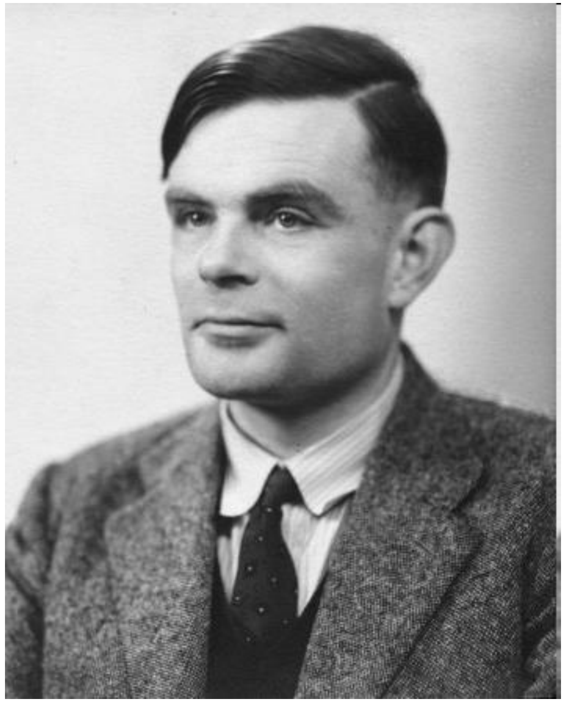
WHAT IS AUTOMATA THEORY?

- *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)

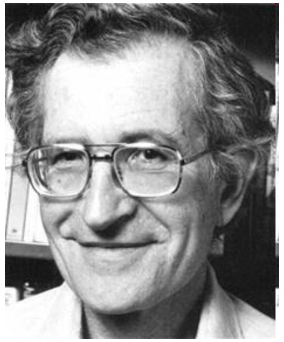
- Father of Modern Computer Science
- English mathematician
- Studied abstract machines called ***Turing machines*** even before computers existed
- Heard of the Turing test?



