Formal languages and automata

Introduction

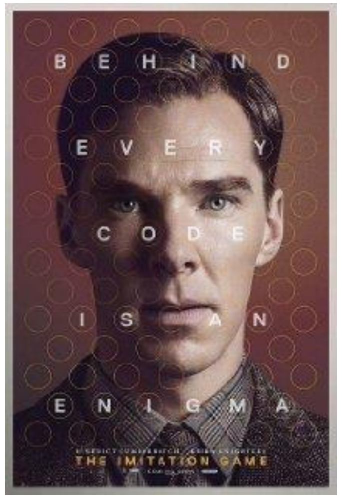
https://t.me/fla_uog
mh.olyaee@gmail.com

What is this course about? – Formal Languages

- *Study of abstract computing devices, or “machines”*
- Automaton = an abstract computing device
  - Note: A “device” need not even be a physical hardware!
- Computations happen everywhere: On your laptop, on your cell phone, in nature, …
- A fundamental question in computer science:
  - Find out what different models of machines can do and cannot do

(A pioneer of automata theory)

Alan Turing (1912-1954)

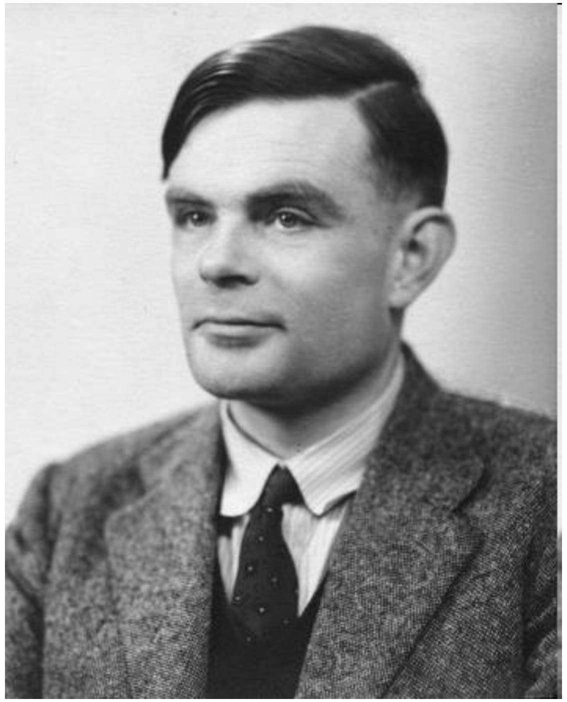


Father of Modern Computer Science

English mathematician

Studied abstract machines called ***Turing machines*** even before computers existed

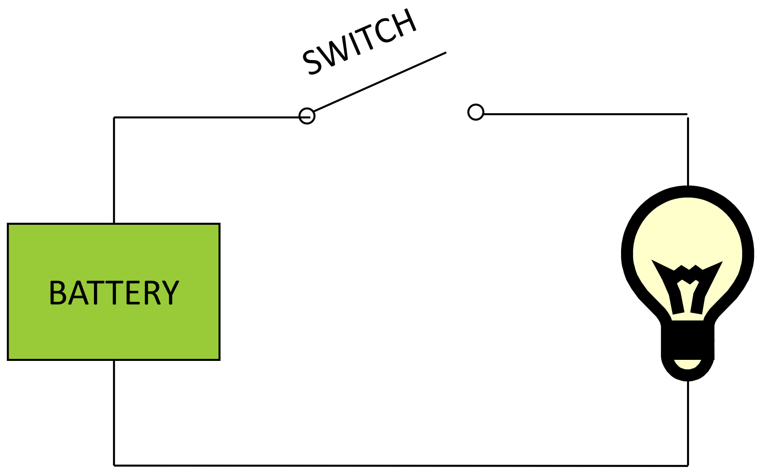
Heard of the Turing test?



- An abstraction of the notion of a “problem”
- Problems are cast **either** as Languages (= sets of “Strings”)
  - ”Solutions” determine if a given “string” is in the set or not
    - e.g., Is a given integer, *n*, prime?
- **Or**, as transductions between languages
  - “Solutions” transduce/transform the input string to an output string
    - e.g., What is 3+5?

