›'c |i^j=k,i>0,j>0}

ii) L—(a°b"c' | n^2ni—k} .

fi) Consider the Grammar E— +EE | *EE|-EE|x|y.

Find the leftmost and riglitniost derivation for the string '+*-xyxy' mid write parse tree.

c) What is ambiguous grmiuiar? Prove that the following p'anuiiar is ambiguous on the string aab S—›aS|aSbS c. [15]

1.a) Define PDA. Discuss about the languages accepted by a PDA. Desipn a Non Deterministic PDA for the language L=(0° l°|n>1}.

h) ColTvert the following giainiiiar to a PDA that accepts the same language by erripty stack. S0S1|A

As lA0|S|c [15]

6.a) What are useless Symbols? Remove all useless Symbols and all ε – productions from the grammar

$$S \rightarrow aA|aB$$
$$A \rightarrow aaA|B|\varepsilon$$
$$B \rightarrow b|bB$$
$$D \rightarrow B$$

b) Define CNF. Convert the following CFG to CNF

$$S \rightarrow ASB|\varepsilon$$
$$A \rightarrow aAS|a$$
$$B \rightarrow SbS|A|bb.$$

[15]

7.a) With a neat diagram, explain the working of a basic Turing Machine. Design a Turing Machine to accept $L=\{1^n2^n3^n \mid n\geq1\}$

b) Explain the differences between PDA and T M. [15]

8. Write short notes on
a) Multi tape Turing Machine
b) Post's correspondence problem
c) Chomsky hierarchy. [15]

****

Code No: R09220504	R09	SET-3

B.Tech II Year - II Semester Examinations, April-May, 2012
FORMAL LANGUAGES AND AUTOMATA THEORY
(Computer Science and Engineering)

Time: 3 hours **Max. Marks: 75**

Answer any five questions
All questions carry equal marks
- - -

1.a) Define the following
i) Power of an alphabet ii) NFA
b) Design a DFA to accept the following language over the alphabet {0,1}
i) L={w / w is an even number}
ii) $L=\{(01)^i 1^{2j} / i\geq 1, j\geq 1\}$
iii) The set of strings either start with 01 or end with 01. [15]

2.a) Define distinguishable and indistinguishable states. Minimize the following DFA.

	0	1
→A	B	F
B	G	C
C	A	C
D	C	G
E	H	F
F	C	G
G	G	E
H	G	C

b) Explain in detail with an example the conversion of NDFA to DFA. [15]

3.a) Write the regular expressions for the following languages
i) The set of all strings over Σ={a,b,c} containing atleast one 'a' and atleast one 'b'
ii) The set of strings of 0's and 1's whose 10^{th} symbol from the right end is 1.
b) Convert the regular expression (0+1)*1(0+1)* to an NFA – ε.
c) State and prove the pumping lemma for regular languages. [15]

4.a) Define CFG. Write CFG for the language $L=\{0^n1^n|n\geq 1\}$ i.e. the set of all strings of one or more 0's followed by an equal number of 1's.
b) Consider the grammar S→aS/aSbS/ε
Is the above grammar ambiguous?
Show in particular that the string 'aab' has no:
i) Parse tree ii) Leftmost derivation iii) Rightmost derivation. [15]

5.a) Discuss the languages accepted by a PDA. Design a PDA for the language that accepts the strings with number of a's less than number of b's where w is in (a+b)* and show the instantaneous description of the PDA on input 'abbab'.
b) Convert the following grammar to a PDA that accepts the same language by empty stack
S→0S1|A
A→1A0|S|ε [15]

6.a) What are useless symbols? Eliminate Null, unit and useless production from the following grammar

$S \rightarrow AaA|CA|BaB$
$A \rightarrow aaBa|CDA|aa|DC$
$B \rightarrow bB|bAB|bb|aS$
$C \rightarrow Ca|bC|D$
$D \rightarrow bD|\varepsilon$

b. What is CNF and GNF? Obtain the following grammar in CNF

$S \rightarrow aBa|abba$
$A \rightarrow ab|AA$
$B \rightarrow aB|a$ [15]

7.a) Explain with neat diagram, the working of a Turing Machine model.

b) Design a Turing machine to accept all set of palindromes over {0,1}*. Also write its transition diagram all Instantaneous description on the string '10101'. [15]

8. Write short notes on the following

a) post's Correspondence problem
b) Recursive languages
c) Universal Turing Machine. [15]

****

Code No: R09220504 | R09 | SET-4

B.Tech II Year - II Semester Examinations, April-May, 2012
FORMAL LANGUAGES AND AUTOMATA THEORY
(Computer Science and Engineering)

Time: 3 hours **Max. Marks: 75**

Answer any five questions
All questions carry equal marks
\- - -

1.a) Define the following terms with an example for each
i) Transition Table ii) Transition Diagram
iii) Power set iv) Language.
b) Mention the differences between DFA, NFA and NFA – ε. [15]

2.a) Prove the equivalence of NFA and DFA.
b) Define Moore and Mealy machines with examples. [15]

3.a) Define a regular expression. Find regular expression for the following languages on {a,b}
i) Language of all strings w such that w contains exactly one 1 and even number of 0's.
ii) Set of strings over {0,1,2} containing atleast one 0 and atleast one 1.
b) Prove that if L is regular language over alphabet Σ then L is also regular language.
c) Prove that the language $L=\{0^n 1^{n+1} \mid n>0\}$ is not regular. [15]

4.a) Construct the CFG for the following languages
i) $L=\{a^{2n}b^m \mid n\geq 0, m\geq 0\}$
ii) $L=\{0^i 1^j 2^k \mid i=j \text{ or } j=k\}$ and generate leftmost derivation for the string 01122.
b) Define ambiguous Grammar. Prove that the following grammar is Ambiguous. Find an unambiguous grammar.
$S\rightarrow aS|aSbS|\varepsilon$ [15]

5.a) Define PDA and Design PDA to accepts the following languages by final state L={W | W is in (a+b)* and number of a's equal to number of b's}.
Draw the graphical representation of PDA. Also show the moves made by the PDA for the string 'abbaba'.
b) Convert the following CFG to PDA
$S\rightarrow aABB|aAA$
$A\rightarrow aBB|a$
$B\rightarrow bBB|A$
$C\rightarrow a$ [15]

6.a) Consider the grammar
$S\rightarrow ABC|BaB$
$A\rightarrow aA|BaC|aaa$
$B\rightarrow bbb|a|D$
$C\rightarrow CA|AC$
$d\rightarrow \varepsilon$
i) Eliminate NULL productions
ii) Eliminate Unit Productions in the resulting grammar
iii) Eliminate Useless Symbols in the resulting grammar.
b) What is CNF? Convert the following grammar into CNF
$S\rightarrow ABa$
$A\rightarrow aab$
$B\rightarrow Ac$ [15]

7.a) With a neat diagram, explain the working of a basic Turing Machine. Design a Turing Machine to accept $L=\{WW^R \mid W \text{ is in } (a+b)^*\}$

b) Explain the general structure of multi-tape and deterministic Turing Machines and show that these are equivalent to basic Turing machine. [15]

8. Write short notes on
a) Post Correspondence problem
b) Chomsky hierarchy
c) Homomorphism. [15]

****

Code No: R09220504

R09

JAWAHARLAL NEHRU TECHNOLOGICAL UNIVERSITY, HYDERABAD

B.Tech II Year II Semester Examinations, May-2013

Formal Languages and Automata Theory

(Computer Science and Engineering)

Time: 3 hours **Max. Marks: 75**

Answer any five questions

All questions carry equal marks

\- - -

1.a) Design NFA to accept strings with a's and b's such that the string end with 'bb'.

b) Design FA to accept string with 'a' and 'b' such that the number of b's are divisible by 3. [15]

