Introduction to Theory of computation (TOC)

(or FLAT, formal languages and automata theory)

and Compiler design

(by K.Revanth, course Instructor)

- Originally computers were not very powerful, people wrote machine instructions by hand in order using 0,1’s and fed them to the processor
- Then human readable assembly language was created and the programs written in assembly language are run through an assembler that maps assembly instructions into machine level instructions

Ex:- Add R1,R2 and Mov R1,R2… instead of series of 16 or 32 bits of 0,1’s

- Eventually programmers realized we need better support to write programs, and the idea of writing programs in specific languages was born
- Programs and Languages need Compilers to understand the language and convert them to binary instructions. And so early day compilers were written in assembly language and for next versions they used their own compilers to generate further versions of their compilers which were written in the same language itself (Bootstrapping)
- Somewhere in that time operating systems and general purpose computers were born too

- Compiler design deals with how we can build a compiler for a given language grammar
- We will need basic definitions of what language and grammar are, those will be covered in TOC part along with some other concepts

Let's start with theory of computation basic terms…

- **Alphabet (Σ)** : a **FINITE SET** of symbols or characters

  Ex:- $\Sigma_1 = \{a,b\}$ or $\Sigma_2 = \{0,1\}$ or $\Sigma_3 = \{A,B,C,D\}$

  - Every language is defined over a set of alphabets and we usually denote that alphabet by Σ
- **String :** a FINITE length string which is made of characters from the given alphabet

  Ex:- "a", "aab", "baab" generated from using Σ = {a,b}

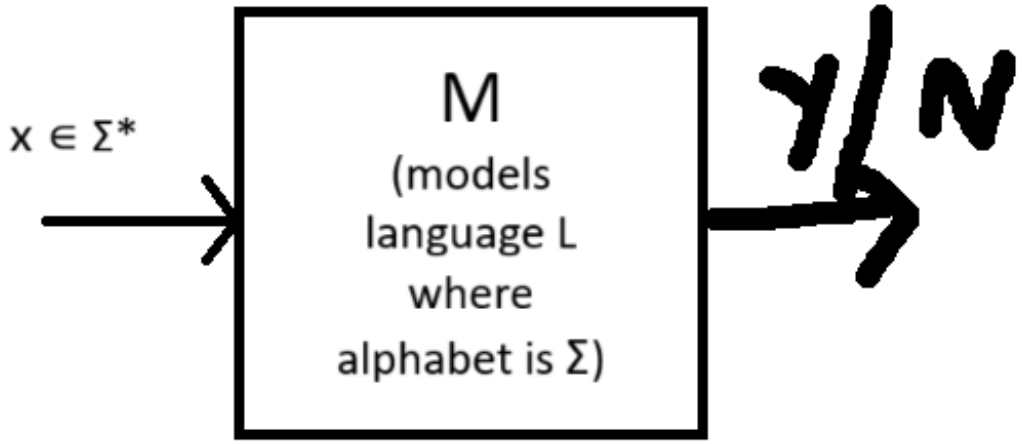
  - All strings are of finite length
  - We use **'ε'(epsilon)** to denote an **empty string** which has length = 0.

