Theory of Automata and Formal Languages

IV semester – CECSC, CACSC, MCCSC 10
(3,1,0)

Course Objectives

- Introduce concepts in automata theory and theory of computation
- Identify different formal language classes and their relationships
- Design grammars and recognizers for different formal languages
- Prove or disprove theorems in automata theory using its properties
- Determine the decidability and intractability of computational problems

In this course we acquire knowledge of the following

- Regular Languages
- FA-Finite Automata
- CFG-Context Free Grammar
- PDA-Push Down Automata
- TM-Turing Machines

Week	Lecture Topic
1	Finite Automata: Deterministic FA, Non deterministic FA, Finite Automaton with €- moves, Kleen's theorem–Conversion of NFA to DFA, Moore and Mealy Machine
2	Regular Expression, Regular Languages. Equivalence of finite Automaton and regular expressions, Minimization of DFA
3	Pumping Lemma for Regular sets, Problems based on Pumping Lemma.
4	Context Free Grammar: Grammar, Types of Grammar, Context Free Grammars and Languages, Derivations, Ambiguity, Relationship between derivation and derivation trees, Simplification of CFG, Elimination of Useless symbols - Unit productions - Null productions, Chomsky normal form (CNF), Greibach Normal form (GNF), Problems related to CNF and GNF.
5	Pushdown Automata: Moves, Instantaneous descriptions, Deterministic pushdown automata, Equivalence of Pushdown automata and CFL, pumping lemma for CFL, problems based on pumping Lemma.
6	Applications of the Pumping Lemma Closure Properties of Regular Languages, Decision Properties of Regular Languages, Equivalence and Minimization of Automata, Context-Free Grammars and Languages: Definition of Context-Free Grammars, Derivations Using a Grammars Leftmost and Rightmost Derivations, The Languages of a Grammar, Parse Trees: Constructing Parse Trees, The Yield of a Parse Tree, Inference Derivations, and Parse Trees, From Inferences to Trees, From Trees to Derivations, From Derivation to Recursive Inferences, Applications of Context-Free Grammars: Parsers, Ambiguity in Grammars and Languages: Ambiguous Grammars, Removing Ambiguity From Grammars, Leftmost Derivations as a Way to Express Ambiguity, Inherent Ambiguity
7	Turing Machine: Definitions of Turing machines, Computable languages and functions, Techniques for Turing machine construction, Multi head and Multi tape
8	Turing Machines, The Halting problem, Partial Solvability, Problems about Turing machine- Chomsky hierarchy of languages.
9	Difficult problems: Unsolvable Problems and Computable Functions, Primitive recursive functions, Recursive and recursively enumerable languages, Universal Turing machine, Measuring and classifying complexity - Tractable and Intractable problems.
10	Tractable and possibly intractable problems, P and NP completeness, Polynomial time reductions, NP-complete problems from other domains: graphs (clique, vertex cover, independent sets, Hamiltonian cycle), number problem (partition), set cover.

SUGGESTED READINGS

- Hopcroft J.E., Motwani R. and Ullman J.D, “Introduction to Automata Theory, Languages and Computations”, Second Edition, Pearson Education. 2.
- John C Martin, “Introduction to Languages and the Theory of Computation”, Third Edition, Tata McGraw Hill Publishing Company, New Delhi.
- Mishra K.L.P., Chandrasekaran N., “Theory of Computer Science Automata, Languages and Computation”, Third Edition, Prentice-Hall of India Pvt Ltd, New Delhi.