2.a) Convert the following NFA with ε to equivalent DFA

	a	b	ε
→A	Φ	A	B
B	B	Φ	C
C	B	A	Φ

b) Construct the minimum state automata for the following. [15]

	a	b
→A	B	A
B	A	C
C	D	B
D	D	A
E	D	F
F	G	E
G	F	G
H	G	D

3.a) Find the regular expression for the Language $L = \{a^{2n}b^{2m} \mid n \geq 0, m \geq 0\}$.

b) Construct NFA for the R.E. that contains odd number of 0's over $\Sigma = \{0\}$.

c) Write a R.E. for the following DFA. [15]

	a	b
→P	Q	P
Q	Q	P

4.a) Write CFG for the language $L=\{a^nb^n \mid n \geq 1\}$ i.e. the set of all strings of one or more a's followed by an equal number of b's.

b) Construct right linear grammar for the following DFA. [15]

	0	1
→A	B	C
B	B	C
C	A	C

5.a) Discuss the languages accepted by a PDA. Design a PDA for the language that accepts the strings with number of a's less than number of b's where w is in $(a+b)^*$ and show the instantaneous description of the PDA on input 'abbab'.

b) Convert the given CFG into GNF. [15]

$S \to CA$

$A \to a$

$C \to aB \mid b$

6.a) Using CFL pumping lemma show that the following language is not context free $L=\{a^i b^j c^k \mid i<j<k\}$

b) Obtain the following grammar in CNF. [15]

$S \to aBa \mid abba$

$A \to ab \mid AA$

$B \to aB \mid a$

7.a) Construct TM for the function $f(x) = (x+3)$.

b) Design a Turing Machine to recognize the language $L = \{a^n b^n a^n \mid n \geq 1\}$. [15]

8.a) Is the language $a^n b^n c^n$ Context Sensitive? Explain.

b) What do you mean by 'decidable' and 'undecidable' problem? Give example.

c) Write short notes on Universal Turing Machine. [15]

---ooOoo---

Code No: R22055 **R10** SET - 1

II B. Tech II Semester, Regular Examinations, April/May – 2013
FORMAL LANGUAGES AND AUTOMATA THEORY
(Computer Science and Engineering)

Time: 3 hours Max. Marks: 75

Answer any **FIVE** Questions
All Questions carry **Equal** Marks
~~~~~~~~~~~~~~~~~~~~~~~~

1. a) Define Relation? Explain about different types of Relations?
   b) Construct a DFA that accepts an identifier of a 'C' programming language. (8M+7M)

2. Construct Minimum state Automata for the following DFA?
   * denotes final state

δ	0	1
→ q 1	q2	q3
q 2	q3	q5
*q 3	q4	q3
q 4	q3	q5
*q5	q2	q5

(15M)

3. a) Show that L={a^{2n}/n<0} is regular?
   b) Show that L={a^p /p is prime} is context free? (8M+7M)

4. a) Define Grammar? Explain about Chomsky classification of Grammars?
   b) Explain about Right linear and Left Linear Grammars? (8M+7M)

5. a) Explain about the decision properties of context free languages?
   b) Explain about Left Factoring and Left Recursion? (8M+7M)

6. a) Explain about PDA?
   b) Convert the grammar S→0AA,A →0S/1S/0 to a PDA that accepts the same language by empty Stack? (4M+11M)

7. a) Define Turing Machine? Explain about the Model of Turing Machine?
   b) Explain about types of Turing Machine? (8M+7M)

8. a) Explain about the Decidability and Undecidability Problems?
   b) Explain about Turing Reducibility? (8M+7M)
~~~~~~~~~~~~~~~~~~~~~~~~

Code No: R22055 **R10** SET - 2

II B. Tech II Semester, Regular Examinations, April/May – 2013
FORMAL LANGUAGES AND AUTOMATA THEORY
(Computer Science and Engineering)

Time: 3 hours Max. Marks: 75

Answer any **FIVE** Questions
All Questions carry **Equal** Marks
~~~~~~~~~~~~~~~~~~~~~~~~~

1. a) Define Finite Automaton? Explain about the model of Finite Automaton?
   b) Define Set? Explain about the Operations on Set? (8M+7M)

2. Explain in detail about Melay and Moore Machines? (15M)

3. Construct Finite Automata for the regular Expression 1(01+10)*00? (15M)

4. a) Define Derivation tree? Explain about LMD and RMD?
   b) Construct a derivation tree for the string abcd from the grammar $S \rightarrow aAB, A \rightarrow bC, B \rightarrow d, C \rightarrow cd$? (8M+7M)

5. a) List out the Applications of CFL?
   b) Construct CNF for the Grammar $S \rightarrow ABC$, $A \rightarrow 0B, B \rightarrow CD/0, C \rightarrow 1$ (8M+7M)

6. a) Show that for every PDA then there exists a CFG such that L(G)=N(P)?
   b) Construct a PDA for $L=\{a^n b^n c^n \ /n > 0\}$ (8M+7M)

7. Construct a Turing Machine that will accept the Language consists of all palindromes of 0's and 1's? (15M)

8. Explain in detail about NP Complete and NP hard problems? (15M)
~~~~~~~~~~~~~~~~~~~~~~~~~

Code No: R22055 **R10** SET - 3

II B. Tech II Semester, Regular Examinations, April/May – 2013
FORMAL LANGUAGES AND AUTOMATA THEORY
(Computer Science and Engineering)

Time: 3 hours Max. Marks: 75

Answer any **FIVE** Questions
All Questions carry **Equal** Marks
~~~~~~~~~~~~~~~~~~~~~~~~~

1. a) Define Relation? Explain about different types of Relations?
   b) Construct a DFA for the Regular Language consisting of any number of a's and b's? (8M+7M)

2. Explain in detail about the Procedure for converting a given Melay to Moore Machine and vice versa? (15M)

3. a) Explain about the identity rules of Regular Expressions?
   b) Explain about the Closure Properties of Regular sets? (8M+7M)

4. a) Explain about LBA?
   b) Explain about Context free and Context Sensitive Grammars? (8M+7M)

5. Explain in detail about Chomsky and Greibach Normal forms? (15M)

6. Construct a PDA for $L=\{wcw^R / w\in (0+1)^*\}$ (15M)

7. a) Design a Turing Machine for $L=\{0^n1^m0^n1^m/m,n>=1\}$?
   b) Explain about Recursively Enumerable Languages? (8M+7M)

8. a) Explain in detail about Halting Problem of Turing machine?
   b) Explain about Universal Turing Machine? (8M+7M)
~~~~~~~~~~~~~~~~~~~~~~~~~

Code No: R22055 **R10** SET - 4

II B. Tech II Semester, Regular Examinations, April/May – 2013
FORMAL LANGUAGES AND AUTOMATA THEORY
(Computer Science and Engineering)

Time: 3 hours Max. Marks: 75

Answer any **FIVE** Questions
All Questions carry **Equal** Marks
~~~~~~~~~~~~~~~~~~~~~~~~~

1. Define the Following:
   i) String ii) Alphabet iii) Languages iv) Grammar v) NP problem (15M)

2. Construct Minimum state Automata for the following DFA? (15M)
   * denotes final state

δ	0	1
→ q 1	q2	q6
q 2	q1	q3
*q 3	q2	q4
q 4	q4	q2
q5	q4	q5
*q6	q5	q4

3. Define Regular Expression? Explain about the Properties of Regular Expressions? (15M)

4. Explain about the Procedure for Converting a Regular Expression in to Automata. (15M)

5. Define Ambiguous Grammar? Check whether the grammar S→aAB,A→bC/cd,C→cd,B→c/d is Ambiguous or not? (15M)

6. a) Explain about DPDA?
   b) Construct PDA for $L=\{a^nb^n/n>0\}$? (8M+7M)

7. a) Construct Turing machine for the languages containing the set of all strings of balanced parenthesis?
   b) Explain about the Design of Turing Machines? (8M+7M)

8. Define LR(0) Grammar? Explain in detail about PCP? (15M)
~~~~~~~~~~~~~~~~~~~~~~~~~

17. Question Bank:Descriptive Type Questions - Unit Wise

UNIT I

1. Explain the Finite automation how the language constructs can be recognized?
2. List out the Finite automata's?
3. Define: string, sub string, transitive closure and reflexive transitive closure?
4. Describe the finite state machine with a block diagram.
5. Construct DFA to accept the language of all strings of even numbers of a's & numbers of b's divisible by three over (a+b)*.