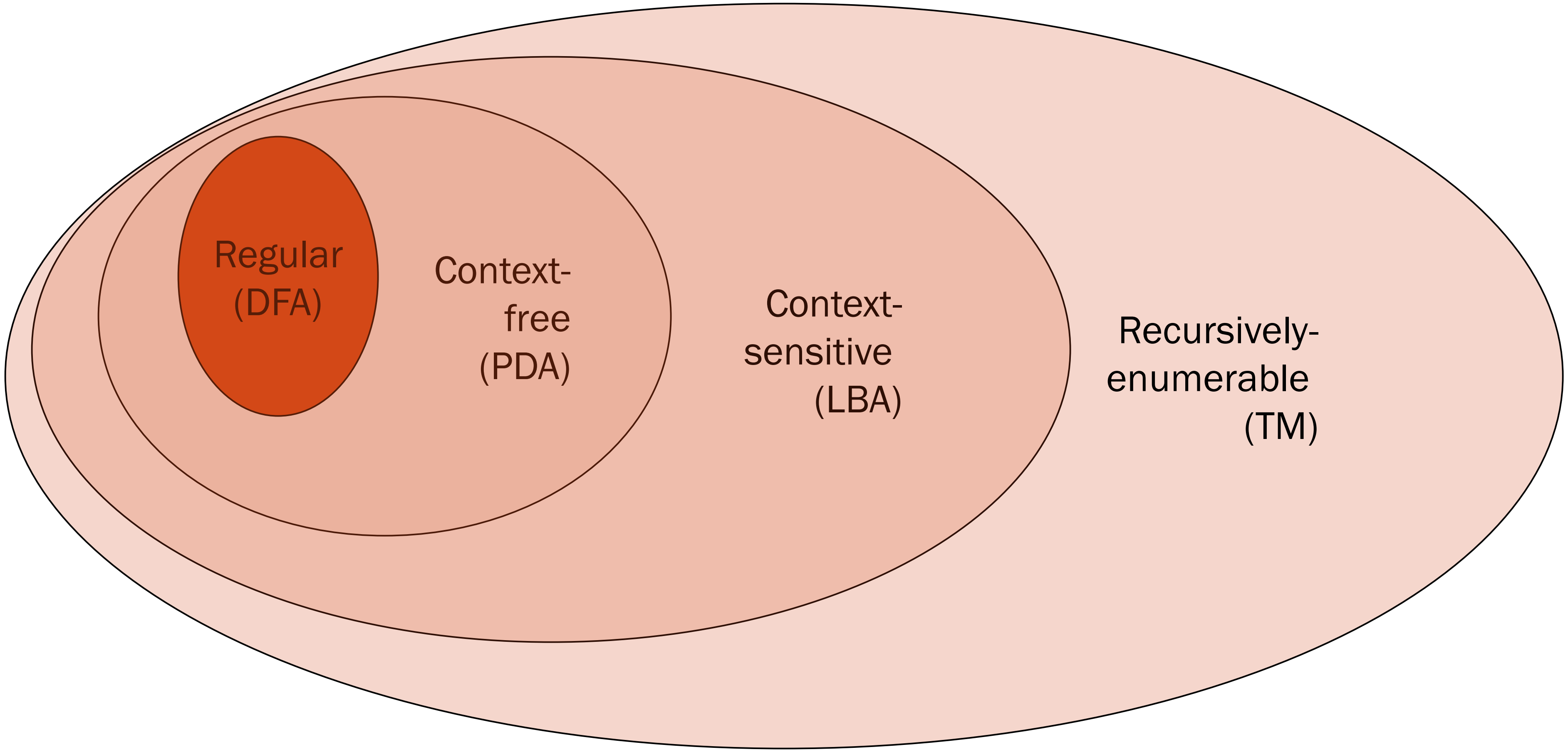
THEORY OF COMPUTATION: A HISTORICAL PERSPECTIVE

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

THE CHOMSKY HIERACHY



- A containment hierarchy of classes of formal languages



BRIEF HISTORY OF THEORY OF COMPUTATION

- 1936 Alan Turing invented the *Turing machine*, and proved that there exists an *unsolvable problem*.
- 1940's Stored-program computers were built.
- 1943 McCulloch and Pitts invented *finite automata*.
- 1956 Kleene invented *regular expressions* and proved the equivalence of regular expression and finite automata.

HISTORY OF THEORY OF COMPUTATION CONTD…

- 1956 Chomsky defined *Chomsky hierarchy*, which organized languages recognized by different automata into hierarchical classes.
- 1959 Rabin and Scott introduced *nondeterministic finite automata* and proved its equivalence to (deterministic) finite automata.
- 1950's-1960's More works on languages, grammars, and compilers

HISTORY OF THEORY OF COMPUTATION

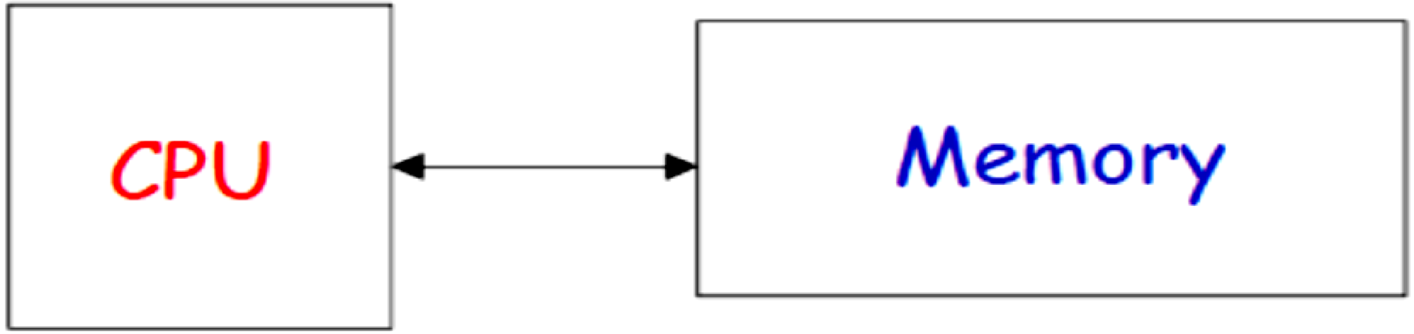
- 1965 Hartmantis and Stearns defined *time complexity*, and Lewis, Hartmantis and Stearns defined *space complexity*.
- 1971 Cook showed the first *NP-complete problem*, the *satisfiability* problem.
- 1972 Karp Showed many other NP-complete problems.

