Welcome to Class CSC257

Theory of Computation (CSC257)

Today's Topic

- Course Description
  - Introduction to Automata Theory
  - Why Study Automata?
  - Basic Concepts of Automata Theory

Course Description in Brief

- **Course Description:** This course includes,
  - **Finite State Machines and their languages.**
  - **Details of finite automata, regular expressions, context free grammars.**
  - **Design of the Push-down automata and Turing Machines.**
  - **Basics of Undecidability and Intractability.**

- **Course Objectives: The main objective of the course is to**
  - **Introduce concepts of the models of computation and formal language approach to computation.**
  - The general objectives of this course are to,
    - **Introduce concepts in automata theory and theory of computation,**
    - **Design different finite state machines and grammars and recognizers for different formal languages.**
    - **Identify different formal language classes and their relationships**
    - **Determine the decidability and intractability of computational problems.**

Course Contents

Course Contents:

Unit I: Basic Foundations (3 Hrs.)

- Mathematical Concepts, Complexity and Computability,
- Basic terminologies in TOC.

Unit II: Introduction to Finite Automata (8 Hrs.)

Unit III: Regular Expressions (6 Hrs.)

Unit IV: Context Free Grammar (9 Hrs.)

Unit V: Push Down Automata (7 Hrs.)

Unit VI: Turing Machines (10 Hrs.)

Unit VII: Undecidability and Intractability (5 Hrs.)

Laboratory Works

- **Design and implementation of DFA, NFA, PDA, and TM.**
- **Students are highly recommended to construct Tokenizers/ Lexers over/for some language.**
- **Students are advised to use regex and Perl (for using regular expressions), or any other higher level language for the laboratory works.**

Text Books:

1. John E. Hopcroft, Rajeev Motwani, Jeffrey D. Ullman, Introduction to Automata Theory, Language and Computation 3rd Edition, Pearson - Addison-Wesley.

Reference Books:

1. **Harry R. Lewis and Christos H. Papadimitriou, Elements of the Theory of Computation, 2nd Edition, Prentice Hall.**
2. **Michael Sipser, Introduction to the Theory of Computation, 3rd Edition, Thomson Course Technology**
3. **Efim Kinber, Carl Smith, Theory of Computing: A Gentle introduction, Prentice- Hall.**
4. **John Martin, Introduction to Languages and the Theory of Computation, 3rd Edition, Tata McGraw Hill.**

Note: You can use any other books related to TOC or Automata available in market/library/online resources for further reference