Automata Theory

Introduction

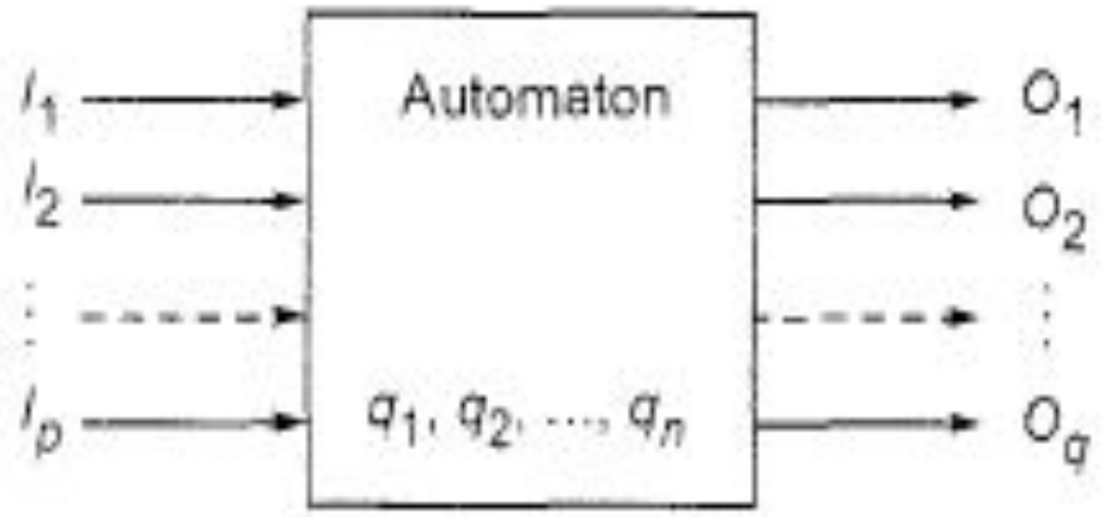
What is Automata Theory?

It is the study of self sufficient abstract computing devices or "machines" that follow a predetermined sequence of steps to solve a problem

In other words *Automaton* is

- A system that performs some function without direct participation of human beings. E.g. Automatic parking machines
- Moving mechanical device made as imitation of a human being.
- A machine which performs a range of functions according to a predetermined set of coded instructions as sophisticated automatons run factory assembly lines effectively
- In Computer Science "automaton" means "discrete automaton"

Model of Discrete Automaton



The characteristics of Automaton are

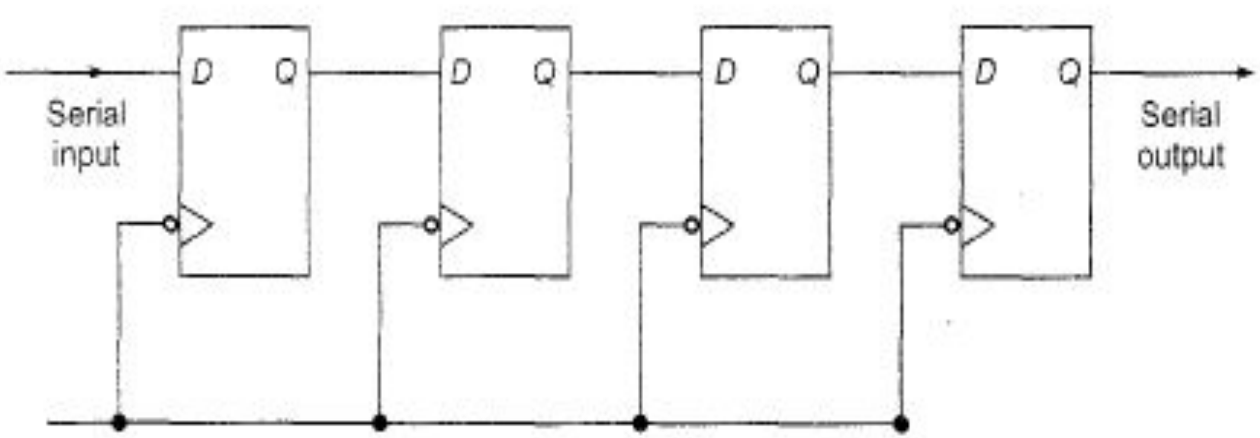
- Input,
- Output,
- States,
- State relations and
- Output relations

Automaton…

- Automaton in which output depends only on the input is automaton without memory
- Automaton in which the output depends on the state as well is called automaton with a finite memory
- Automaton in which the output depends only on the states of the machine is called a *Moore Machine*.
- Automaton in which the output depends on the states as well as on the input at any instant of time is called a *Mealy Machine*.