- Automata (singular *Automaton*) are abstract mathematical devices that can
  - Determine membership in a set of strings
  - Transduce strings from one set to another
- They have all the aspects of a computer
  - input and output
  - memory
  - ability to make decisions
  - transform input to output
- Memory is crucial:
  - Finite Memory
  - Infinite Memory
    - Limited Access
    - Unlimited Access

Example: A simple computer



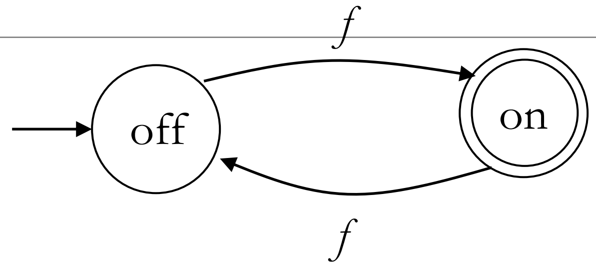
input: switch

output: light bulb

actions: flip switch

states: on, off

Example of a finite automaton



There are states off and on, the automaton starts in off and tries to reach the "good state" on

What sequences of fs lead to the good state?

Answer: $\{f, fff, fffff, \ldots\} = \{f^{\,n}\!: n \text{ is odd}\}$

This is an example of a deterministic finite automaton over alphabet $\{f\}$

Different kinds of automata

This was only one example of a computational device, and there are others

We will look at different devices, and look at the following questions:

- What can a given type of device compute, and what are its limitations?
- Is one type of device more powerful than another?

Some devices we will see

finite automata	Devices with a finite amount of memory. Used to model "small" computers.
push-down automata	Devices with infinite memory that can be accessed in a restricted way. Used to model parsers, etc.
Turing Machines	Devices with infinite memory. Used to model any computer.
time-bounded Turing Machines	Infinite memory, but bounded running time. Used to model any computer program that runs in a "reasonable" amount of time.

Some highlights of the course

Finite automata

- We will understand what kinds of things a device with finite memory can do, and what it cannot do
- **Introduce simulation**: the ability of one device to “imitate” another device
- **Introduce nondeterminism**: the ability of a device to make arbitrary choices

Push-down automata

- These devices are related to grammars, which describe the structure of programming (and natural) languages

Some highlights of the course

Turing Machines

- This is a general model of a computer, capturing anything we could ever hope to compute
- Surprisingly, there are many things that we cannot compute, for example:

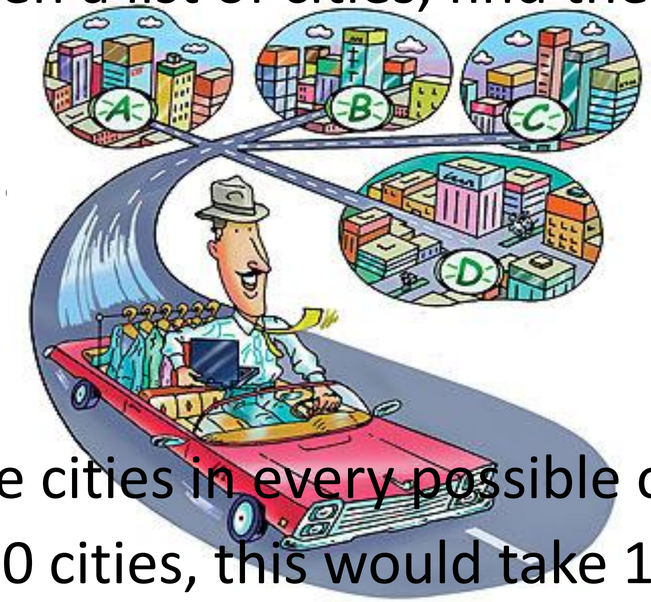
Write a program that, given the code of another program in C, tells it is **complete** or not!

- It seems that you should be able to tell just by looking at the program, but it is impossible to do!

