An Introduction to Formal Languages and Automata

Peter Linz

What is Automata?

- **A machine or control mechanism designed to follow automatically a predetermined sequence of operations or respond to encoded instructions.**
- Thermostats, automatic pilots of aircraft (device for controlling an aircraft), missile guidance systems, telephone networks, and controls of certain kinds of automatic elevators are all forms of automata.
- The best known general automaton is the modern electronic computer, the internal states of which are determined by the data input and which operates to produce a certain output.

Finite automata are a useful model as……

1. Software for designing and checking the behaviour of digital circuits

2. The lexical analyser of a typical compiler, that is, the compiler component that breaks the input text into logical units

3. Software for scanning large bodies of text, such as collections of Web pages, to find occurrences of words, phrases or other patterns

4. Software for verifying systems of all types that have a finite number of distinct states, such as communications protocols for secure exchange information

Automata theory teaches you the very important equivalence between….

→a language: some -- usually -- infinite set of strings

→a grammar: the finite set of rules to generate that language

→an automaton: the abstract processing device that can recognize that language

- Automata theory and formal languages are the base of current

→compilers,

→regular expressions,

→parsers,

→web-scrappers,

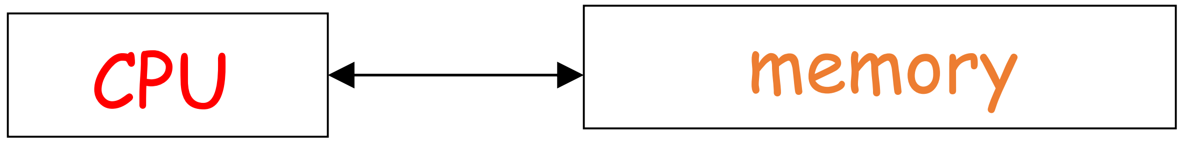
→natural language processing (NLP),

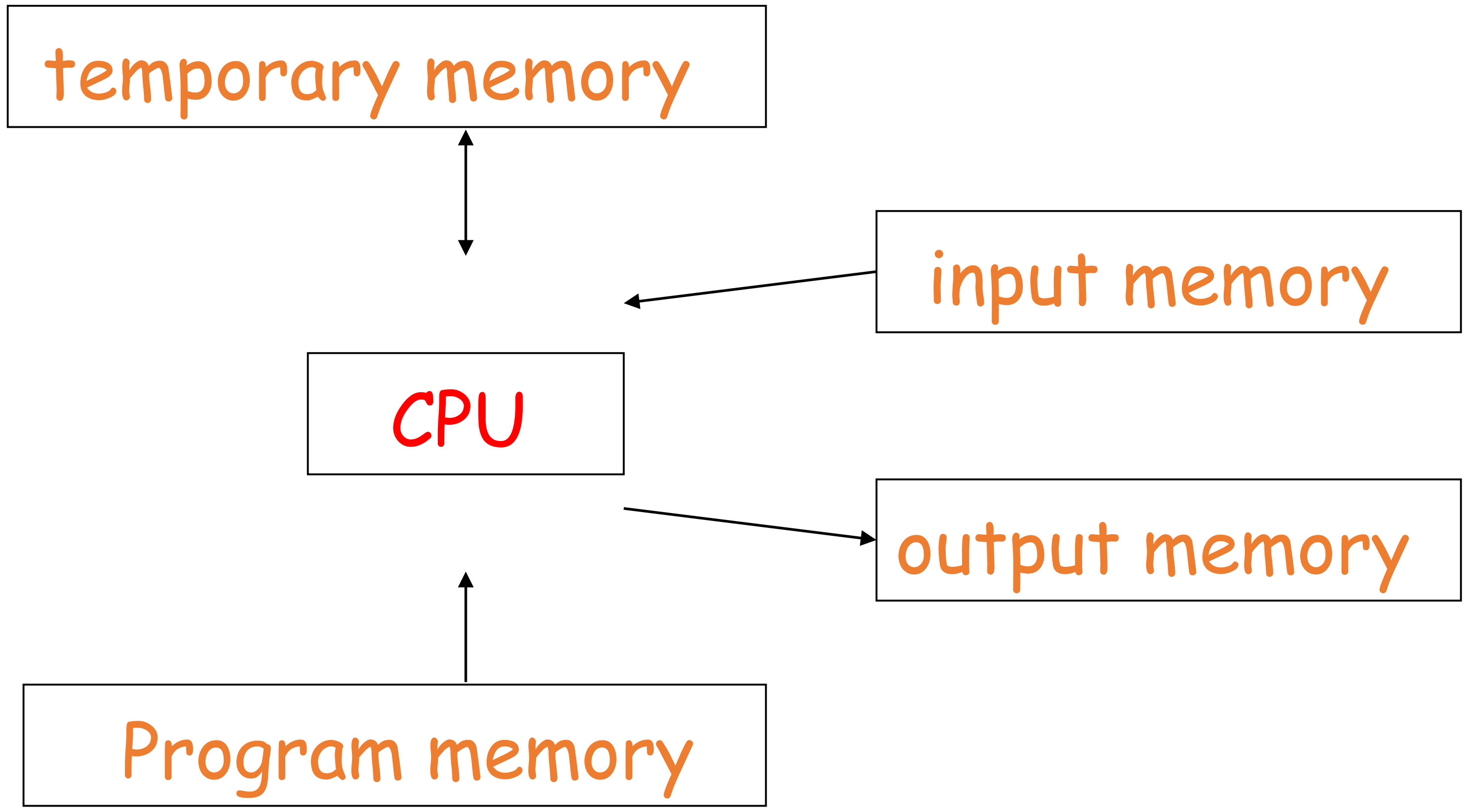
→Automata play a major role in theory of computation, compiler construction, artificial intelligence, parsing and formal verification.

- **Theory of computation** deals with how efficiently problems can be solved on a model of computation **- that** describes how a set of outputs are computed for a given a set of inputs….
- A **compiler** is a computer program that transforms computer code written in one programming language (the source language) into another programming language (the target language).
- **Parsing**, **syntax analysis**, or **syntactic analysis** is the process of analysing a string of symbols, either in natural language, computer languages or data structures, conforming to the rules of a formal grammar

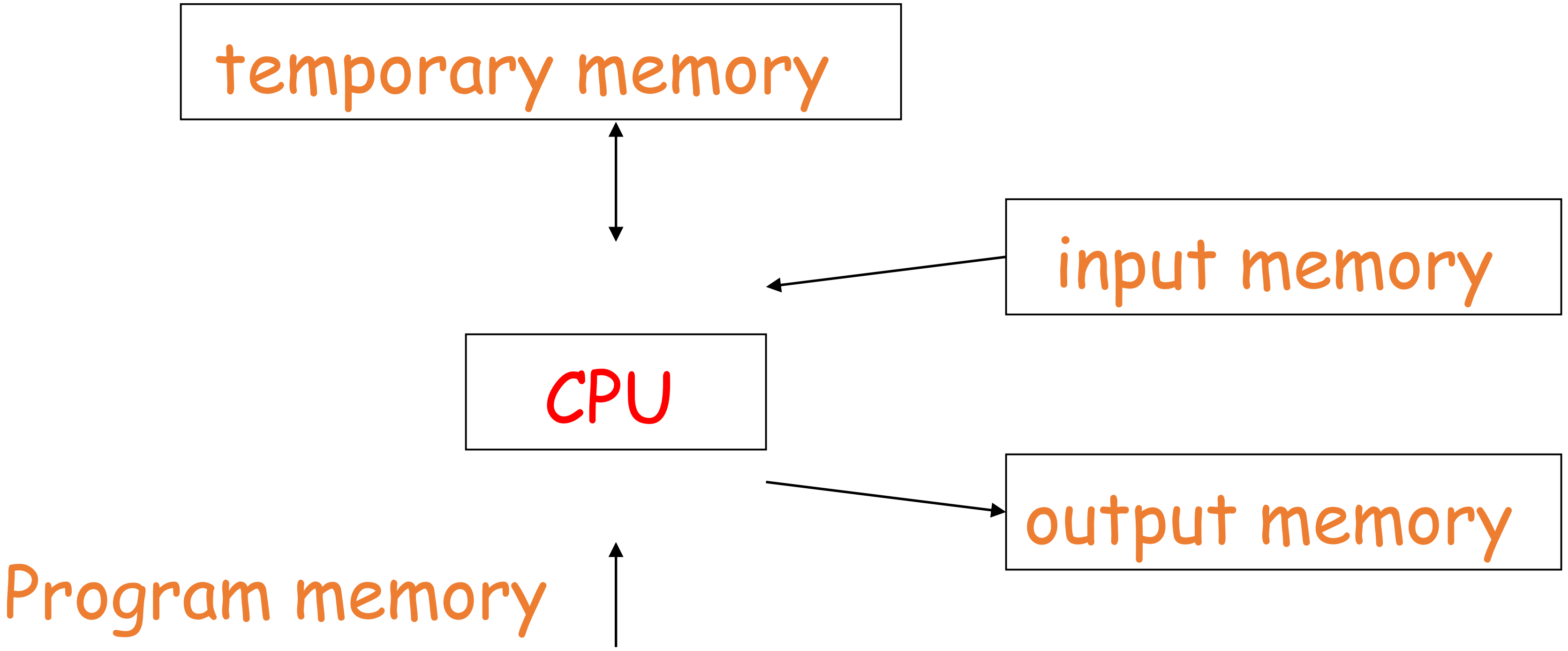
- **Automaton** that contains all features of a digital computer . It accepts input, produces output, and have temporary storages and can make decisions in transforming the input into output.
- **Formal Language** is the general characteristics of programming language which consist of set of symbols and some rules where symbols can be combined into sentences.
- Finally we formalize the concept of mechanical computation by giving a precise definition of the term **algorithm**.

