ANKARA UNIVERSITY COM364 AUTOMATA THEORY

Week 1

Introduction, Course Outline, and Math Revision

Kurtuluş KÜLLÜ

AUTOMATA THEORY

Theory of Computation

Students often find theoretical topics boring and difficult to understand BUT,

- there are some fascinating big ideas and
- theory is relevant to practice.

For example,

- *Grammars* → programming language and compiler design, natural language processing
- *Finite automata and regular expressions* → string searching, pattern matching, digital logic (circuit) design
- *Undecidability, complexity, and intractability* → what can we compute and how fast?

We will try to make it understandable and interesting

AUTOMATA, COMPUTABILITY, AND COMPLEXITY

These are central topics in theory of computation

What are computers capable of?
What are their limits?

Mathematicians started working on these questions in the 1930s (before an ordinary "computer" of today existed)

Since then, technology improved a lot and these questions are now not only about theory but also has practical effects.

Each of Automata, Computability, and Complexity deals with slightly different questions. Let's look at them in reverse order (to emphasize the reason for the given order)

COMPLEXITY THEORY

There are many different problems,

- some are easy: e.g., sorting a list
- some are hard: e.g., scheduling classes for the entire university satisfying certain constraints (no two classes can be in the same room at the same time, a person cannot be in two places at the same time, etc.)

What makes some problems
computationally hard and others easy?

- The interesting thing is that we still don't know the exact answer. But we have a very useful classification.
- For example, this has important effects in cryptography.
- We will explore this topic towards the end of the course.

COMPUTABILITY THEORY

What are the limits of computers?

I.e., what can they solve and cannot solve?

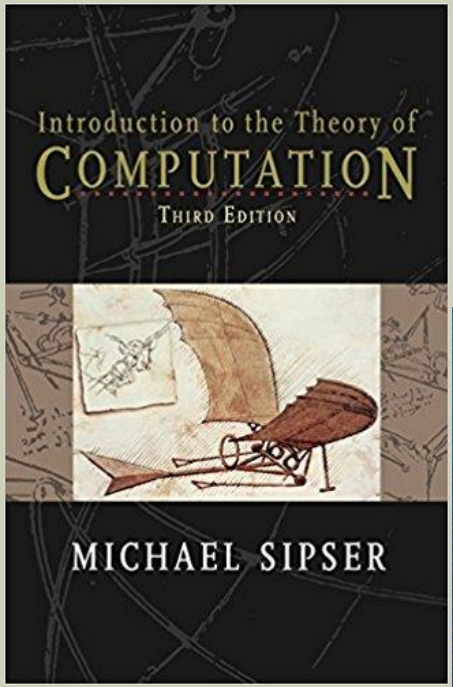
- Between 1930s and 1960s, mathematicians such as Gödel, Turing, and Church discovered that some problems cannot be solved by computers.
- Theoretical computer models were developed in this field which eventually lead to actual computers.

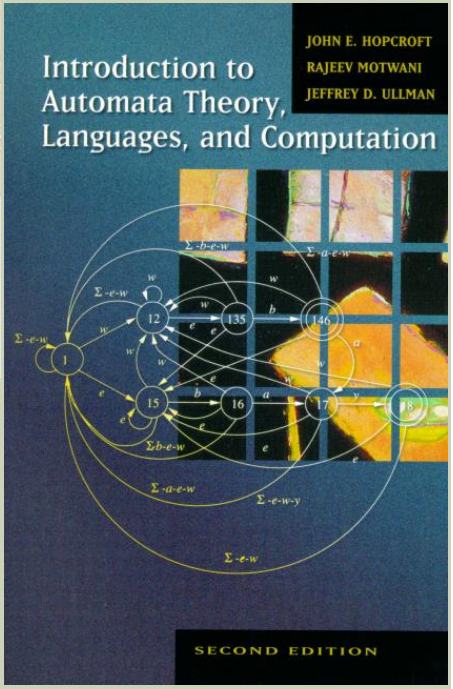
AUTOMATA THEORY

- This is about the mathematical computation models and their properties
- These models have direct applications in practice
- For example,
  - *finite automata* are (automata is plural, singular version is automaton) used in text processing, compilers, and hardware design;
  - *context-free grammars* are used in programming languages and artificial intelligence.
- We will start with this topic because the other two will use the computer models we will see here.