Some highlights of the course

Time-bounded Turing Machines

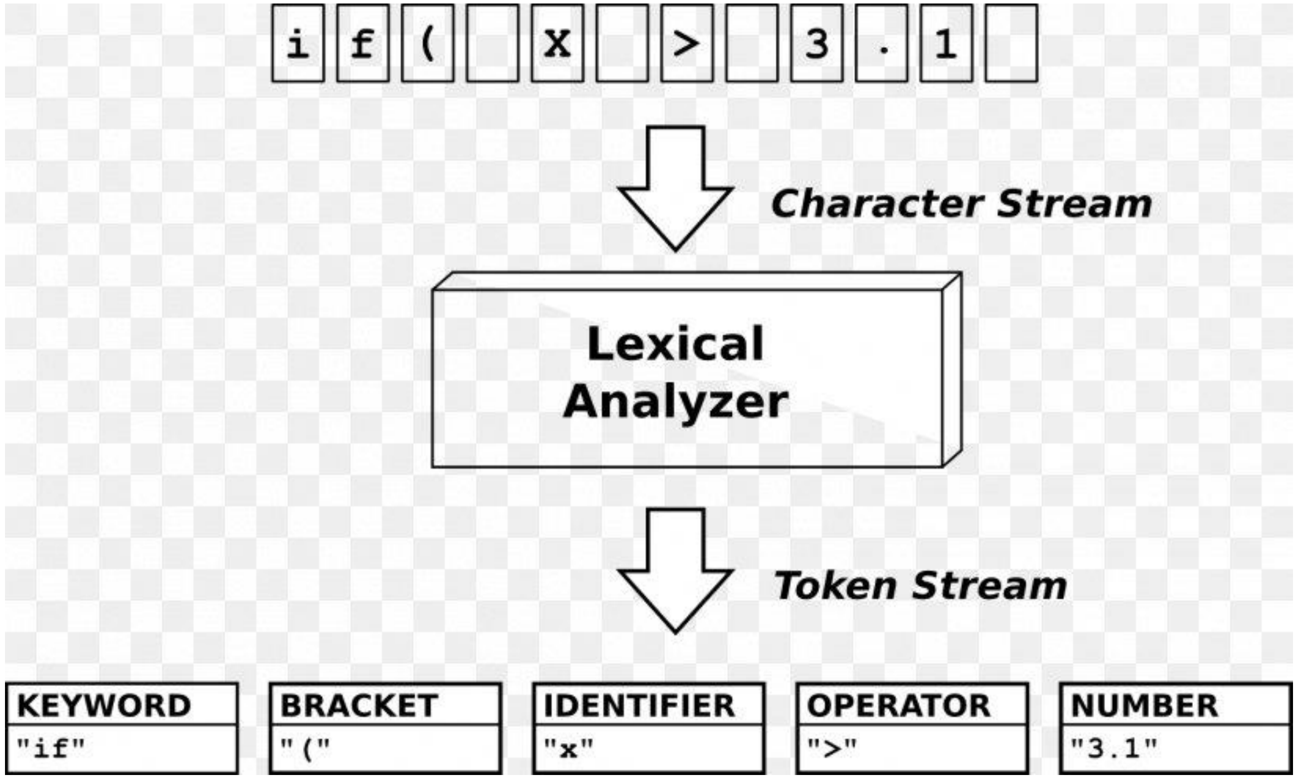
- Many problems are possible to solve on a computer in principle, but take too much time in practice
- Traveling salesman: Given a list of cities, find the shortest way to visit them and come back home



- Easy in principle: Try the cities in every possible order
- Hard in practice: For 100 cities, this would take 100+ years even on the fastest computer!