6. Explain the procedure to convert NFA to DFA.
7. What are the Finite automates with output and explain them with the suitable Examples.
8. Explain the procedure to minimize the DFA for the given regular expression.
9. a) Construct a Mealy machine similar to (well equivalent to except for Ms's initial output) the following Moore machine.

	0	1	
A	B	C	0
B	C	B	1
C	A	C	0

b) Construct a Moore machine similar to the following Mealy machine.

	0	1
A	B, 0	C, 1
B	C, 1	B, 1
C	A, 1	C, 0

10. Give Mealy and Moore machines for the following processes:

a) For input from $(0 + 1)^*$, if the input ends in 101, output A; if the input ends in 110, output B; otherwise output C.

b) For input from (0 + 1 + 2)*, print the residue modulo 5 of the input treated as a ternary (base 3, with digits 0, 1, and 2) number.

UNIT II

1. Define the Regular Expression.
2. Write the Identity Rules for RE
3. Construct the FA for the Regular Expression (a/b)*abb.
4. Obtain the minimized DFA for the RE (a/b)*abb.
5. Explain the Pumping Lemma for the regular sets.
6. What are the properties of regular sets?

7. Define the grammar and what are the types of grammars?

8. Consider the grammar E->E + E | E * E | id.
   Write the right-most derivation and left most derivation for the sentence id*id+id.
9. Explain right linear and left linear grammar, with a example?
10. Construct a regular grammar G generating the regular set represented by a*b (a+b)*.
11. If a regular grammar G is given by S □ aS/a, find regular expression for L(G).

UNIT III

1. What is an ambiguity?
2. What does an ambiguity trouble in the CFG?
3. What are the techniques used to minimize the CFG?
4. Explain the CNF and GNF with an example.

5. Explain Pumping Lemma for context free grammars?

6. Explain the concept of push down automata?

7. Write the push down automata to accept the language {ww* | w e {0, 1}}
8. Explain the equivalence of CFL and PDA.
9. Construct PDA equivalent to the following grammar: S □ aAA, A □aS/bS/a.

   Show that the set of all strings over {a, b} consisting of equal numbers of a's and b's accepted by a PDA.

UNIT IV

1. Solve the problem using the TM, [anbcn | where n is an odd]
2. Explain the steps required to design the TM.
3. Explain the Counter machines with suitable example.
4. Design a Turing Machine to accept the string that equal number of 0's and 1's.
5. Design a Turing Machine to recognize the language $\{1^n 2^n 3^n / n \geq 1\}$.

6. What is meant by linear bounded automata?

UNIT V

1. Explain the Chomsky hierarchy of languages
2. Explain the Universal TM?
3. Explain the P and NP problems?
4. Explain the Decidability of Problems. Give an example.
5. Explain Post Correspondence Problem.

18. ASSIGNMENT QUESTIONS

UNIT-I

1. a) Given L1={a,ab,a^2} and L2={b^2,aa} are the languages over

   A={a,b}. Determine i) L1L2 and ii) L2L1.

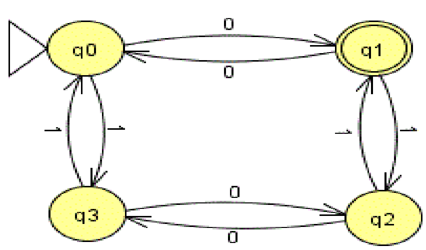
   b) Given A={a, b, c} find L^* where i)L={b^2} ii) L={a, b} and iii) L={a,b,c^3}.

   c) Let L= {ab, aa, baa} which of the following strings are in L^*

   i) abaabaaabaa and ii) aaaaabaaaab.

2. Determine which of the following strings are accepted by the given Finite Automata

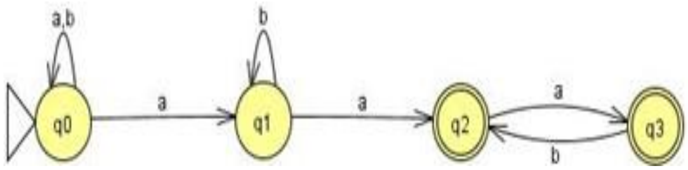
   i) 0011 ii) 0100 and iii) 0101011.



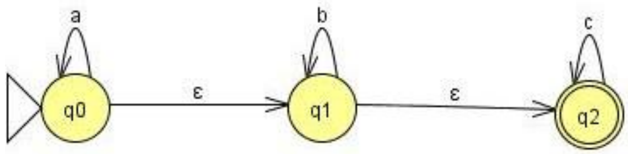
3. a) Define The following terms: i) DFA and ii)NFA.

   b) Design a DFA which accepts set of all strings containing odd number of 0's and odd number of 1's.

4. a) Convert the following NFA to DFA



b) Convert the following NFA with ε- transitions to without ε- transitions.



5. a) Construct the minimum state automata for the following : Initial State :A Final State: D

Q/∑	a	b
A	B	A
B	A	C
C	D	B
D	D	A
E	D	F
F	G	E
G	F	G
H	G	D

b) Design FA to accept strings with ‘a’ and ‘b’ such that the number of b’s are divisible by 3

6. a) Design DFA for the following languages shown below: ∑={a,b}

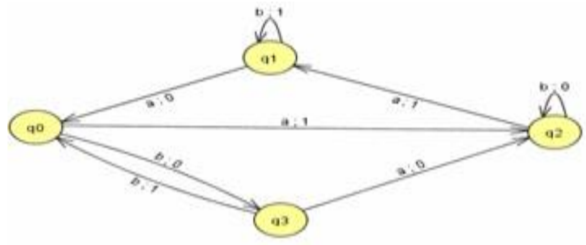
   i) L= {w| w does not contain the substring ab}.

   ii) L= {w| w contains neither the substring ab not ba}.

   iii) L= {w| w is any string that does not contain exactly two a’s}.

7. Design a Moore and Mealy machine to determine the residue mod 5 for each ternary string (base 3) treated as ternary integer.

8. Construct the Moore machine for the given Mealy machine



9. Construct the Mealy machine for the following Moore machine

Present State	Next State i/p=0	p=1	output
q0	q1	q2	1
q1	q3	q2	0
q2	q2	q1	1
q3	q0	q3	1

10. Design an NFA for the following

    i) L={ $abaa^n$ | n≥ 1}

    ii) To accept language of all strings with 2 a's followed by 2 b's over {a,b}.

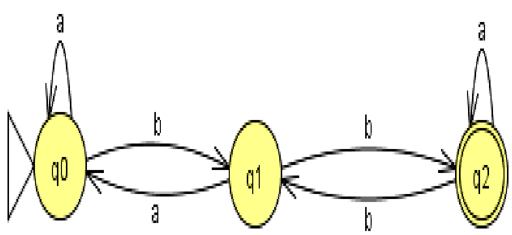
    iii) To accept strings with a's and b's such that the string end with bb.