Example- Mealy/Moore Machine

A 4-bit serial shift register using D flip-flops



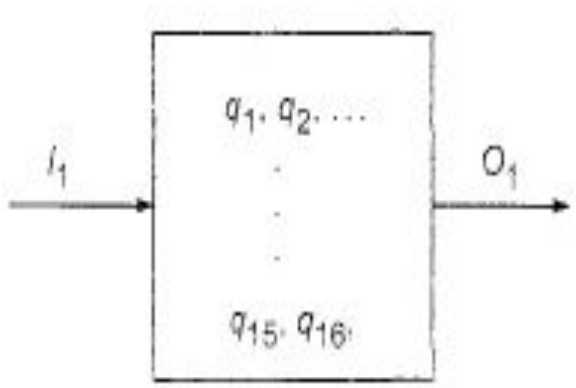
2^4= 16 states (0000,0001,….1111) and one serial input and one serial output.

if

input alphabet is {0,1}

Output alphabet is {0,1}

4-bit serial shift register can be represented as



From the operation it is clear that the output depends upon both the input and the state so it is a *Mealy Machine*

Theory of Computation: A Historical Perspective

1930s	• Alan Turing studies Turing machines • Decidability • Halting problem
1940-1950s	• “Finite automata” machines studied • Noam Chomsky proposes the “Chomsky Hierarchy” for formal languages
1969	Cook introduces “intractable” problems or “NP-Hard” problems
1970-	Modern computer science: compilers, computational & complexity theory evolve

Languages & Grammars

Or "**words**"

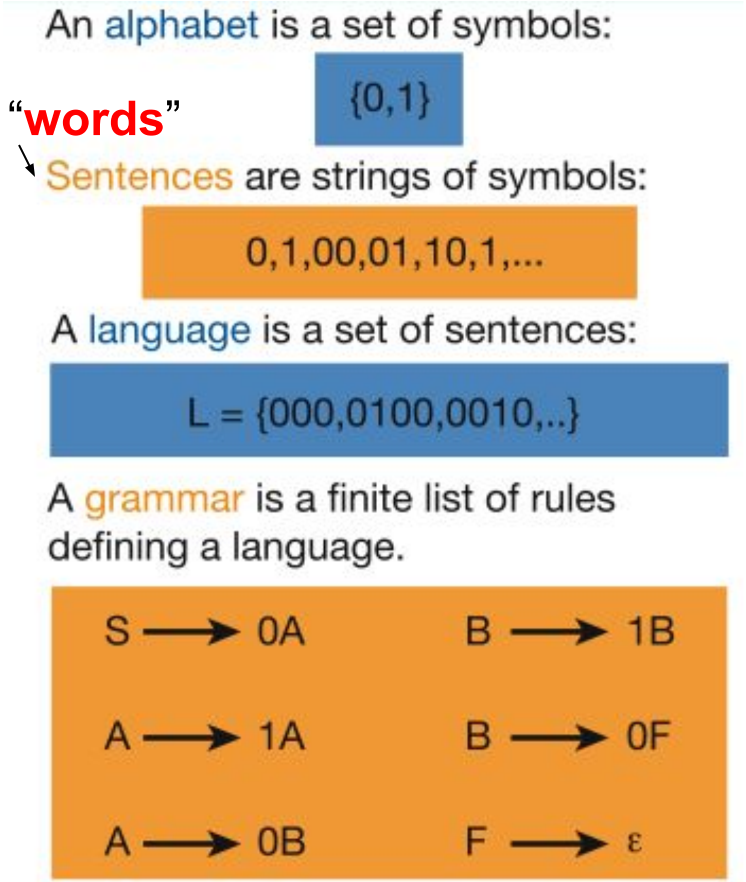
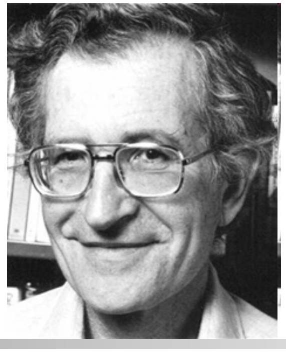