ABOUT THE COURSE

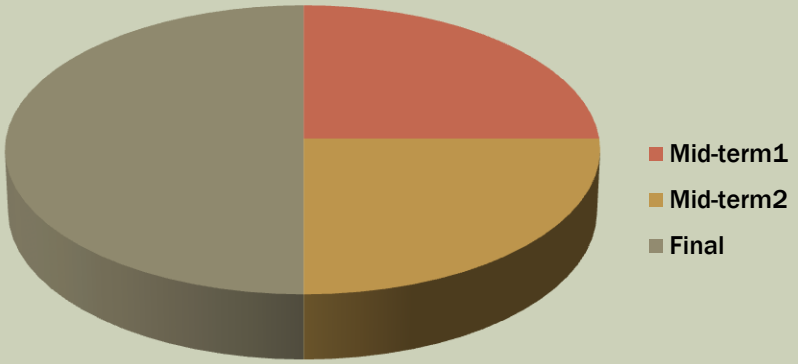
- We will mostly use the board instead of slides
- I will try to produce some summary notes each week but the main resources to study are the suggested books.
- There are many examples in these books that will be very helpful for exams

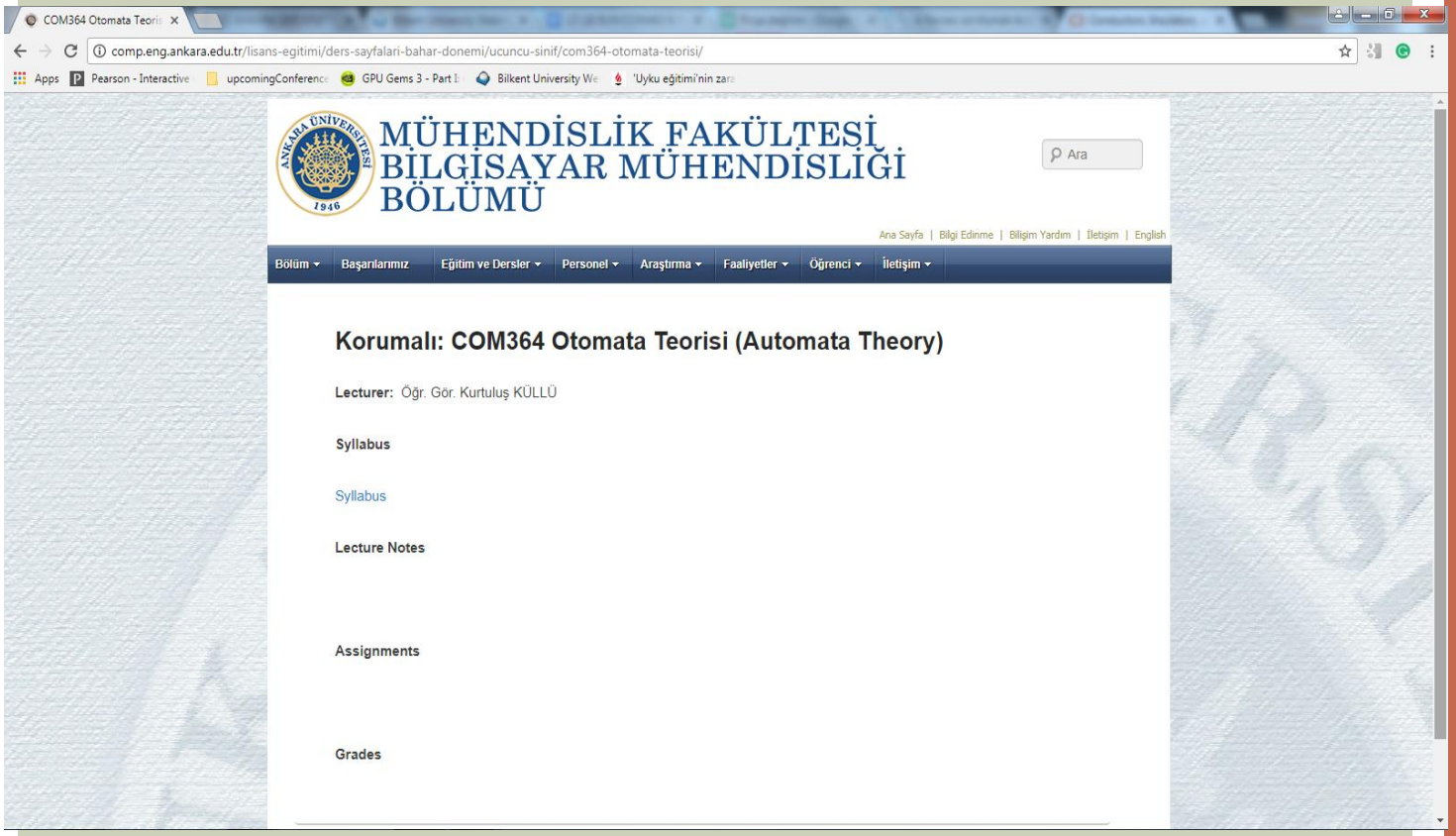




EXAMS AND GRADING

- There will be
  - two mid-term exams and
  - a final exam
- Each mid-term will count for 25% and
- the final will be 50% of your course grade
- Exam dates are determined by the department, so they can be different from the weeks in the syllabus





COURSE WEB PAGE

Follow the course web page on the department web site for announcements and documents

(Check it at least twice a week, especially once before a class hour)

OUTLINE OF TOPICS

- Finite Automata (FA)
  - Deterministic and Nondeterministic FA
- Regular Expressions (REs) and Languages (RLs)
  - Including relations between these and FAs
  - Including: Pumping Lemma (PL) for RLs; Examples of non-regular languages; Proving that a language is not regular using PL
- Context-Free Grammars (CFGs) and Languages (CFLs)
- Pushdown Automata (PDA)
  - And their relation to CFLs, Pumping Lemma for CFLs
- Turing Machines
- Decidability
- Reductions
- Complexity (Time and Space)
- Intractability (P and NP classes, NP-completeness)
- Advanced Topics (if there is time)

WARNINGS

- This course will be very mathematical and we will work with proofs
- A good proof should be correct and clear (easy to understand)

- When writing (or describing) a proof, it is helpful to give three levels of detail
  - 1st level: A short phrase/sentence providing a "hint" of the proof
