Introduction to Automata Theory

Reading: Chapter 1

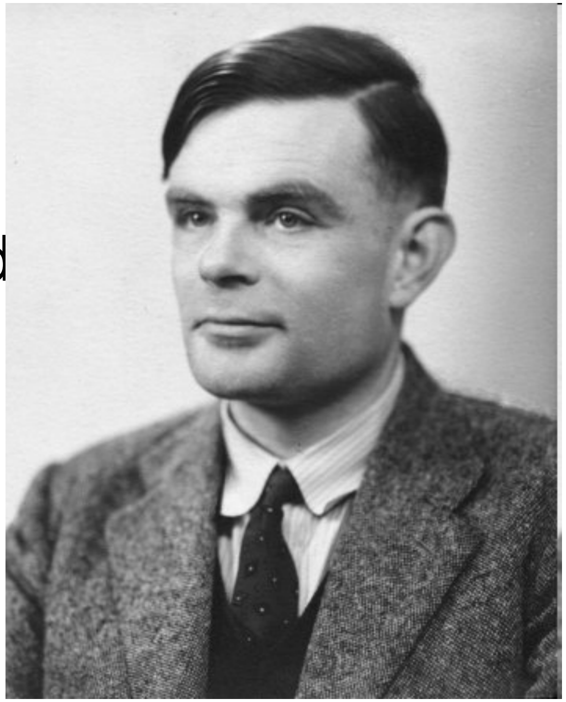
What is Automata Theory?

- *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

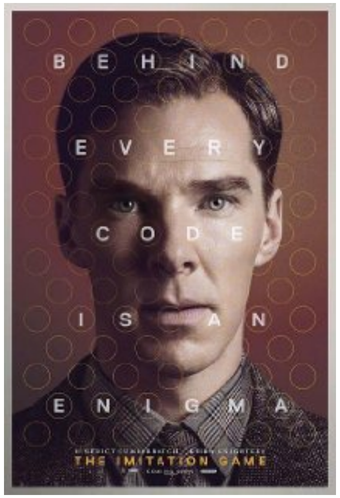
(A pioneer of automata theory)

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	• 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**"

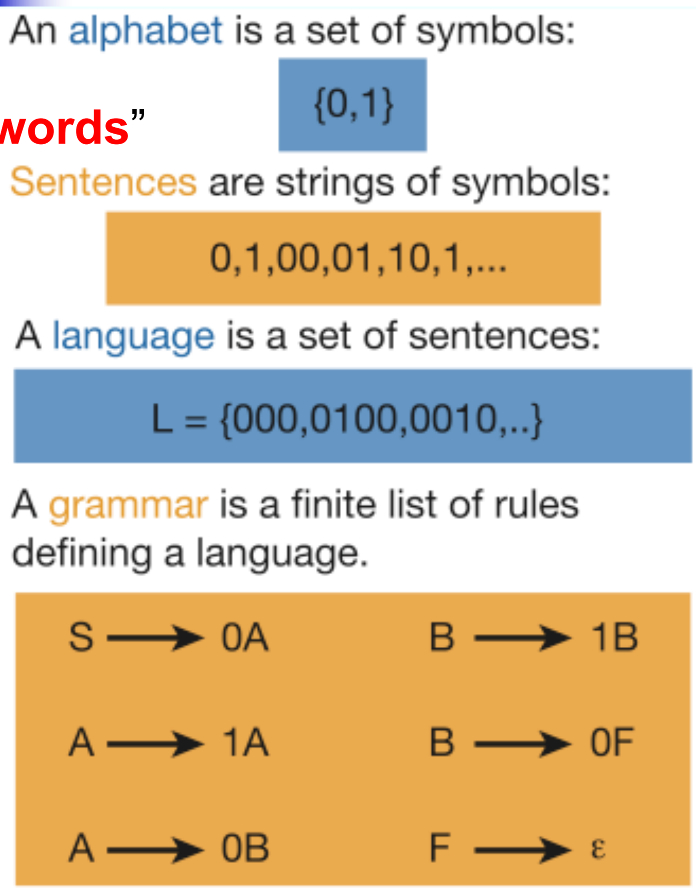
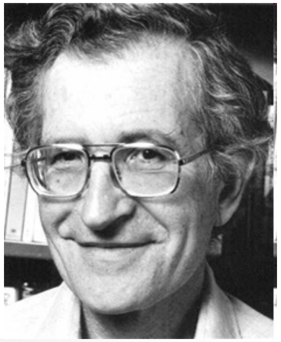