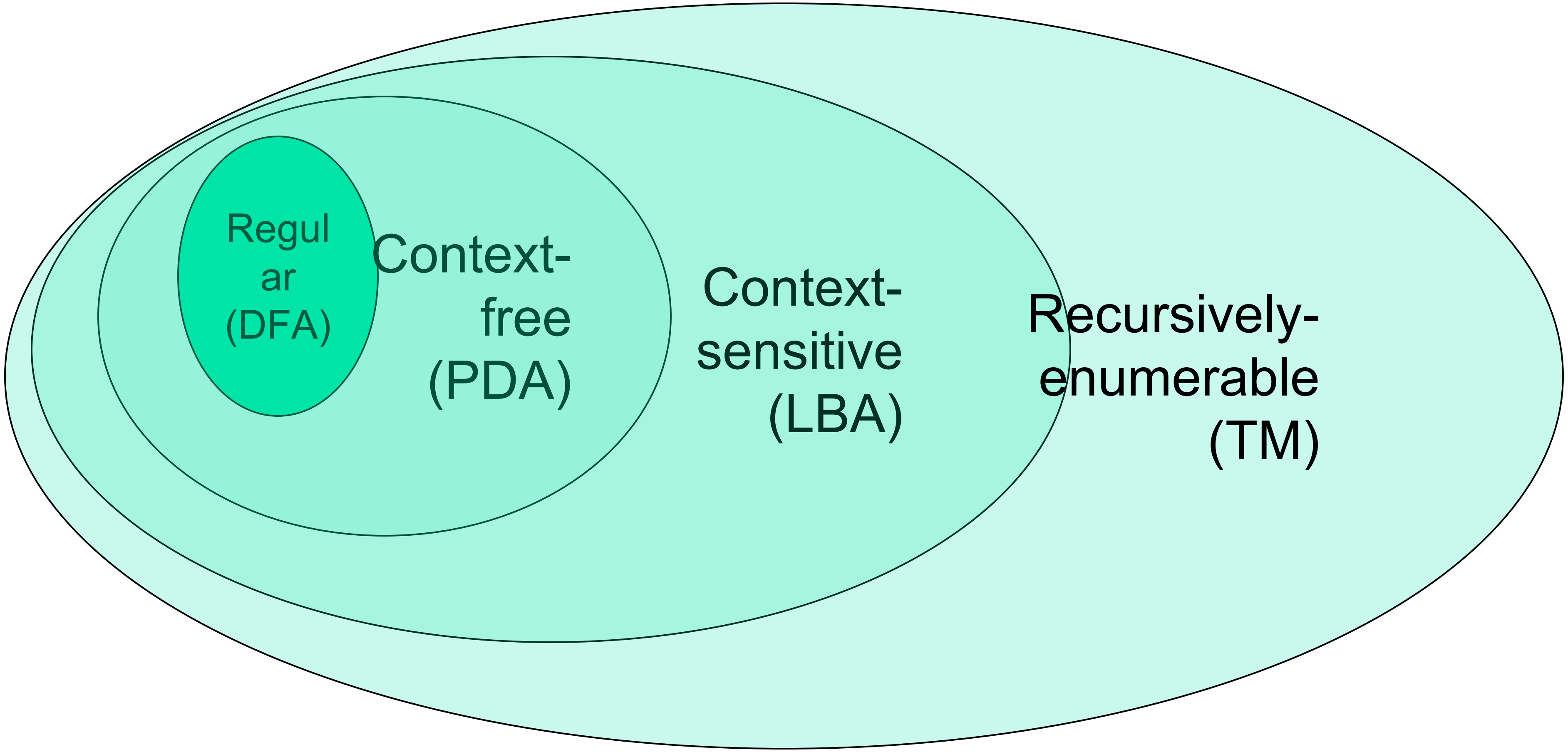
Image source: Nowak et al. Nature, vol 417, 2002