Computation




temporary memory
input memory
CPU
output memory
Program memory

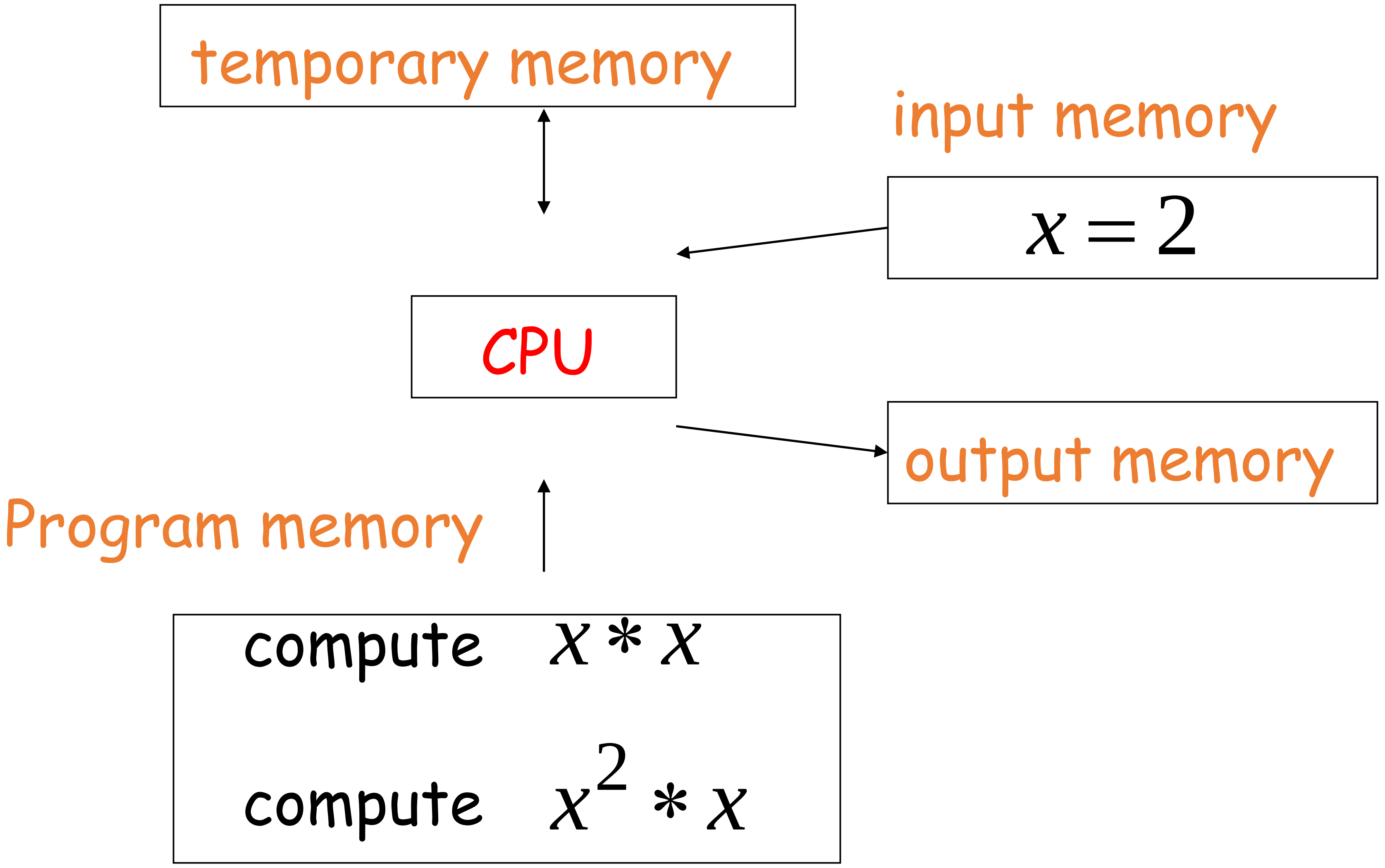
Example: $f(x) = x^3$



compute $x * x$

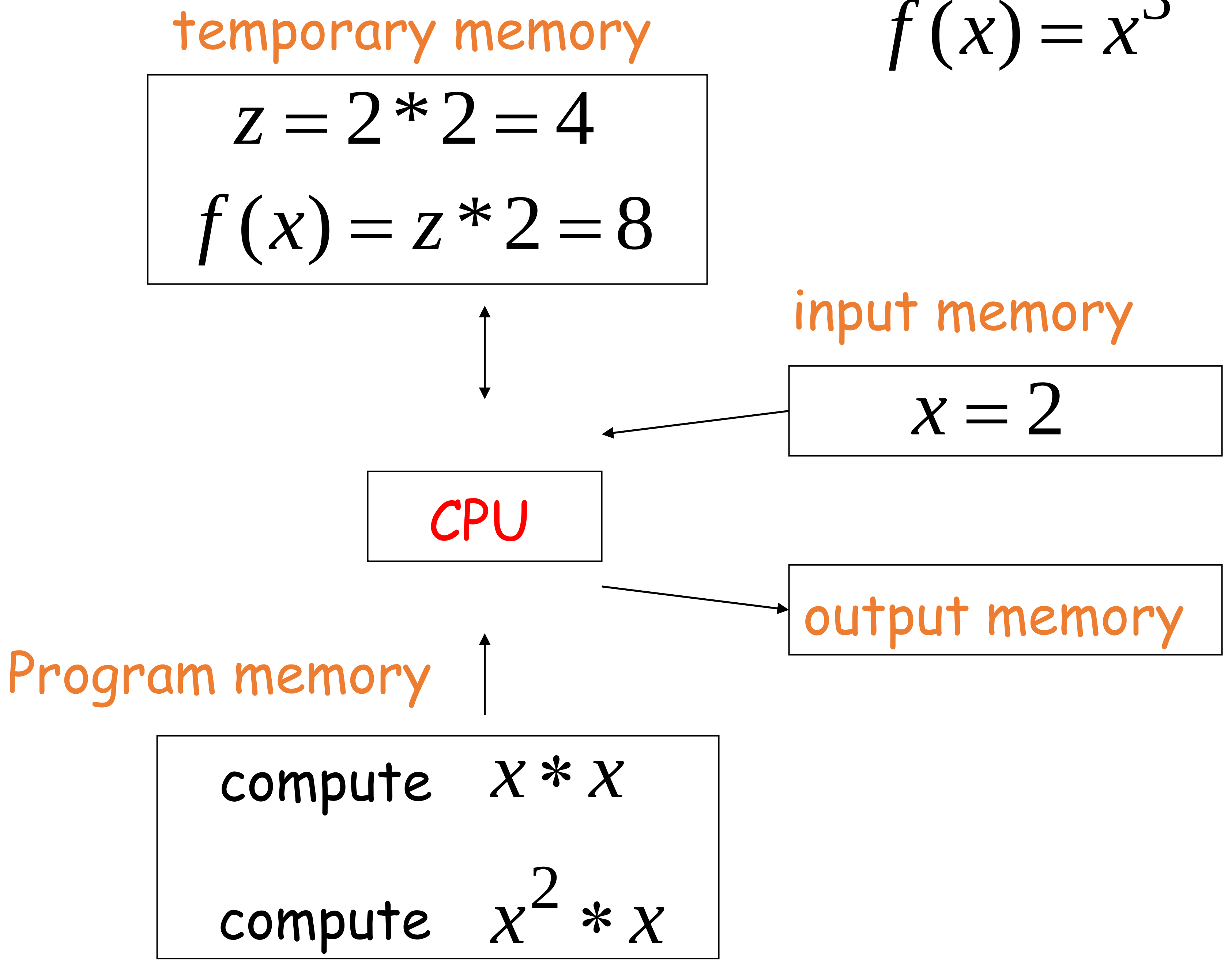
compute $x^2 * x$

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

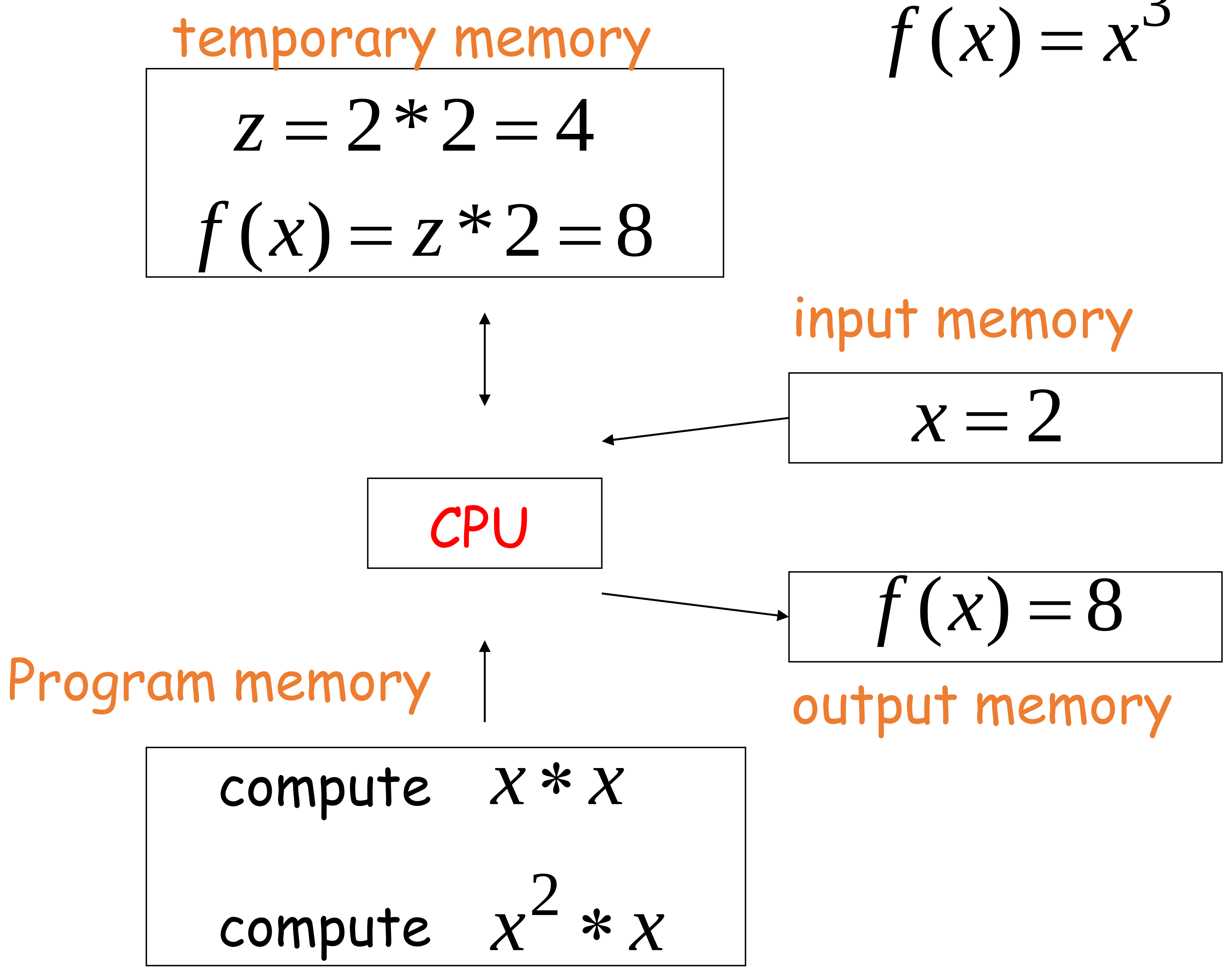


temporary memory
input memory
$x = 2$
CPU
output memory
Program memory
compute $x * x$
compute $x^2 * x$