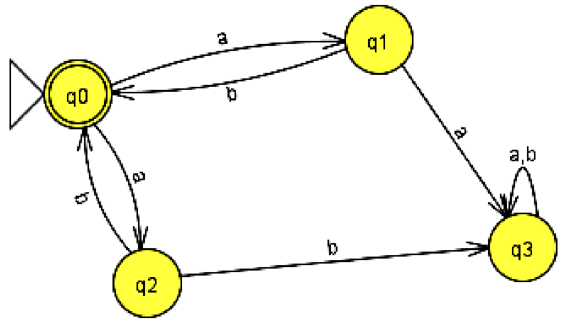
UNIT-II

1. a)Define Regular Expression.
   b) List the Identity Rules of Regular sets.
   c) Prove the following
   i) ε+1*(011)*(1* (011)*)* = (1+011)*
   ii) (1=00*1)+(1+00*1)(0+10*1)*(0+10*1) = 0*1(0+10*1)*
   iii) (rs+r)*r=r(sr+r)*

2. a) Explain equivalence of NFA and regular expression.
   (OR)
   Prove that every language defined by a regular expression is also defined by Finite Automata
   b) Construct DFA for (a+b)*abb.

3. Find the regular expression accepted by following DFA
   a) b)

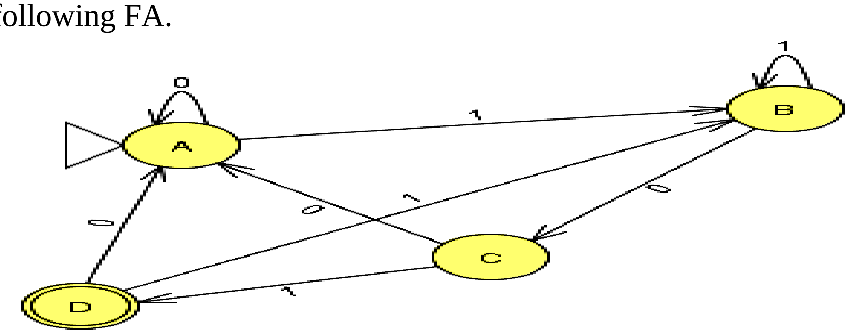




4. a) State and prove pumping lemma for regular languages. Apply pumping lemma for following
language and prove that it is not regular L={$a^m b^n$ | gcd(m,n) = 1}.
b) Show that L= {$a^{n!}$ |n>=1} is not regular.

5. a) Obtain a regular expression to accept strings of a's and b's such that every block of four consecutive symbols contains at least two a's.
b) Give regular expression for representing the set L of strings in which every 0 is immediately at least two 1's.
c) Find the regular expression for the language L={$a^{2n}b^{2m}$|n≥0, m≥0}.
d) Find the regular expression for L= {w | every odd position of w is a 1}

6. a) Define Regular Grammar. Explain in detail obtaining a right linear and left linear grammar for the
following FA.



b) Find the right linear grammar and left linear grammar for the regular expression (0+1)*010(1(0+1))*

7. a) Explain the process of obtaining a DFA from the given Regular Grammar.
b) Construct a DFA to accept the language generated by CFG:

i) S□ 01A,A□ 10B,B□ 0A|11. ii). S□ Aa,A□ Sb|Ab|ε.

8. a) Define Context Free Grammar.

b) i)What is CFL generated by the grammar S □ abB, A □ aaBb,B □ bbAa,A □ ε.
ii) State in English about the language corresponding to below given grammar

S□ aB|bA, A□ a|aS|bAA, B□ b|bS|aBB.

iii) Describe the language generated by the grammar S□ aAB, A□ bBb, B□ A|ε.

c) i) Given the grammar G as S□ 0B|1A, A□ 0|0S|1AA, B□ 1|1S|0BB. Find leftmost and rightmost

derivation and derivation tree for the string 00110101.

ii) Construct the leftmost, rightmost derivation and parse tree for the following grammar which

accepts the string aaabbabbba S□ aB|bA, A□ aS|bAA|a, B□ bS|aBB|b.

9. Write the Context Free Grammar for the following languages

i) L= $\{a^n b^n | n \geq 1\}$

ii) L= $\{a^i b^j c^k | i=j\}$

iii) Language of strings with unequal number of a's and b's.

iv) L= $\{a^i b^j c^k | \; i+j=k, i \geq 0, \; j \geq 0\}$

v) L= $\{ww^R |$ w is in (a,b)* and w^R is the reversal of w$\}$

10. a) Write and explain all properties of regular sets.

b) State and prove Arden's theorem.

UNIT-III

1. a) Discuss Ambiguity, left recursion and factoring in context free grammar.

b) Check whether the following grammars are ambiguous or not?

i) S□ aAB, A□ bC|cd, C□ cd, B□ c|d.

ii) E□ E+E|E-E|E*E|E/E|(E)|a.

iii) S□aS|aSbS|ε.

c) Explain the process of eliminating ambiguity.

2. a) Explain minimization or simplification of context free grammars.

b) i) Eliminate Null productions in the grammar S□ ABaC,A□ BC,B□ b|ε,C□ D|ε,D□ d.

ii) Eliminate Unit productions in the grammar S→ AB, A→ a, B→ C,B→ b,C→ D,D→ E,E→ a.

iii) Find a reduced grammar equivalent to the grammar G whose productions are
S→ AB|CA, B→ BC|AB, A→ a,C→ aB|b.

c) Simplify the following grammar: S→ AaB|aaB, A→ D, B→ bbA|ε, D→ E, E→F, F→ aS.

3. a) Explain Chomsky Normal Form.

b) i) Find a grammar in CNF equivalent to the grammar S→ ~S|[S∩S]|p|q.

ii) Find a grammar in CNF equivalent to G= S→ bA|aB, A→ bAA|aS|a, B→ aBB|bS|b.

4. a) Explain Griebach Normal Form

b) i) Convert the following grammar into GNF: E→ E+T|T, T→ T*F|F, F→ (E)|a.

ii) Convert the following grammar into GNF: S→ Ba|ab, A→ aAB|a, B→ ABb|b.

5. a) Explain and prove the pumping lemma for context free languages.

b) Show that the following languages are not CFL

i) L= $\{a^i b^j \mid j=i^2\}$ ii) L=$\{a^n b^n c^j \mid n \le j \le 2n\}$

c) Consider the following grammar and find whether it is empty, finite or infinite

i) S→ AB, A→ BC|a, B→ Cc|B,C→ a.

ii) S→ AB, A→ BC|a, B→ CC|b, C→ a,C→ AB.

6. a) Define Push Down Automata. Explain its model with a neat diagram.

b) Explain ID of PDA

c) Construct a PDA which accepts

i) L= $\{a^3 b^n c^n \mid n \ge 0\}$ ii) L=$\{ a^p b^q c^m \mid p+m=q\}$ iii) L= $\{a^i b^j c^k \mid i+j=k; i \ge 0, j \ge 0\}$

7. a) Construct a CFG for the following PDA M=({q0,q1},{0,1},{Z0,X},δ,q0,Z0,ϕ) and δ is given by

δ (q0,1,Z0)=(q0,XZ0), δ(q0,ε,Z0)=(q0, ε), δ (q0,1,X)=(q0,XX)

δ(q1,1,X)=(q1, ε), δ (q0,0,X)=(q1,X), δ (q1,1,Z0)=(q0,Z0).

b) Construct PDA for the grammar S→ aA, A→ aABC|bB|a, B→ b, C→ c.

8. a) Construct a Two Stack PDA which accepts L=$\{a^n b^n c^n \mid n \varepsilon N\}$

b) Design a Two Stack PDA which accepts L=$\{a^n b^n a^n b^n \mid n \varepsilon N \}$

9. a) Differentiate Deterministic PDA and Non- Deterministic PDA.

b) Explain acceptance of PDA by empty state and final state.

c) Prove the equivalence of acceptance of PDA by empty state and final state.

10. a) Explain the closure properties of Context Free Languages.

b) Design a Non Deterministic PDA for the language $L=\{0^n1^n \mid n \geq 1\}$.