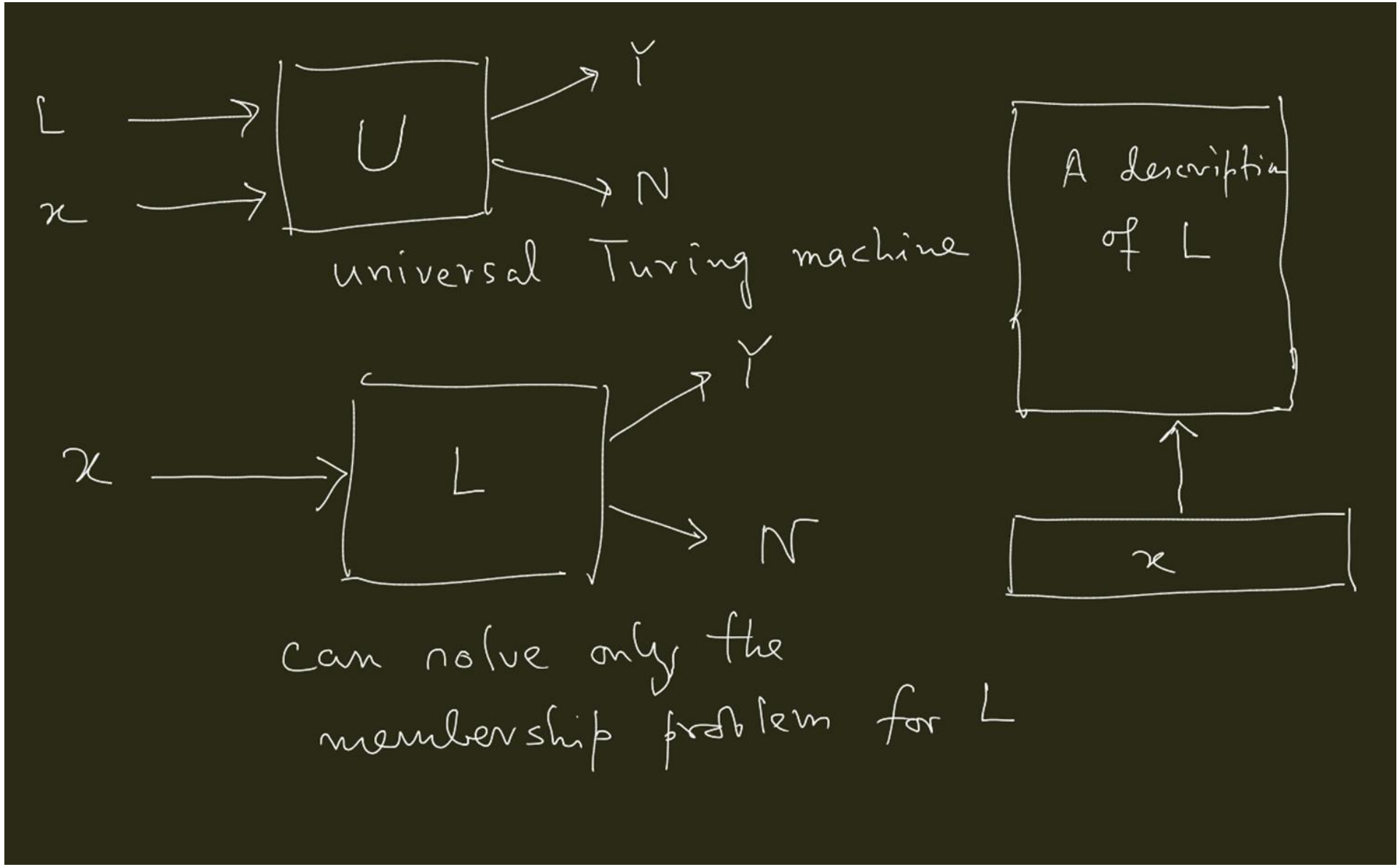
- **Σ* or U** (universal set) : we use sigma star to denote set of all possible strings that can be generated from given alphabet Σ.
  - Ex: given Σ = {a,b} then Σ* is set of all possible strings that are made from given Σ and including ε(empty string)
    - Σ* = {ε,"a","b", "ab", "ba", "aaa".....} set of all possible length strings made of Σ.

- Formal Definition of **Language :** given an alphabet set Σ, any subset L of U or Σ*.
  - Ex: given Σ = {a,b}
  - Language of all strings that does not contain 'b' is L = {x | x ∈ Σ* and x does not contain character 'b'}
  - We can see that L here is a subset of Σ*. L ⊆Σ*
  - Language is basically a set
- definition of a **Computational Problem :** given a Σ and a string x ∈ Σ* and a language L⊆Σ*

Decide whether string x ∈ L language or not

- When we talk about Computational problem (also called as Language-Membership problem), we will only deal with decision problems as opposed to functional problems because, we can always solve or prove computability of functional problems using only decision problems.
- **Computational Machine :**
  - For every language we can build separate machines that can model that language, and they take any particular string as input and outputs YES if the string is part of the given language or NO, after all a language is just a set of strings





Whenever we say machines in this course we will only talk about a machine representing a particular language (L)

It takes a string $x \in \Sigma^*$ and outputs whether or not $x \in L$

Later there will be Universal Turing Machines that are more capable of Taking both a finite representation of a language L and a string and solve the language membership problem

Another reason to say a language description has to be finite

- Sounds easy, but this is not an algorithms course.
- How do we represent these languages and computational problem and machines formally and mathematically?
- We can represent everything in a finite strings, by encoding in strings of some finite alphabet set.

- Any description of Language L must be finite.
- If L set is finite – no problem, we can just give the finite set itself
- but infinite – we will see…

Finite description of languages

- Given $L \subseteq \Sigma^*$ and $x \in \Sigma^*$, decide whether $x \in L$ or not. – definition of a computational problem

Any description of a Language set must be Finite

Finite representation of L :

- Describe in English sentences – informal description, not mathematical
- Mathematical description – use set notations to describe a language set

  Ex: $L = \{ a^n b^m \mid n \geq 0, m \geq 0 \}$ and $L = \{ w \in \Sigma^* \mid w = \text{reverse}(w) \}$…
- Recursive definition (Grammars (will see in future)) – mathematical and finite
- Machines – automata

The conclusion here is that we can mathematically represent languages using finite strings that are made of another alphabet set say $\top$(tau), not necessarily Σ.