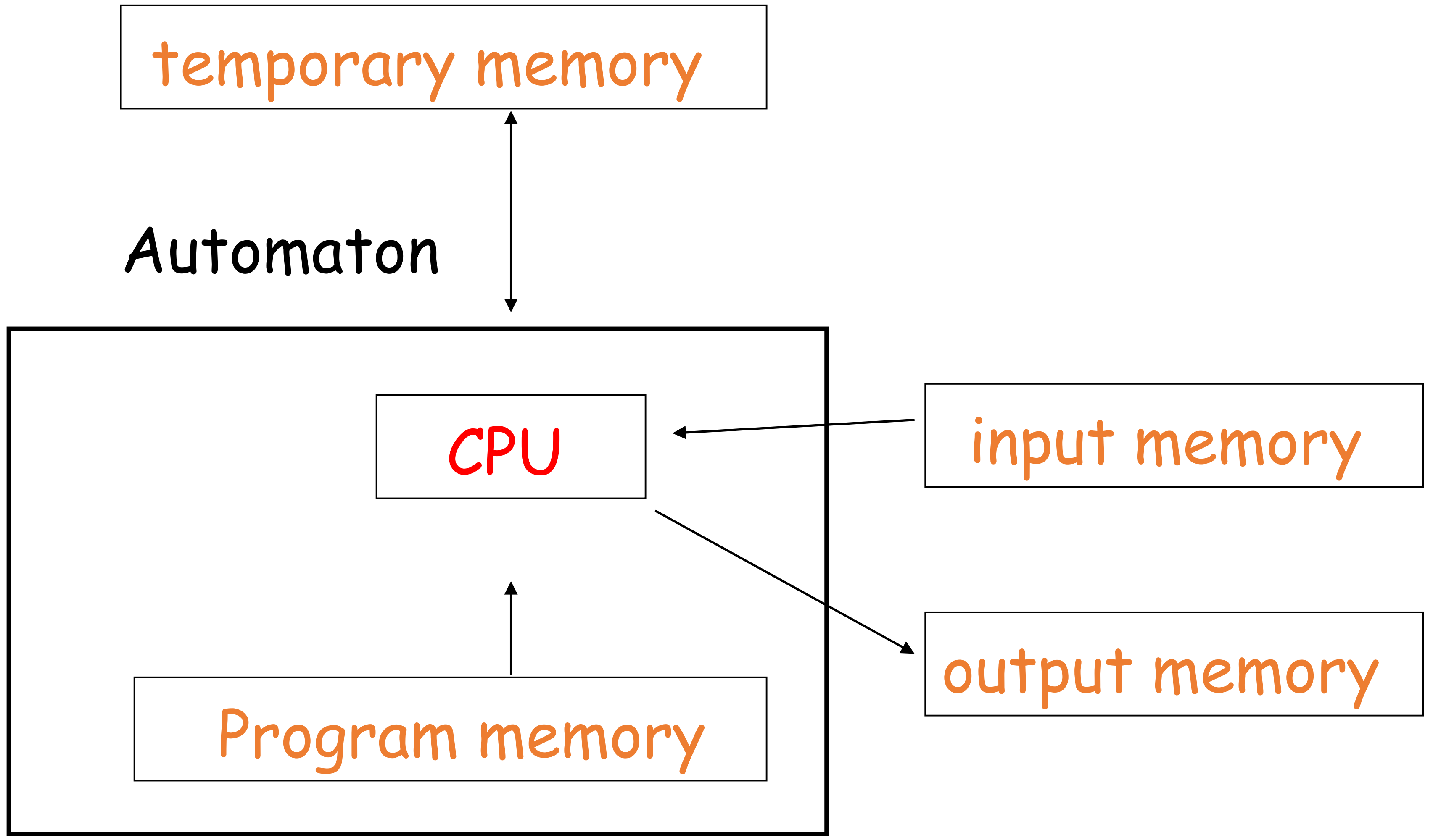
$$f(x) = x^3$$


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

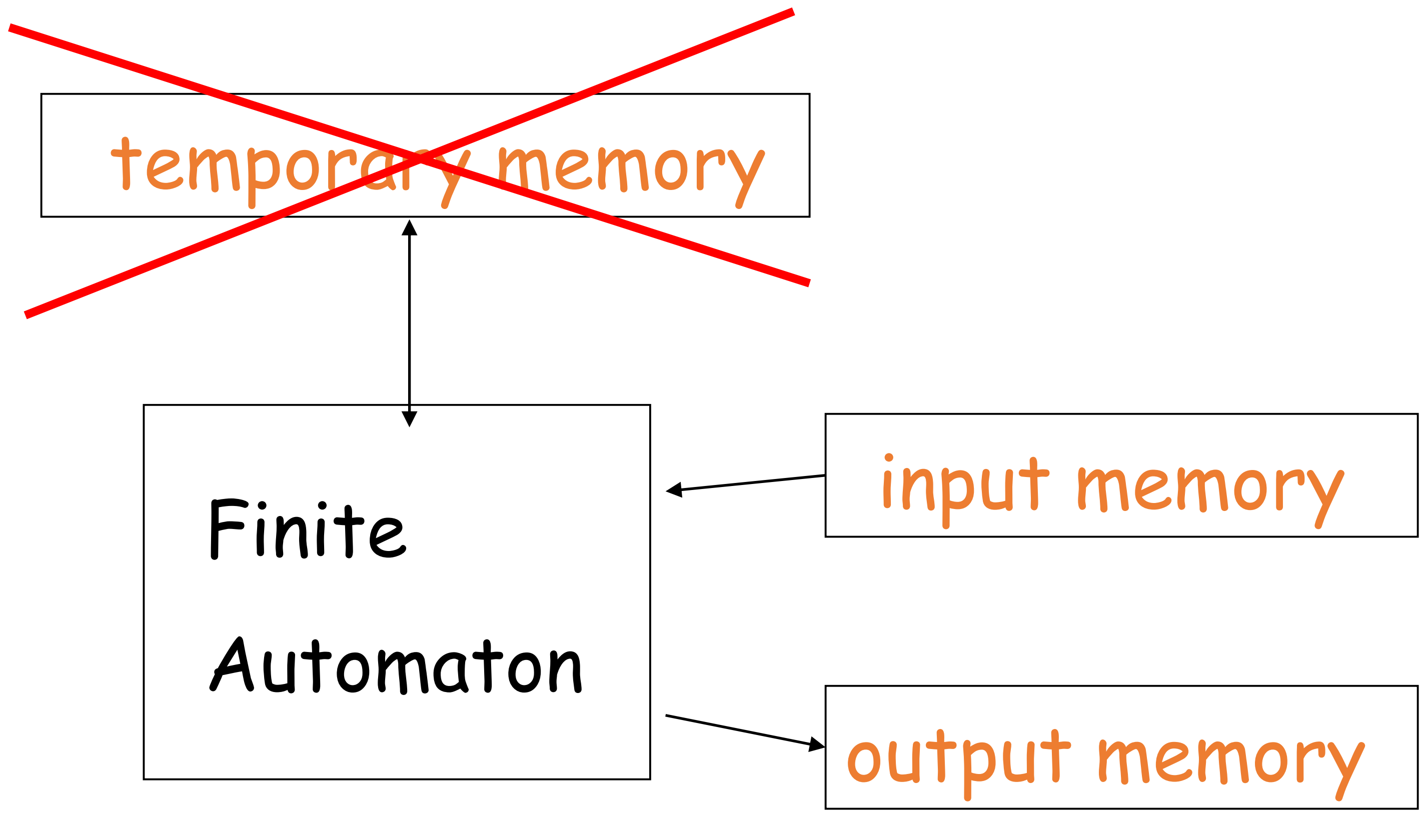

temporary memory
Automaton
CPU
input memory
output memory
Program memory

Different Kinds of Automata

Automata are distinguished by the temporary memory

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

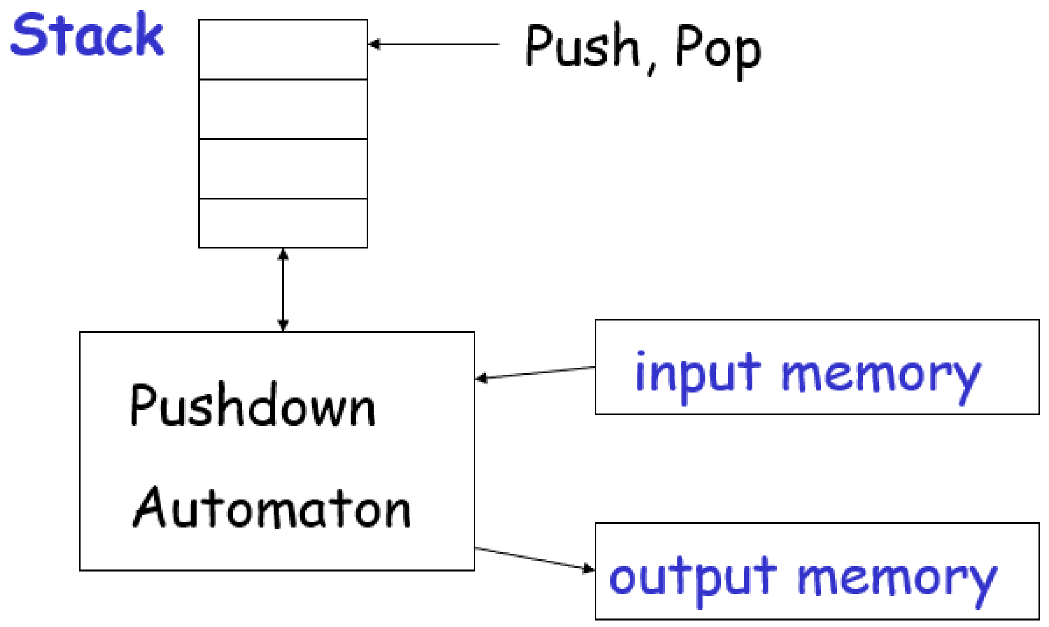
Finite Automaton



Example: Vending Machines

(small computing power)

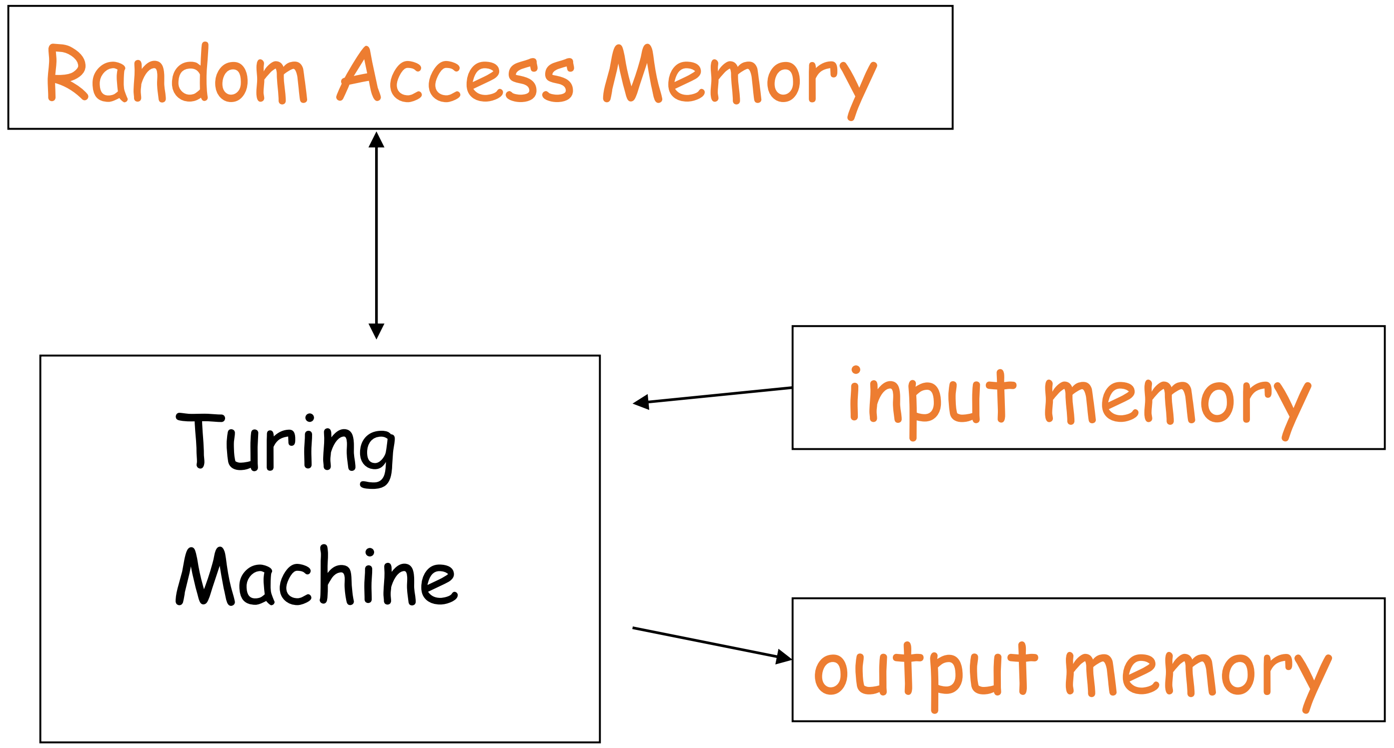
Pushdown Automaton



Example: Compilers for Programming Languages

(medium computing power)

Turing Machine



Examples: Any Algorithm

(highest computing power)

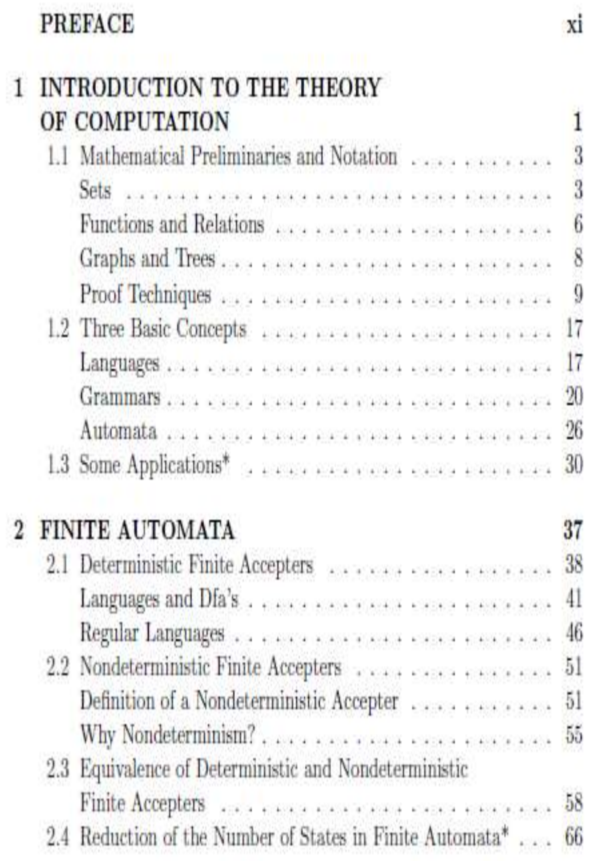
Power of Automata

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

Less power $\longrightarrow$ More power

Solve more computational problems

Module -1	Teaching Hours
INTRODUCTION TO THE THEORY OF COMPUTATION AND FINITE AUTOMATA: Three basic concepts, Some Applications, Deterministic Finite Accepters, Nondeterministic Finite Accepters, Equivalence of Deterministic and Nondeterministic Finite Accepters, Reduction of the Number of States in Finite Automata. **Text Book 1**: Chapter 1:1.2 - 1.3, Chapter 2: 2.1 - 2.4	**08 Hours**



Three Basic Concepts

→Languages

→Grammar

→Automata

Finite non empty sets **Σ** of symbols called **alphabets**

From individual symbols we construct **strings** which are finite sequence of symbols from the alphabet.

If Σ = { a,b} then abab, aaabbba are some of the **strings** on **Σ**

We use lowercase letters a, b, c… for elements of Σ

We use u, v , w for string names

Example w = abaaa

Languages:

The **concatenation** of two strings w and v is the string obtained by appending the symbols of v to the right end of w, that is, if

$$w = a_1 a_2 \cdots a_n$$

$$v = b_1 b_2 \cdots b_m,$$

then the concatenation of w and v, denoted by wv, is

$$wv = a_1 a_2 \cdots a_n b_1 b_2 \cdots b_m.$$

The **reverse** of a string is obtained by writing the symbols in reverse order; if w is a string as shown above, then its reverse w^R is

$$w^R = a_n \cdots a_2 a_1.$$

The **length** of a string w, denoted by $|w|$, is the number of symbols in the string.

Empty string is the string with no symbols and denoted by **λ**

$$\boxed{\begin{aligned} |\lambda| &= 0, \\ \lambda w &= w\lambda = w \end{aligned}}$$

- If **w = vu** then the substrings **v** and **u** are **prefix** and **suffix** of **w**

Example 1: **w = abbab**

Prefix : **{ λ, a,ab,abb,abba, abbab}** is the prefixes of w

Suffix : **{abbab, bbab, bab, ab, b, λ }**

Example 2: **w = aabcd**

Prefix : **{ λ, a,aa,aab,aabc, aabcd}**

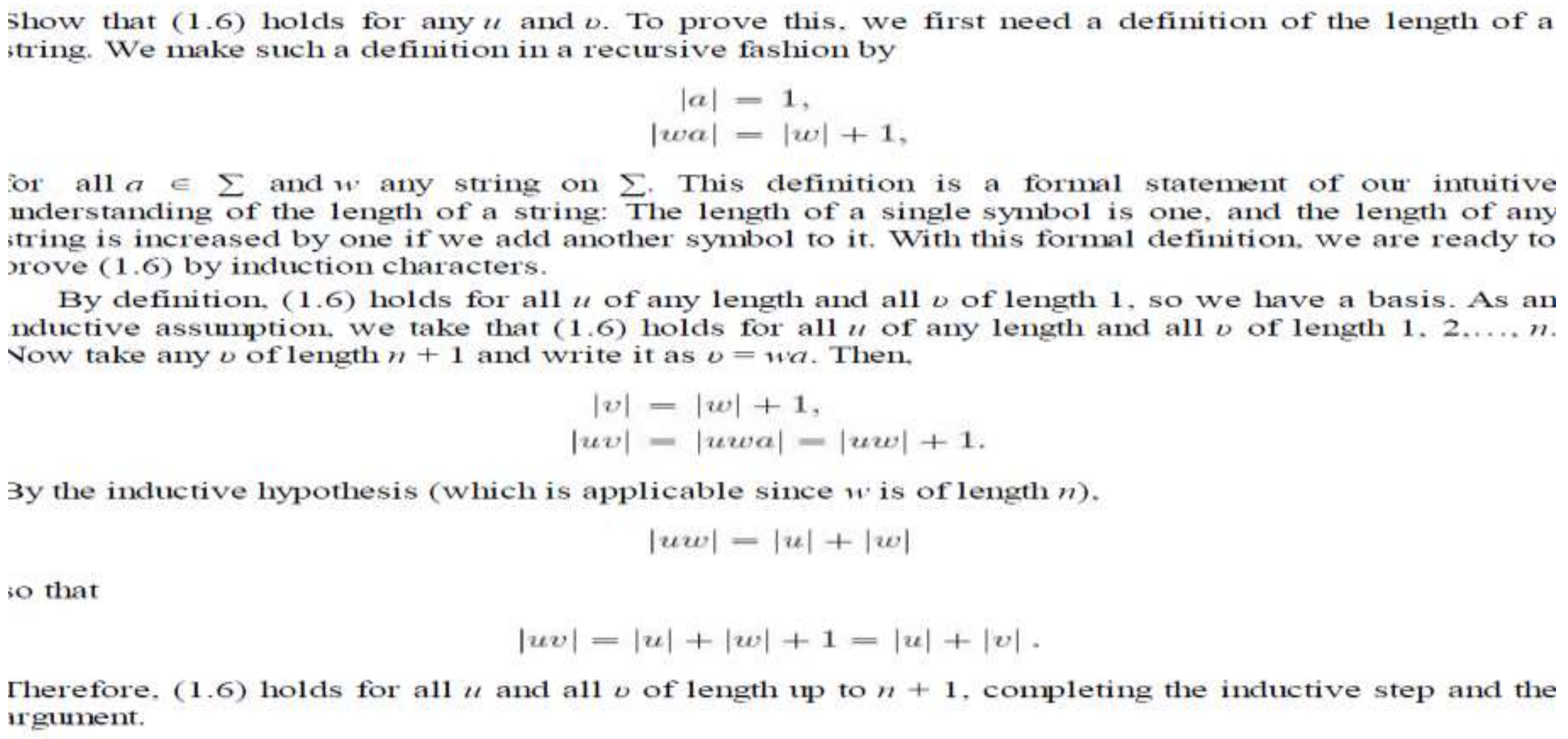
Suffix : **{aabcd, abcd, bcd, cd,d, λ }**

Recursive Definition of Length: (Exam point of view no proofs required. We solve problems based on proofs.)