UNIT-IV

1. a) Define Turing Machine. Explain its model with a neat diagram.
b) Explain ID of a Turing Machine.
c) Design a Turing machine which accepts the following languages
i) $L= \{a^nb^nc^n \mid n\geq 0\}$.
ii) $L= \{a^{2n}b^n \mid n\geq 1\}$.
iii) accepting palindrome strings over {a ,b}.

2. a) Explain how a Turing Machine can be used to compute functions from integers to integers.

b) Design a Turing Machine to perform proper subtraction m – n, which is defined as m-n for $m \geq n$ and zero for $m < n$.

c) Design a Turing Machine to perform multiplication.

3. Design a Turing machine to compute the following

a) Division of Two integers b) 2's complement of a given binary number

4. Design a Turing machine to compute the following

a) x^2 b) n! c) $\log_2 n$

5. a) Explain in detail various types of Turing Machines.

b) List the properties of Recursive and Recursively Enumerable Languages.

c) Explain the following

i) Church's Hypothesis ii) Counter Machine.

UNIT-V

1. Explain the Chomsky Hierarchy with a neat diagram.

2. Explain in detail the Universal Turing Machine.

3. Explain the following

   a) Decidability   b) Post Correspondence Problem c) Turing Reducibility

4. Explain P and NP Classes.

5. a) Define NP-Complete and NP-Hard Problems.

   b) Explain some NP-Complete Problems in detail.

19. Unit Wise Objective Type Questions

UNIT - I

1. The prefix of abc is _ _ _ _ _ _ _ _ _ _ _ _ _ (d)
a. c
b. b
c. bc
d. a

2. Which of the following is not a prefix of abc? (d)
a. e
b. a
c. ab
d. bc

3. Which of the following is not a suffix of abc ? (d)
a. e
b. c

c. bc

d. ab

4. Which of the following is not a proper prefix of doghouse ? (d)
a. dog
b. d
c. do
d. doghouse

5. Which of the following is not a proper suffix of doghouse ? (d)
a. house
b. se
c. e
d. doghouse

6. If then the number of possible strings of length 'n' is _____________(d)
a. n
b. n * n
c. n n
d. 2 n

7. The concatenation of e and w is _ _ _ _ _ _ (b)
a. e
b. w
c. ew
d. can't say

8. _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ is a set of strings . (a)
a. Language
b. grammar
c. NFA
d. DFA

9. _ _ _ _ _ _ _ _ _ _ _ _ _ is a finite sequence of symbols. (c)
a. Language
b. grammar
c. string
d. NFA

10. Let a is any symbol, x is a palindrome then which of the following is not a Palindrome. (d)
a. e
b. a
c. axa
d. xa

11. Let a is any symbol , x is a palindrome then which of the following is a palindrome. (a)
a. e
b. xa
c. ax
d. aax

12. The basic limitation of FSM is that _ _ _ _ _ _ _ _ (a)
a. it can't remember arbitrary large amount of information
b. it sometimes recognizes grammars that are not regular
c. it sometimes fails to recognize grammars that are regular
d. it can remember arbitrary large amount of information

13. The number of states of the FSM required to simulate the behavior of a computer witha memory capable of storing m words each of length n bits is _________(b)
a. m
b.
c. 2mn
d. 2m

14. We formally denote a finite automaton by (Q, ,q0 , F) Where is the transition Function mapping from Q X to______ (a)
a. Q
b.
c. q0
d. F

15. Application of Finite automata is __________________ (a)
a. Lexical analyzer
b. parser
c. scanner
d. semantic analyzer

16. An FSM can be used to add two given integers .This is_________________________ (b)
a. true
b. false
c. may be true
d. can't say

17. We formally denote a finite automaton by a____________ tuple. (c)
a. 3
b. 4
c. 5
d. 6

18. We formally denote a finite automaton by Where Q is _ _ _ (a)
a. a finite set of states
b. finite input alphabet
c. initial state
d. A set of final states

19. We formally denote a finite automaton by Where is _ _ _ (b)
a. a finite set of states
b. finite input
acl.p ihniatbiaelt state
d. A set of final states

20. We formally denote a finite automaton by Where Q is _ _ _ (c)
a. a finite set of states
b. finite input alphabet
c. initial state
d. A set of final states

21. We formally denote a finite automaton by Where F is _ _ _ (d)
a. a finite set of states
b. finite input alphabet
c. initial state
d. A set of final states

22. An automation is a _ _ _ _ _ _ _ _ _ _ _ _ _ _ device (b)
a. generative
b. cognitive
c. acceptor
d. can't say

23. A grammar is a _ _ _ _ _ _ _ _ _ _ _ _ _ _ device (a)
a. generative
b. cognitive
c. acceptor
d. can't say

24. An FSM can be used to add two given integers .This is_______________ (b)
a. true
b. false
c. may be true
d. can't say

25. An FSM can be used to perform subtracttion of given two integers .This is _ _   (b)
a. true
b. false
c. may be true
d. can't say

26. The word formal in formal languages means _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _   (c)
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. only the form of the string of symbols is not significant

27. The recognizing capability of NDFSM and DFSM [04S02]   (c)
a. may be different
b. must be different
c. must be same
d. may be same

28. Any given transition graphs has an equivalent _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _   (d)
a. RE
b. DFA
c. NFA
d. DFA, NFA, RE

29. Finite state machine________________________ recognize palindromes
(b)
a. can
b. can't
c. may
d. may not

30. FSM can recognize _ _ _ _ _ _ _ _ _ _ _   (d)
a. any grammar
b. only CFG
c. any unambiguous grammar
d. only regular grammar

31. Palindromes can _ t be recognized by any FSM because   (a)
a. FSM can't remember arbitrarily large amount of
b FSM cannot deterministically fix the mid point
c even of the mid-point is known, an FSM cannot find whether the second half of the string matches the first half
d FSM can remember arbitrarily large amount of information

32. Let M = (Q,S, ,q0 , F) , F= { q0 } , S= {0,1 }. :
Then (q0 , 110101) _ _ _ _ _ _ _ _ _ _ _ _ _   (a)
a. q0

b. q1

c. q2
d. q3

33. Let M = (Q,S, ,q0 , F) , F= { q0 } , S= {0,1 }. :
Then L(M) is the set of strings with _ _ _ _ number of 0's and ______________ Number of 1's .
(c)
a. odd, odd
b. odd, even
c. even, even
d. even, odd

34. Let M = (Q,S, ,q0 , F) , F= { q0 } , S= {0,1 }. :
Then (q0 , 110) _ _ _ _ _ _ _ _ _ _ _ _ _ (c)
a. q0
b. q1
c. q2
d. q3

35. Let M = (Q,S, ,q0 , F) , F= { q0 } , S= {0,1 }. :
Then which of the following is accepted _ _ _ _ _ _ _ _ _ _ _ _ _ (a)
a. 110101
b. 11100
c. 00011
d. 111000

36. Let M = (Q,S, ,q0 , F) , F= { q0 } , S= {0,1 }. :
Then which of the following is not accepted ______________________(d)
a. 11101
b. 110001
c. 0011
d. 1101

37. In transition diagrams states are represented by _ _ _ _ _ _ _ _ _ _ _ _ _ (b)
a. ellipses
b. circles
c. triangles
d. rectangles

38. In transition diagrams a state pointed by an arrow represents the _ _ _ _ _ _ _ state. (c)
a. final
b. interior
c. start
d. final or start

39. In transition diagrams a state encircled by another represents _ _ _ _ _ _ _ state. (a)
a. final
b. interior

c. start
d. final or start

40. NFA stands for _ (a)
a. Non deterministic finite automaton
b. Non deterministic finite analysis
c. Non deterministic finite acceptance
d. Non deterministic finite authorization