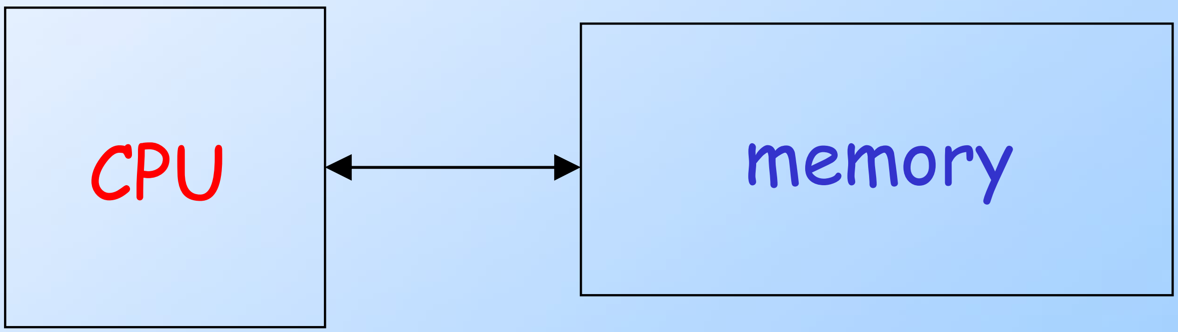
Basic Concepts of Automata Theory

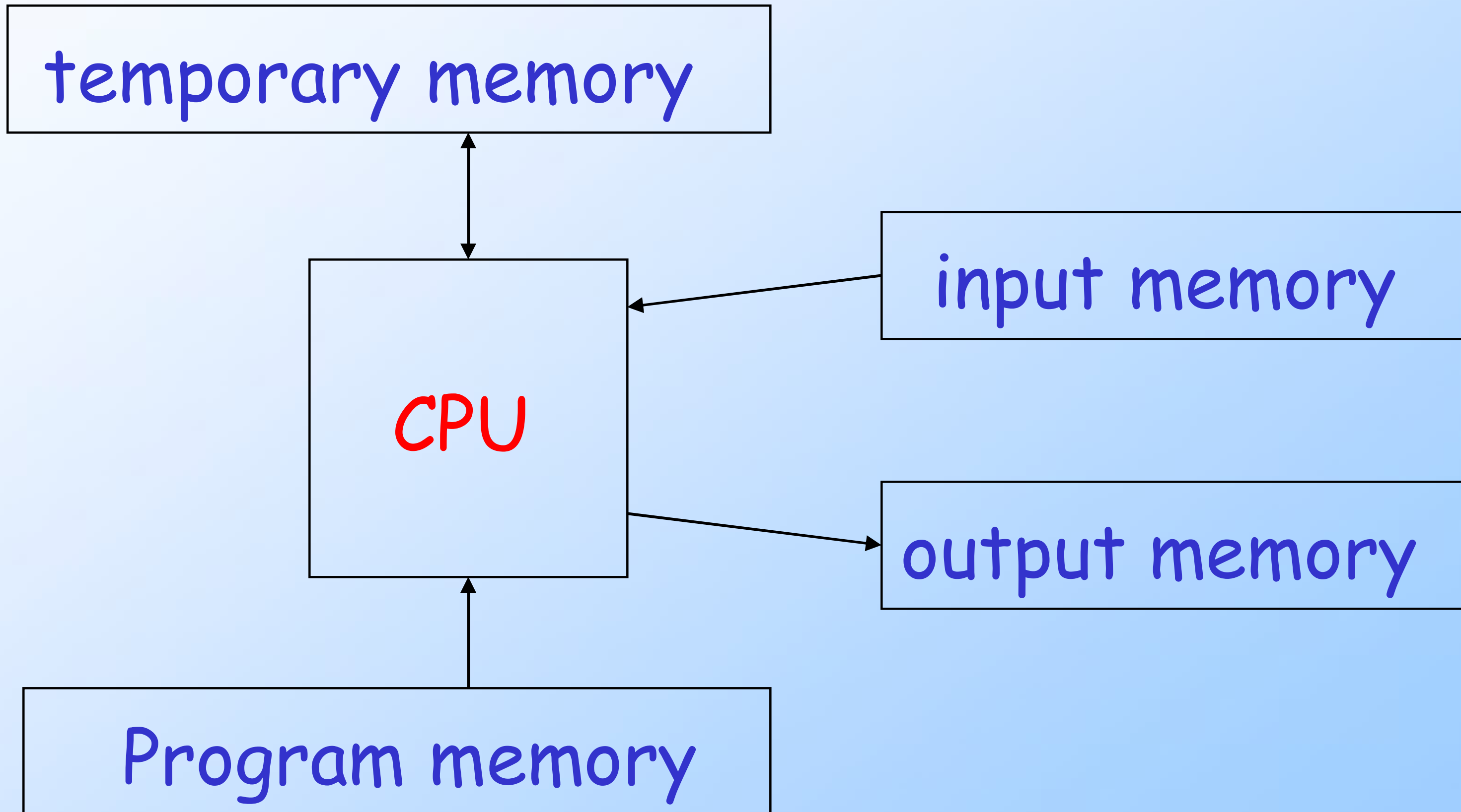
- **Automata and Automata Theory:**
  - Automaton – a self-operating machine or mechanism(Dictionary meaning) and the plural is **Automata**.
  - Automata- Abstract computing Device or Machine i.e. a mathematical model of computation.
  - Automata Theory -It deals with the study of automata and the computational problems that can be solved using the automata.

Historical Aspect

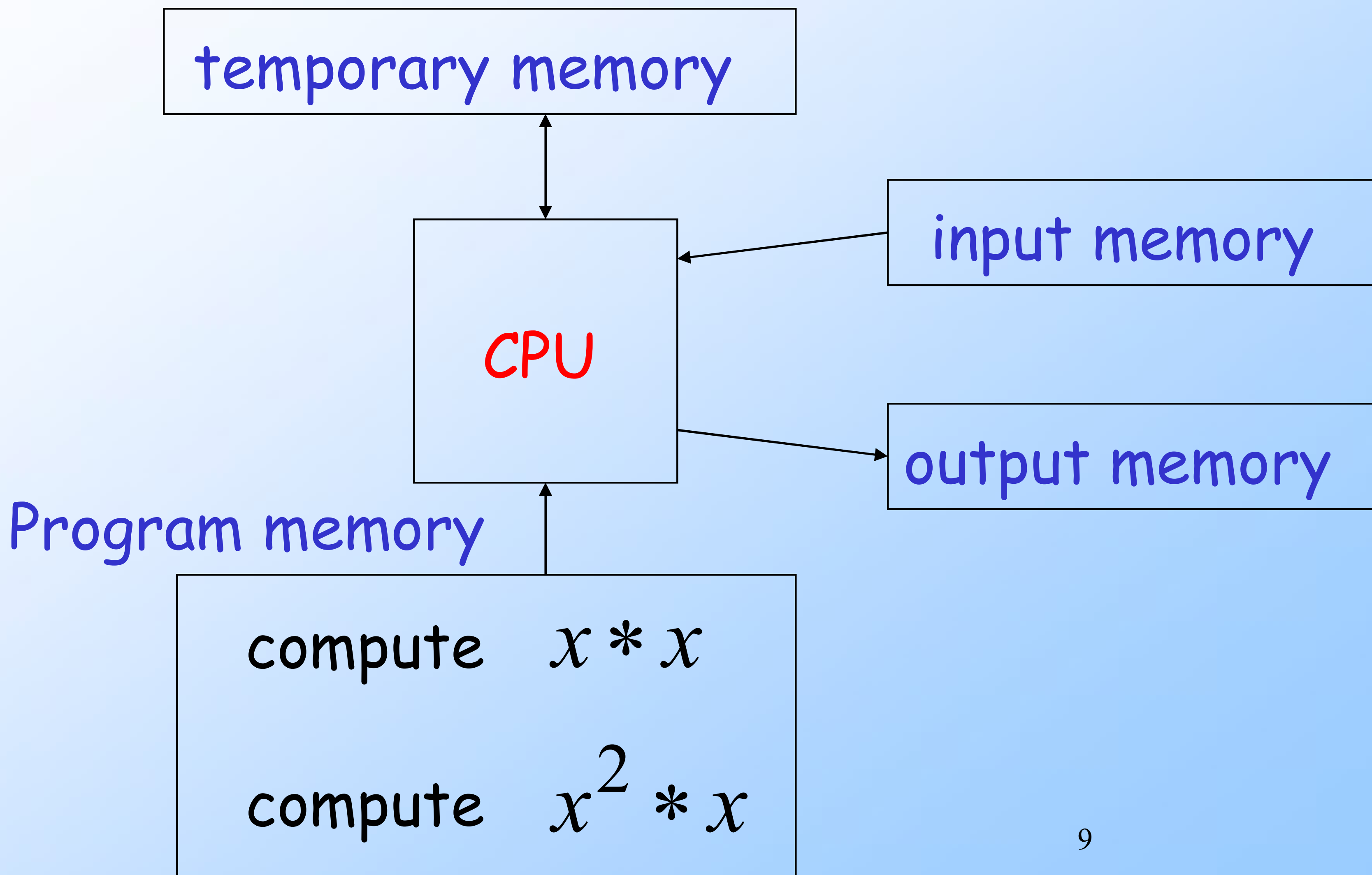
- Before 1930's, there were no computers, no any specific model, either real or abstract model of computation.
- In 1936,Godel and American mathematician Stephen Kleen proposed the *Theory of partial recursive functions* based on an inductive mechanism for the definition of functions, which has become the standard tool for studying computability.
- In 1936, the American logician Alonzo Church proposed his lambda calculus , based on constrained type of inductive definitions. *Lambda calculus* later became the inspiration for the programming language Lisp.
- The British mathematician Alen Turing proposed an abstract machine "*Turing Machine*" based on mechanistic model of problem solving that has all the capabilities of today's computer.
- **In 1940s and 1950s simpler kinds of machines, "*finite automata*" were studied by a number of researchers.**
- **In late 1950s the linguist N. Chomsky began the study of formal "*grammars*", which are closely related to abstract automata.**
- **In 1969 S. Cook extended Turing's study of what could and what could not be computed. He separated those problems as:**
  - **Those can be solved efficiently by computers, "*decidable*".**
  - **Those problems that can be solved, but in practice take so much time, that computers are useless for all-"*intractable*"**