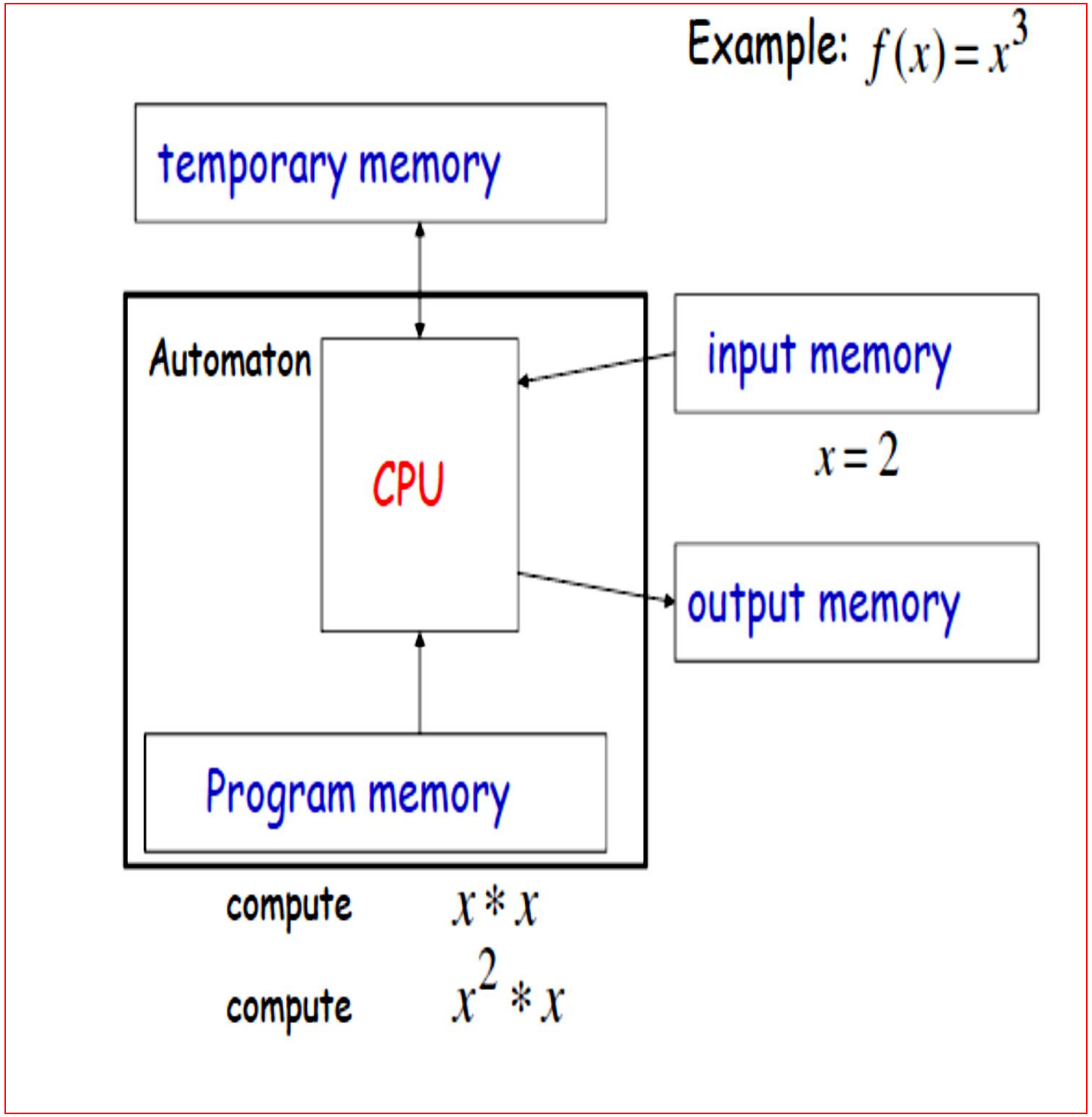
WHAT IS COMPUTATION ?