    - E.g. "proof by contradiction", "proof by induction", "follows from the pigeonhole principle"
  - 2nd level: A short, one paragraph description of the main ideas
  - 3rd level: The full proof
- The book by Sipser is written in this way and I suggest you do the same when needed
- In the classroom, we will generally talk about the proofs using the first two levels (the details will mostly be excluded)
  - When studying, you should think (and look at the books) about how to complete these details because you might be asked to give complete proofs in exams.
- We will go over some standard proof methods (Both textbooks have parts on these)

REVISION OF RELATED MATHEMATICAL TOPICS

- Sets
- Sequences and Tuples
- Relations and Functions
- Graphs
- Alphabets, Strings, and Languages

SETS

A set is an unordered collection of objects (numbers, symbols, functions, or anything, even other sets)

$$S = \{a, b, c\} \text{ or } A_1 = \{1, f, \{1\}\}$$

Objects in the set are its *elements* or *members.*

Elements in a set don't have to be related.

$$S = \{\text{turkish}, \text{red}, \pi\}$$

Symbols $\in$ and $\notin$ denote membership and non-membership.

$$7 \in \{7,21,57\}$$
$$8 \notin \{7,21,57\}$$

Cardinality of a set is the number of its elements and it is shown with two bars (| symbols)

$$|\{a, b, c\}| = 3$$

If $A = \{\{1,2\}, 3\}$, then $|A| = ?$

SETS

A set is an unordered collection of objects (numbers, symbols, functions, or anything, even other sets)

$$S = \{a, b, c\} \text{ or } A_1 = \{1, f, \{1\}\}$$

Objects in the set are its *elements* or *members.*

Elements in a set don't have to be related.

