Introduction to Automata Theory

Purpose and motivation

- What are the mathematical properties of computer hardware and software?
- What is a **computation** and what is an **algorithm**? Can we give rigorous mathematical definitions of these **notions**?
- What are the limitations of computers? Can “everything” be computed?

Purpose: Develop formal mathematical models of computation that reflect real-world computers.

Alan Turing (1912-1954)

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

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

Theory of Computation, nowadays

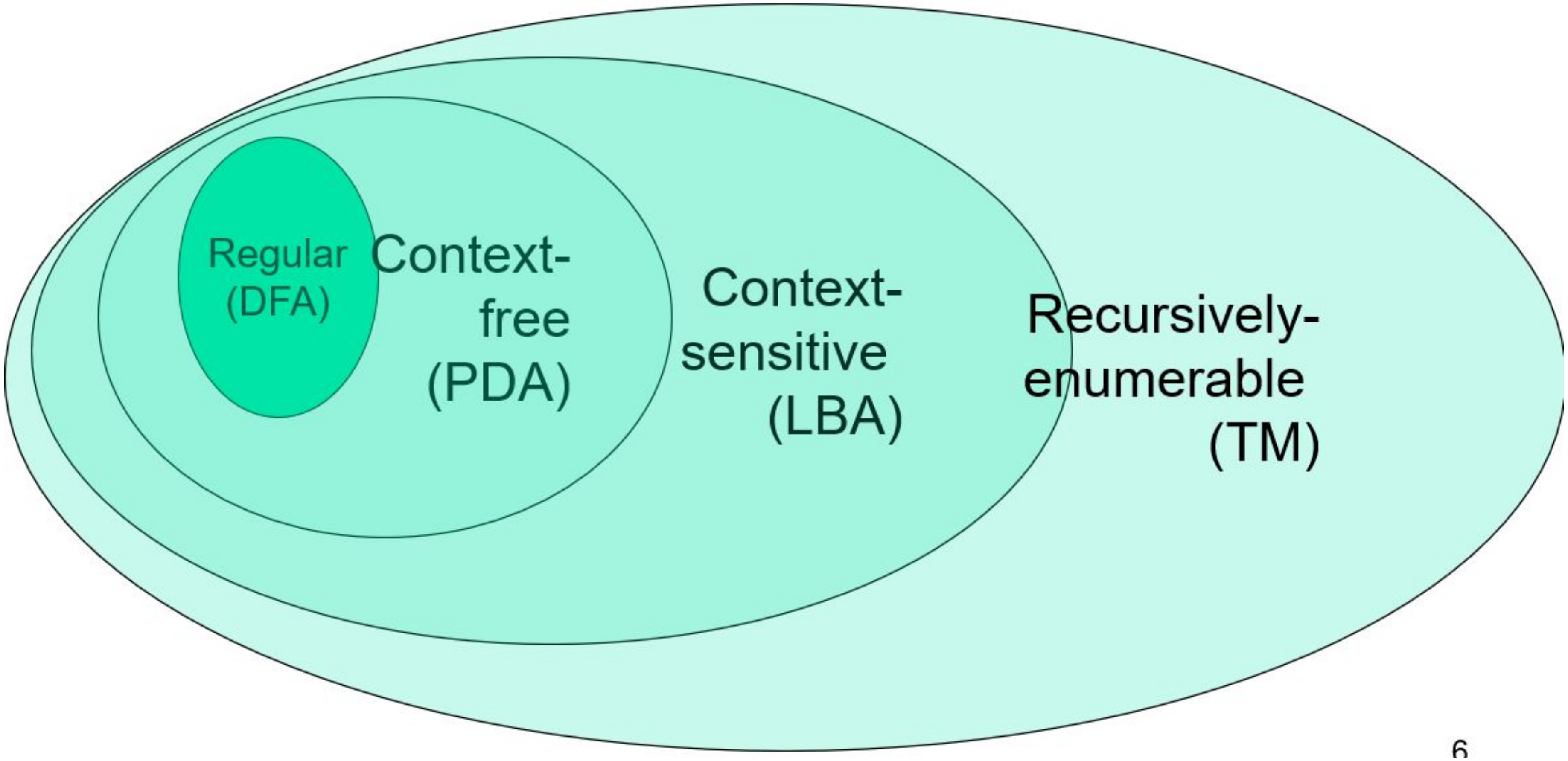
- Nowadays, the Theory of Computation can be divided into the following three areas:
- **Complexity theory**: What makes some problems computationally **hard** and other problems **easy**?
- **Computability theory**: Classify problems as being **solvable** or **unsolvable**.
- **Automata theory**: It deals with definitions and properties of different types of **computation models**.