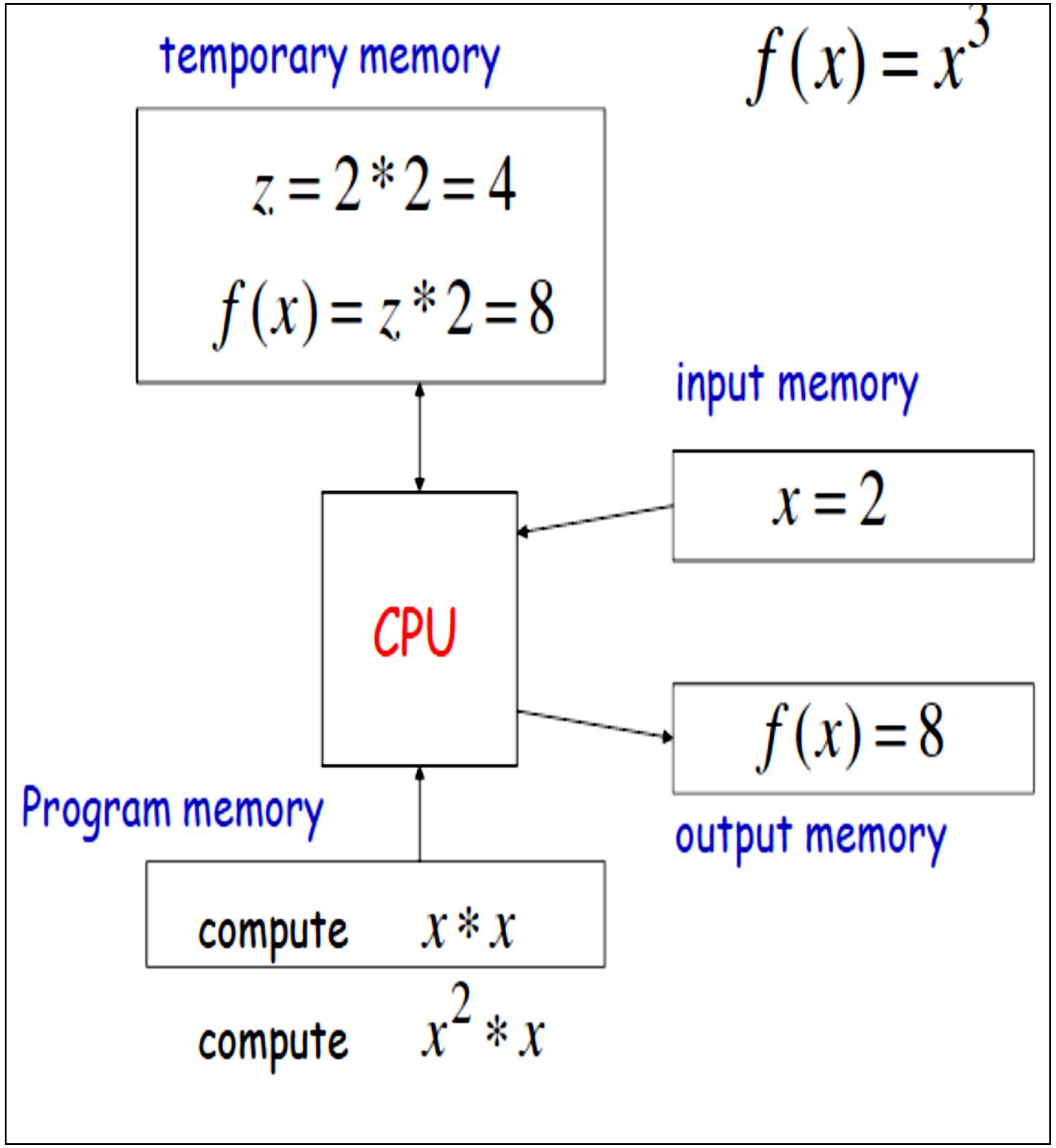
- Computation is nothing but any task that can be performed by a computer or calculator.
- **TOC:- The study of mathematical representation of computing system and their capabilities is TOC.**
  - Sequence of mathematical operations ?
    - What are, and are not, mathematical operations?
  - Sequence of well-defined operations
    - How many operations ?
      - The fewer, the better.
    - Which operations ?
      - The simpler, the better.

