

Course 1

Introduction to Automata Theory



The structure and the content of the lecture is based on <http://www.eecs.wsu.edu/~ananth/CptS317/Lectures/index.htm>



What is Automata Theory?

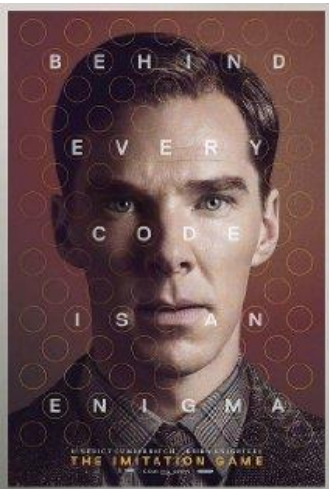
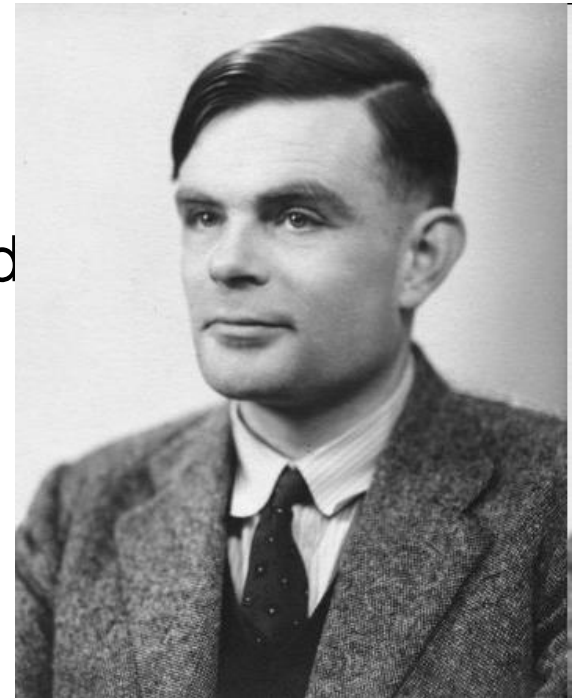
- *Study of abstract computing devices, or “machines”*
- **Automaton = an abstract computing device**
 - Note: A “device” need not even be a physical hardware!
- **A fundamental question in computer science:**
 - Find out what different models of machines can do and cannot do
 - The *theory of computation*
- Computability vs. Complexity

(A pioneer of automata theory)

Alan Turing (1912-1954)

- Father of Modern Computer Science
- English mathematician
- Studied abstract machines called **Turing machines** even before computers existed

Heard of the Turing test?





Theory of Computation: A Historical Perspective

1930s	<ul style="list-style-type: none">• Alan Turing studies Turing machines• Decidability• Halting problem
1940-1950s	<ul style="list-style-type: none">• “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

An **alphabet** is a set of symbols:

$\{0,1\}$

Or “**words**”

↓
Sentences are strings of symbols:

0,1,00,01,10,1,...

A **language** is a set of sentences:

$L = \{000,0100,0010,.. \}$

A **grammar** is a finite list of rules defining a language.