- If u and v are strings, then the length of their concatenation is the sum of the individual length.

ie.

$$|uv| = |u| + |v| . \qquad (1.6)$$



Show that (1.6) holds for any u and v. To prove this, we first need a definition of the length of a string. We make such a definition in a recursive fashion by

$$|a| = 1,$$
$$|wa| = |w| + 1,$$

or all $a \in \Sigma$ and w any string on Σ. This definition is a formal statement of our intuitive nderstanding of the length of a string: The length of a single symbol is one, and the length of any string is increased by one if we add another symbol to it. With this formal definition, we are ready to prove (1.6) by induction characters.

By definition, (1.6) holds for all u of any length and all v of length 1, so we have a basis. As an nductive assumption, we take that (1.6) holds for all u of any length and all v of length $1, 2, \ldots, n$. Now take any v of length $n+1$ and write it as $v = wa$. Then,

$$|v| = |w| + 1,$$
$$|uv| = |uwa| = |uw| + 1.$$

By the inductive hypothesis (which is applicable since w is of length n),

$$|uw| = |u| + |w|$$

so that

$$|uv| = |u| + |w| + 1 = |u| + |v| .$$

Therefore, (1.6) holds for all u and all v of length up to $n+1$, completing the inductive step and the rgument.

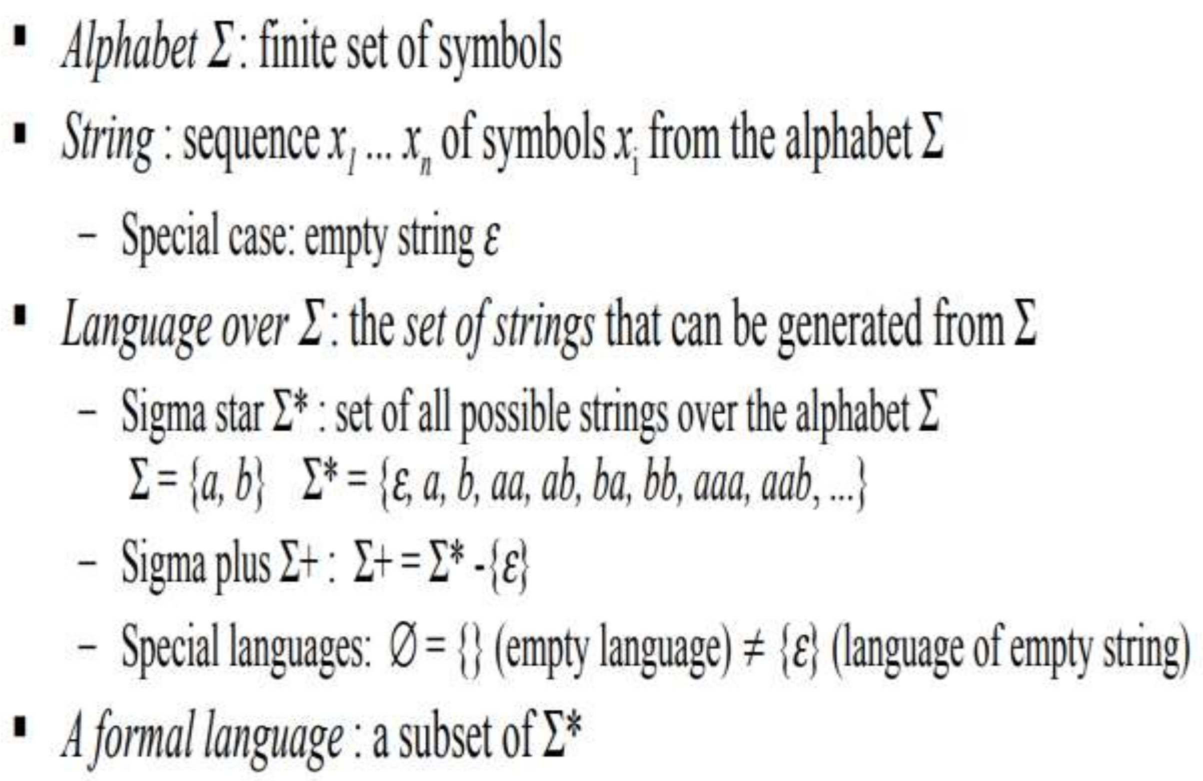
- If **w** is a string, then **w**^{n} stands for string obtained by repeating **w, n** times.
  $w^0 = \lambda$.

- **Σ*** ➔ includes **λ** and **Σ$^+$ → Σ* - λ**

- **Σ** is finite where **Σ*** and **Σ$^+$** are always infinite.

- Let **L = { a, aa, ab, abb}** on **Σ** then
- $\overline{L}$ = **Σ* - L**
- **$L^* = L^0 \cup L^1 \cup L^2 \cup$ ….. (star closure)**
- **$L^0 = \{\lambda\}$**
- **$L^1 = L$**
- **$L^2 = L . L$**
- **$L^+ = L^1 \cup L^2 \cup$….. (positive closure)**

- **A string in a Language L will be called as sentence of L.**



- *Alphabet* Σ: finite set of symbols
- *String* : sequence $x_1 \ldots x_n$ of symbols x_i from the alphabet Σ
  - Special case: empty string ε
- *Language over* Σ: the *set of strings* that can be generated from Σ
  - Sigma star Σ^* : set of all possible strings over the alphabet Σ
    $\Sigma = \{a, b\}$ $\Sigma^* = \{\varepsilon, a, b, aa, ab, ba, bb, aaa, aab, \ldots\}$
  - Sigma plus $\Sigma+$: $\Sigma+ = \Sigma^* - \{\varepsilon\}$
  - Special languages: $\emptyset = \{\}$ (empty language) $\neq \{\varepsilon\}$ (language of empty string)
- *A formal language* : a subset of Σ^*

Note: in place of Epsilon (ε) use Lambda (λ)

Example 1.9

Let $\sum = \{a, b\}$. Then

$$\Sigma^* = \{\lambda, a, b, aa, ab, ba, bb, aaa, aab, ...\}.$$

The set

$$\{a, aa, aab\}$$

is a language on $\sum$. Because it has a finite number of sentences, we call it a finite language. The set

$$L = \{a^n b^n : n \geq 0\}$$

is also a language on $\sum$. The strings *aabb* and *aaaabbbb* are in the language L, but the string *abb* is not in L. This language is infinite. Most interesting languages are infinite.

Since languages are sets, the union, intersection, and difference of two languages are immediately defined. The complement of a language is defined with respect to $\sum^*$; that is, the complement of L is

$$\overline{L} = \Sigma^* - L.$$

The reverse of a language is the set of all string reversals, that is,

$$L^R = \{w^R : w \in L\}.$$

The concatenation of two languages L_1 and L_2 is the set of all strings obtained by concatenating any element of L_1 with any element of L_2; specifically,

$$L_1L_2 = \{xy : x \in L_1, y \in L_2\}.$$

We define L^n as L concatenated with itself n times, with the special cases

$$L^0 = \{\lambda\}$$

and

$$L^1 = L$$

for every language L.

Finally, we define the **star-closure** of a language as

$$L^* = L^0 \cup L^1 \cup L^2 \cdots$$

and the **positive closure** as

$$L^+ = L^1 \cup L^2 \cdots.$$

- If L = {λ, ab, abab …..} then find L^2

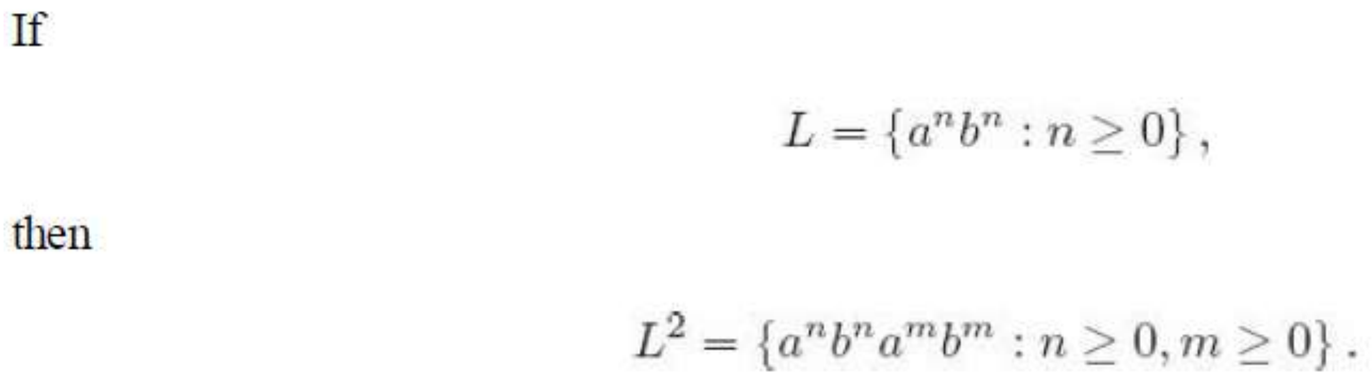
  L^2 = L.L = {λ, ab, abab …..} {λ, ab, abab …..}

  = {λ, ab, abab …..}

  = L

  Therefore L^2 = L

Now, L = {λ, ab, a^2b^2, a^3b^3 …..} then find L^2



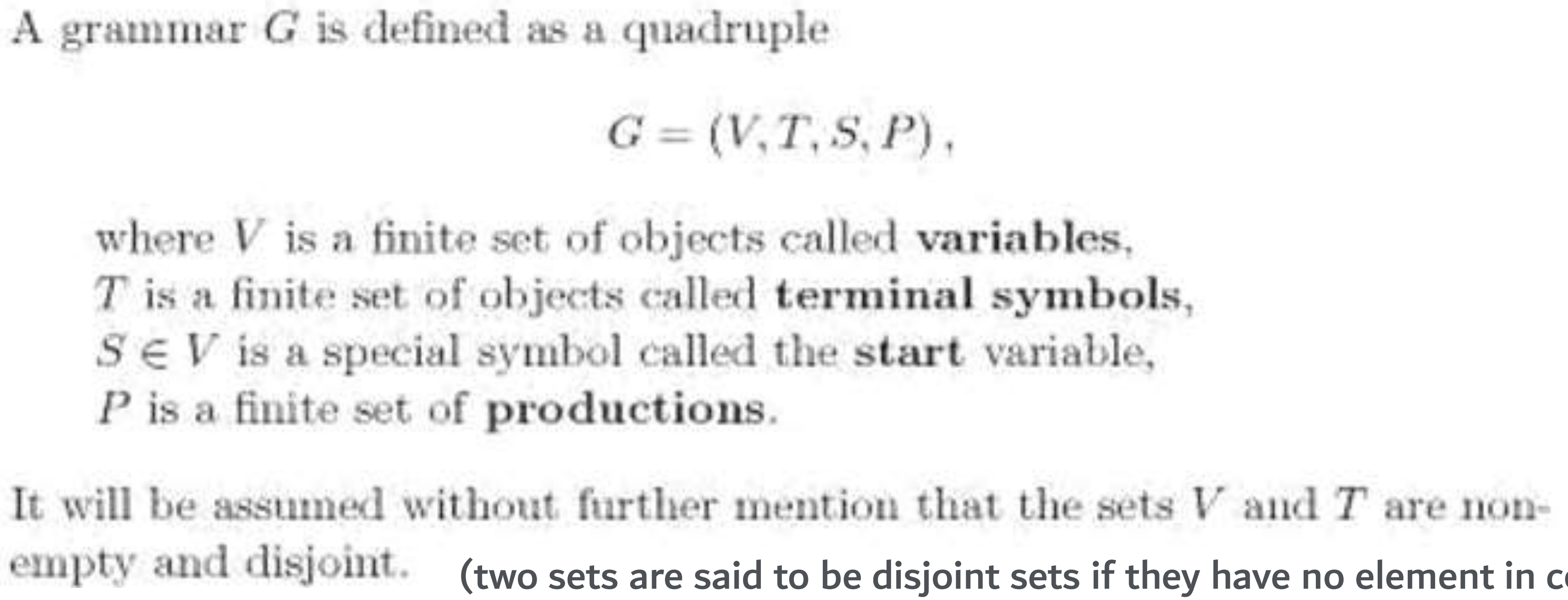
If

$$L = \{a^n b^n : n \geq 0\},$$

then

$$L^2 = \{a^n b^n a^m b^m : n \geq 0, m \geq 0\}.$$

Grammars



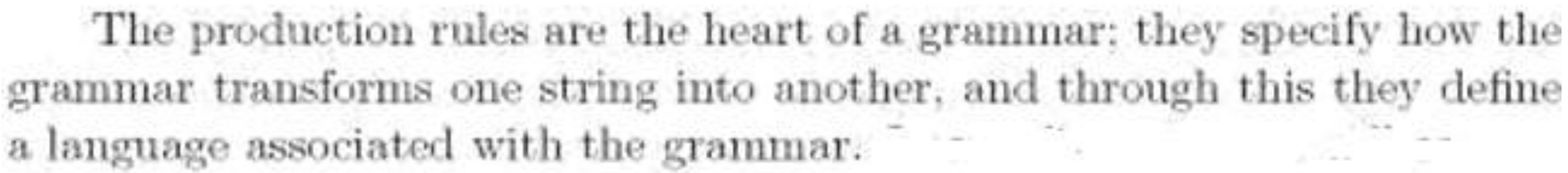
A grammar G is defined as a quadruple

$$G = (V, T, S, P),$$

where V is a finite set of objects called **variables**,
T is a finite set of objects called **terminal symbols**,
$S \in V$ is a special symbol called the **start** variable,
P is a finite set of **productions**.

