Module 0: Introduction

Text reading:

A ***string over an alphabet*** is a finite sequence of symbols from that alphabet, usually written next to one another and not separated by commas. If $\Sigma_1 = \{0,1\}$, then 01001 is a string over Σ_1. If $\Sigma_2 = \{\mathtt{a}, \mathtt{b}, \mathtt{c}, \ldots, \mathtt{z}\}$, then abracadabra is a string over Σ_2. If w is a string over Σ, the ***length*** of w, written $|w|$, is the number of symbols that it contains. The string of length zero is called the ***empty string*** and is written ε. The empty string plays the role of 0 in a number system. If w has length n, we can write $w = w_1 w_2 \cdots w_n$ where each $w_i \in \Sigma$. The ***reverse*** of w, written $w^{\mathcal{R}}$, is the string obtained by writing w in the opposite order (i.e., $w_n w_{n-1} \cdots w_1$). String z is a ***substring*** of w if z appears consecutively within w. For example, cad is a substring of abracadabra.

If we have string x of length m and string y of length n, the ***concatenation*** of x and y, written xy, is the string obtained by appending y to the end of x, as in $x_1 \cdots x_m y_1 \cdots y_n$. To concatenate a string with itself many times, we use the superscript notation x^k to mean

$$\overbrace{xx\cdots x}^{k}.$$

A ***language*** is a set of strings over alphabet which satisfy some property.

For example:
Consider alphabet $\Sigma = \{0,1\}$.

1. Set of strings over this alphabet of odd length:
   L1 = {0, 1, 000, 001, 011, 100, …}
2. Set of strings with equal number of 0 and 1:
   L2 = {01, 10, 0011, 0101, 1010, 1100, …}
3. Set of strings that are binary representation of a prime number:
   L3 = {10, 11, 101, 111, 1011, 1101, 10001, …}
4. Set of all strings over this alphabet:
   L4 = {ε, 0, 1, 00, 01, 10, 11, 000, 001, 010, 011, 100, 101, …}
   also written as Σ^*.

Important task – **membership testing:** given a string **s** over alphabet Σ, and a language L, is it true that $s \in$ L?

How is it related to the theory of computation?
Recall from CP312:

Problem: Given a problem instance, carry out a particular computational task.

Problem Instance: Input for the specified problem.

Problem Solution: Output (correct answer) for the specified problem.

Algorithm: An algorithm is a step-by-step process (e.g., described in pseudocode) for carrying out a series of computations, given some appropriate input.

Algorithm solving a problem: An Algorithm *A solves* a problem Π if, for every instance I of Π, A finds a valid solution for the instance I in finite time.

We restrict ourselves to decision problems only – those problems that have yes/no answer. For example:
- given a binary string, does it have odd length?
- given a binary string, does it contain the same amount of 0 and 1?
- given a number in binary representation, is it prime?

Yes-instance is an input to the problem that have answer **yes**.
Yes-instances of problems above are exactly string which belong to corresponding languages. You can treat the input to the problem as a string that encodes this input and if we have membership test for the language, we also have decision algorithm for the decision problem.

Remark. We will later show that considering only decision problems is not very restrictive.

Short proof that there are undecidable decision problems…
This is simple **counting** argument.

Consider the set of all decision problems (languages over fixed alphabet). Is it countable?

We are talking the cardinality of a set of infinite sets over alphabet… which is the same as the cardinality of the set of real numbers …

NO!

Consider the set of all decision algorithms (finite strings representing it – eq. Java program). Is it countable?

YES! (Sort the algorithms descriptions as the strings, as every algorithm is JUST FINITELY LONG STRING OF CHARACTERS). Count them …

So, there are MUCH MORE problems than the algorithms …

BINGO! We MUST have decision problems that do not have decision algorithms!!!!

NOT EVERY PROBLEM IS SOLVABLE!!!
NOT EVERY LANGUAGE IS DECIDABLE!!!