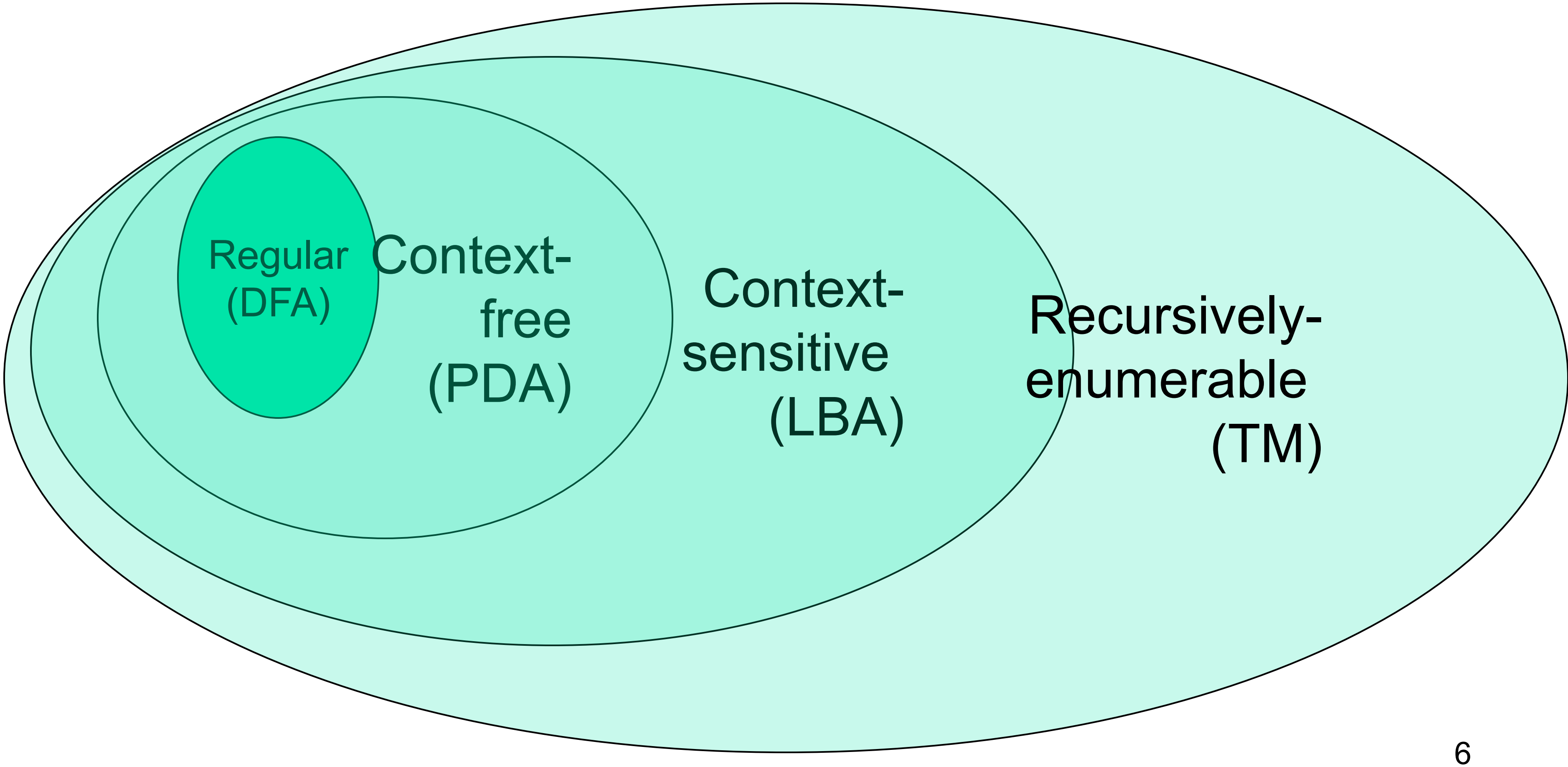
Image source: Nowak et al. Nature, vol 417, 2002

- 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
- *N. Chomsky, Information and Control, Vol 2, 1959*

The Chomsky Hierachy



- 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 ∑ (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 $\sum$ be an alphabet.