It will be assumed without further mention that the sets V and T are non-empty and disjoint.

(two sets are said to be disjoint sets if they have no element in common.)



The production rules are the heart of a grammar; they specify how the grammar transforms one string into another, and through this they define a language associated with the grammar.

production rules are of the form

$$x \to y,$$

where x is an element of $(V \cup T)^+$ and y is in $(V \cup T)^*$. The productions are applied in the following manner: Given a string w of the form

$$w = uxv,$$

we say the production $x \to y$ is applicable to this string, and we may use it to replace x with y, thereby obtaining a new string

$$z = uyv.$$

The production x → y implies

$$w \Rightarrow z.$$

ie. **w** derives **z** or **z** is derived from **w**

$$w_1 \Rightarrow w_2 \Rightarrow \cdots \Rightarrow w_n,$$

if

we say that w_1 derives w_n and write

$$w_1 \overset{*}{\Rightarrow} w_n.$$

The $*$ indicates that an unspecified number of steps (including zero) can be taken to derive w_n from w_1.

Let $G = (V, T, S, P)$ be a grammar. Then the set

$$L(G) = \left\{ w \in T^* : S \overset{*}{\Rightarrow} w \right\}$$

is the language generated by G.

If the strings contains variables and terminals then they are called as **sentential form**

Example:

$$\left.\begin{array}{l} S \rightarrow aAb \mid \lambda \\ A \rightarrow aAb \mid \lambda \end{array}\right\} \quad \textbf{productions}$$

S → aAb → aaAbb → aabb   in this aaAbb and aAb are in sentential form

a , b are terminals

A - Variables

aabb is a string ie. w

- G = ({S}, {a,b}, S, P) with P given by

S $\rightarrow$ aSb | λ

Option 1:

S $\rightarrow$ λ

S $\rightarrow$ aSb

S $\rightarrow$ a λb

S $\rightarrow$ ab

Option 2:

S $\rightarrow$ aSb

S $\rightarrow$ a aSbb

S $\rightarrow$ aa λbb

S $\rightarrow$ aabb

The language generated by this grammar is

L= { $a^n b^n$ | n >= 0}

L = { λ, ab, aabb, aaabbb…..}

- <u>Language Definition:</u>

If G = (V, T, S P) is a grammar then language L generated by grammar G is

$L(G) = \{ w \in T^* \mid S \overset{*}{\rightarrow} w \}$

<u>Example 1:</u>

G = ({ S,A}, {a,b}, S, P}

$S \rightarrow Ab$

$A \rightarrow aAb$

$A \rightarrow \lambda$

$S \rightarrow Ab \rightarrow aAbb \rightarrow aaAbbb \rightarrow aa\ A\ bbb \rightarrow aaa\ A\ bbbb \rightarrow aaa\ \lambda\ bbbb \rightarrow aaabbbb$

So

$L = \{ a^n b^{n+1} \mid n >= 0\}$

- Let G be a grammar G = ({S}, {a,b}, S, P)

$S \rightarrow SS$

$S \rightarrow aSb$

$S \rightarrow bSa$

$S \rightarrow \lambda$

$S \rightarrow SS \rightarrow aSbS \rightarrow aaSbbS \rightarrow aabbS \rightarrow aabb$

$S \rightarrow SS \rightarrow SaSb \rightarrow S\ abSa\ b \rightarrow abSab \rightarrow abab$

So The language generated by this grammar is

$L = \{ w \mid n_a (w) = n_b (w) \}$

- <u>Equivalent Grammar</u>

Let G1 and G2 be any 2 grammars. Then if L (G1) = L (G2) then 2 grammars are equivalent.

Let G1 be…..

$$\boxed{S \rightarrow aSb \mid \lambda}$$

$S \rightarrow aSb \rightarrow aaSbb \rightarrow aa\,\lambda\,bb \rightarrow aabb$

$$\boxed{L = \{ a^n b^n \mid n >= 0\}}$$

Let G2 be…..

$$\boxed{\begin{array}{l} S \rightarrow aAb \mid \lambda \\ A \rightarrow aAb \mid \lambda \end{array}}$$

$S \rightarrow aAb \rightarrow aaAbb \rightarrow aa\,\lambda\,bb \rightarrow aabb$

$$\boxed{L = \{ a^n b^n \mid n >= 0\}}$$

So L (G1) = L (G2)

Some Examples

Find the Language generated by this grammar

1. $G = (\{S\}, \{a,b\}, S,P)$

   $S \rightarrow SS \mid aSb \mid bSa \mid \lambda$

<u>Find the Grammar for the Language</u>

2. $L = \{ (01)^n \mid n >= 0\}$ or $L= \{ w \mid w \in \{01\}^* \}$

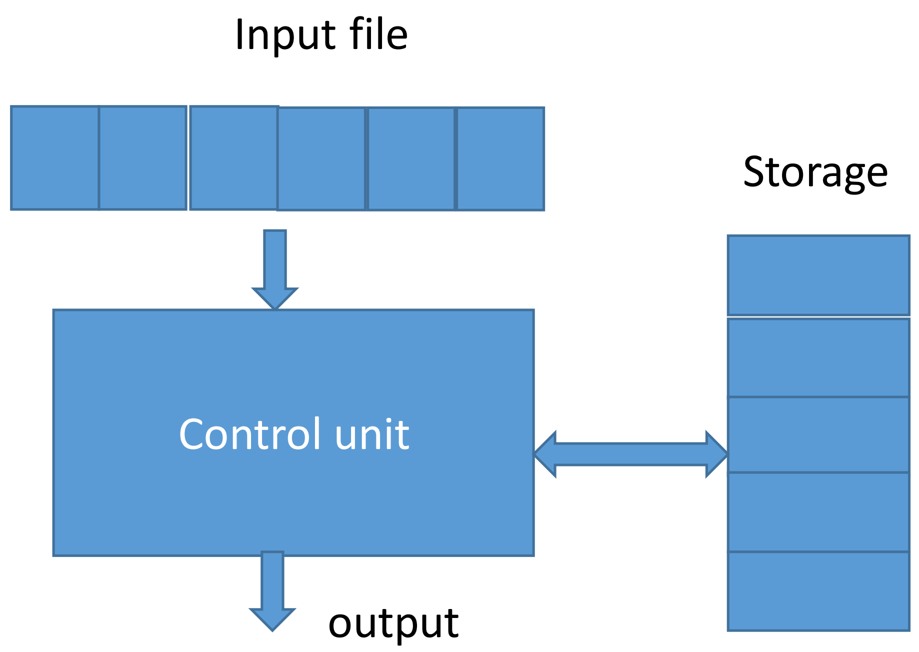
3. $L = \{ ww^r \mid w \in \{a,b\}^*\}$

4. $L = \{ 0^n 1^{2n} \mid n >= 0\}$

5. $L = \{ a^n b^{n-3} \mid n >= 3\}$

6. $L = \{ w \mid |w| \bmod 3 = 0 \}$

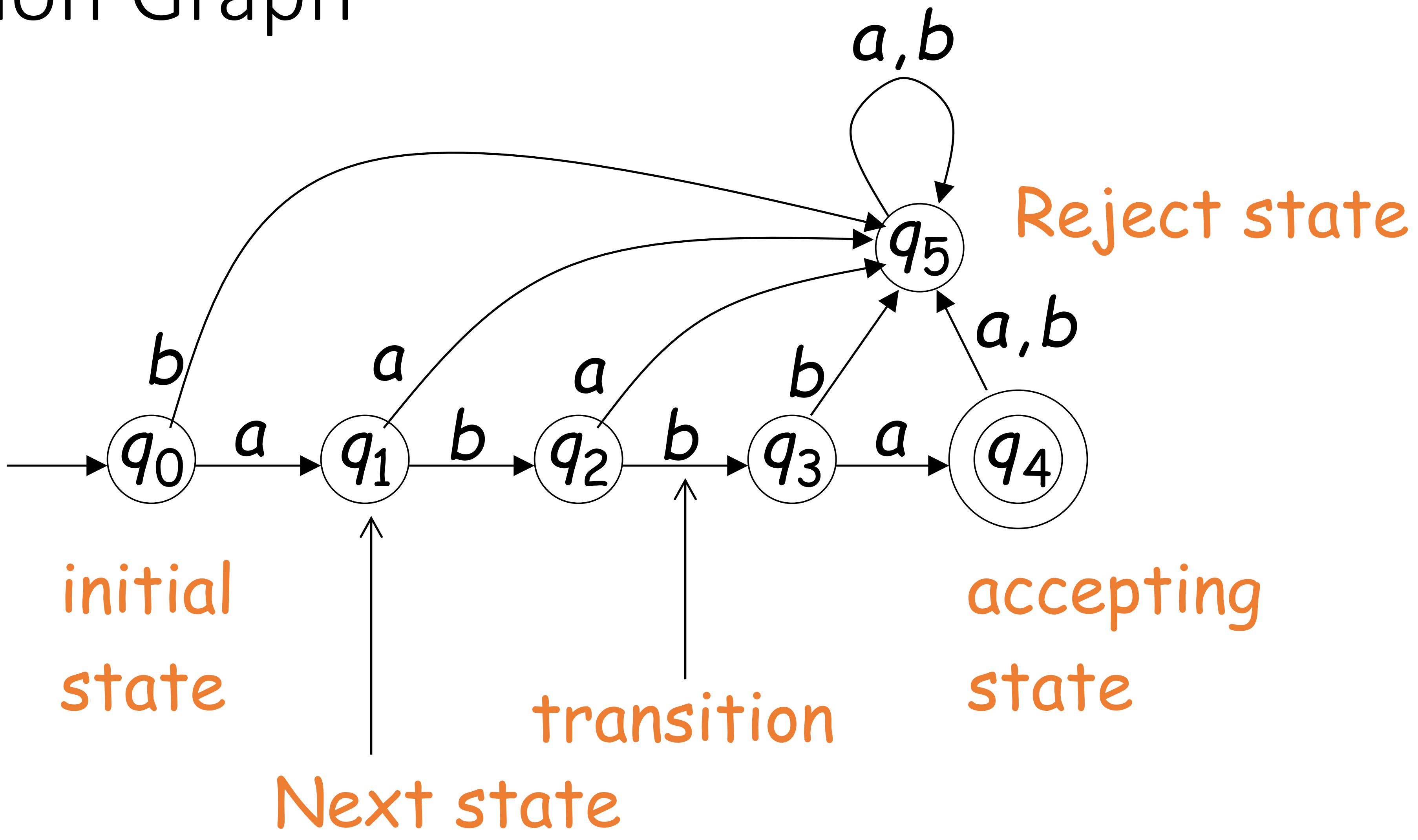
Automata



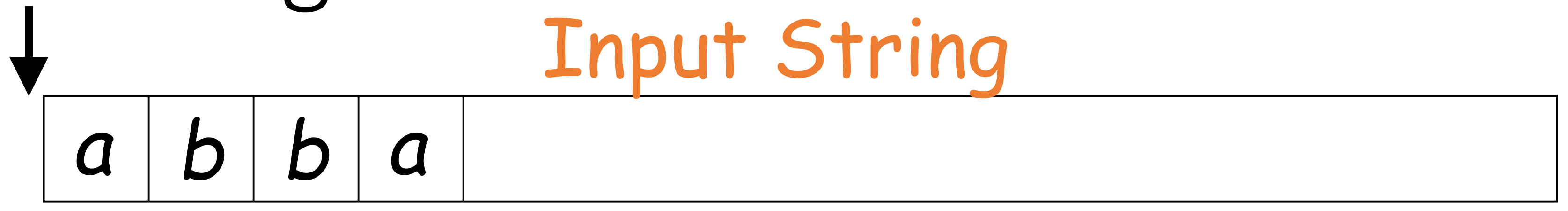
- Abstract model of digital computer
- Input file is divided into cells each can hold one symbol
- Read the input file from left to write one at a time
- Temporary storage device with unlimited no. of cells each capable of holding one symbol
- Automaton can read and change the content of storage cells