- Languages: "*A language is a collection of sentences of finite length all constructed from a finite alphabet of symbols*"
- Grammars: "*A grammar can be regarded as a device that enumerates the sentences of a language*" - nothing more, nothing less

The Chomsky Hierachy



- A containment hierarchy of classes of formal languages



Basic Concepts of Automata Theory

Alphabet

An alphabet is a finite, non-empty set of symbols

- We use the symbol ∑ (sigma) to denote an alphabet
- Examples:
  - Binary: ∑ = {0,1}
  - All lower case letters: ∑ = {a,b,c,..z}
  - Alphanumeric: ∑ = {a-z, A-Z, 0-9}
  - DNA molecule letters: ∑ = {a,c,g,t}
  - …

Strings

A string or word is a finite sequence of symbols chosen from ∑

- ***Empty string is ε (or "epsilon")***

- Length of a string *w*, denoted by "|*w*|", is equal to the *number of (non- ε) characters in the string*
  - *E.g., x = 010100* |*x*| = *6*
  - *x = 01 ε 0 ε 1 ε 00 ε* |*x*| = *?*

- *xy* = concatentation of two strings *x* and *y*

Powers of an alphabet

Let Σ be an alphabet.

- Σ^k = the set of all strings of length k
- $\Sigma^* = \Sigma^0 \cup \Sigma^1 \cup \Sigma^2 \cup \dots$
- $\Sigma^+ = \Sigma^1 \cup \Sigma^2 \cup \Sigma^3 \cup \dots$

Languages

L is a said to be a language over alphabet ∑, only if L ⊆ ∑ (subset)*

- this is because ∑* is the set of all strings (of all possible length including 0) over the given alphabet ∑

Examples:

1. Let L be *the* language of all strings consisting of *n* 0's followed by *n* 1's:
   L = {ε, 01, 0011, 000111,…}
2. Let L be *the* language of all strings of with equal number of 0's and 1's:
   L = {ε, 01, 10, 0011, 1100, 0101, 1010, 1001,…}

Definition: Ø denotes the Empty language

The Membership Problem

Given a string $w \in \Sigma^*$ and a language L over Σ, decide whether or not $w \in L$.

($\in$- belongs to)

<u>Example:</u>

Let w = 100011

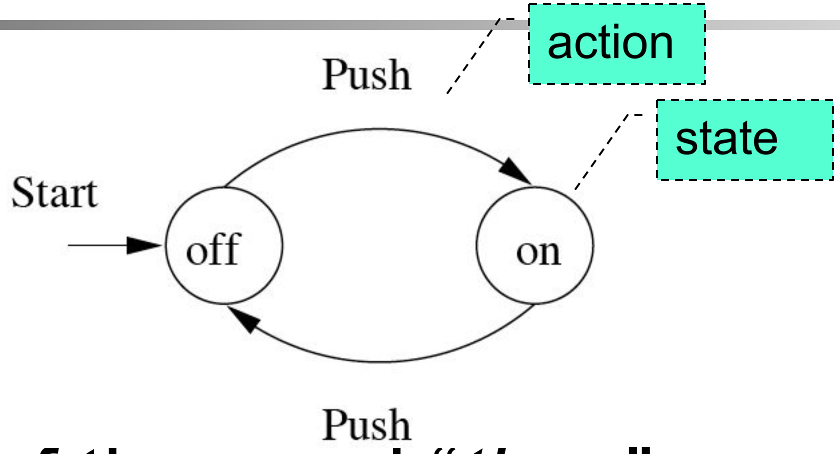
Q) Is w $\in$ the language of strings with equal number of 0s and 1s?