- $\sum^{k}$ = the set of all strings of length k
- $\sum^{*} = \sum^{0} \cup \sum^{1} \cup \sum^{2} \cup \ldots$
- $\sum^{+} = \sum^{1} \cup \sum^{2} \cup \sum^{3} \cup \ldots$

Languages

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

➔ 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,…}

   Canonical ordering of strings in the language

Definition: Ø denotes the Empty language

- Let L = {ε}; Is L=Ø? 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

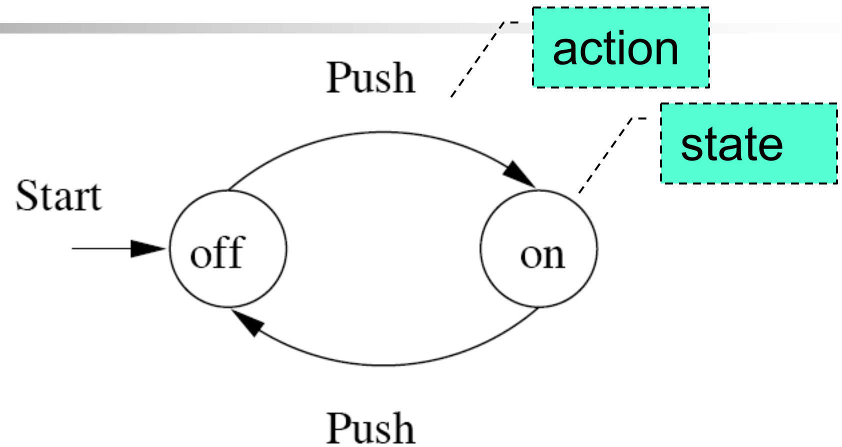
Q) Is $w \in$ the language of strings with equal number of 0s and 1s?

Finite Automata

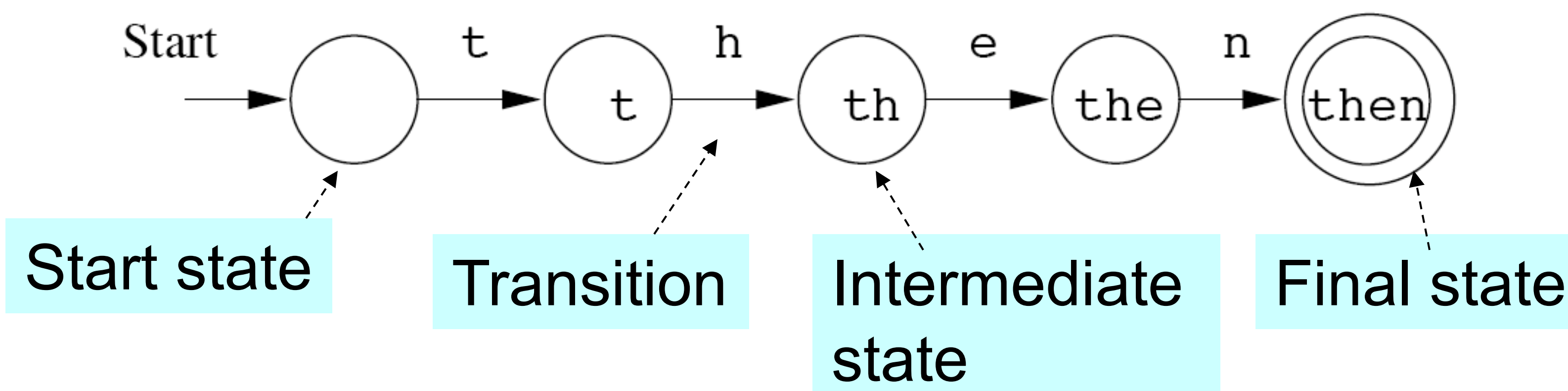
- Some Applications
  - Software for designing and checking the behavior of digital circuits
  - Lexical analyzer of a typical compiler
  - Software for scanning large bodies of text (e.g., web pages) for pattern finding
  - Software for verifying systems of all types that have a finite number of states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

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:

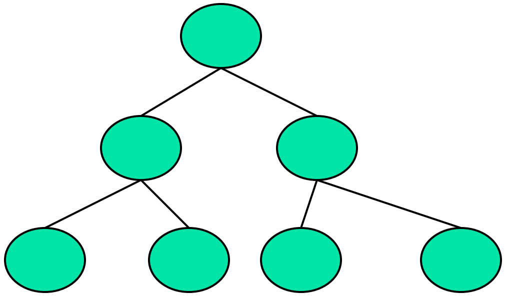
- A major result as a "**theorem**"
- An intermediate result that we show to prove a larger result as a "**lemma**"
- A result that follows from an already proven result as a "**corollary**"

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”:
  1. H *implies* C
  2. *H => C*
  3. C *if* H
  4. H *only if* C
  5. *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"
- 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"

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

- Read chapter 1 for more examples and exercises