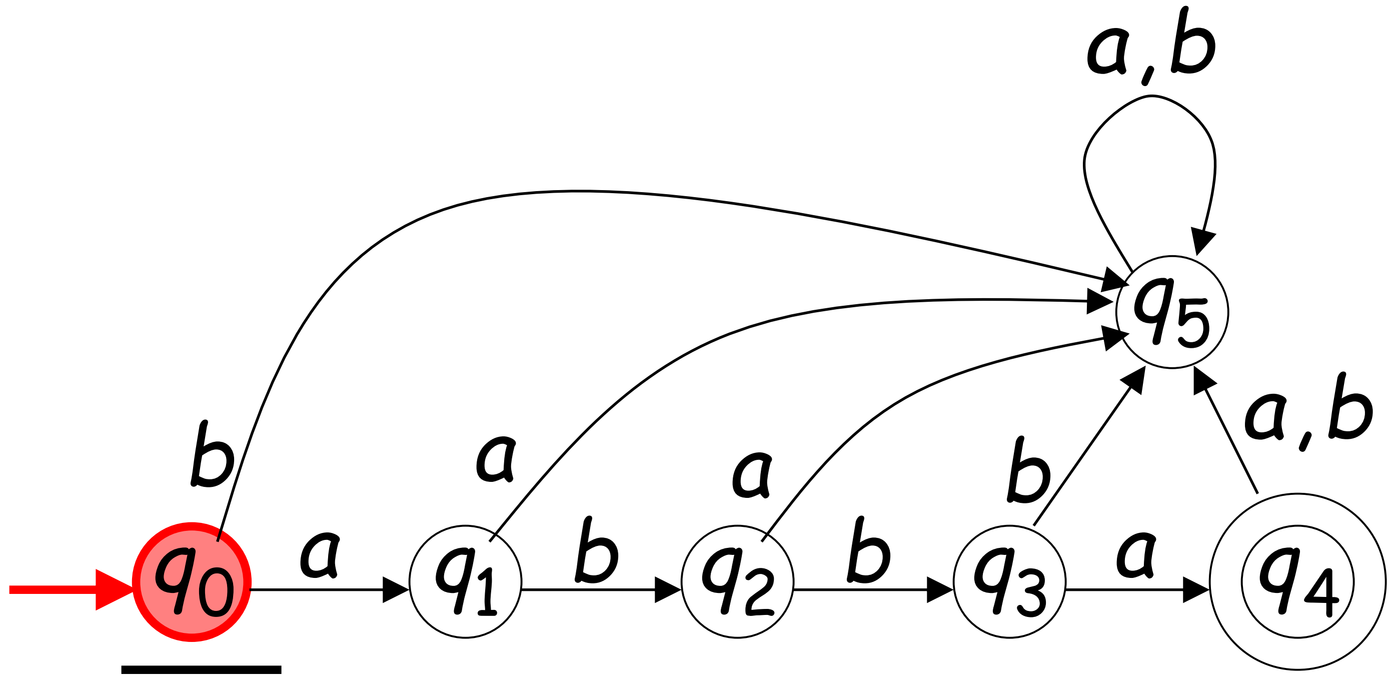
- Control unit will be in some internal state
- The internal state of CU at the next time step is determined by **transition function**
- Transition function gives the next state in terms of current state, current input symbol, current information in the storage.
- The term **configuration** is used to refer to a particular state of CU, input file and temporary storage.
- The transition of automaton from one configuration to next is called a **move**.