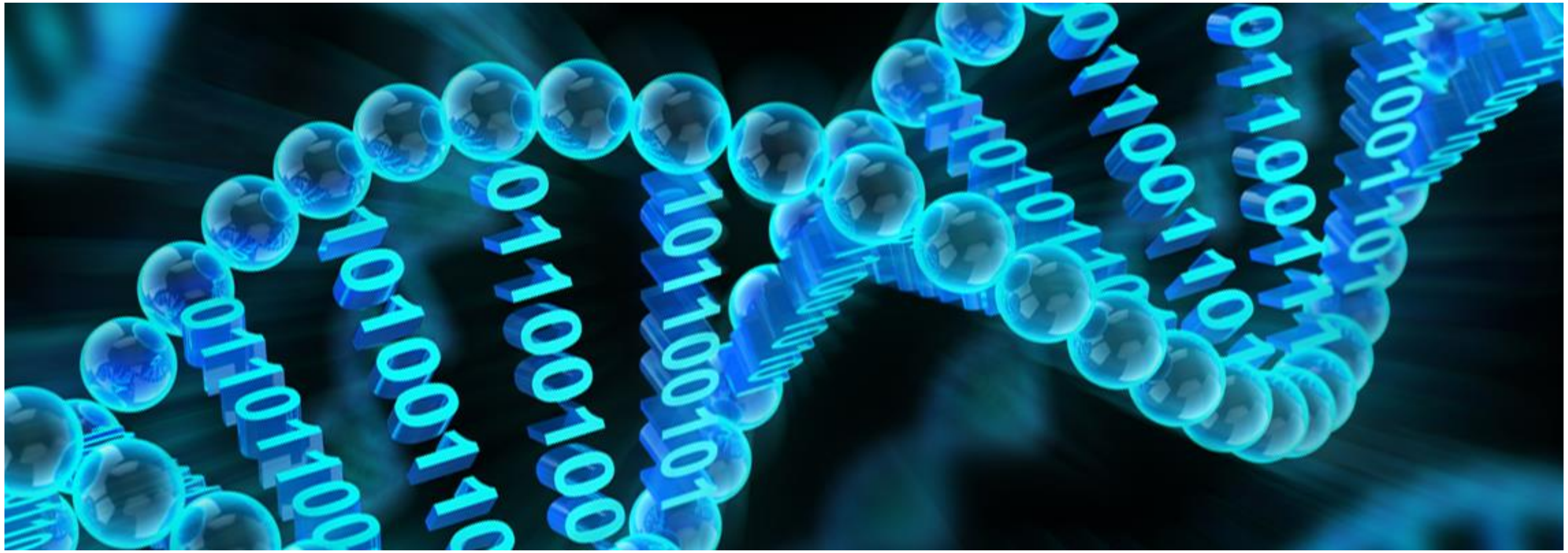
Applications

Lexical analyzer



Applications

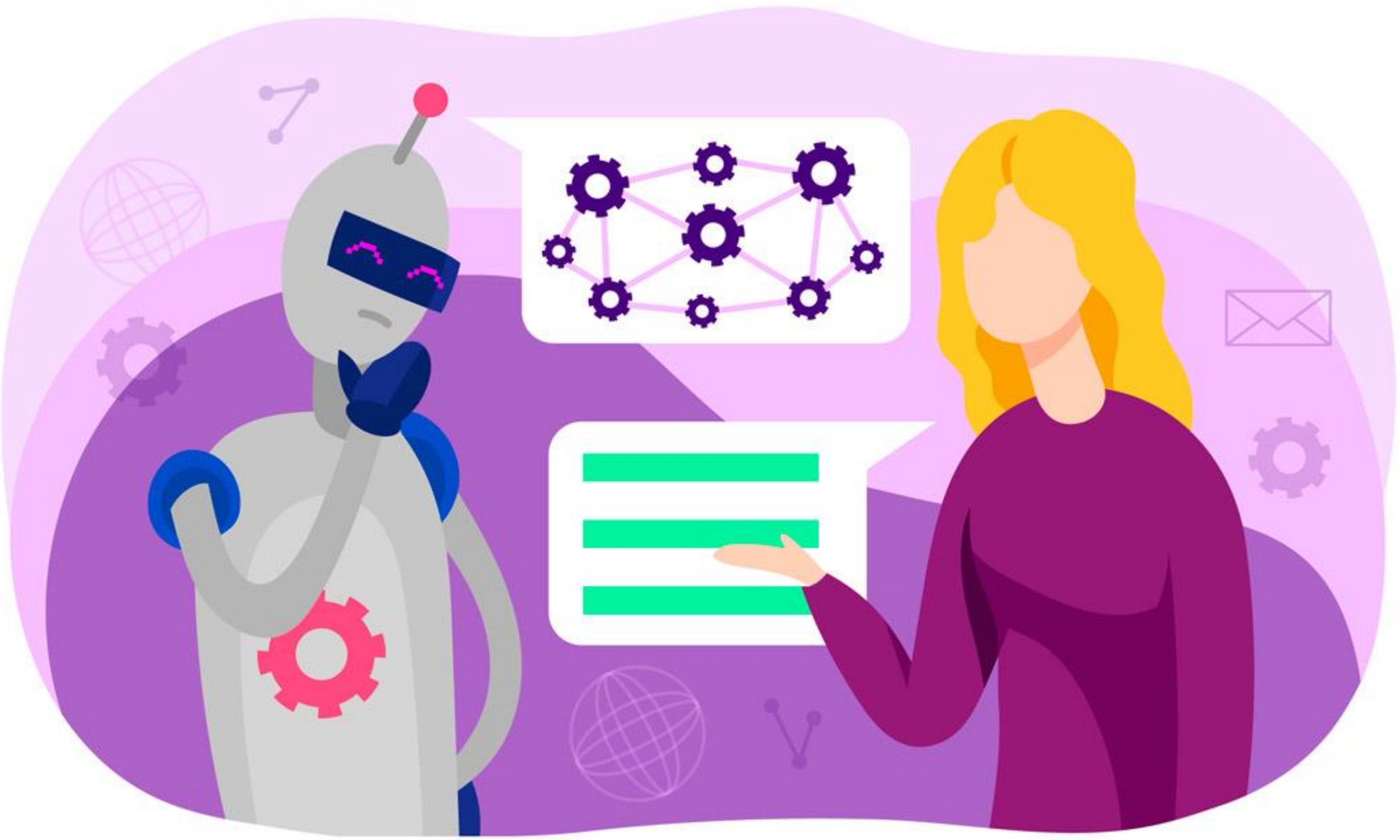
Bioinformatics



Applications

Software verification

Natural language processing

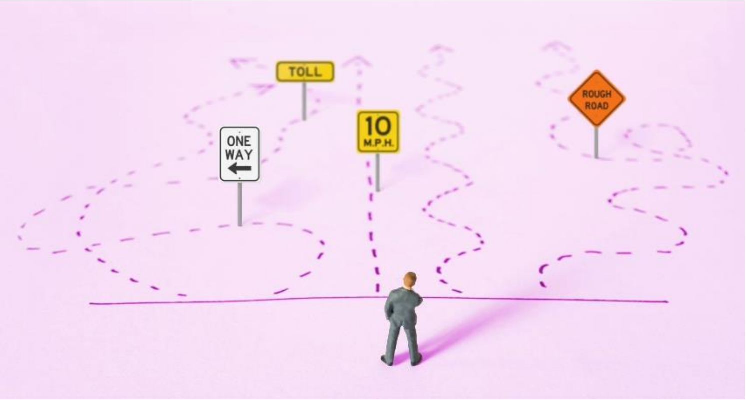


- An abstraction of the notion of a “problem”
- Problems are cast **either** as Languages (= sets of “Strings”)
  - ”Solutions” determine if a given “string” is in the set or not
    - e.g., Is a given integer, *n*, prime?
- **Or**, as transductions between languages
  - “Solutions” transduce/transform the input string to an output string
    - e.g., What is 3+5?

Decision problems

- A decision problem is a function with a YES/NO output
- We need to specify
  - the set A of possible inputs (usually A is infinite)
  - the subset $B \subseteq A$ of YES instances (usually B is also infinite)
- The subset B should have a finite description!

- A: set of all pairs (G, t)
  - G is a $\{$finite set of triples of the sort $(i, j, w)\}$,
  - i and j are integers and w is real
  - The finite set encodes the edges of a weighted directed graph G.
  - $A = \{\dots(\{\dots,(3,4,5.6),\dots\},8.0),\dots\}$
- Each pair in A, (G, t), represents a graph G and a threshold t
- Does G have a path that goes through all nodes once with total weight $< t$?
  - Travelling Salesperson Problem
- A is the set of all TSP instances.