COMPUTATION

Solving problems through the mechanical, preprogrammed execution of a series of small, unambiguous steps.




Example: $f(x)=x^3$
temporary memory
Automaton
CPU
input memory
$x=2$
output memory
Program memory
compute $x*x$
compute x^2*x


temporary memory
$f(x)=x^3$
$z=2*2=4$
$f(x)=z*2=8$
input memory
$x=2$
CPU
$f(x)=8$
Program memory
output memory
compute $x*x$
compute x^2*x

WHAT DO WE STUDY IN THEORY OF COMPUTATION ?

- What is computable, and what is not ?
- What a computer can and can not do
- Can you make your program more efficient?

- Basis of
  - Algorithm analysis
  - Complexity theory

WHAT DO WE STUDY IN COMPLEXITY THEORY ?

- What is easy, and what is difficult, to compute ?

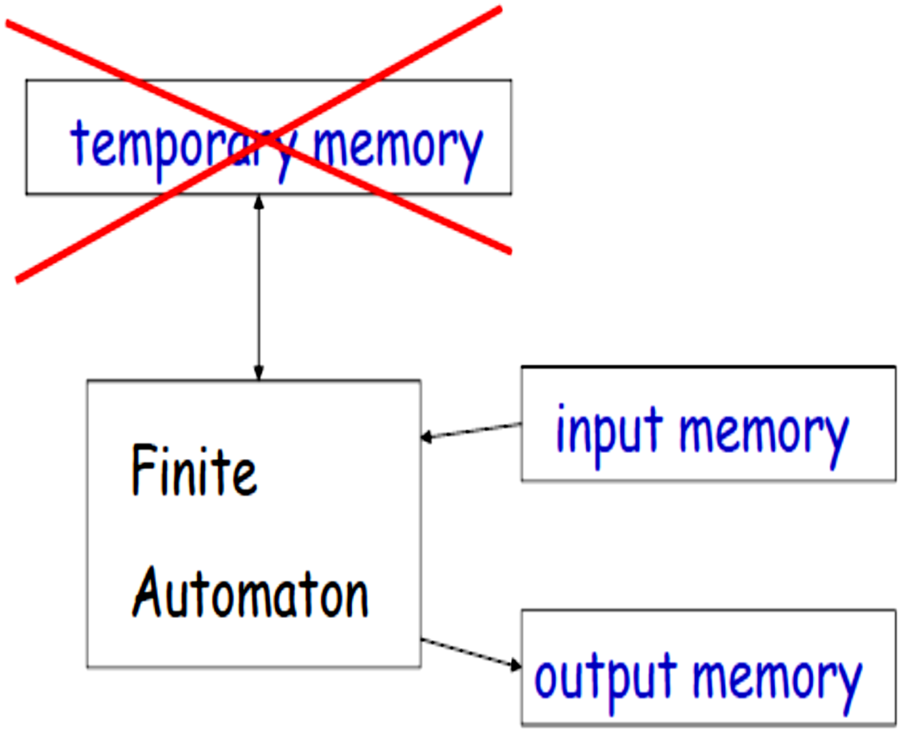
- What is easy, and what is hard for computers to do?