Computation

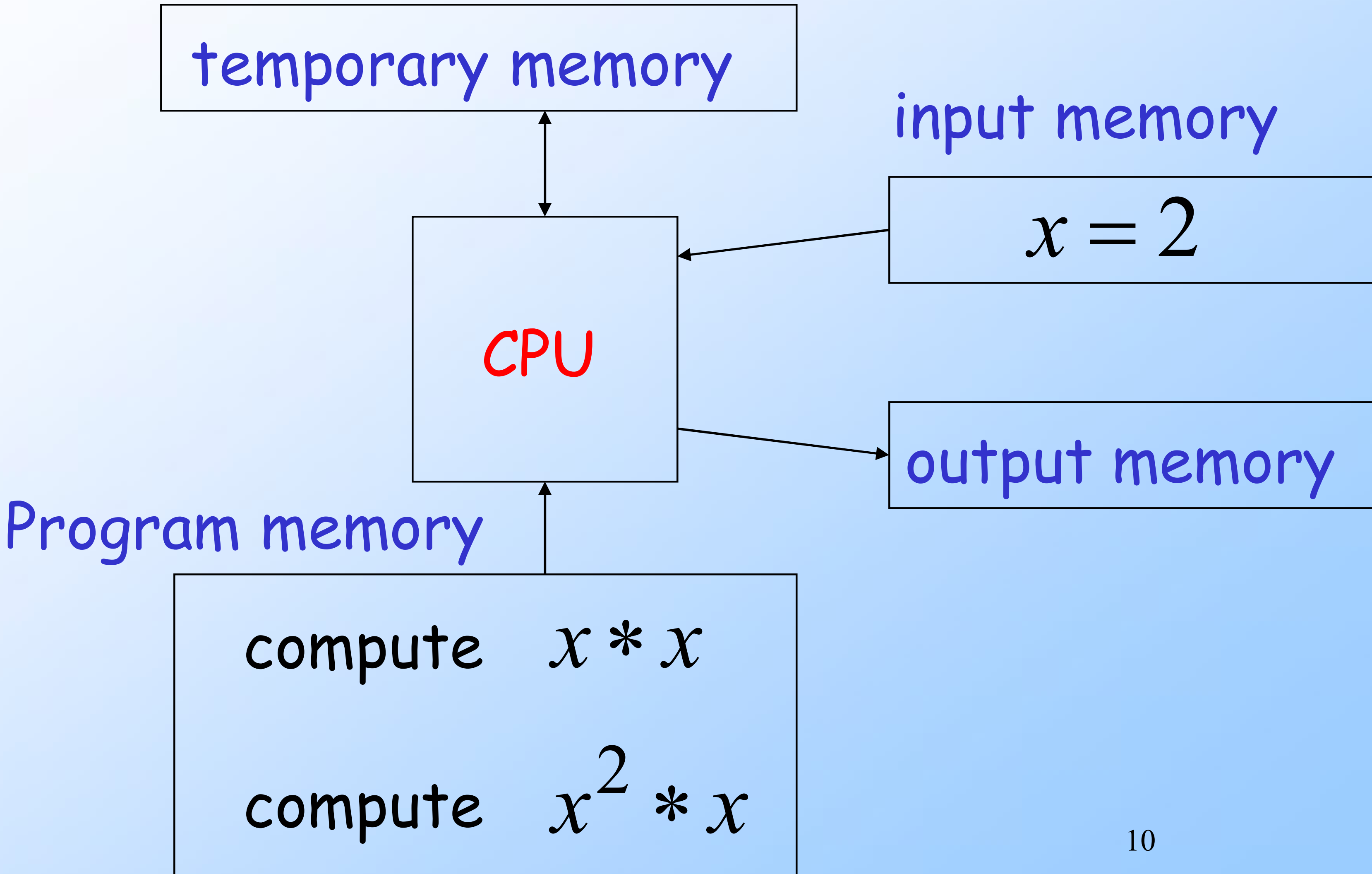



temporary memory
input memory
CPU
output memory
Program memory

Example: $f(x) = x^3$

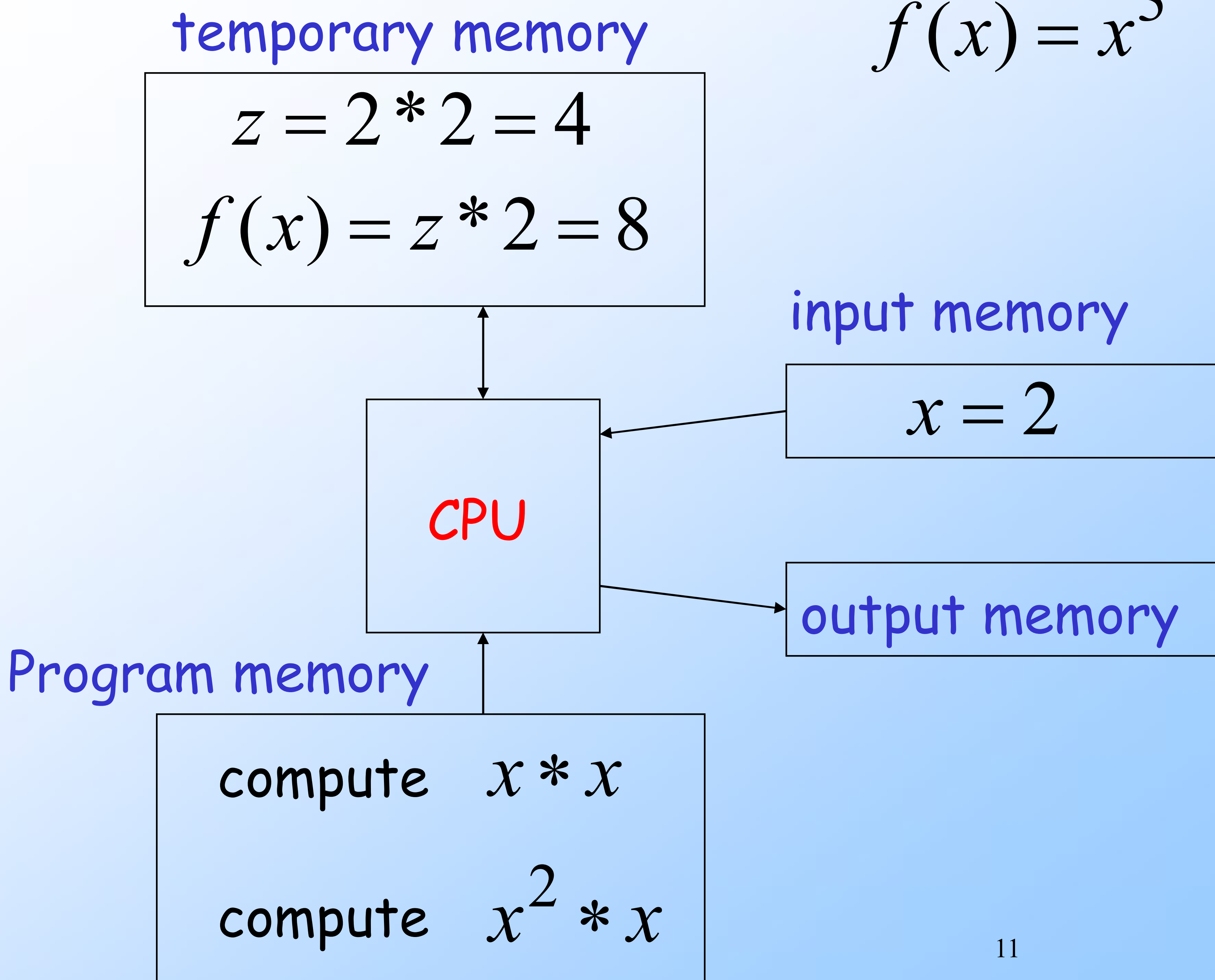


$$f(x) = x^3$$



temporary memory
input memory
$x = 2$
CPU
output memory
Program memory
compute $x * x$
compute $x^2 * x$