Problems

- Examples of problems we will consider
  - Given a word s, does it contain the subword "Hello"?
  - Given a number n, is it divisible by 7?
  - Given a pair of words s and t, are they the same?
  - Given an expression with brackets, e.g. (( ) ( )), does every left bracket match with a subsequent right bracket?
- All of these have "yes/no" answers.
- There are other types of problems, that ask "Find this" or "How many of that" but we won't look at those.

- In real life, we use many different types of data: integers, reals, vectors, complex numbers, graphs, programs (your program is somebody else’s data).
- These can all be encoded as strings
- So we will have only one data type: strings

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}
  - DNA molecule letters: ∑ = {a,c,g,t}
  - …

Strings

A string or word is a finite sequence of symbols chosen from ∑

- ***Empty string is λ (or "lambda")***

- Length of a string *w*, denoted by "|*w*|", is equal to the *number of (non- λ) characters in the string*
  - *E.g., x = 010100* |*x*| = *6*
  - *x = 01 λ 0 λ 1 λ 00 λ* |*x*| = *?*

- *xy* = concatentation of two strings *x* and *y*

- If $a \in \Sigma$, we use a^n to denote a string of n a's concatenated
  - $\Sigma = \{0, 1\}, 0^5 = 00000$
  - $a^0 =_{def} \epsilon$
  - $a^{n+1} =_{def} a^n a$
- The reverse of a string ω is denoted by ω^R.
  - $\omega^R = a_n, \ldots, a_1$

Powers of an alphabet

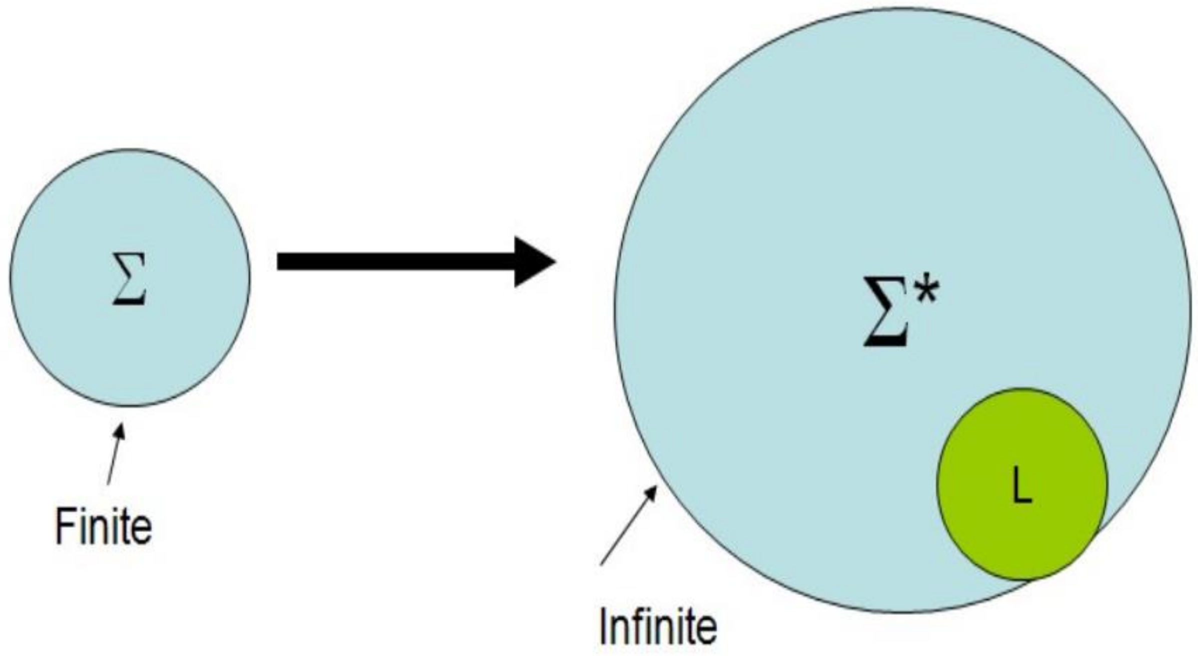
Let ∑ be an alphabet.