Proof that there exists unsolvable problems, or that we can not describe all languages formally

- For an alphabet set Σ, Σ* denotes the universal set of all strings possible that can be made using Σ.
- Even though Σ is finite, Σ* is an infinte set because the length of strings that can be made from Σ has no upper limit. We call this countable infinite (will see in class)
- And the set of all possible languages from this Σ* i.e., the set of all subsets of Σ* is the powerset of Σ*, whose cardinality is 2^ Σ*. And this is uncountable infinite, and it is strictly larger than Σ* (will see in class)

- Now, from previous slide, we saw that all language representations are again a finite strings over another finite alphabet set say ⊤, so set of all possible language descriptions we can give are ⊤*, which is again countable infinite.

Conclusion : We have uncountable infinite many languages (2^ Σ*) and countably infinite many language descriptions (⊤*).

- So there must exist some languages that does not have any finite mathematical descriptions
- For those we can not give any mathematical representations or build machines
- Don’t worry about finite descriptions and these too much, we will understand this part’s significance when we get to turing machines part

Different language classes – Chomsky hierarchy of languages

We will not study that part,
We will only study first regular languages,
then context free grammars and turing machines

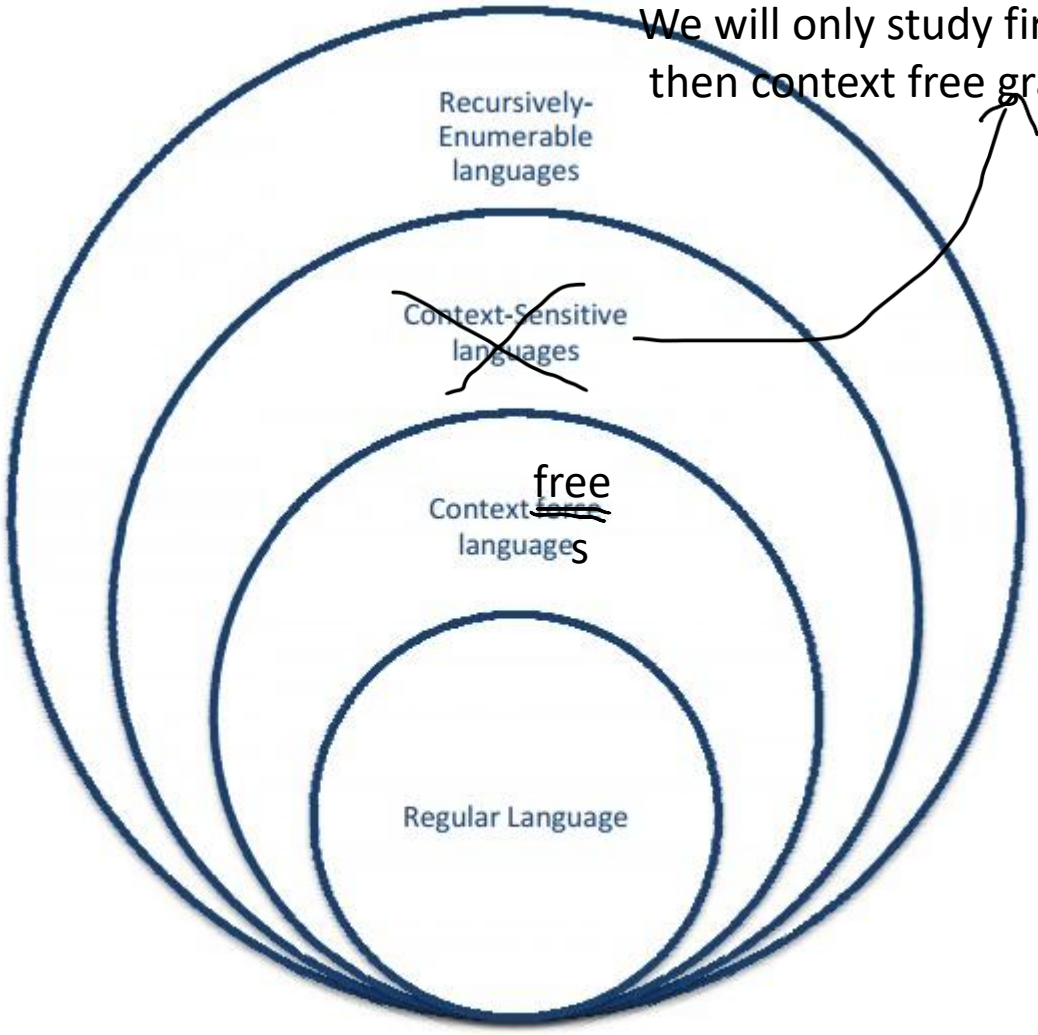
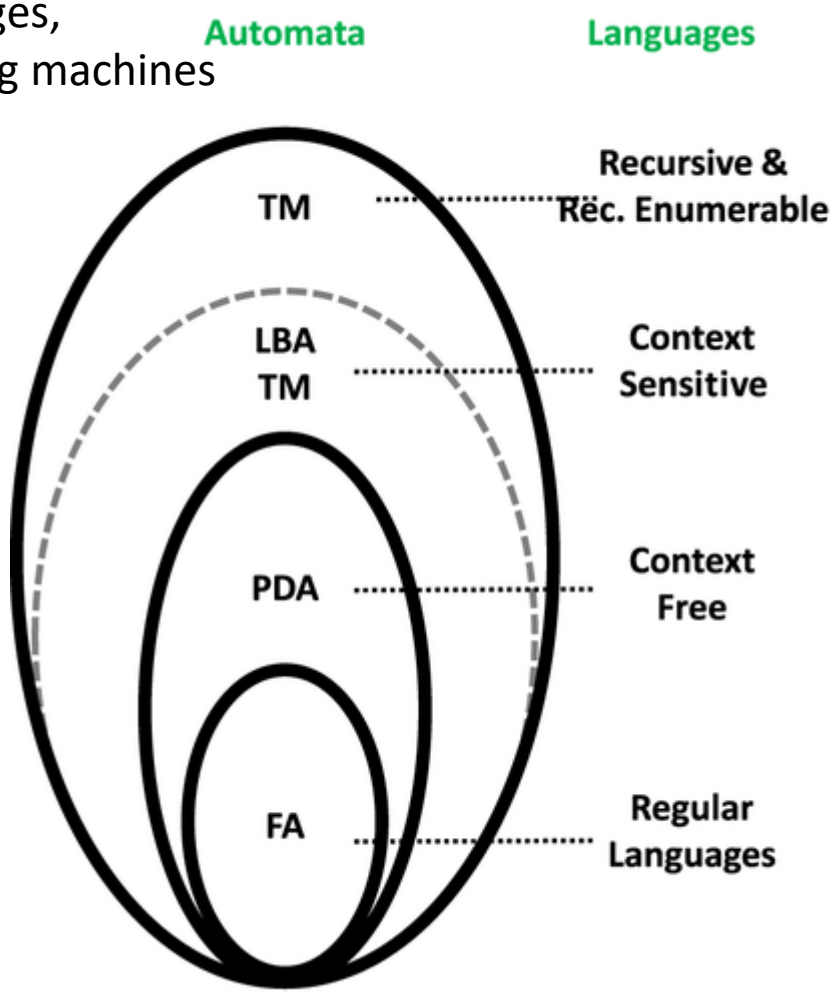