$$f(x) = x^3$$



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

$$f(x) = x^3$$

temporary memory

$$z = 2 * 2 = 4$$

$$f(x) = z * 2 = 8$$

input memory

$$x = 2$$

CPU

$$f(x) = 8$$

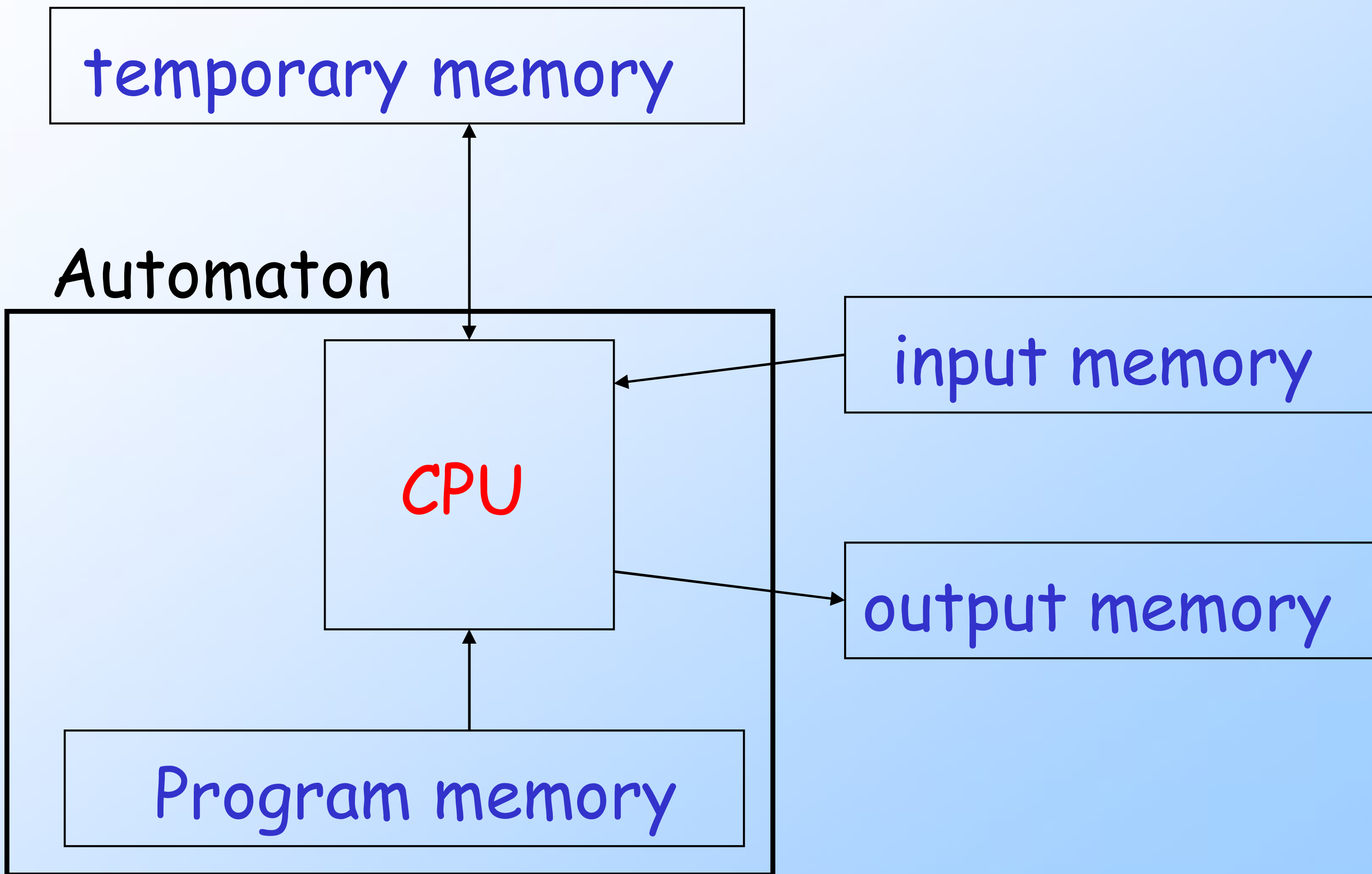
output memory

Program memory

compute $x * x$

compute $x^2 * x$

Automaton

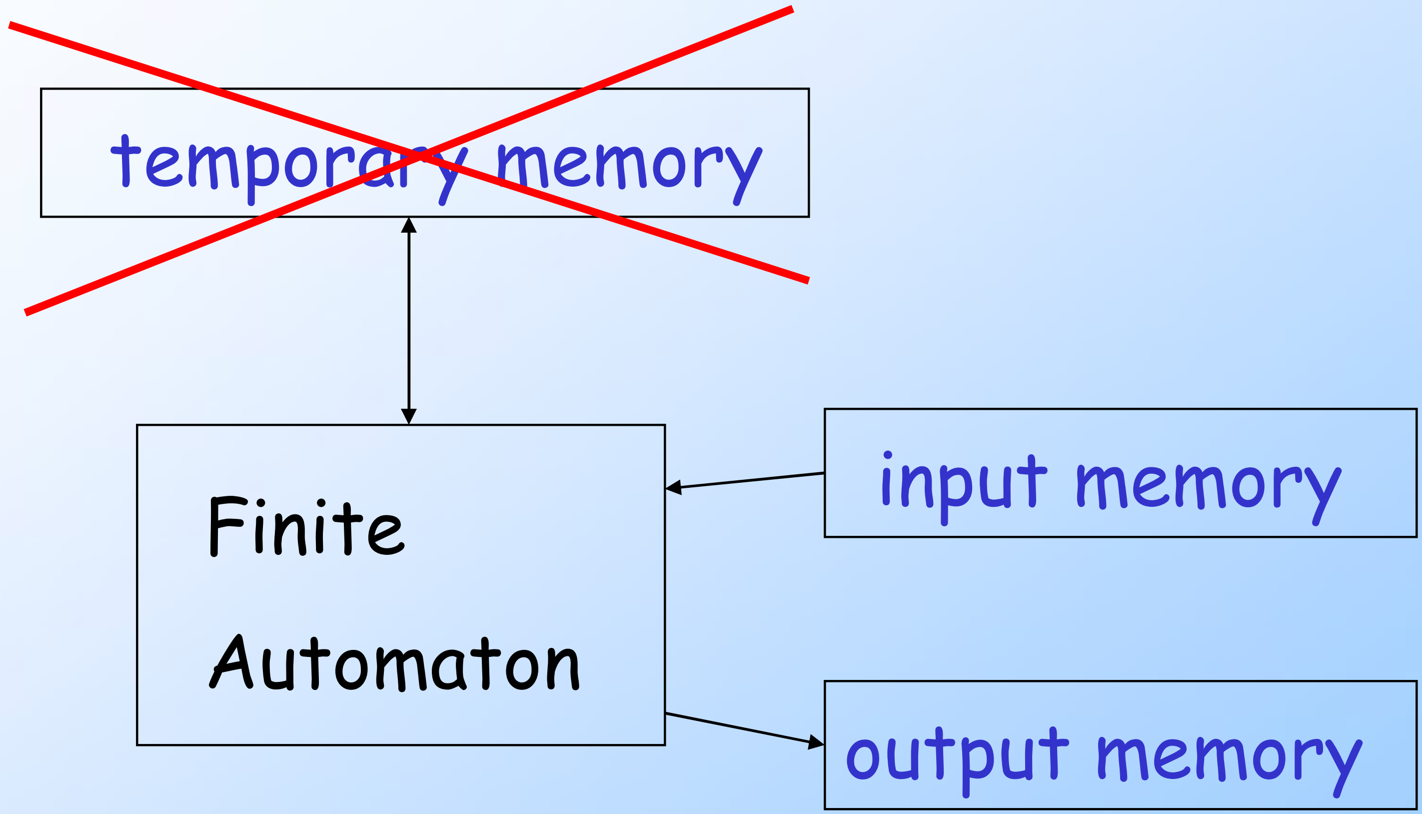