$$S = \{\text{turkish}, \text{red}, \pi\}$$

Symbols $\in$ and $\notin$ denote membership and non-membership.

$$7 \in \{7,21,57\}$$
$$8 \notin \{7,21,57\}$$

Cardinality of a set is the number of its elements and it is shown with two bars (| symbols)

$$|\{a, b, c\}| = 3$$

If $A = \{\{1,2\}, 3\}$, then $|A| = 2$

EMPTY SET, SINGLETONS, INFINITE SETS

Empty set is a set with no elements

$\{\}$ or $\emptyset$

$|\emptyset| = 0$

Singleton is a set with only one element

E.g., $\{1\}$ or $\{\{\}\}$

An ***infinite set*** is a set with infinite number of elements. We cannot write all elements of a list like this, so we sometimes use three dots (…)

E.g., the set of natural numbers, $\mathbb{N} = \{1,2,3, \dots\}$

$|\mathbb{N}| = \infty$

$S = \{01, 0011, 000111, \dots\}$

DESCRIBING A SET

Sometimes, we describe a set by writing a rule about its elements

$$\{n | \text{rule about } n\}$$

A colon (:) is sometimes used instead of the bar (|) above

$$\{n : \text{rule about } n\}$$

For example,

$$\{n | n = m^2 \text{ for some } m \in \mathbb{N}\}$$
$$S = \{x | x \in \mathbb{Z} \text{ and } |x| < 10\}$$
$$A = \{a | a \text{ is prime}\}$$

SET OPERATIONS

Union of two sets, ***A*** and ***B***

$$A \cup B = \{n | n \in A \text{ or } n \in B\}$$

Set ***intersection***

$$A \cap B = \{n | n \in A \text{ and } n \in B\}$$

Complement of a set, A

$$\bar{A} = \{n | n \notin A\}$$

Set difference (or ***relative complement***)

$$A - B = \{n | n \in A \text{ and } n \notin B\}$$

SET RELATIONS

A is a ***subset*** of ***B*** ($A \subseteq B$) if every element of ***A*** is also a member of ***B***. (***B*** is said to be a ***superset*** of ***A***)

$\emptyset$ is a subset of any set (including itself)

A is a ***proper subset*** of ***B*** ($A \subset B$) if ***A*** is a subset of ***B*** and not equal to ***B*** (***B*** must have at least one element that is not in ***A***).

$\emptyset$ is the only set that does not have a proper subset

A and ***B*** are ***disjoint*** (sets) if $A \cap B = \emptyset$

SEQUENCES AND TUPLES

A ***sequence*** is an ordered collection of objects

$$(7,21,57)$$

Note that we use parenthesis, (), not brackets, {}

For a set, order is not important, but for a sequence, it is

$$\{7,21,57\} = \{21,57,7\}$$
$$(7,21,57) \neq (21,57,7)$$

Sequences can be finite or infinite.

Finite sequences are often called ***tuples***.

A sequence with *k* elements is called a ***k-tuple***.

For example, $(7,21)$ is a 2-tuple, which is more commonly called an ***(ordered) pair.***

3-tuples are often called ***(ordered) triple***,

4-tuples , ***(ordered) quadruple***,

…

POWER SET

The ***power set*** ($\mathcal{P}$) of a set (A) is the set that contains all subsets of that set (A).

If $A = \{0,1\}$, then
$\mathcal{P}(A) = \{\emptyset, \{0\}, \{1\}, \{0,1\}\}$

If $|S| = k$, then
$|\mathcal{P}(S)| = ?$

POWER SET

The ***power set*** ($\mathcal{P}$) of a set (A) is the set that contains all subsets of that set (A).

If $A = \{0,1\}$, then
$\mathcal{P}(A) = \{\emptyset, \{0\}, \{1\}, \{0,1\}\}$

If $|S| = k$, then
$|\mathcal{P}(S)| = 2^k$

CARTESIAN PRODUCT

Cartesian product **of two sets, *A* and *B*, is the set of all ordered pairs where the first element is from set *A* and second element is from set *B***

$$A \times B = \{(x, y) | x \in A \text{ and } y \in B\}$$

For example, if $A = \{0,1\}$ **and** $B = \{a, b, c\}$

$$A \times B = \{(0, a), (0, b), (0, c), (1, a), (1, b), (1, c)\}$$

RELATIONS

A ***relation*** is a set of ordered pairs.