- Σ^k = the set of all strings of length k
- $\Sigma^* = \Sigma^0 \cup \Sigma^1 \cup \Sigma^2 \cup \ldots$
- $\Sigma^+ = \Sigma^1 \cup \Sigma^2 \cup \Sigma^3 \cup \ldots$

- Σ^* is a countably infinite set of finite length strings

Language

- A language L over Σ is any subset of Σ^*

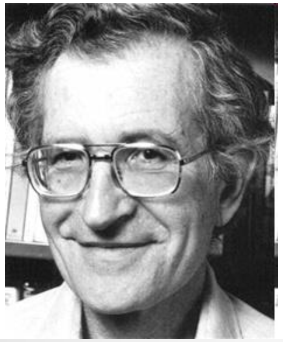


- L can be finite or (countably) infinite

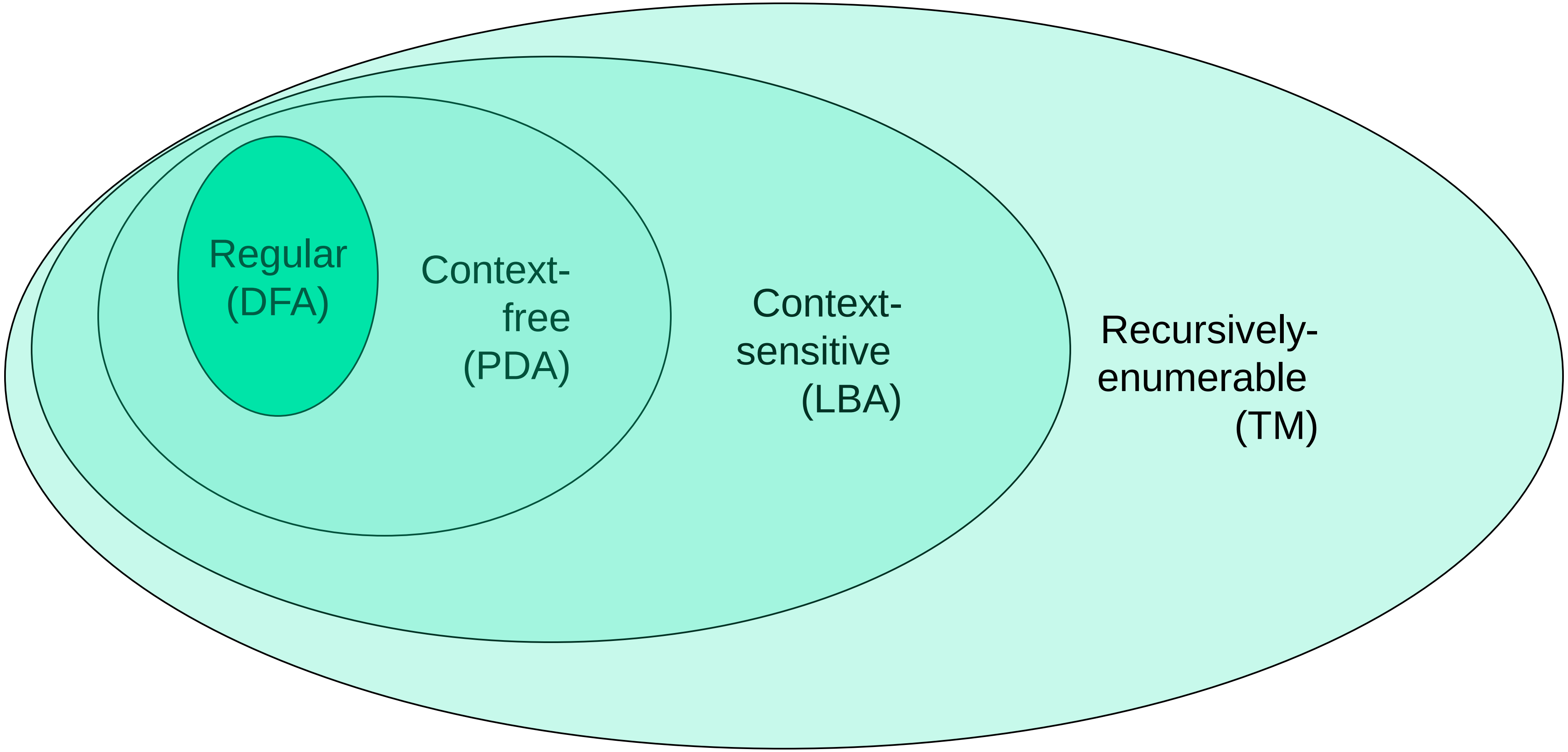
- $L = \Sigma^*$ – The mother of all languages!
- $L = \{a, ab, aab\}$ – A fine finite language.
  - Description by enumeration
- $L = \{a^n b^n : n \geq 0\} = \{\epsilon,\ \mathrm{ab},\ \mathrm{aabb},\ \mathrm{aaabbb}, \ldots\}$
- $L = \{\omega | n_a(\omega) \text{ is even}\}$
  - $n_x(\omega)$ denotes the number of occurrences of x in ω
  - all strings with even number of a's.
- $L = \{\omega | \omega = \omega^R\}$
  - All strings which are the same as their reverses – palindromes.