41. Consider the following NFA
Now (q0, 01) = _ (a)
a. {q0, q1}
b. {q0 , q3,q4 }
c. {q0 , q1, q4 }
d. {q4 }

42. Consider the following NFA
Now (q0, 010) = _ _ _ _ _ _ _ _ _ _ _ (b)
a. {q0 , q1 }
b. {q0 q3}
c. {q0 , q1, q4 }
d. {q4 }

43. Consider the following NFA
Now (q0, 01001) = _ _ _ _ _ _ _ _ _ _ _ (c)
a. {q0 , q1 }
b. {q0 , q3 }
c. {q0 , q1,q4}
d. {q4 }

44. Consider the following NFA
Now (q0, 0) = _ _ _ _ _ _ _ _ _ _ _ (c)
a. {q0 , q1 }
b. {q0 , q3 }
c. {q0 , q1,q4}
d. {q4 }

45. Let NFA has a finite number n of states ,the DFA will have at most ____________ states.
(d)
a. 2n
b. n/2
c. n 2
d. 2 n
46. Let NFA has a finite number 6 of states ,the DFA will have at most ____________ states.
(d)
a. 12
b. 2

c. 36
d. 64

47. Can a DFA simulate NFA ? [08S01] (b)
a. No
b. Yes
c. sometimes
d. depends on NFA

48. The DFA start state = _ (c)
a. NFA start state
b. NFA final state
c. closure(NFA start state)
d. closure (NFA final state)

49. Let maximum number of states in a DFA =64 .
Then it's equivalent NFA has _ _ _ _ _ _states. (d)
a. 2
b. 4
c. 8
d. 6

50. Let maximum number of states in a DFA =128 .
Then its equivalent NFA has _ _ _ _ _ _ states. (b)
a. 5
b. 7
c. 8
d. 9

51. Let maximum number of states in a DFA =1024.
Then it's equivalent NFA has _ _ _ _ _ states. (c)
a. 5
b. 7
c. 10
d. 11

52. Choose the wrong statement (d)
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

53. Choose the wrong statement (d)
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. A Mealy machine has terminal state

54. The major difference between a mealy and a moore machine is that (b)
a. The output of the former depends on the present state and present input
b. The output of the former depends only on the present stste
c. The output of the former depends only on the present input
d. The output of the former doesn't depends on the present state

55. In moore machine shows _ (c)
a. states
b. input alphabet
c. output alphabet
d. Final state

56. A melay machine is a _ _ _ _ _ _ _ _ _ _ _ tuple. (d)
a. 4
b. 5
c. 7
d. 6

UNIT- II

57. In case of regular sets the question ' is the intersection of two languages a language of the same type ?' is _ (c)
a. Decidable
b. Un decidable
c. trivially decidable
d. Can't say

58. In case of regular sets the question 'is the complement of a language also a language of the same type ? ' is _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ (c)
a. Decidable
b. Un decidable
c. trivially
dd.e Cciadna'tb slaey

59. In case of regular sets the question ' is L1 n L2 = F ? ' is _ _ _ _ _ _ _ _ _ _ _ _ _ _ (a)
a. Decidable
b. Undecidable
c. trivially decidable
d. Can't say

60. In case of regular sets the question ' is L=R where R is a given regular set ?' is ________ (a)
a. Decidable
b. Undecidable
c. trivially decidable
d. Can't say

61. In case of regular sets the question ' is L regular?' is _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ (c)
a. Decidable
b. Undecidable
c. trivially decidable
d. Can't say
62. In case of regular sets the question 'Is w in L? 'Is _
(a)
a. Decidable
b. Undecidable
c. trivially decidable
d. Can't say

63. In case of regular sets the question 'is L = F? 'Is _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ (a)
a. Decidable
b. Undecidable
c. trivially decidable
d. Can't say

64. In case of regular sets the question 'is L = *? Is _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ (a)
a. Decidable
b. Undecidable
c. trivially decidable
d. Can't say

65. In case of regular sets the question ' is L1 = L2? ‘is _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ (a)
a. Decidable
b. Undecidable
c. trivially decidable
d. Can't say

66. In case of regular sets the question 'is L1subset or equal to L2? ‘Is _ _ _ _ _ _ _ (a)
a. Decidable
b. Undecidable
c. trivially decidable
d. Can't say

67. The regular expression (1 + 10) * denotes all strings of 0's and 1's beginning with _ _ _ _ _ _
_ _ _ _ _ _ _ _ and not having two consecutive _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ (a)
a. 1, 0's
b. 0, 1's
c. 0, 0's
d. 1, 1's

68. Let r and s are regular expressions denoting the languages R and S.
Then (r + s) denotes _ _ _ _ _ _ _ _ _ _ _ _ (c)
a. RS
b. R*
c. RUS
d. R+

69. Let r and s are regular expressions denoting the languages R and S.
Then (r s) denotes _ _ _ _ _ _ _ _ _ _ _ _ _ (a)
a. RS
b. R*
c. RUS
d. R+

70. Let r and s are regular expressions denoting the languages R and S.
Then (r*) denotes _ _ _ _ _ _ _ _ _ _ _ (b)
a. RS
b. R*
c. RUS
d. R+

71. _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ denotes all strings of 0,s and 1,s. (d)
a. (0+1)
b. 01
c. 0* 1
d. (0+ 1)*

72. (0+1) * 011 denote all strings of 0's and 1's ending in _ _ _ _ _ _ _ _ _ _ _ _ (c)
a. 0
b. 0111
c. 011
d. 111

73. Let r, s, t are regular expressions. (r* s *) * =_________________________________ (c)
a. (r-s)*
b. (r s)*
c. (r +s)*
d. (s-r)*

74. Let r, s, t are regular expressions. (r + s)* = _ (c)
a. r *s*
b. (rs)*
c. (r* s *) *
d. r *+s*

75. Let r, s, t are regular expressions. (r*)* = _ (b)
a. r
b. r*
c. F
d. can’t say

76. Let r, s, t are regular expressions. (e + r)* = _ (c)

a. r
b. e
c. r*
d. e r

77. Let r, s, t are regular expressions. r + s = _ (b)
a. r s
b. s + r
c. s r
d. r / s

78. Let r, s, t are regular expressions. (r + s) +t = _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ (a)
a. r +(s +t)
b. r s t
c. r t
d. s t

79. Let r, s, t are regular expressions. (r s) t = _ (c)
a. r s
b. r t
c. r(st)
d. s t

80. Let r, s, t are regular expressions. r(s+ t) = _ (d)
a. r s
b. r t
c. rs - r t
d. rs +r t

81. Let r, s, t are regular expressions. (r + s) t = _ (a)
a. r t +st
b. (r-s)t
c. (rs) t
d. t(rs)

82. In NFA for r=e the minimum number of states are________________________________(b)
a. 0
b. 1
c. 2
d. 3

83. In NFA for r=F the minimum number of states are _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ (c)
a. 0

b. 1
c. 2
d. 3

84. In NFA for r=a the minimum number of states are________________________________ (c)
a. 0
b. 1
c. 2
d. 3

85. (e + 00)* = _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ (d)
a. e
b. 0
c. e 0
d. (00)*

86. 0 (00)* (e + 0)1 + 1 = _ (a)
a. 00* 1 + 1
b. 00* 1
c. 0 *1 +1
d. 00*+1

87. 1 + 01 = _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ (b)
a. e + 0
b. (e + 0) 1
c. 1 (e +0)
d. 101

88. Let f(0) =a and f(1) =b* Then f(010) = _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ (c)
a. a
b. b*
c. a b* a
d. aba