temporary memory
Automaton
CPU
input memory
output memory
Program memory

Different Kinds of Automata

Automata are distinguished by the temporary memory

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

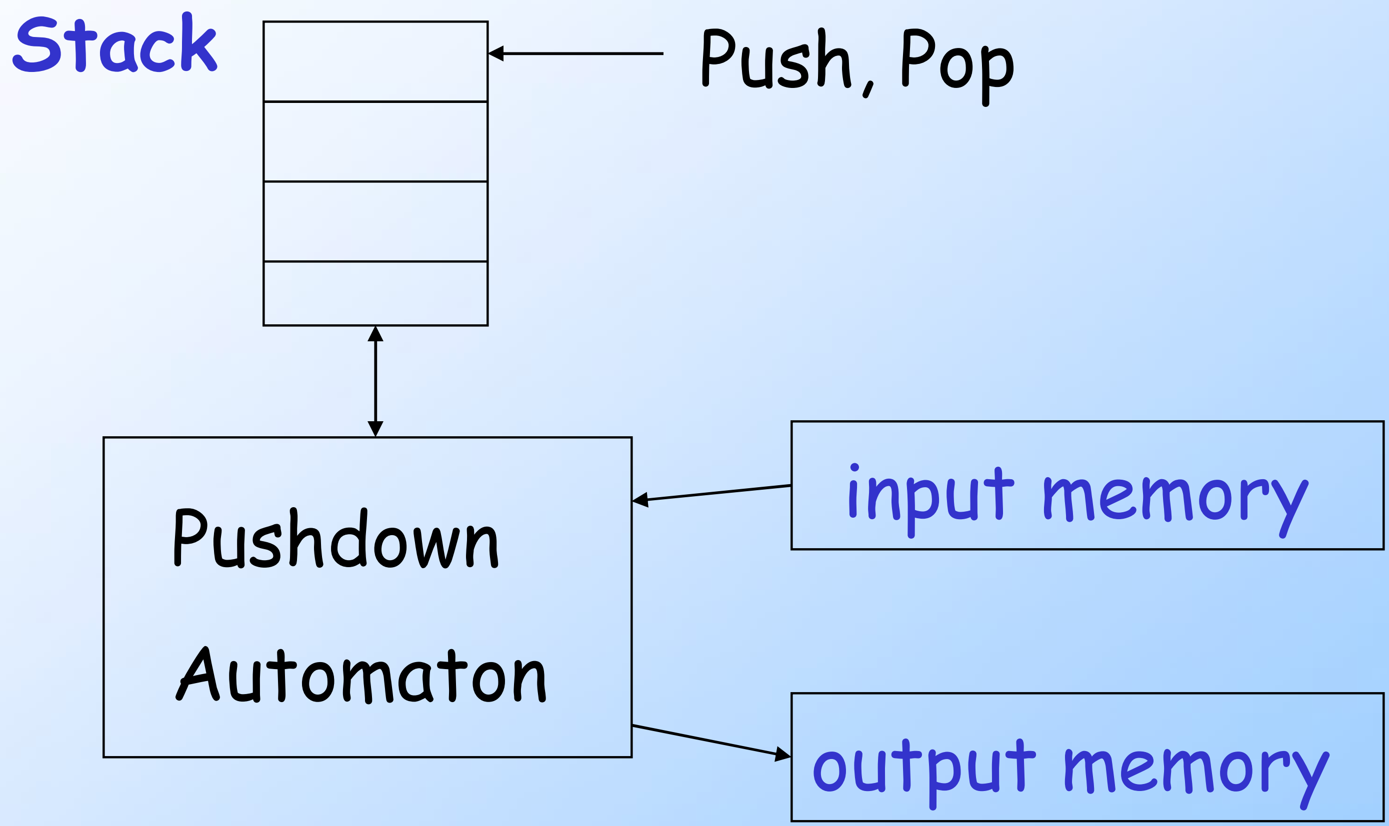
Finite Automaton



Example: Automatic Door, Vending Machines

(small computing power)