More formally, a relation from set *A* to set *B* is a subset of *A*x*B*.

If $A = \{x, y, z\}$ and $B = \{1,2\}$,

some relations from *A* to *B* are

$$R_1 = \{(x, 2), (y, 1), (z, 1)\}$$
$$R_2 = \{(x, 1), (y, 2)\}$$
$$R_3 = \emptyset$$

When *A*=*B*, and *R* is a subset of *A*x*A*, it is called a relation on *A*.

FUNCTIONS

A ***function*** (***mapping***) is a relation with some special properties.

If $R \subseteq A \times B$ (i.e., ***R*** is a relation from ***A*** to ***B***) and

- for each $a \in A$, there is exactly one ordered pair in ***R*** with a,

then ***R*** is a function.

A function ***f*** from set ***A*** (***domain***) to set ***B*** (***range***) is written as

$$f : A \to B$$

and instead of writing $(a, b) \in f$, we write

$$f(a) = b \ (f \text{ maps } a \text{ to } b)$$

FUNCTIONS

A function $f : A \rightarrow B$ may not use all elements in ***B***

If it does, it is an ***onto*** function.

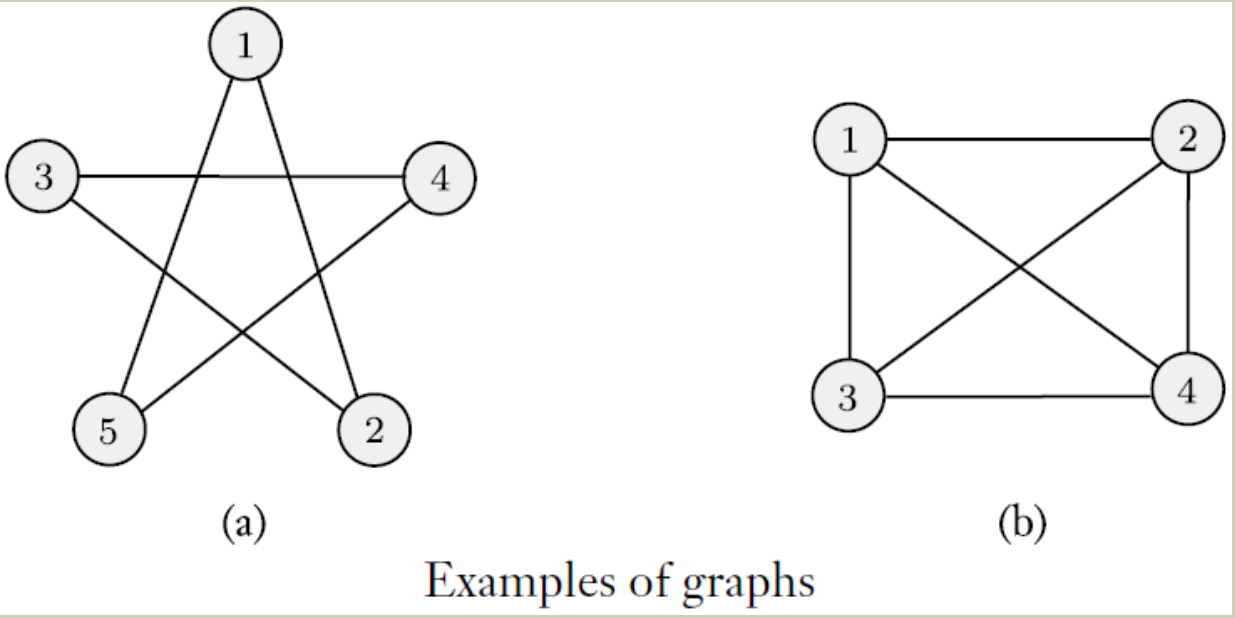
In an ordinary function, two input values can map to the same result ($f(x) = f(y)$ can happen)

If it doesn't, *f* is ***one-to-one***.