89. Let f(0)=a and f(1) = b* If L is the language 0*(0+1)1* then f(L)= _ _ _ _ (d)
a. ab
b. a b*
c. b*
d. a* b*

90. Let L1 be 0*10* and L2 be 1 0* 1 The quotient of L1 and L2 is _ _ _ _ _ _ _ (a)
a. empty
b. 0*
c. 1
d. 10*

91. Let L1 be 0*10* and L2 be 0* 1 The quotient of L1 and L2 is _ _ _ _ _ _ _ (b)
a. empty

b. 0*
c. 1
d. 10*

92. Let L1 be 10* 1 and L2 be 0* 1 The quotient of L1 and L2 is _ _ _ _ _ _ _ (d)
a. empty
b. 0*
c. 1
d. 10*

93. 'The regular sets are closed under union' is _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ (a)
a. True
b. False
c. True or False
d. can't say

94. 'The regular sets are closed under concatenation' is _ _ _ _ _ _ _ _ (a)
a. True
b. False
c. True or False
d. can't say

95. 'The regular sets are closed under kleene closure' is _ _ _ _ _ _ _ _ _ _ (a)
a. True
b. False
c. True or False
d. can't say

96. 'The regular sets are closed under intersection' is _ _ _ _ _ _ _ _ _ _ _ _ _ _ (a)
a. True
b. False
c. True or False
d. can't say

97. The class of regular sets is closed under complementation .That is if L is a regular set and L is
subset or equal to * then _ _ _ _ _ _ _ _ _ _ _ _ _ _ is regular set (d)
a.
b. *
c. * + L
d. * - L

UNIT – III

98. Regular grammars also known as _ _ _ _ _ _ _ _ _ _ _ _ _ grammar. (d)

a. Type 0
b. Type 1
c. Type 2
d. Type3

99. _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ grammar is also known as Type 3 grammar. (d)
a. un restricted
b. context free
c. context sensitive
d. regular grammar

100. Which of the following is related to regular grammar ? (c)
a. right linear
b. left linear
c. Right linear & left linear
d. CFG

101. Regular grammar is a subset of _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ grammar. (d)
a. Type 0 .
b. Type 1
c. Type 2
d. Type 0,1 &2

102. P,Q, R are three languages .If P and R are regular and if PQ=R then (c)
a. Q has to be regular
b. Q cannot be regular
c. Q need not be regular
d. Q has to be a CFL

103. Let A = {0,1 } L= A * Let R = { 0 n1n , n >0 } then LUR is regular and R is _ _ (b)
a. regular
b. not regular
c. regular or not regular
d. can`t say

104. Let L1 =(a+b) * a L2 =b*(a+b)
L1 intersection L2 = _ _ _ _ _ _ _ _ _ _ _ (d)
a. (a+b) * ab
b. ab (a+b) *
c. a (a+b) * b
d. b(a+b)*a

105. Let L denote the language generated by the grammar S0s0100 then (c)
a. L= 0 +
b. L is CFL but not regular
c. L is regular but not 0 +
d. L is not context free

106. Let A = {0,1 } L= A * Let R = { 0 n1n , n >0 } then LUR _ _ _ _ _ _ _ _ _ _ _ _ (a)
a. regular
b. not regular
c. regular or not regular
d. can`t say

107. Which of the following are regular? (d)
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 prime number
d. string of odd number of zeros

108. Pumping lemma is generally used for proving (b)
a. a given grammar is regular
b. a given grammar is not regular
c. whether two given regular expressions are equivalent are not
d. a given grammar is CFG

109. Pick the correct statement the logic of pumping lemma is a good example of (a)
a. the pigeon hole principle
b. divide and conquer
c. recursion
d. iteration

110. The logic of pumping lemma is a good example of _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ (d)
a. iteration
b. recursion
c. divide and conquer
d. the pigeon hole principle

111. Let L1 = { n.m =1,2,3 .....}
L2 = { n ,m=1,2,3..... }
L3 = { n =1,2,3..... }
Choose the correct answer (a)
a. L3= L1 intersection L2
b. L1, L2 , L3 are CFL
c. L1, L2 not CFL L3 is CFL
d. L1 is a subset of L3

112. Choose the wrong statement (a)
a. All languages can be generated by CFG
b. Any regular language has an equivalent CFG
c. Some non regular languages can _ t be generated by CFG
d. Some regular languages can be simulated by an FSM

113. In CFG each production is of the form Where A is a variable and is string of Symbols from _ _ _ _ _ _ _ _ _ _ _ (V, T are variables and terminals) (d)
a. V
b. T
c. VUT
d. *(VUT)

114. Any string of terminals that can be generated by the following CFG (d)
a. has atleast one b
b. should end in a 'a'
c. has no consecutive a's or b's
d. has atleast two a's

115. CFG is not closed under (c)
a. union
b. kleene star
c. complementation
d. product

116. The set A= { n=1,2,3 ..... } is an example of a grammar that is (c)
a. regular
b. context free
c. not context free
d. can`t say

117. Let G=(V,T,P,S) be a CFG. A tree is a derivation (or parse) tree for G if If vertex n has label ? then n is a _ _ _ _ node (d)
a. root
b. interior
c. root or interior
d. leaf

118. The vernacular language English ,if considered a formal language is a (b)
a. regular language

b. context free language
c. context sensitive language
d. can`t say

119. The language constructs which are most useful in describing nested structures such as balanced parentheses matching begin ends etc are _ _ _ _ _ _ _ _ (b)
a. RE
b. CFG
c. NM CFG
d. CSG

120. CFL are closed under (c)
a. Union, intersection
b. kleene closure
c. Intersection, complement
d. complement, kleene closure

121. Recursively enumerable languages are accepted by? (a)
a. TM
b. FA
c. PDA
d. None

122. The statement –'ATM can't solve halting problems (a)
a. true
b. false
c. still an open question
d. none of the above

123. The language { 1n 2n 3n / n>=1} is recognized by? (c)
a. FA
b. PDA
c. TM
d. None of the above

124. The language L (0^n 1^n 2^n where n>0) is a (b)
a. context free language
b. context sensitive language
c. regular language
d. recursively enumerable language

125. Recursively enumerable languages are not closed under. (c)
a. Union
b. Intersection
c. Complementation
d. concatenation

126. The class of languages generated by ---- grammar is exactly the linear bounded languages. (b)
a. RG
b. CFG
c. CSG
d. PSG

127. Which of the following is the most general phase-structured grammar? (b)
a. regular
b. context-sensitive
c. context free
d. none of the above

128. The number of internal states of a UTM should be atleast (b)
a. 1
b.2
c. 3
d.4

129. Context Sensitive Grammar (CSG) can be recognized by (b)
a. Finite state automata
b. 2-way linear bounded automata
c. push down automata
d. none of the above

130. The language L= (0^n 1^n 2^R 3^R where n, R>0) is a (a)
c. context free language
d. context sensitive language
c. regular language
d. recursively enumerable language

130.A Pushdown automata is.... if there is at most one transition applicable to each configuration ?

a. Deterministic (a)

b. Non Deterministic

c. Finite

d. Non Finite

131. The idea of automation with a stack as auxiliary storage? (b)

a. Finite automata

b. Push down automata

c. Deterministic automata

d. None of these

132. Suppose $((p,a,□),(q,□))$ is a production in a push-down automaton. True or

false: a □ is popped from the stack if this production isused.
b □ is pushed onto the stack if this production is
used. c □ is popped from the stack if this production
is used. d □ is pushed onto the stack if this
production is used.