Different Kinds of Automata

Automata are distinguished by the temporary memory

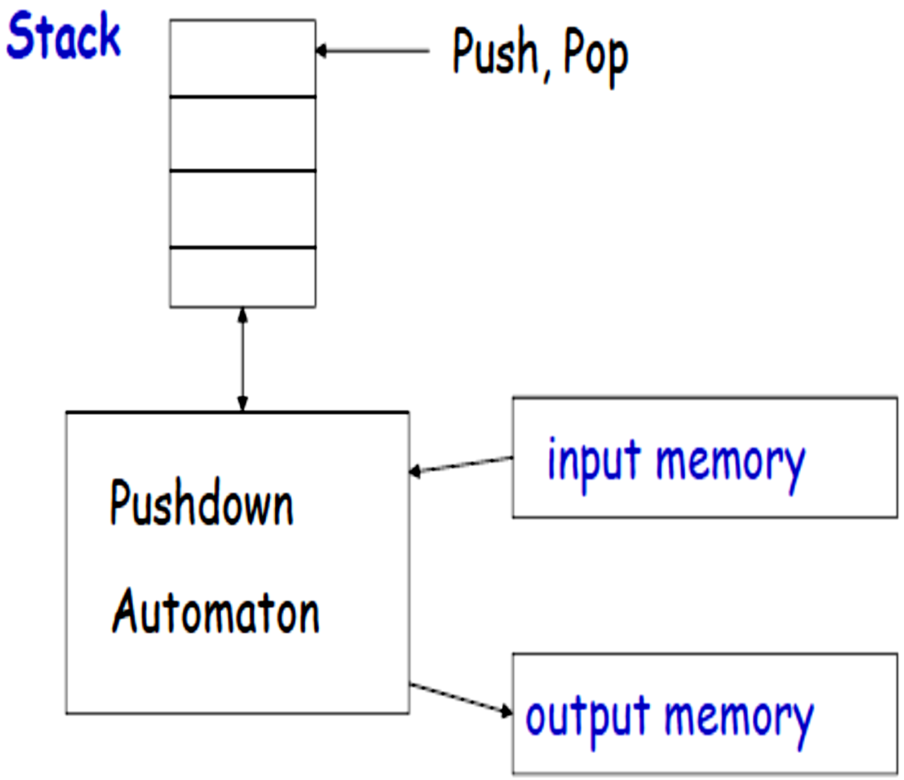
- **Finite Automata:** no temporary memory
- **Pushdown Automata:** stack
- **Turing Machines:** random access memory

Finite Automaton



Example: String Search, Decision Making, etc

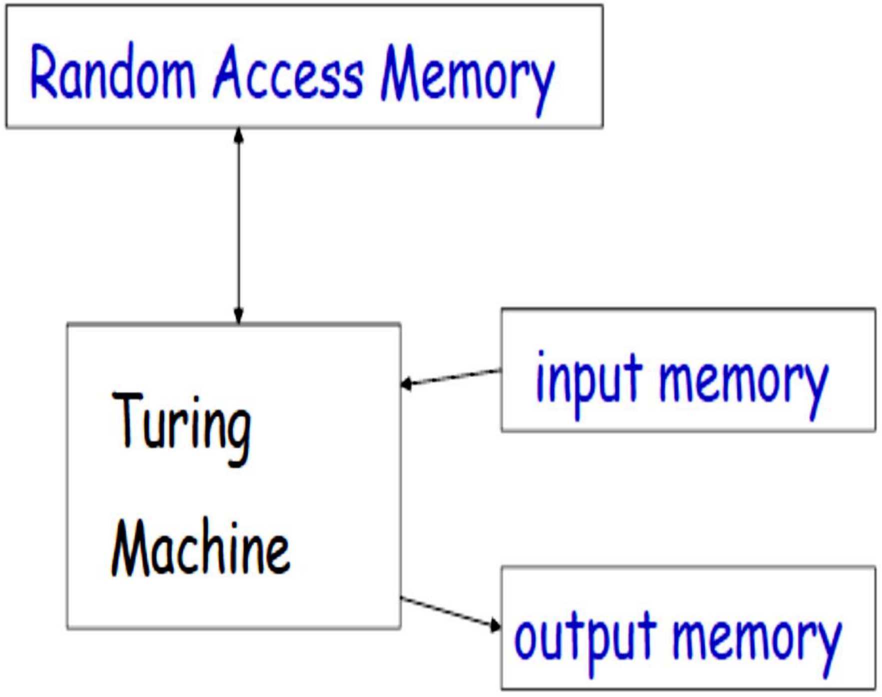
Pushdown Automaton



Example: Compilers for Programming Languages

(medium computing power)