Finite Automata

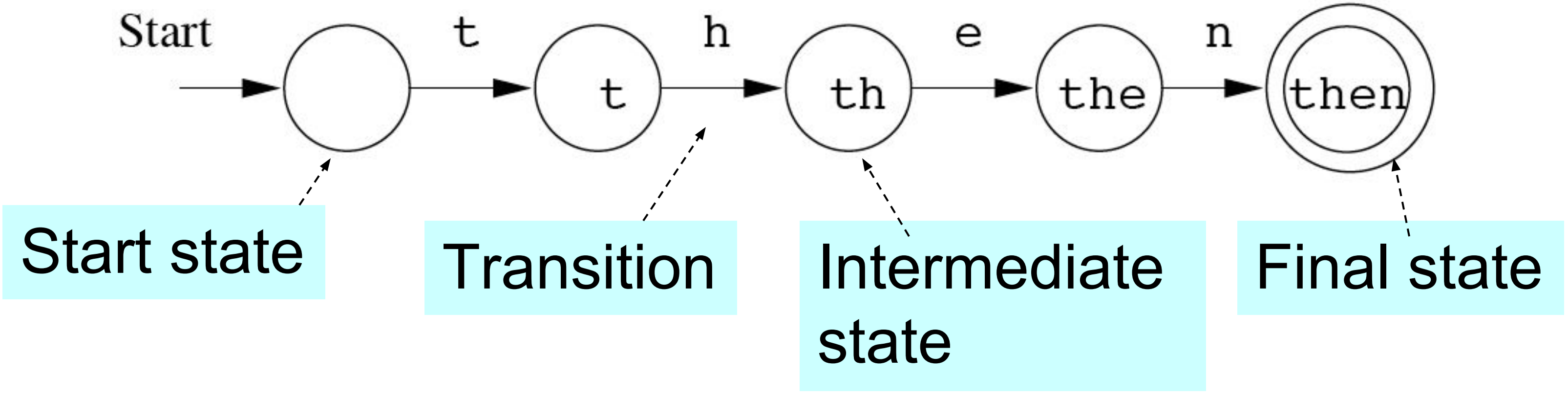
- Some Applications
  - Software for designing and checking the behavior of digital circuits
  - Lexical analyzer of a typical compiler
  - Software for scanning large bodies of text (e.g., web pages) for pattern finding
  - Software for verifying systems of all types that have a finite number of states (e.g., stock market transaction, communication/network protocol)

Finite Automata : Examples

- On/Off switch



- Modeling recognition of the word “*then*”



Structural expressions

- Grammars
- Regular expressions
  - E.g., unix style to capture city names such as “Palo Alto CA”:
    - [A-Z][a-z]*([ ][A-Z][a-z]*)*[ ][A-Z][A-Z]

Start with a letter

A string of other letters (possibly empty)

Other space delimited words (part of city name)

Should end w/ 2-letter state code

Structural expressions

- Grammars

E.g.

S->E+E

Where S is a statement of a language, E is any Expression in a language

- Regular expressions

E.g. identifier can be defined as a RE as

L(L/D)*

Where L=(A..Z, a…z)

D=(0…9)

Formal Proofs

Deductive Proofs

From the given statement(s) to a conclusion statement (what we want to prove)

- Logical progression by direct implications

Example for parsing a statement:

- "If $y \geq 4$, then $2^y \geq y^2$."

given *conclusion*

(there are other ways of writing this).

Example: Deductive proof

Let <u>Claim 1:</u> If $y \geq 4$, then $2^y \geq y^2$.