Transition Graph

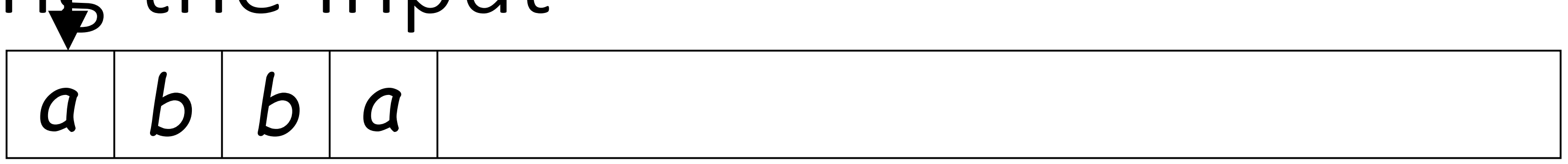


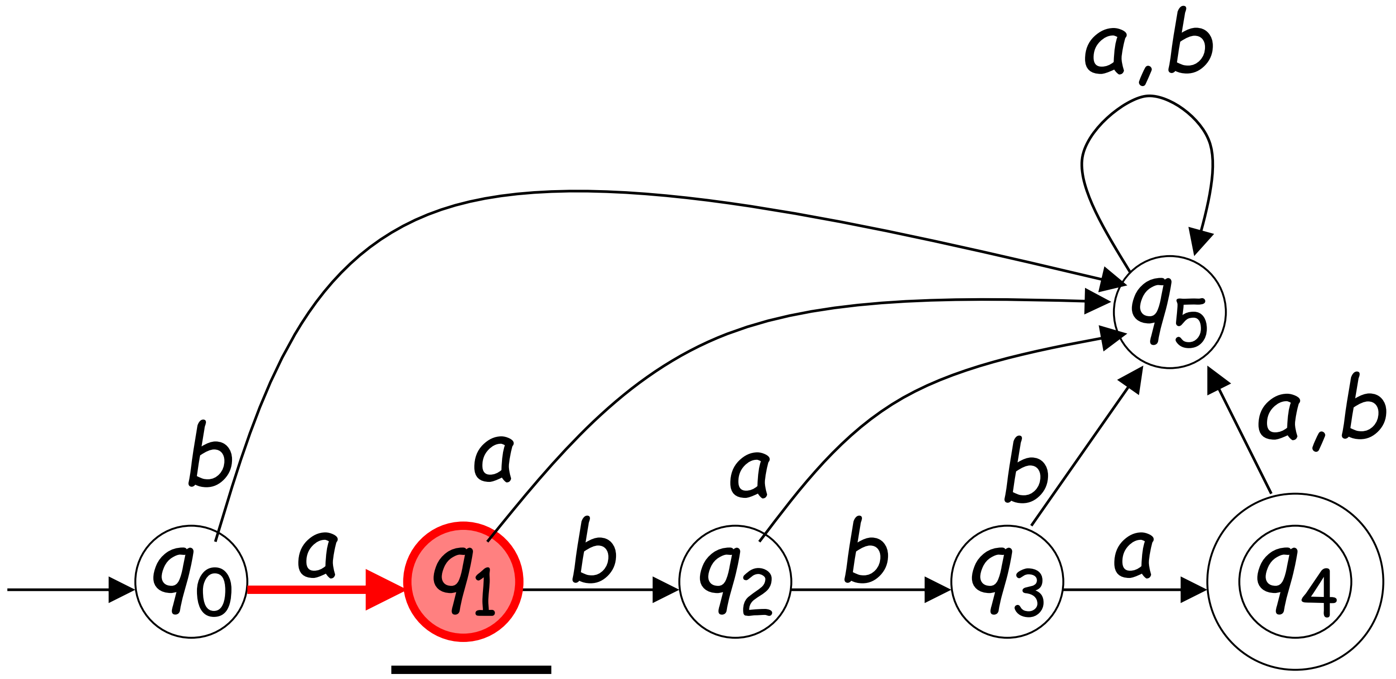
Initial Configuration

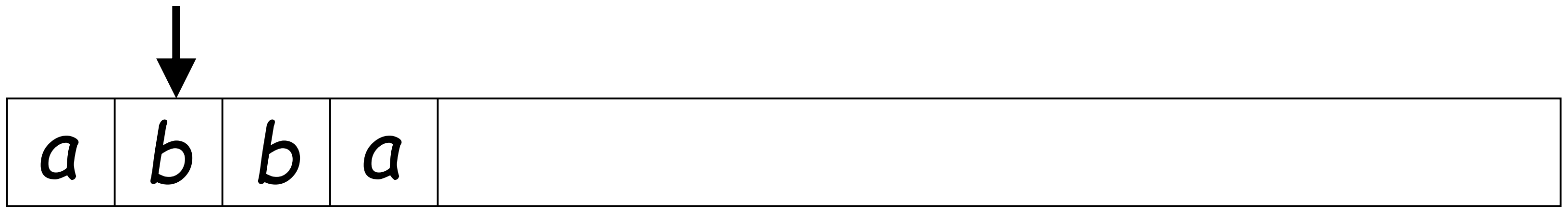




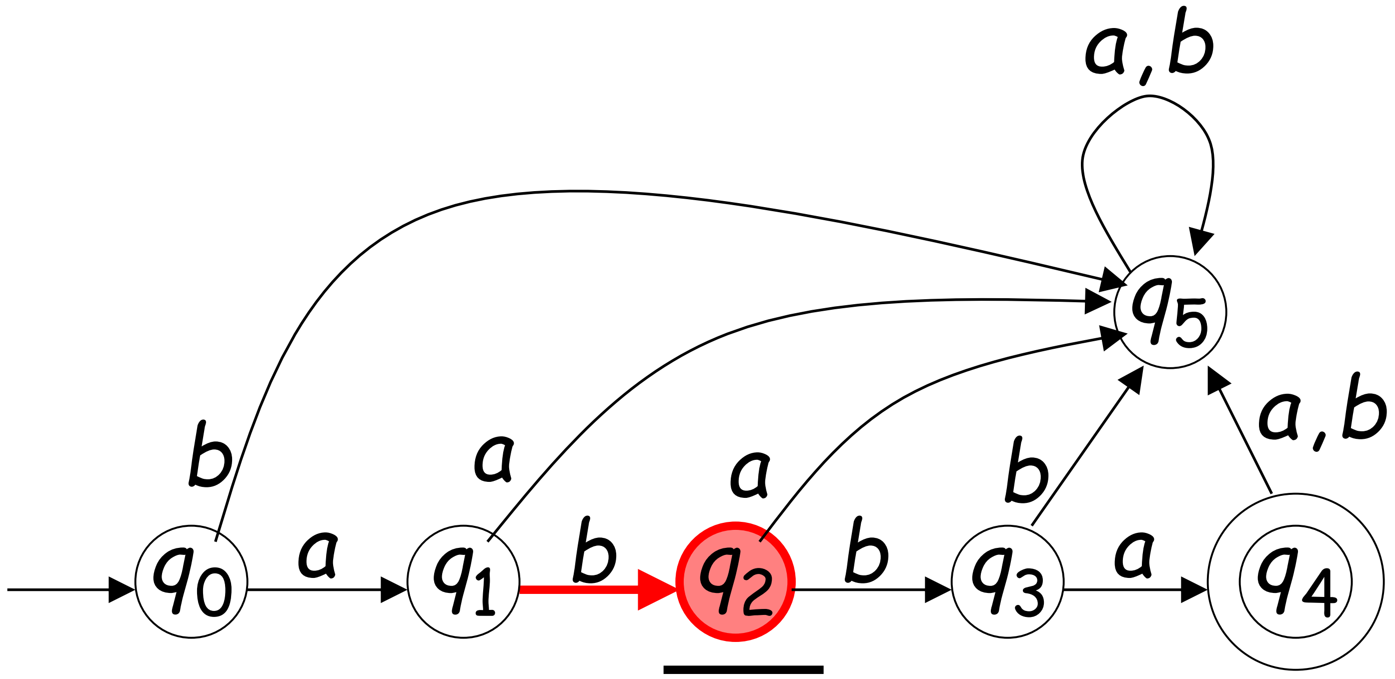
Reading the Input

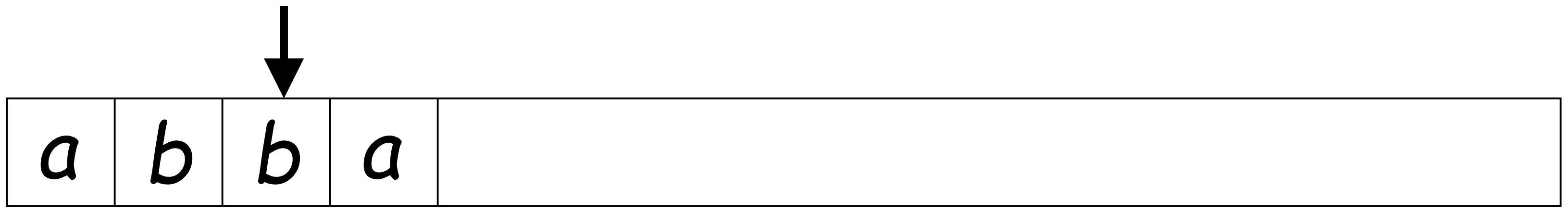


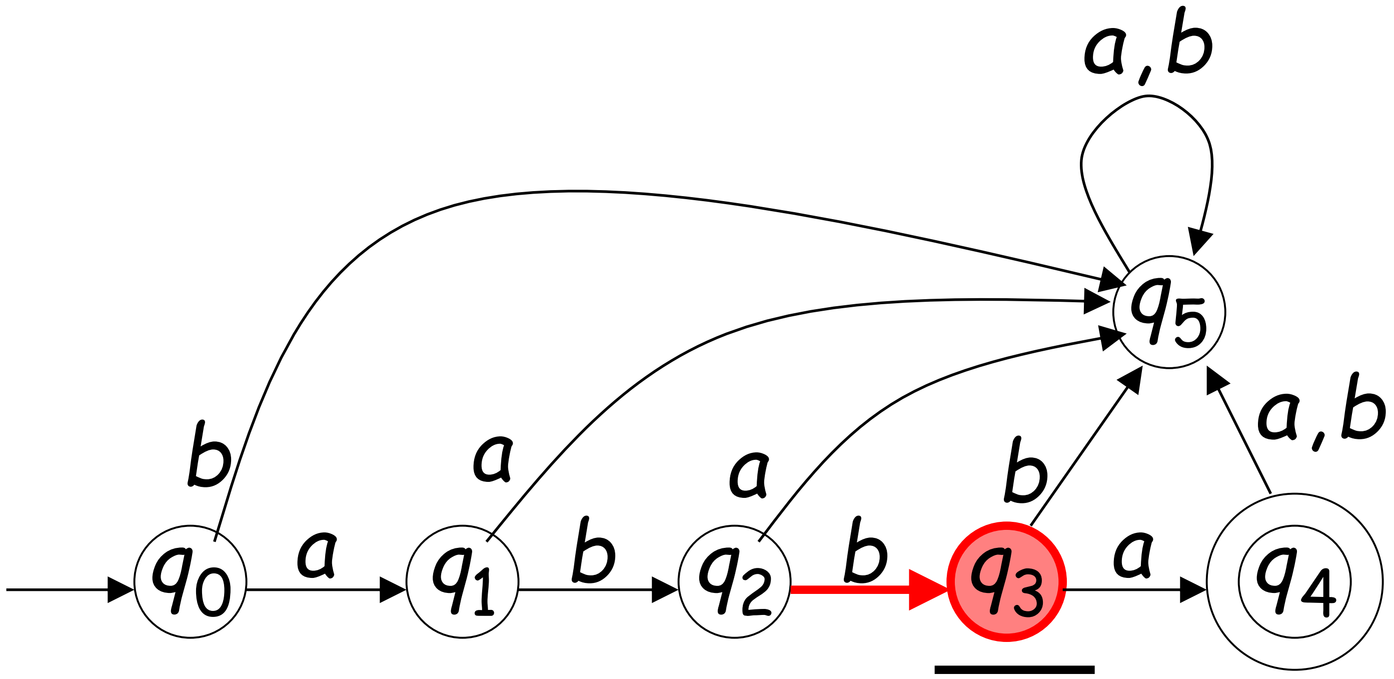


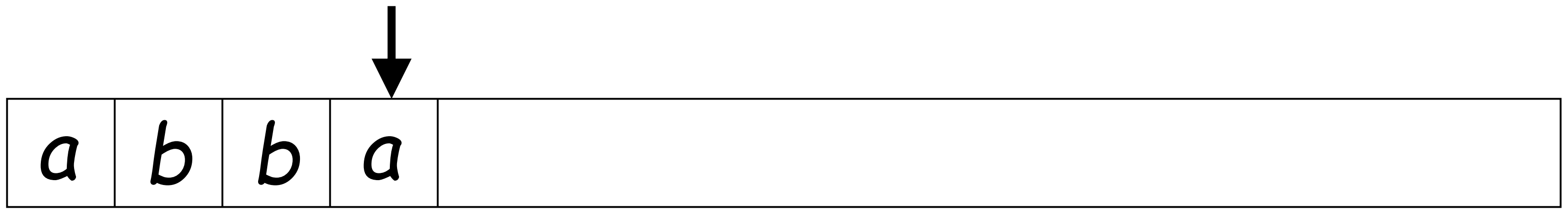


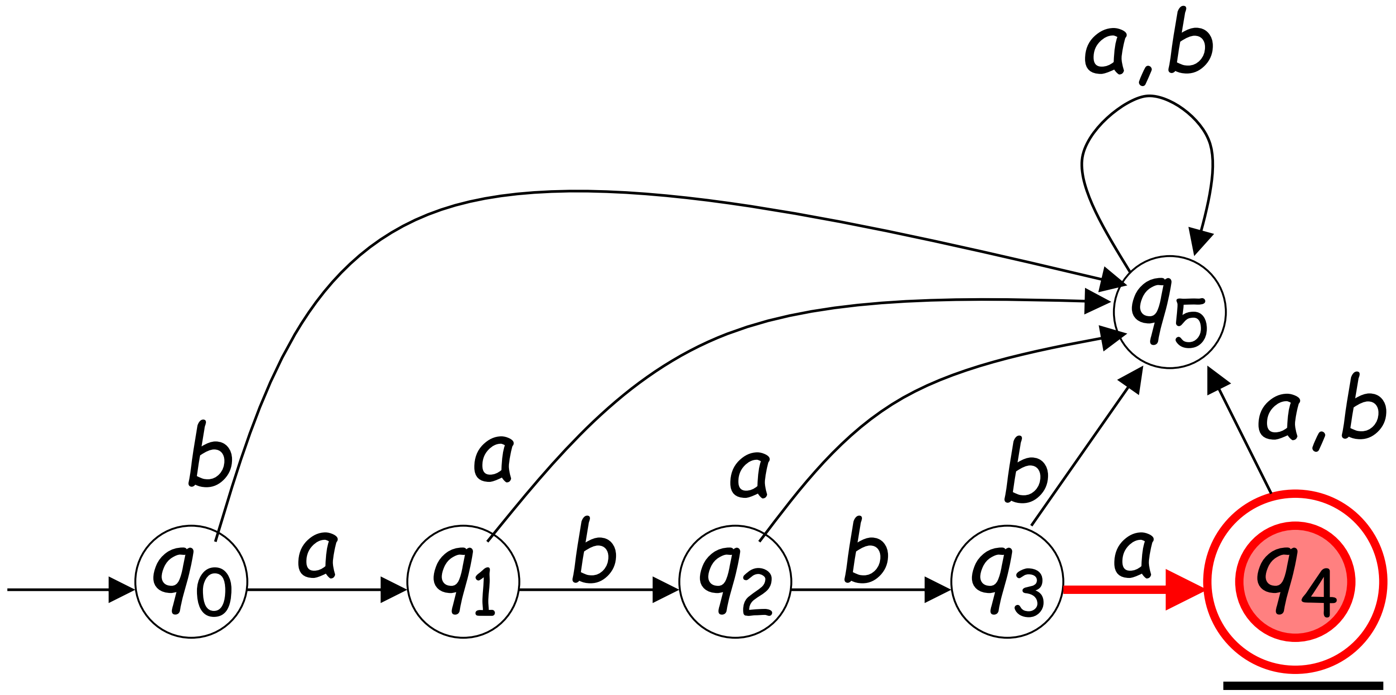
a	b	b	a	



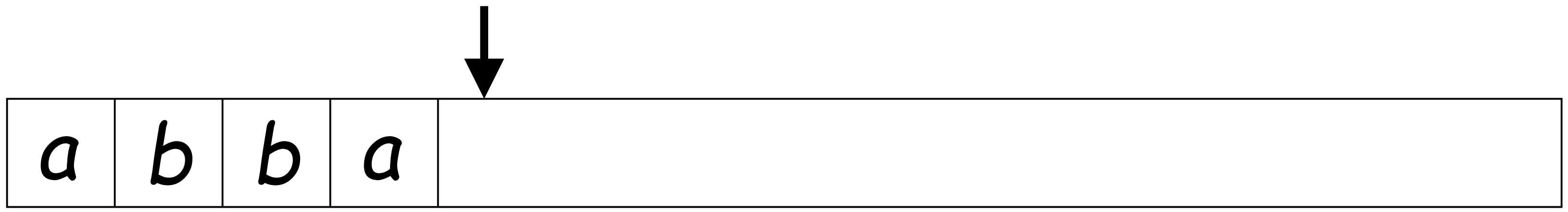

a
b
b
a

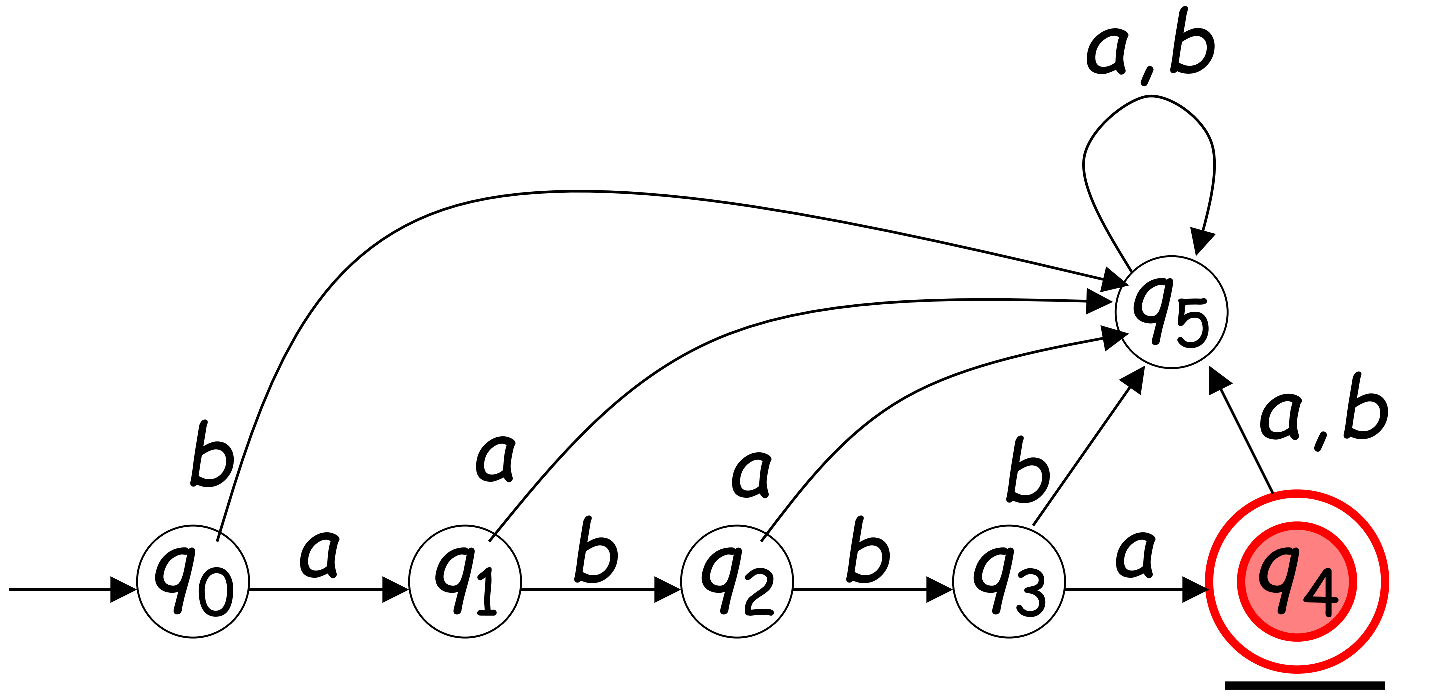

a,b
q5
b
a
a
b
a,b
q0
a
q1
b
q2
b
q3
a
q4


a
b
b
a


a,b
q5
b
a
a
b
a,b
q0
a
q1
b
q2
b
q3
a
q4

Input finished



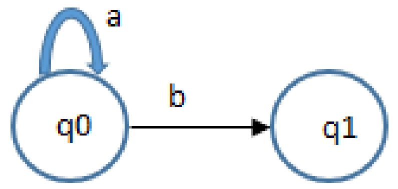


accept

Two types of Automata

1. Deterministic Automata (DFA)
2. Nondeterministic Automata (NFA)

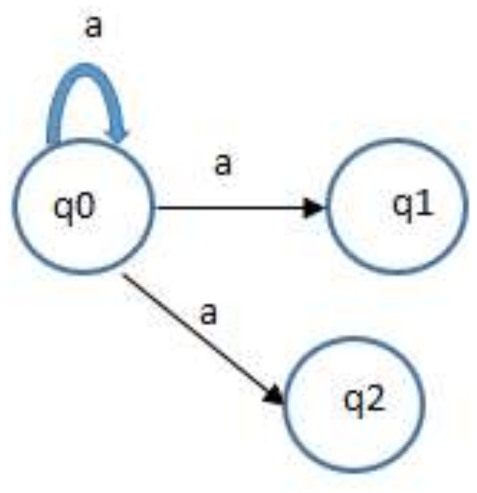
DFA – In this each move is uniquely determined by the current configuration.



δ (q0 ,a) = q0

δ (q0 ,b) = q1

NFA – It has several possible moves. So we can predict a set of possible actions.