- Since languages are sets, all usual set operations such as intersection and union, etc. are defined.
- Complementation is defined with respect to the universe Σ^* : $\overline{L} = \Sigma^* - L$
- If L, L_1 and L_2 are languages:
  - $L_1 \cdot L_2 = \{xy | x \in L_1 \text{ and } y \in L_2\}$
  - $L^0 = \{\epsilon\}$ and $L^n = L^{n-1} \cdot L$
  - $L^* = \bigcup_0^\infty L^i$
  - $L^+ = \bigcup_1^\infty L^i = L^* - \{\epsilon\}$

The Chomsky Hierachy



- A containment hierarchy of classes of formal languages



Languages & Grammars

Or "**words**"

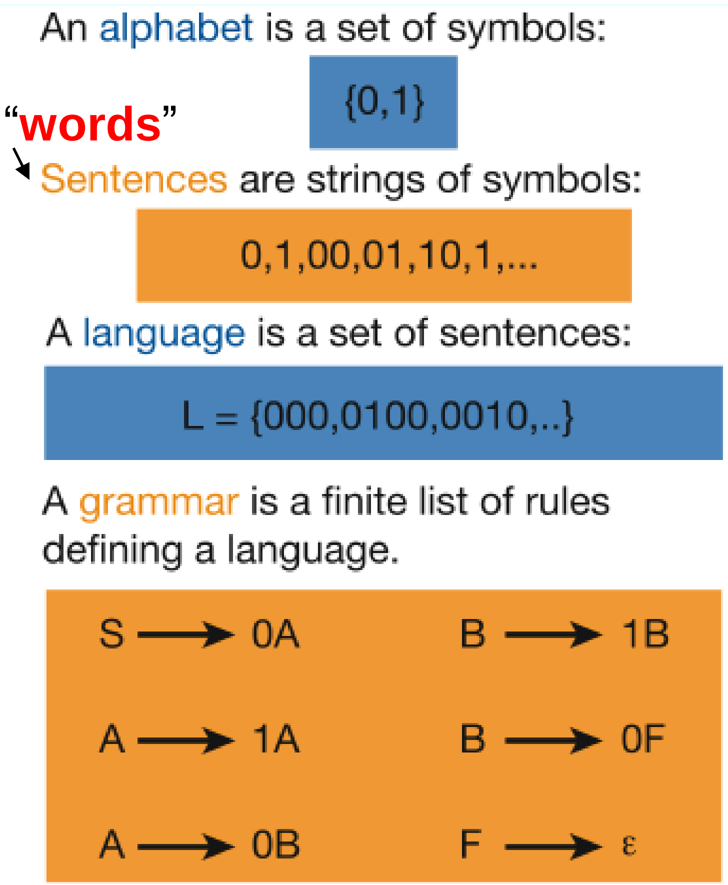


Image source: Nowak et al. Nature, vol 417, 2002

- <u>Languages</u>: "*A language is a collection of sentences of finite length all constructed from a finite alphabet of symbols*"
- <u>Grammars</u>: "*A grammar can be regarded as a device that enumerates the sentences of a language*" - nothing more, nothing less

- *N. Chomsky, Information and Control, Vol 2, 1959*