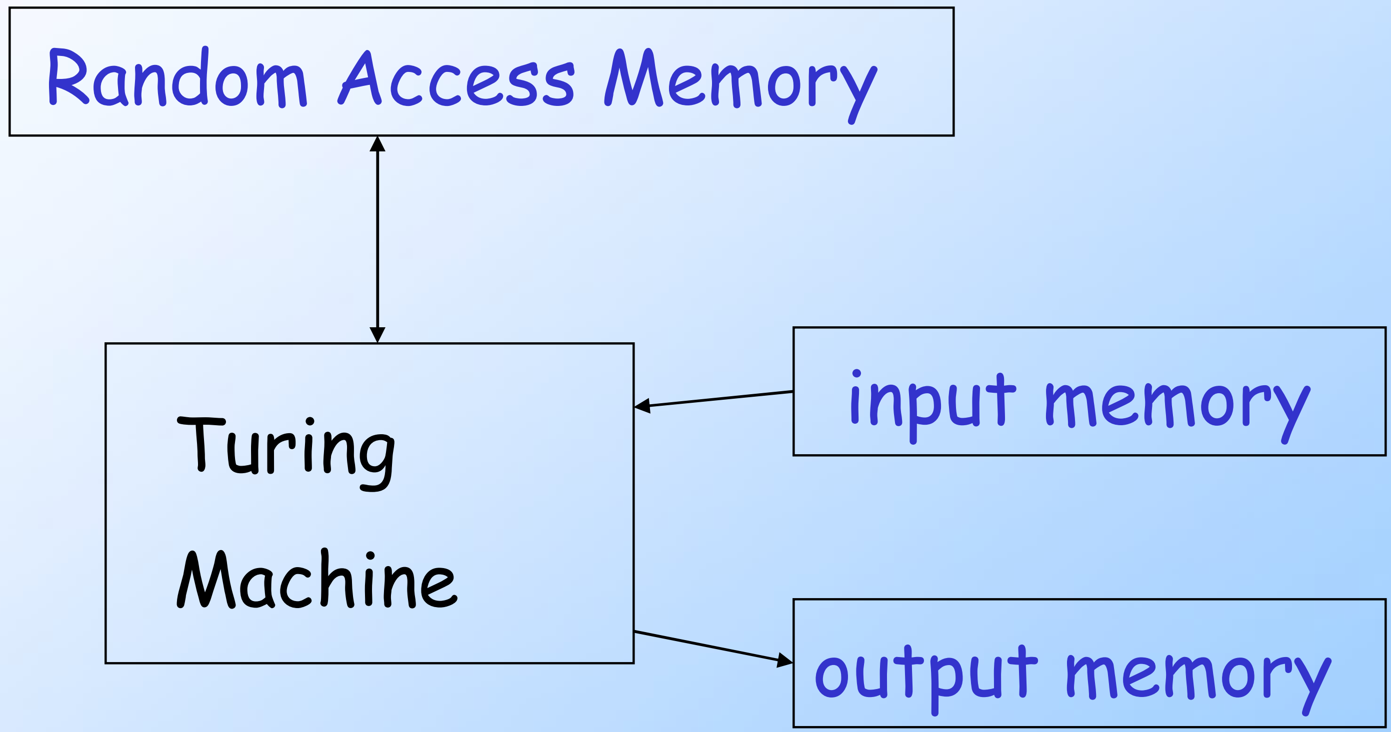
Pushdown Automaton



Example: Compilers for Programming Languages

(medium computing power)

Turing Machine



Examples: Any Algorithm

(highest computing power)

Power of Automata

Finite Automata $<$ Pushdown Automata $<$ Turing Machine

Less power $\longrightarrow$ More power

Solve more computational problems

Why Study Automata Theory?

- The study of automata and complexity is an important part of computer science. The motivation towards the study of automata theory can be summarized as:

  - Automata theory plays an important role when we are making software for designing and checking behavior of digital circuit.

  - The "*lexical analyzer*" of a typical compiler, that is, the compiler component that breaks the input text into logical units called "*tokens*".

  - Software for scanning large bodies of text, such as collections of web pages, to find occurrence of wards, phrases or patterns*(searching)*.

  - Automata theory is key to software for verifying systems of all types that have a finite number of distinct states, such as communication protocols, protocols for secure exchange of information.

  - Automata Theory is most useful concept of software for *natural language processing.*

Definitions and preliminaries

- ***Alphabet***: **An alphabet is a finite, non-empty set of symbols, that make up the language of concern.**

  e.g. A= {0, 1 } defines a Binary Alphabet

  B={+,−,×↵,Γ,α} - Alphabet of 6 symbols.

  C= { All ASCII Symbols } etc.

- For convention, we shall use ∑ in this course for alphabet.
- ***String*** **:** **(or word): A string is a finite sequence of symbols taken from an alphabet. For example,**

011010, 00,**11**,01,**10**,1,0 are strings from binary alphabet **{ 0,1}**

asdfg – A string of lower case English alphabet.