$S \longrightarrow 0A$

$B \longrightarrow 1B$

$A \longrightarrow 1A$

$B \longrightarrow 0F$

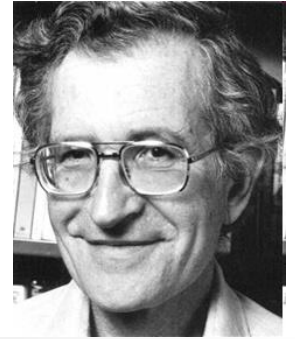
$A \longrightarrow 0B$

$F \longrightarrow \epsilon$

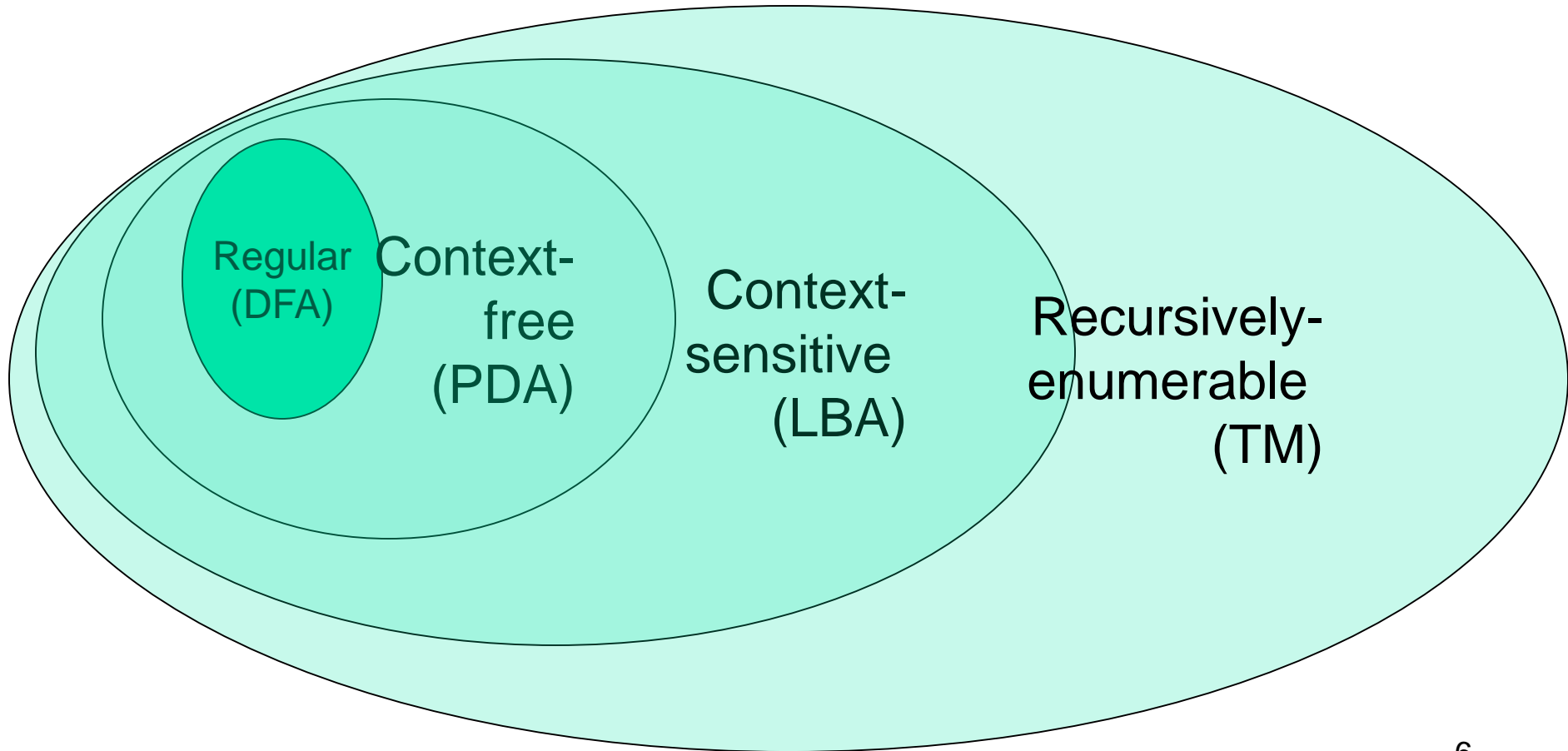
- Languages: “A language is a collection of sentences of finite length all constructed from a finite alphabet of symbols”
- Grammars: “A grammar can be regarded as a device that enumerates the sentences of a language” - nothing more, nothing less
- N. Chomsky, *Information and Control*, Vol 2, 1959



The Chomsky Hierarchy



- A containment hierarchy of classes of formal languages



The Central Concepts of Automata Theory





Alphabet

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

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



Strings

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

- **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., $x = 010100$ $|x| = 6$
 - $x = 01\ \varepsilon\ 0\ \varepsilon\ 1\ \varepsilon\ 00\ \varepsilon$ $|x| = ?$
- xy = concatenation 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 said to be a language over alphabet Σ , only if $L \subseteq \Sigma^$*

→ this is because Σ^* is the set of all strings (of all possible length including 0) over the given alphabet Σ

Examples:

1. Let L be *the* language of all strings consisting of n 0's followed by n 1's:

$$L = \{\epsilon, 01, 0011, 000111, \dots\}$$

2. Let L be *the* language of all strings of with equal number of 0's and 1's:

$$L = \{\epsilon, 01, 10, 0011, 1100, 0101, 1010, 1001, \dots\}$$

→
Canonical ordering of strings in the language

Definition: \emptyset denotes the Empty language

- Let $L = \{\epsilon\}$; Is $L = \emptyset$?