Turing Machine



Examples: Any Algorithm (highest computing power)

THE CENTRAL CONCEPTS OF AUTOMATA THEORY

LANGUAGES & GRAMMARS

- A system suitable for expressing certain ideas, facts, or concepts, which includes a set of symbols and rules to manipulate them.
- Types -
  - Natural languages
  - Programming / Computer languages
  - Formal languages
    - (letters, words, and sentences)

- **Formal languages** : is a set of strings of symbols that is constrained by some specific rules.
- **Language structure** - the decision of **whether a given set of words constitutes a valid sentence?**

LANGUAGES & GRAMMARS

- **Symbols**
  - User defined entity (indivisible objects)
  - Symbols are the atoms of the world of languages.
- **Alphabets Σ**
  - An alphabet is a finite, nonempty set of fundamental symbols.
  - Standardized set of letters
  - E.g. Roman alphabet, Binary alphabet
- **Strings**
  - A string over an alphabet Σ is a finite sequence of concatenated symbols **from** Σ.
  - String over some alphabet should contain all the symbols from the alphabet.
- Those strings that are permissible in the language are called **words**.

Alphabet → String → Language

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.

$S \longrightarrow 0A$	$B \longrightarrow 1B$
$A \longrightarrow 1A$	$B \longrightarrow 0F$
$A \longrightarrow 0B$	$F \longrightarrow \varepsilon$

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}
  - …

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
- $\sum^* = \sum^0 \cup \sum^1 \cup \sum^2 \cup \dots$
- $\sum^+ = \sum^1 \cup \sum^2 \cup \sum^3 \cup \dots$

Powers of Alphabets:

$\Sigma^k = \{ w \mid w \text{ is a string over } \Sigma \text{ and } |w| = k\}$.

- For any alphabet, Σ^0 - all strings of length zero. $\Sigma^0 = \{e\}$
- For the binary alphabet $\{0, 1\}$

$\Sigma^0 = \{e\}$

$\Sigma^1 = \{0, 1\}$

$\Sigma^2 = \{00, 01, 10, 11\}$

$\Sigma^3 = \{000, 001, 010, 011, 100, 110, 111\}$

POWERS OF AN ALPHABET

Closure of sets
The * Operation
Kleene Closure

Σ^* : the set of all possible strings from alphabet Σ

$\Sigma = \{ a, b \}$

$\Sigma^* = \{ \varepsilon, a, b, aa, ab, ba, bb, aaa, aab, abab, \ldots \}$

The + Operation
Positive Closure

Σ^+ :
The set of all possible strings from alphabet Σ except ε

$\Sigma^+ = \Sigma^+ - \varepsilon$

$\Sigma = \{ a, b \}$

$\Sigma^* = \{ \varepsilon, a, b, aa, ab, ba, bb, aaa, aab, abab, \ldots \}$

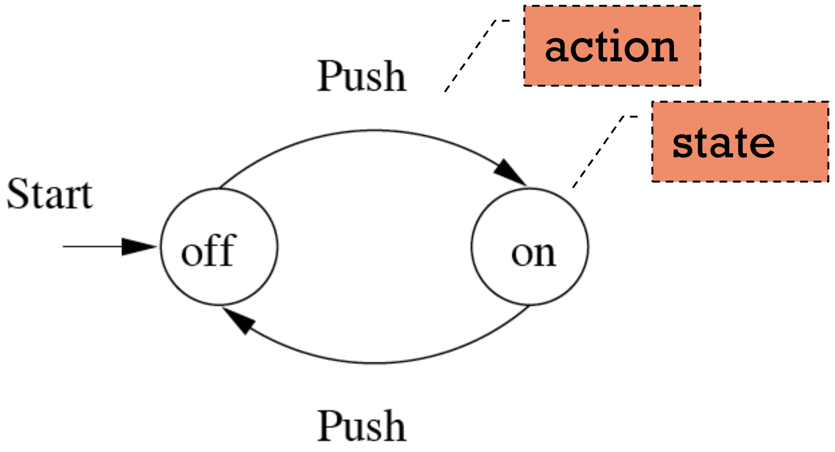
$\Sigma^+ = \{ a, b, aa, ab, ba, bb, aaa, aab, abab, \ldots \}$