10+5×3÷6 − string of digits and mathematical operator symbols from alphabet ∑ **= {0,1,2,3,4,5,6,7,8,9,+,-,×,÷ }**

Definitions Contd…

- The ***length*** of a string w, denoted by $|w|$, is the no of symbols in w.
  - e.g. w=010010 , u=10+5×3÷6
  - $|w| = 6$ $|u| = 8$

- ***Empty string*:** It is a string consisting of zero occurrences of symbols from an alphabet. i.e. the empty string has length zero. It is denoted by $\in$ so length $|\in| = 0$.
  - *So empty string is a string obtained from the symbols of any alphabets.*

Definitions Contd…

- ***Power of an Alphabet*:** The set of all strings of a certain length say k, from an alphabet is the k^{th} power of that alphabet.

  i.e. $\Sigma^k = \{ w \mid |w| = k\}$

  - e.g. Let $\sum = \{0,1\}$, *then*,
  - $\sum^0 = \{\in\}$
  - $\sum^1 = \{0,1\}$
  - $\sum^2 = \{00,01,10,11\}$
  - $\sum^3 = \{000,001,\ldots\ldots\ldots\ldots\ldots\ldots,111\}$ and so on.

Definitions Contd…

- ***Kleene Closure*:** The set of all strings(strings with any length) over an alphabet Σ, denoted by Σ^* is called *Kleene closure* of Σ. i.e. *Kleene Closure* is set of all strings over alphabet Σ with length 0 or more. Mathematically we can write,
  - $\Sigma^* = \Sigma^0 \cup \Sigma^1 \cup \Sigma^2 \cup$........ An Infinite set

- ***Positive closure*:** The set of all strings from an alphabet Σ. except empty string is called positive closure of Σ. and denoted by Σ^+
- i.e.
  - $\Sigma^+ = \Sigma^* - \Sigma^0 = \Sigma^1 \cup \Sigma^2 \cup$......... An Infinite set of strings from Σ except empty string

Definitions Contd…

- ***<u>Suffix of string</u>*****<u>:</u>** A string *s* is called a *<u>suffix</u>* of string *w* if it is obtained by removing zero or

  more leading symbols in *w*.

  e.g. $w = \alpha\beta\gamma\delta$; $s = \beta\gamma\delta$, is a suffix of *w*.

  Let w= **apple**, then all suffix of w are (apple, pple, ple, le , e , ε)

  - *s* is proper suffix since $s \neq w$.

- ***<u>Prefix of a string</u>*****<u>:</u>** The string *s* is called a *<u>prefix</u>* of string *w* if it is obtained by removing zero or more trailing symbol from *w*.

  e.g. $s = \alpha\beta\gamma\delta$, $s = \alpha\beta\gamma$, is a prefix of w.

  Let w= apple, then all prefix of w are { apple, appl, app, ap,a , ε)

  - s is proper prefix since $s \neq w$.

- Look above in both suffix and prefix example, ε is both suffix and prefix of any string.

Definitions Contd…

- ***Substring:*** The string *s* is called a *substring* of *w* if it is obtained by removing zero or more leading or trailing symbols from *w*. It is proper substring if $s \neq w$.
- e. g. *w* = *abcdefgh*, Here the strings *ab, bc, cdef, fgh* are the substrings of *w.*
- ***Let w = patan***

The Valid Substrings:

- *All prefixes and suffixes of the strings*
- *More like ata, at, ta*

- *Not valid substrings: pn, aa, pt, aan ,pan etc*

Definitions Contd…

- ***Language:*** A language L over an alphabet ∑ is a subset of the set of all strings that can be formed out of ∑. i.e. A language is the subset of *Kleene closure* over an alphabet ∑. i.e. $L \subseteq \Sigma^*$.

  Formally, L={w |something about w}

- A language may be empty: $L = \varphi$.
- The language $L_1 = \{\in\}$ is not an empty language since it contains one string ∈. So L is empty but L_1 is a language of empty string.

- A sting over an alphabet ∑ is any string $w \in \Sigma^*$.
- Σ^* is the language over ∑ consisting of all stings.
- Any string that can be made from alphabet ∑ is in Σ^*

Definitions Contd…

Some examples of language:

- *English Language is a language from finite set of {A..Z, a..z} alphabet which consists of the collection of legal English words(strings) from its alphabet.*
- *A programming language like C consists of legal C strings from its alphabet(ASSCI character set)*

- Set of all strings over $\Sigma = \{0,1\}$with equal no of 0's&1s.
  L={∈,01,0011,10,1100,0101,1010,..........}

- ***φ***, the empty language, is a language over any alphabet.

- **{**∈**}** is the language consisting of only empty string and also a language from any alphabet.
- Set of all strings over binary alphabet with length of each exactly 2 is **{00,01,10,11}** **Formally , L = {w| w** ∈ {0,1}* and |w|=2 }
- Set of all strings over binary alphabet with length at most 2 is **{ ε, 0, 1, 00,01,10,11}** **Formally L={w|w is in {0,1}* and |w|<=2}**

Definitions Contd…

- **Problem**: A problem in automata theory is the question of deciding whether a given string is a member of some particular language.

- In other words, if Σ is an alphabet and L is a language over Σ , then problem is.

  - Given a string w in Σ^* , decide whether or not w is in L.

Types of Problems

- Decision Problem
  - When the answer are simply Yes(Accept) or No(Reject) then such problem is decision problem. E.g S={a,e,i,o,u} , Is x a member of S?
- Search Problem
  - A solution algorithm must search for the correct structure to return. The return value by the algorithm may be single or structured. e.g. Searching text in web
- Optimization Problem
  - When the return value optimizes some objective function, such problem is optimization problem. e.g. finding shortest path in graph rather than any path
- Enumeration Problem
  - When the answer is list of all satisfactory structures, it is called enumeration problem. e.g. return all paths from A to B
- Counting Problem
  - When the result is the count of such structures rather than a list, we called it counting problem. E. g. return no of possible paths from A to B.

- Thank You