NO



The Membership Problem

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

Example:

Let $w = 100011$

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

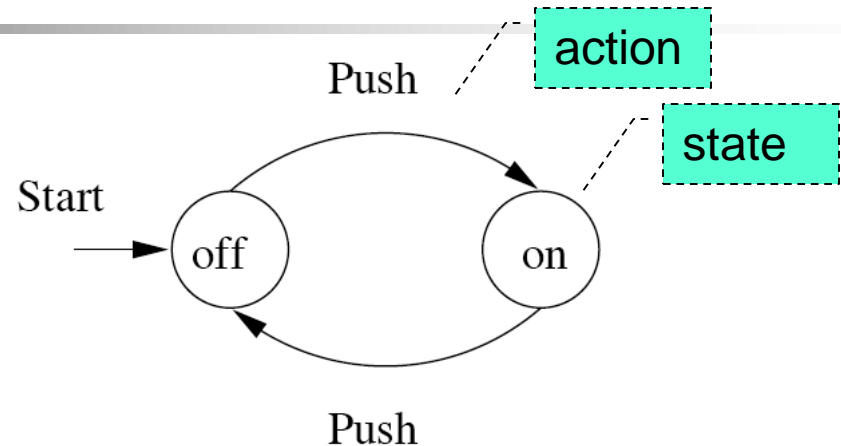


Finite Automata

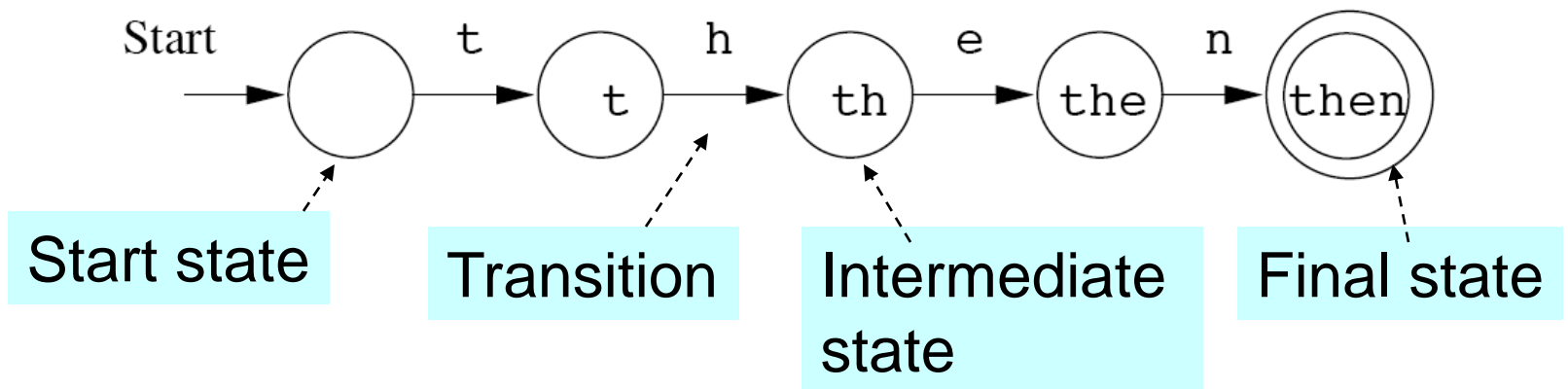
- Some Applications
 - Software for designing and checking the behavior of digital circuits
 - Lexical analyzer of a typical compiler
 - Software for scanning large bodies of text (e.g., web pages) for pattern finding
 - Software for verifying systems of all types that have a finite number of states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



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).