FINITE AUTOMATA

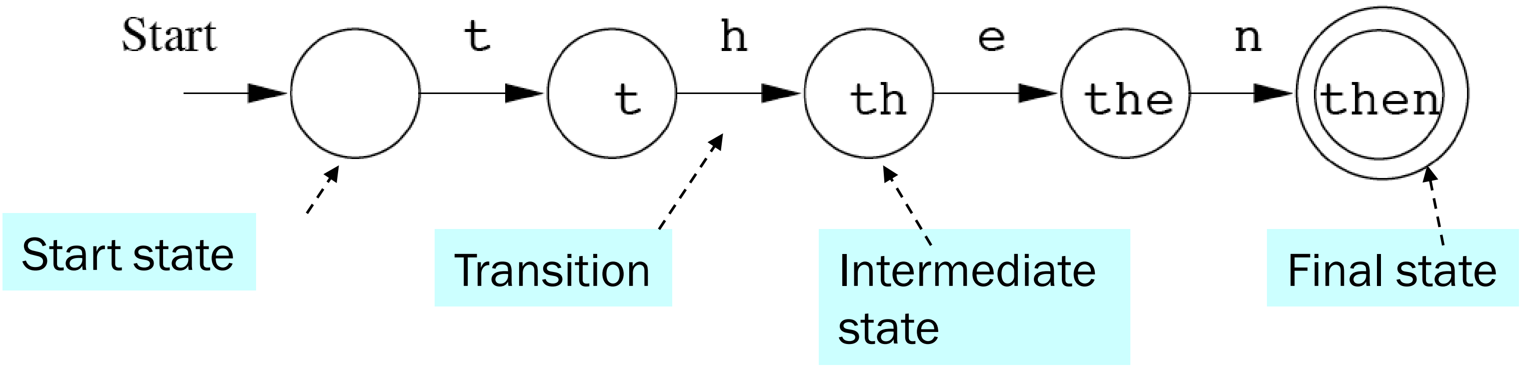
- Some Applications
  - Software for designing and checking the behavior of digital circuits
  - Lexical analyzer of a typical compiler
  - Software for scanning large bodies of text (e.g., web pages) for pattern finding
  - Software for verifying systems of all types that have a finite number of states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

FINITE AUTOMATA : EXAMPLES

- On/Off switch



- Modeling recognition of the word "*then*



LANGUAGES

Symbols : Letters and digits are examples of frequently used symbols.

Eg: A..Z, a..z, 0..9, some special characters.

The finite set of symbols forms the alphabet ($\sum$).

Word : A word or string is a finite sequence of symbols.

Language : A set of strings of symbols from the alphabet.

TYPES OF MACHINES

- Basic Machine (Combinatorial Machine).

A machine in which output at any instant of time depends only on the current input.

Eg: Basic logic gates.

- Finite State Machine (Sequential Machine).

A machine in which ouput depends on not only the current input but also on the current state of the machine.

In general, a state machine is any device that stores the status of something at a given time and can operate on input to change the status and/or cause an action or output to take place for any given change.

- Turing Machine : e.g. Computer.

FINITE STATE MACHINE

An **FSM is represented by a pair of functions, namely:**

- Machine function: MF: $I \times S \rightarrow O$
- State transition function: STF : $I \times S \rightarrow S$
- Where S:Finite set of internal states of the machine
- I : Finite set of input symbols (or input alphabet)
- O: Finite set of output symbols (or output alphabet)