Let x be any number which is obtained by adding the squares of 4 positive integers.
<u>Claim 2:</u>
Given x and assuming that Claim 1 is true, prove that $2^x \geq x^2$

- <u>Proof:</u>
  1) Given: $x = a^2 + b^2 + c^2 + d^2$
  2) Given: $a \geq 1$, $b \geq 1$, $c \geq 1$, $d \geq 1$
  3) □ $a^2 \geq 1$, $b^2 \geq 1$, $c^2 \geq 1$, $d^2 \geq 1$ (by 2)
  4) □ $x \geq 4$ (by 1 & 3)
  5) □ $2^x \geq x^2$ (by 4 and Claim 1)

"implies" or *"follows"*

On Theorems, Lemmas and Corollaries

We typically refer to:

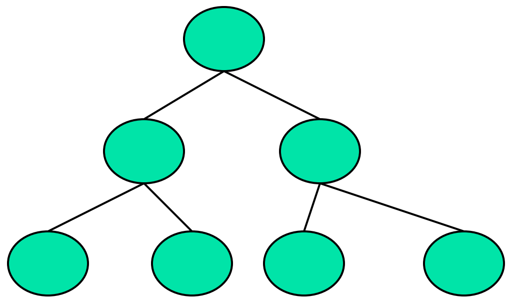
- A major result as a "**theorem**"
- An intermediate result that we show to prove a larger result as a "**lemma**"
- A result that follows from an already proven result as a "**corollary**"

An example:

Theorem: *The height of an n-node binary tree is at least floor(lg n)*

Lemma: *Level i of a perfect binary tree has* 2^i *nodes.*

Corollary: *A perfect binary tree of height h has* $2^{h+1}-1$ *nodes.*



Quantifiers

- *"For all"* or *"For every"*
  - Universal proofs
  - Notation= $\forall$
- *"There exists"*
  - Used in existential proofs
  - Notation= $\exists$
- Implication is denoted by =>
  - E.g., "IF A THEN B" can also be written as "A=>B"

Proving techniques

- By contradiction
  - Start with the statement contradictory to the given statement
  - E.g., To prove (A => B), we start with:
    - (A and ~B)
    - … and then show that could never happen

    What if you want to prove that “(A and B => C or D)”?

- By induction
  - (3 steps) Basis, inductive hypothesis, inductive step
- By contrapositive statement
  - If *A* then *B* ≡ If ~*B* then ~*A*

Proving techniques…

- By counter-example
  - Show an example that disproves the claim
- Note: There is no such thing called a "proof by example"!
  - So when asked to prove a claim, an example that satisfied that claim is *not* a proof

Different ways of saying the same thing

- “*If* H *then* C”:
  i. H *implies* C
  ii. *H => C*
  iii. C *if* H
  iv. H *only if* C
  v. *Whenever* H *holds*, C *follows*

“If-and-Only-If” statements

- “A if and only if B” (A <==> B)
  - *(if part)* if B then A (<=)
  - *(only if part)* A only if B (=>)
    (same as “if A then B”)
- “If and only if” is abbreviated as “iff”
  - i.e., “A iff B”
- <u>Example:</u>
  - <u>Theorem:</u> *Let x be a real number. Then floor of x = ceiling of x <u>if and only if</u> x is an integer.*
- Proofs for iff have two parts
  - One for the “if part” & another for the “only if part”

Summary

- Automata theory & a historical perspective
- Finite automata
- Alphabets, strings/words/sentences, languages
- Membership problem
- Proofs:
  - Deductive, induction, contrapositive, contradiction, counterexample
  - If and only if

Internet sources

1.https://www.eecs.wsu.edu/~ananth/CptS317/Lectures/index.htm

2. https://nptel.ac.in/courses/106/104/106104028/

3. https://nptel.ac.in/courses/106/104/106104148/