If *f* is both one-to-one and onto, it is a ***bijection*** (***one-to-one correspondence***)

GRAPHS

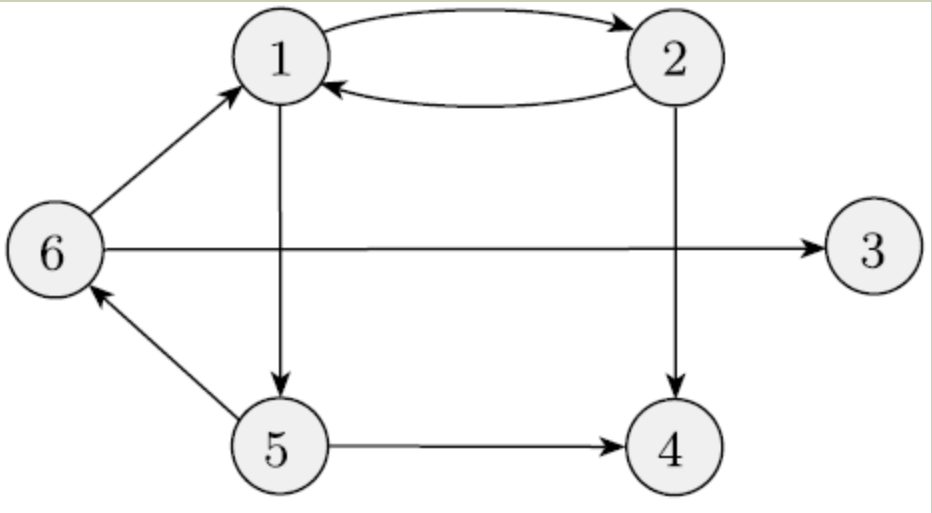
- An ***(undirected) graph***, *G=(V,E)*, is a set of points, *V*, with lines connecting some of these points, *E*. (*E* is a relation on *V*)
- The points are called ***nodes*** or ***vertices*** (elements of *V*)
- The lines are called ***edges*** (elements/pairs in *E*)
- ***Degree*** of a node is the number of edges connected to that node (Each node in (a) below has degree 2; How about (b)?)



Examples of graphs

GRAPHS

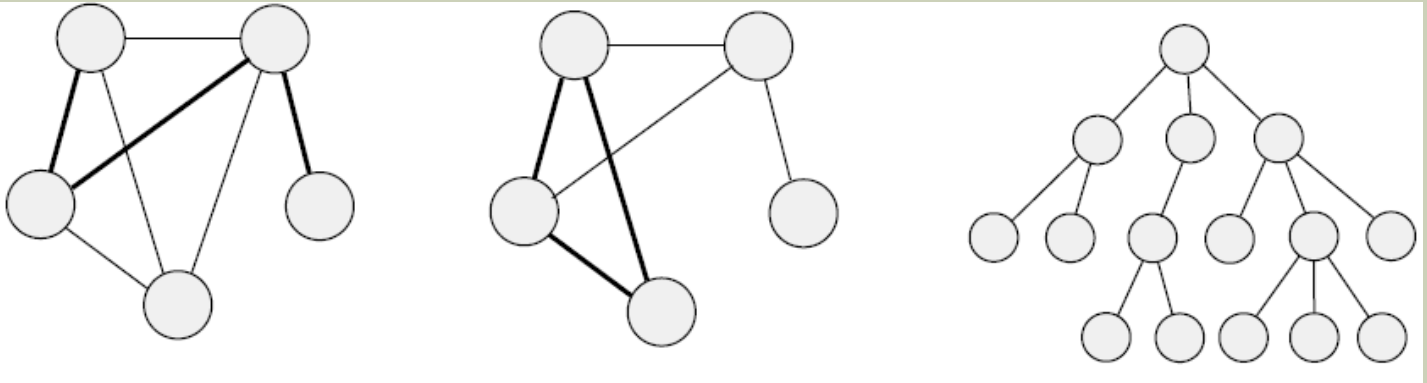
- A ***directed graph*** is a graph where the direction of edges is important (so we speak of ***outdegree*** and ***indegree***)
- Graphs are frequently used to represent data (cities and roads, devices and connections, people and relationships, …)



