THEORY OF AUTOMATA

What does Theory of Automata mean?

- The word **"Theory"** means that this subject is a more mathematical and less practical.
- It is not like other courses such as programming. However, this subject is the foundation for many other practical subjects.
- In general, this subject focuses on the theoretical aspects of computer science.
- The word **"Automata"** derived from Greek word automaton which means "self-acting" / "something that works automatically".
- **Automation** like the real system accepts input, produces output having some temporary storage and can make decisions about input to output transformation.

For Example: ATM Machine

- Enter card and password
- Check amount
- Gives money (if have)
- Return card

Why we study Theory of Automata / Applications of Theory of Automata?

- The aim of this course (Theory of Automata) is to design such machine that works automatically. **Like** ATM machine.
- Behind all computer languages such as C, C++, JAVA etc., the theory that work is Theory of Automata.

This subject plays a major role in:

- Theory of computation
- Compiler construction
- Parsing
- Formal verification
- Defining computer languages

Importance of Automata Theory

Automata theory is a fundamental area of computer science and mathematics that explores abstract machines and the problems they can solve. Its importance lies in several key areas:

1. **Understanding Computation**: Automata theory provides a framework for understanding how machines process information, which is essential for designing efficient algorithms and systems.
2. **Formal Language Processing**: It is closely related to formal language theory, helping in the development of programming languages and compilers.
3. **Applications in AI and Verification**: Automata are used in artificial intelligence, parsing, and formal verification to ensure systems behave as intended.
4. **Problem Solving**: It aids in solving computational problems using mathematical models and techniques.

What is Language?

Language is a combination of Alphabets (∑) and Rules.

For Example:

If ∑ = {a , b} ***(Alphabets)***

All string which end with "**a**" ***(Rule)***

Types of Language:

There are two types of languages that are:

- Formal Languages (Syntactic languages)
- Informal Language (Semantic languages)

1. Formal Languages:

- Formal languages are such languages that we can represent it in mathematical form.
- They are used as a basis for defining computer languages.
- They consist of Predefined set of symbols and strings. (it means outside those symbols and strings nothing is allowed in that language)
- Formal languages are called **syntactic languages** because it only concern with structure. (it means that let's we have a word **aabd**, now we will check that this word is allowed in that particular language, if yes than we don't need to know it's meaning).

2. Informal Language:

Informal languages are such languages that we can't represent it in mathematical form.

For example: English, Urdu etc.

- They have not proper structure, with the passage of time new words are introduced.

- ❖ Informal languages are called **semantic languages** because it only concern with meaning.

⇨ ***Now to study formal language further, that how we represent it, we must know some basic elements of formal language that are:***

1: Alphabets:

A finite non-empty set of symbols is called alphabets.

It should not contain empty symbol Λ.

It is denoted by Greek letter sigma (∑)

For Example:

- ∑ = {a, b, c}
- ∑ = {1, 2, 3}
- ∑ = {0, 1} // Binary Digits

2: Letter:

The **symbols** in the **set of alphabets** is called letter.

For Example:

- ∑ = {a,b}
- ∑ = {0,1}

In the above examples the **"a, b, 0 and 1"** are letters.

3: String:

Any combination of alphabets but it should be finite is called string.

OR we can say that by joining the letters we get string.

For Example:

- ∑ = {a,b} aaabbbba , bbbbaaab ***(These are strings)***
- ∑ = {0,1} 00011110 , 11101111 ***(These are strings)***

4: Word:

A string that belongs to a language is called word.

For Example:

All strings ending with "a" if ∑ = {a, b} ***(Here we defined language)***

Strings:

- **aaba**
- **bbab**
- **bbba**

In the above strings **“aaba”** and **“bbba”** are words but **“bbab”** is not a word because it not follows language. :

Note: All words are strings but all strings are not words.

5: Length of string:

The number of symbols present in string is called length of string.

It is denoted by |S| or |w|.

For Example:

- If ∑ = {a, b} **what will be the length of string “aaaba**”?
- The easy and accurate method to finding the length of string is, to tokenize the string as;
- (a) (a) (a) (b) (a) // **now count it**.
- The length of the string “aaaba” is **5**.

6: Empty / Null string:

A string having no character / symbol is called empty / null string.

It is denoted by Small Greek lambda (λ) **OR** Capital Greek lambda (Λ) **OR** epsilon (**ε**).

|S| = 0

7: Prefix of a string

A string B is called a prefix of a string A if it is obtained by removing 0 or more trailing symbols of A.

For example:

- A = abcd
- B = abc is prefix of A.
- Here, B is proper Prefix if B ≠ A.

8: Suffix of a string

A string B is called a suffix of a string A if it is obtained by removing 0 or more leading/starting symbols in A.

For example:

- A= abcd

- B= bcd is suffix of A.
- Here, B is proper suffix if B ≠ A.

9: Substring:

The part of a string is called substring.

For Example:

- The list of all substrings of the string **"apple"** would be "apple", "appl", "pple", "app", "ppl", "ple", "ap", "pp", "pl", "le", "a", "p", "l", "e", "".

OR

A string ***B*** is called substring of a string ***A*** if it is obtained by removing 0 or more leading or trailing symbols in ***A***. It is proper substring of ***A*** if ***B*** ≠ ***A***.

SOME OPERATIONS ON STRING

1: Concatenation of String:

When two or more string combine together ***in such order as they are given*** and form a new string is known as **concatenation of string.**

It is denoted by dot.

For Example:

- S1 = ab and S2 = bc
- S1.S2 = ab.bc or abbc

> **Note:** S1.S2 is not equal to S2.S1*(Because S1.S2 gives **"abbc"** but S2.S1 gives **"bcab")***

2: Union of String:

It gives you the facility / option to choose the desire string.

It is denoted by "**U**", "**+**", "**,**", "|".

For Example:

Union of string 1 and string 2 (S1 = aa and S2 = bb)

Will be

S1 | S2 = aa | bb **(Now we will only choose one string, either S1 or S2)**

3: Reverse of string:

Putting or writing back a given string from right to left is called reverse of string.

For Example:

- Let's we have a string **"aabbba**"
- Reverse of the string will be **(abbbaa)**

Note: in some cases we have such situations:

If ∑ = {aa,bba} Then we make a string "aabbaaa"

- Then make sure the reverse of such string will be "aabbaaa" not like it "aaabbaa" because we have symbols of "aa" and "bba" not "a" and "b". so it must be sure.
- The above reverse string will be more clear with tokenization (aa) (bba) (aa), now it easy to reverse each string.

4: Palindrome:

Putting or writing back a given string from right to left and gives the same string is called palindrome.

For Example:

- **aba** when we reverse it, it gives us the same string **"aba"**
- **bab** when we reverse it, it gives us the same string **"bab"**
- **madam** when we reverse it, it gives us the same string **"madam"**