On Theorems, Lemmas and Corollaries

We typically refer to:

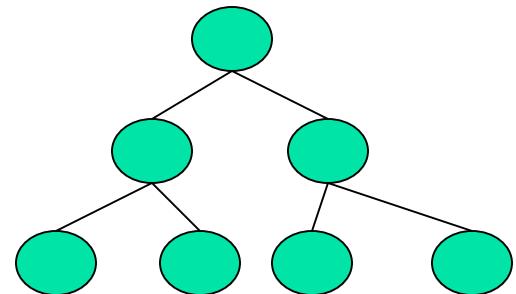
- A major result as a “**theorem**”
- An intermediate result that we show to prove a larger result as a “**lemma**”
- A result that follows from an already proven result as a “**corollary**”

An example:

Theorem: *The height of an n -node binary tree is at least $\text{floor}(\lg n)$*

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 \Rightarrow

- E.g., “IF A THEN B” can also be written as “ $A \Rightarrow B$ ”



Proving techniques

- **By contradiction**

- Start with the statement contradictory to the given statement
- E.g., To prove $(A \Rightarrow B)$, we start with:
 - $(A \text{ and } \sim B)$
 - ... and then show that could never happen

What if you want to prove that “ $(A \text{ and } B \Rightarrow C \text{ or } D)$ ”?

- **By induction**

- (3 steps) Basis, inductive hypothesis, inductive step

- **By contrapositive statement**

- If A then B \equiv If $\sim B$ then $\sim A$



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. $H \Rightarrow C$
 - iii. *C if H*
 - iv. *H only if C*
 - v. *Whenever H holds, C follows*

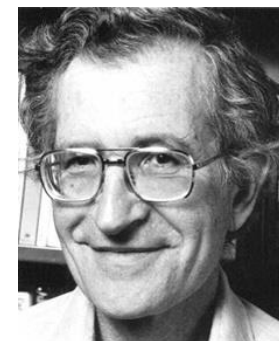


“If-and-Only-If” statements

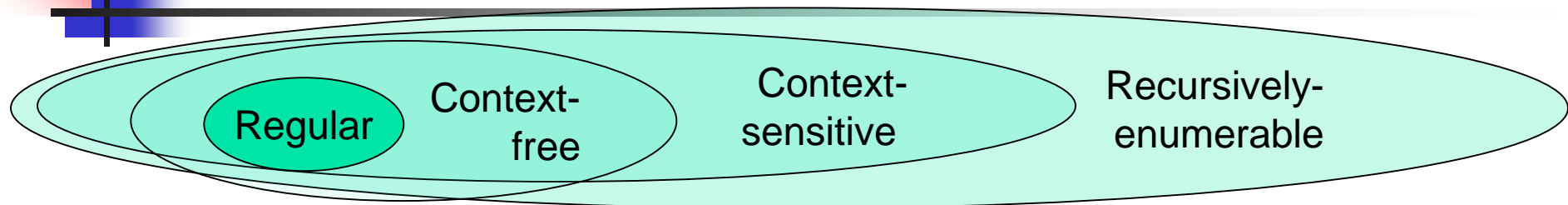
- “A if and only if B” ($A \iff B$)
 - (if part) if B then A (\implies)
 - (only if part) A only if B (\implies)
(same as “if A then B”)
- “If and only if” is abbreviated as “iff”
 - i.e., “A iff B”
- Example:
 - Theorem: *Let x be a real number. Then floor of x = ceiling of x if and only if x is an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



The Chomsky Hierarchy



The Chomsky Hierarchy



Grammar	Languages	Automaton	Production Rules
Type-0	Recursively enumerable \mathcal{L}_0	Turing machine	$\alpha \rightarrow \beta$
Type-1	Context sensitive \mathcal{L}_1	Linear-bounded non-deterministic Turing machine	$\alpha A \beta \rightarrow \alpha \gamma \beta$
Type-2	Context-free \mathcal{L}_2	Non-deterministic push down automaton	$A \rightarrow \gamma$
Type-3	Regular \mathcal{L}_3	Finite state automaton	$A \rightarrow a$ and $A \rightarrow aB$



The Chomsky Hierarchy (cont'd)

Classification using the structure of their rules:

- **Type-0 grammars**: there are no restriction on the rules;
- **Type-1 grammars/Context sensitive grammars**: the rules for this type have the next form:

$$uAv \rightarrow upv, u, p, v \in V_G^*, p \neq \lambda, A \in V_N$$

- **Type-2 grammars/Context free grammars**: the rules for this type are of the form:

$$A \rightarrow p, p \in V_G^*, A \in V_N$$

- **Type-3 grammars/regular grammars**: the rules for this type have one of the next two forms:

Cat. I rules
Cat. II rules

$$A \rightarrow Bp \\ C \rightarrow q$$

or

$$A \rightarrow pB \\ C \rightarrow q$$

$$A, B, C \in V_N, p, q \in V_T^*$$

Examples: see whiteboard



Summary

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