A directed graph

GRAPHS

- A ***path*** is a sequence of nodes connected by edges
- If a path doesn't have repeated nodes, it is a ***simple path*** (or ***trail***)
- A graph is ***connected*** if any two nodes have a path between them
- A ***cycle*** is a path that starts and ends in the same node
- A ***tree*** is a connected graph with no cycles
- A tree often has a specific top node called the ***root***
- The nodes at the bottom with degree 1 are called the ***leaves***



(a) A path in a graph, (b) a cycle in a graph, and (c) a tree

ALPHABETS AND STRINGS

An ***alphabet*** is a simple non-empty and finite set. For example,

$$\Sigma_1 = \{a, b, c, \dots, z\}$$

$$\Sigma_2 = \{0,1\}$$

The elements of an alphabet are called ***symbols***.

A ***string (over an alphabet)*** is a finite sequence of symbols from that alphabet (usually written together without any punctuation or spaces).

For example, if $\Sigma = \{\mathrm{a}, \mathrm{b}, \mathrm{c}\}$, then baba is a string over Σ.

If *w* is a string, then the ***length*** of *w*, written $|w|$, is the number of symbols in *w*.

The string with no symbols is called the ***empty string***, often shown with ε. So, $|\varepsilon| = 0$.

ALPHABETS AND STRINGS

Let w be a string over alphabet Σ

- w^R denotes the ***reverse*** of w (the string obtained by writing w in opposite order)
- z is a ***substring*** of w if it appears somewhere inside w.
  - E.g., "cad" is a substring of "abracadabra"

If $x=x_1x_2...x_m$ and $y=y_1y_2...y_n$ are two strings where x_i and y_j are symbols, the ***concatenation*** of x and y, written xy, is the string simply obtained by appending y to the end of x

$$xy = x_1x_2 \dots x_my_1y_2 \dots y_n$$

If $|x| = m$ and $|y| = n$, then $|xy| = m + n$

POWERS OF ALPHABETS AND STRINGS

When we concatenate a string with itself, we use superscript notation

$$xx = x^2$$

$$w^k = \overbrace{ww \dots w}^{k}$$

Similar notation applies to alphabets.

Σ^k is the set of strings of length k, where each symbol is from Σ

For example, if $\Sigma = \{0,1\}$,

- $\Sigma^1 = \Sigma$, $\Sigma^2 = \{00,01,10,11\}$, $\Sigma^3 = \{000,001, \dots, 111\}$

For any Σ, $\Sigma^0 = \{\varepsilon\}$

POWERS OF ALPHABETS AND STRINGS

Σ^* is the set of all strings over alphabet Σ.

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

Σ^+ is similar but there is one important difference

$$\Sigma^* = \Sigma^0 \cup \Sigma^+$$

In other words,

$$\Sigma^* - \Sigma^+ = \{\varepsilon\}$$

LANGUAGES

If Σ is an alphabet, then any $L \subseteq \Sigma^*$ is a ***language*** over Σ.

So, a language is essentially a set of strings.

Examples:

- Turkish or English
- C programming language
- The language of all strings consisting of n 0's followed by n 1's, for some n≥0: {ε, 01, 0011, 000111, 00001111,……}
- The set of binary numbers whose value is a prime {10,11,101,111,1011,…}
- Σ* (set of all strings)
- ∅, the empty language, is a language over any alphabet
- {ε}, the language consisting of only the empty string, is also a language over any alphabet.

PROBLEMS AS LANGUAGE RECOGNITION

In Automata Theory, a problem is considered as recognizing whether a given string is a member of a particular language

For example, consider the problem of testing a given number and deciding if it is a prime number or not.

We can take all the prime numbers as a language, L_p.

And let the given number be a string, x.

The problem is now the same as deciding whether $x \in L_p$ or not.

If we have a machine that can recognize members of L_p, then we can give it x and it will give us the answer.