Computation models

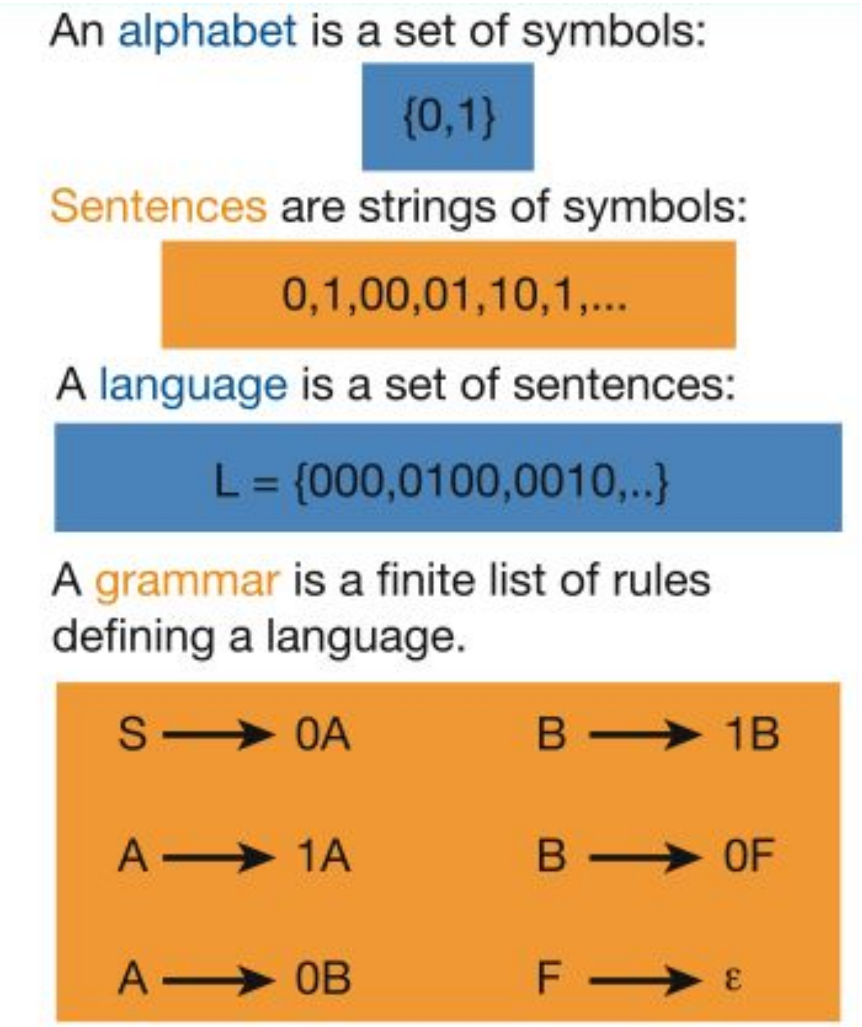
- **Finite Automata**: These are used in text processing, compilers, and hardware design.
- **Context-Free Grammars**: These are used to define programming languages and in Artificial Intelligence.
- **Turing Machines**: These form a simple abstract model of a “real” computer, such as our PC at home.

The Chomsky hierarchy

- A hierarchy of classes of formal languages



Languages & Grammars



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

Reference: N. Chomsky, Information and Control, Vol 2, 1959

Rudimentary Elements 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 = concatenation* 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?

The membership problem

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

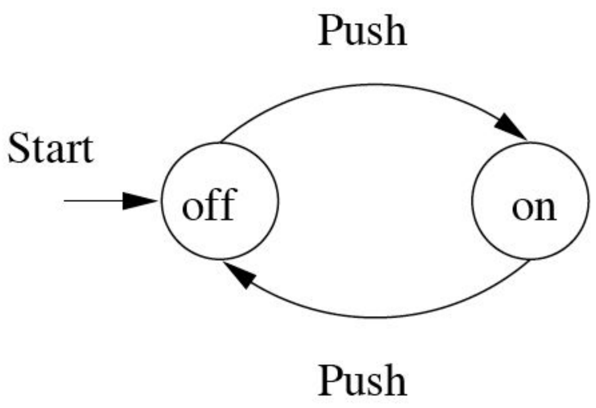
<u>Example:</u>

Let w = 100011

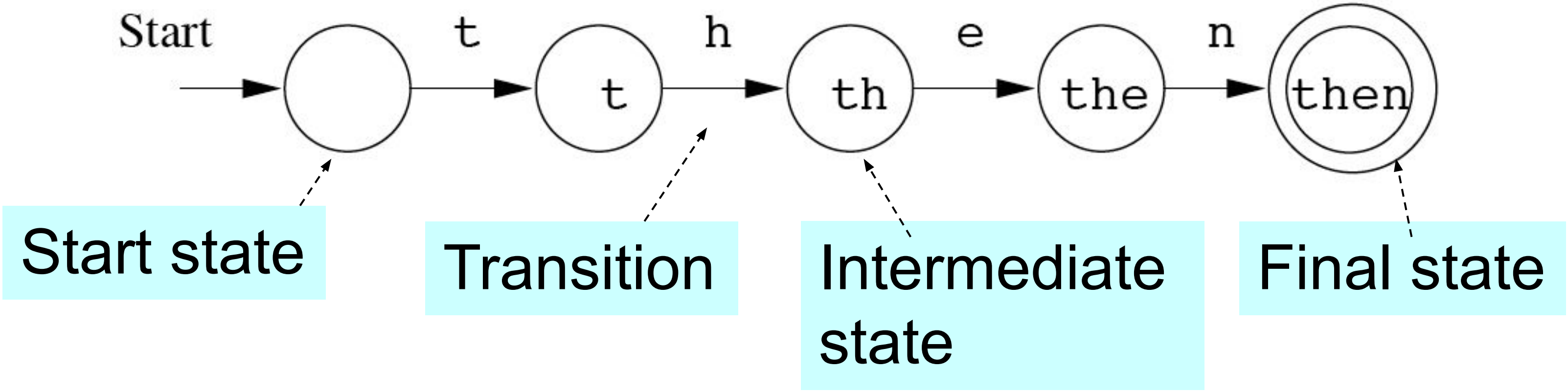
Q) Is w $\in$ the language of strings with equal number of 0s and 1s?

Finite Automata : Examples

- On/Off switch



- Modeling recognition of the word "**then**"



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).