133. Which of the following is not accepted by DPDA but accepted by NDPDA ()

a. Strings end with a particular alphabet
b. All strings which a given symbol present at least twice
c. Even palindromes
d. None

134. PDA maintains (d)

a. Tape
b. Stack
c. Finite Control Head
d. All the ab

UNIT - IV

135.A Turing machine can be used to (c)

a. Accept languages
b. Compute functions
c. a & b
d. none

136. Any turing machine is more powerful than FSM because (c)

a. Tape movement is confined to one direction
b. It has no finite state control
c. It has the capability to remember arbitrary long input symbols
d. TM is not powerful than FSM

137. In which of the following the head movement is in both directions (d)

a. TM
b.FSM
c.LBA
d.a& c

138. A turing machine is (a)

a. Recursively enumerable language
b. RL
c.CFL
d.CSL

139. Any Turning machine with m symbols and n states can be simulated by another TM with just
2 s symbols and less than (d)

a. 8mn states
b.4mn+8states
c. 8mn+ 4 states
d. mn states

UNIT - V

134. Push down automata represents

a. Type 0 Grammar
b. Type 1 Grammar
c. Type 2 Grammar
d. Type 3 Grammar

135. If every string of a language can be determined whether it is legal or illegal in finite time the language is called

a. Decidable
b.undecidable
c.Interpretive
d. Non deterministic

136. PCP having no solution is called (b)

a. undecidability of PCP
b.decidability of PCP
c.Semi-decidability of PCP
d None

137. Which of the following is type- 2 grammar? (b)

a. $A \rightarrow \alpha$ where A is terminal
b. $A \rightarrow \alpha$ where A is Variable
c. Both
d. None

20. Tutorial Problems

UNIT-I

1. Define epsilon closure. Find NFA without ε for the following NFA with

ε where q0-initial state q3-final state

	a	b	ε
qo	qo	Ø	q1
q1	Ø	{q3,q1}	q2
q2	q2	Ø	{q1,q3}
q3	Ø	Ø	Ø

2 a) Construct DFA equivalent where initial and final state is q0

	0	1
qo	q0	q1
q1	q1	{q0,q1}

b)Construct DFA equivalent where initial state is A and final state is C

	0	1	ε
A	A,B	A	C
B	C	Ø	Ø
C	C	C	A

3. Minimize the FA given below and show both given and reduced FA'S are equivalent or

not where q0-initial state q6-final state

	0	1
qo	q1	q2
q1	q3	q4
q2	q5	q6
q3	q3	q4
q4	q5	q6
q5	q3	q4
q6	q5	q6

4.a) Discuss about FA with output in detail

b) Convert the following melay machine to moore machine

	Input symbol=0		Input symbol=1	
	Nextstate	output	Nextstate	output
q0	q1	N	q2	N
q1	q1	Y	q2	N
q2	q1	N	q2	Y

5. a) Explain significance of NFA with ε transitions and write differences between NFA with ε and ordinary NFA. Define NFA-ε transitions

b) Convert the following moore machine to melay machine

	a=0	a=1	output
qo	q1	q2	1
q1	q3	q2	0
q2	q2	q1	1
q3	q0	q3	1

UNIT-II

1. Define grammar, regular grammar, right linear grammar, left linear grammar with examples.

2. a) what are the rules to construct regular grammar for a given finite automata

   b) Construct regular grammar for the given TT where q3 is final state

	0	1
qo	q1	φ
q1	q2	q1
q2	q2	q3
q3	q2	q1

3. a) What are the rules to construct finite automata for a given regular grammar

   b) Construct FA recognizing L (G) where the grammar is

S→ aS|bA|b

A→ aA|bS|a

4. a) Write short notes on context free grammar

   b) Obtain CFG to obtain balanced set of parentheses (that is every left parentheses should match with the corresponding right parentheses

5.a) Define derivation, derivation tree, sentential form, LMD, RMD

   b) Find LMD, RMD, and DT for the string: 00110101 where the grammar is

S→ 0B|1A

A→ 0|0S|1AA

B→ 1|1S|0BB

UNIT-III

1. What is CFL generated by the grammar $S \rightarrow abB$, $A \rightarrow aaBb$, $B \rightarrow bbAa$, $A \rightarrow \varepsilon$

2. Given the grammar G as S→ 0B|1A, A→ 0|0S|1AA, B→ 1|1S|0BB. Find leftmost and rightmost derivation and derivation tree for the string 00110101.
3. Construct the leftmost, rightmost derivation and parse tree for the following grammar which accepts the string aaabbabbba S→ aB|bA, A→ aS|bAA|a, B→ bS|aBB|b.
4. Simplify the following grammar: S→ aA|aBB, A→ aAA|ε, B→ bB|bbC, C→ B.
5. Simplify the following grammar: S→ AaB|aaB, A→ D, B→ bbA|ε, D→ E, E→ F, F→ aS.
6. Convert the following grammar into CNF

   S→ aA|a|B|C, A→ aB|ε, B→ aA, C→ cCD, D→ abd.
7. Convert the following grammar into GNF: S→ AB, A→ BS|b, B→ SA|a.
8. Show that L={$a^n b^n c^n$|n≥1} is not CFL.
9. Construct a PDA accepting {$a^n b^n$|n≥1} by Empty Stack and by final state.
10. Construct PDA for the grammar S→ aA, A→ aABC|bB|a, B→ b, C→ c.

UNIT-IV

1. Design a Turing Machine M to accept the language L= {$0^n 1^n$|n≥1}.
2. Design a Turing Machine M to accept strings of the language L= {$a^n b^n c^n$ | n≥0}.
3. Design a Turing Machine to perform proper subtraction m – n, which is defined as m-n for m ≥ n and zero for m < n.
4. Design a Turing Machine to perform multiplication.
5. Design a Turing Machine that gives two's complement for the given binary representation

UNIT-V

1. Show that the PCP with two lists $x=(b,bab^3,ba)$ and $y=(b^3,ba,a)$ has a solution. Give the solution sequence.

2. Find the solution for PCP problem given below

	List A	List B
i	w_i	x_i
1	a	aaa
2	abaaa	ab
3	ab	b

3. Explain why the PCP with two lists $x= (ab,b,b)$ and $y=(ab^2,ba,b^2)$ has no solution?

4. Consider the following Turing machine defined as M=({q0,q1,qA},{0,1},{0,1,B},,q0,B,{qA})

	a	b	B
q0	(q1,b,R)	(q1,a,L)	(q1,b,L)
q1	(q_A,a,L)	(q0,a,R)	(q1,a,R)
q_A			

State whether for the string w=ab, Turing Machine halts?

5. Show that the satisfiability problem is in Class NP?

21. Known Gaps if any

No Gaps for this course.

22. Discussion topics

1) Importance of formal languages and it use.
2) Applications of automata theory.
3) Types of finite automata and its application.
4) Importance of FSM with outputs & what are they?
5) Importance of grammar & its formalism.

6) Grammar Normalisation techniques
7) Significance of push down automata
8) Types of PDA & its conversions
9) Significance of Turing machine
10) Types of languages & its importance.

23. References, Journals, websites and E-links

- **References:**

1) "Introduction to Automata Theory Languages and Computation".Hopcroft H.E. and Ullman J.D.Pearson Education.
2) "Theory of computer Science- Automata Languges and computation"- Mishra and Chandrashekaran, second edition,PHI.
3) "Elements of Theory of Computations",Lewis H.P. &Papadimition C.H.Person/PHI.
4) An introduction to formal languages and automata by Peter Linz

- **Journals**:

1) **On external contextual grammars with subregular selection languages.**
2) **Nonterminal complexity of tree controlled grammars**
3) Weighted grammars and automata with threshold interpretation.

- **Aspects of Language and Automata Theory – Special Issue Dedicated to Jürgen Dassow.**

- **Websites:**

http://www.cse.chalmers.se/edu/course/TMV027/

http://www.eecs.wsu.edu/~ananth/CptS317/

http://www.nptel.iitm.ac.in/downloads/106106049/

- **E-links**

http://books.google.co.in/books?id=tzttuN4gsVgC&source=gbs_similarbooks

http://en.wikipedia.org/wiki/Formal_language

http://en.wikipedia.org/wiki/Automata_theory

http://cs.fit.edu/~dmitra/FormaLang/

http://www.computersciencemcq.com/mcq.aspx?name=Theory_of_Comp utati on_MCQ_14