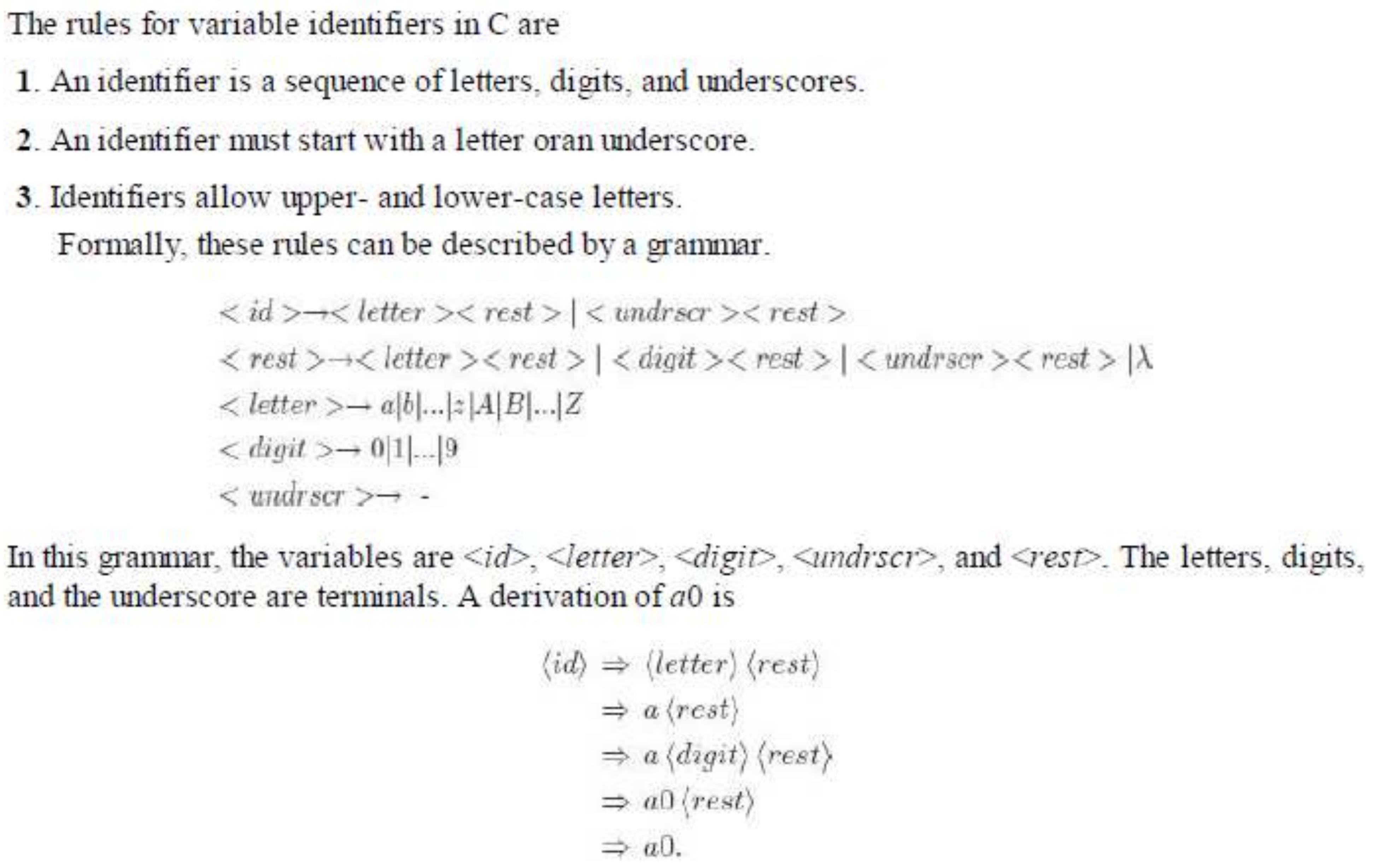


δ (q0 ,a) = { q0, q1, q2 }

Accepter: An automaton whose output response is limited to sample “Yes” or “No” is called accepter ie. Either accept the string or rejects it.

Transducer: Automaton capable of producing strings of symbols as output .

- Some Applications



The rules for variable identifiers in C are

1. An identifier is a sequence of letters, digits, and underscores.
2. An identifier must start with a letter oran underscore.
3. Identifiers allow upper- and lower-case letters.

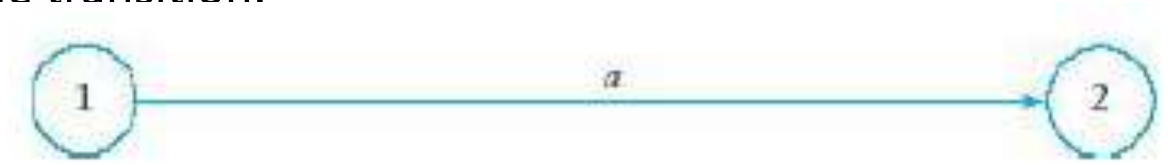
Formally, these rules can be described by a grammar.

$$
\begin{aligned}
&< id > \rightarrow < letter >< rest > \mid < undrscr >< rest > \\
&< rest > \rightarrow < letter >< rest > \mid < digit >< rest > \mid < undrscr >< rest > \mid \lambda \\
&< letter > \rightarrow a|b|...|z|A|B|...|Z \\
&< digit > \rightarrow 0|1|...|9 \\
&< undrscr > \rightarrow \text{-}
\end{aligned}
$$

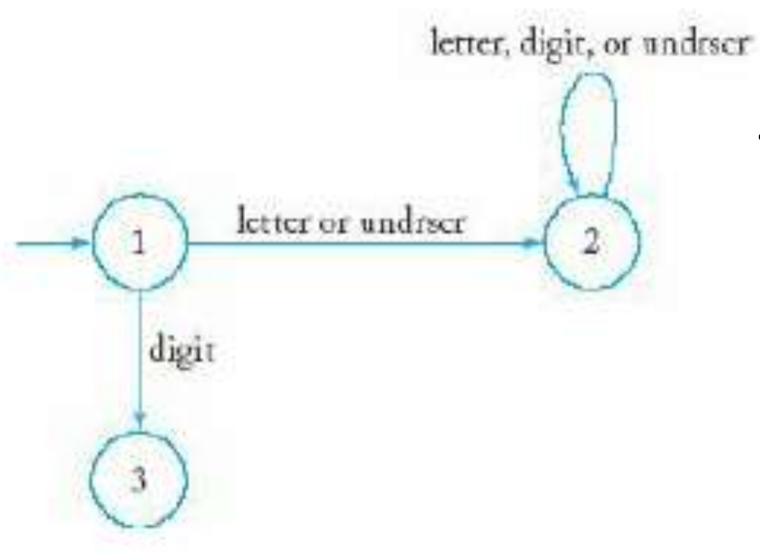
In this grammar, the variables are <*id*>, <*letter*>, <*digit*>, <*undrscr*>, and <*rest*>. The letters, digits, and the underscore are terminals. A derivation of $a0$ is

$$
\begin{aligned}
\langle id \rangle &\Rightarrow \langle letter \rangle \langle rest \rangle \\
&\Rightarrow a \langle rest \rangle \\
&\Rightarrow a \langle digit \rangle \langle rest \rangle \\
&\Rightarrow a0 \langle rest \rangle \\
&\Rightarrow a0.
\end{aligned}
$$

- An **automaton** can be represented by a **graph** in which the **vertices** give the **internal states** and the **edges** give the **transitions**. The **labels** on the **edges** show what happens (in terms of input and output) during the transition.



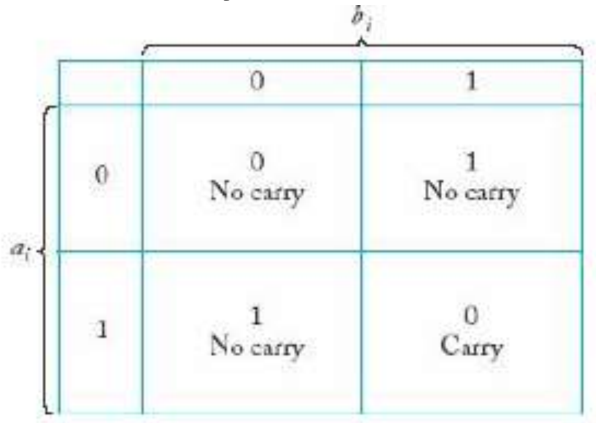
- The above Figure represents a transition from State 1 to State 2, which is taken when the input symbol is *a*.



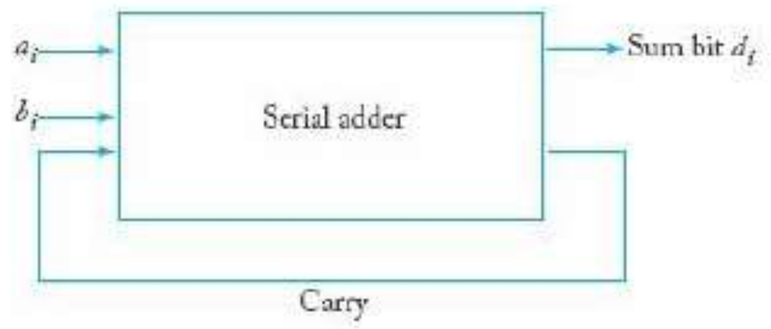
This is an automaton that accepts all legal C identifiers. When the **first** symbol is a **letter** or an **underscore**, the automaton goes into State 2, after which the rest of the string is immaterial. State 2 therefore represents the "yes" state of the accepter. Conversely, if the **first** symbol is a **digit**, the automaton will go into State 3, the "no" state, and remain there.

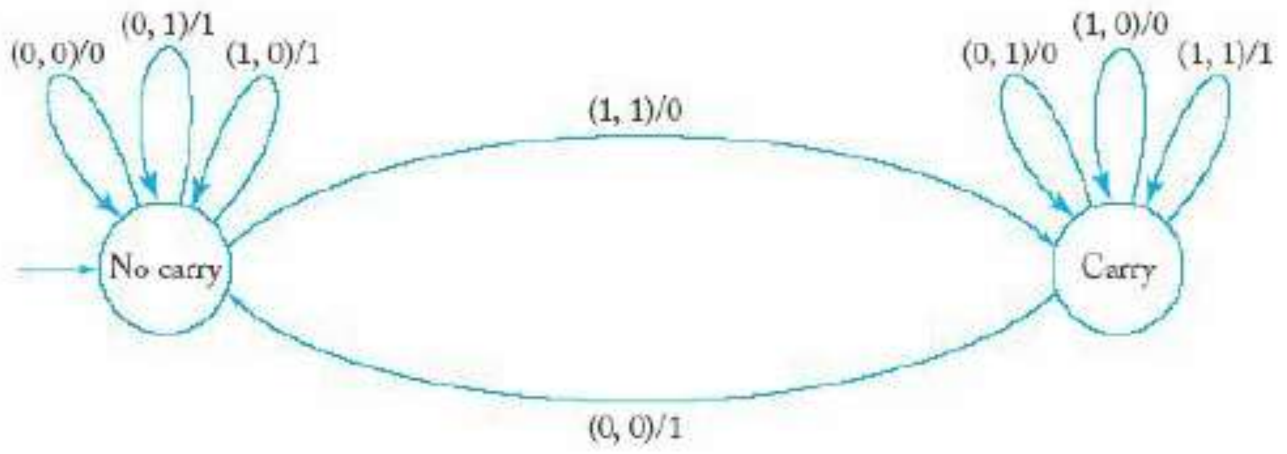
Serial Binary Adder:

→ A binary addition table summarizes the process



a_i \ b_i	0	1
0	0 No carry	1 No carry
1	1 No carry	0 Carry

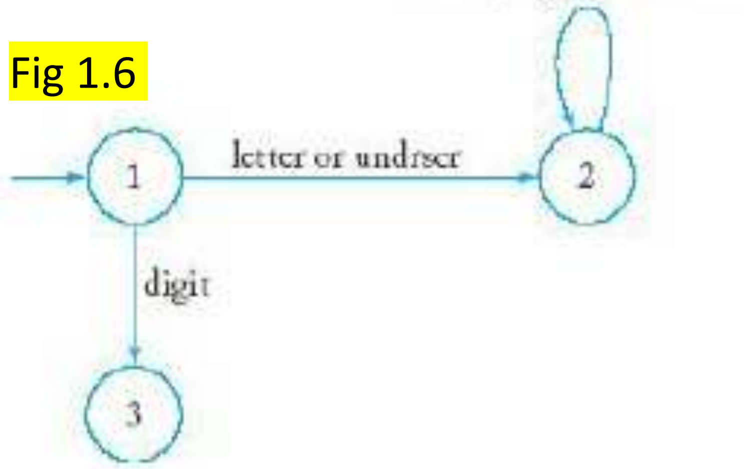




We represent the automaton by a graph now labeling the edges (a_i, b_j)/d_i.

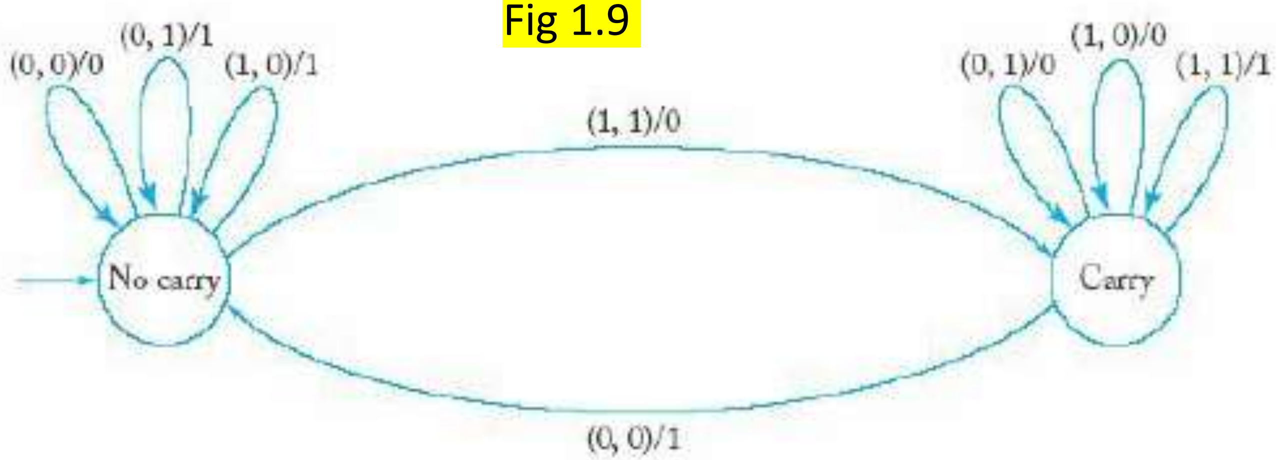
Comparison Acceptor and Transducer

Fig 1.6



Acceptor

Fig 1.9



Transducer

- Both have a finite number of internal states.
- Both process an input string, consisting of a sequence of symbols.
- Both make transitions from one state to another, depending on the current state and the current input symbol.
- Both produce some output, but in a slightly different form. The automaton in Figure 1.6 only accepts or rejects the input; the automaton in Figure 1.9 generates an output string.