Figure Chomsky Hierarchy of Languages



Some operations/notations on languages, since they are just sets

- We take Σ to be the alphabet here. (again alphabet is just a finite set of symbols)
- Concatenation on strings, if $x \in \Sigma^*$ and $y \in \Sigma^*$ are two strings. Then xy is the concatenation of strings x,y
  - Concatenation is an operation on strings, it follows associative property, not commutative,
- If L is a language set, then
  - L.L (or) $L^2 = \{xy \mid x \in \Sigma^*, y \in \Sigma^*\}$
  - Similarly, $L^n = L.\ L^{n-1} = L.L\ldots\ldots L$ (n times)

  Ex: L={ a, b } then L^2 set contains { aa, ab, ba, bb }
- And by definition for any $L \subseteq \Sigma^*$, $L^0 = \{\varepsilon\}$. Any language set to power 0 is the single element set, where the element is 'ε' the empty string. This is a choice to make sure that L=L. L^0 valid.
- And L* asterate of L is . $L^* = \bigcup_{n\geq 0} L^n = L^0 \cup L^1 \cup L^2 \cup \ldots\ldots\ldots$
- This above definition satisfies notation wise if L is Σ too.
- $\Sigma^* = \bigcup_{n\geq 0} \Sigma^n = \Sigma^0 \cup \Sigma^1 \cup \Sigma^2 \cup \ldots\ldots\ldots$

  Where Σ^n is basically set of all strings of length n (as per definition of Σ^n)

Please download the given text books and start reading chapter 1 of TOC book, this is a complete theoretical course, your effort is needed

Thank you