Lecture 1

Gidon Rosalki

2025-03-23

Notice: If you find any mistakes, please open an issue at `https://github.com/robomarvin1501/notes_computability_compl`

1 Introduction to the course content

Some of these concepts will only be defined further in the course, since in order to learn them you need to complete the course. Such is life.

Definitions: A clique is a collection of nodes in a graph where there is an edge between every pair of nodes.
Computational tasks: Examples

1. To sort a given array. We saw in DAST that this can be done in $O(n \log(n))$
2. Given a graph G, with 2 points s, t, find a shortest path from s to t. We can do this in $O(n^4)$ (possibly better as seen in DAST, but this is useful for comparison to the next problem)
3. Given a graph G what is the largest clicker in the graph. $O(2^n \cdot n^2)$ We know not an algorithm that solves this in less than exponential time.
4. Given a program in python, establish whether or not the program will stop. There is no solution to this problem (Google halting problem).

Above is examples with respect to complexity and computation. Models are useful, for example let us consider sorting an array. In DAST we used the comparison model, and as a result had a lower bound for sorting arrays in $O(n \log(n))$. In the macaroni model (collect the macaroni in your hand, push them against a flat surface with your other hand until you've reached the height of the longest, remove it, and repeat), we may sort the array in $O(n)$

2 Computational problem

Definition 2.1 (Alphabet)**.** *The finite non empty set Σ, or sometimes Γ*

Definition 2.2 (Word)**.** *A word on Σ is a finite string of letters from Σ. A word without any letters is still correct, and is ε.*

Definition 2.3 (Computational problem)**.** *An input Σ, an output Γ (can be, but doesn't need to be the same alphabet), and a function that finds for every input (word over Σ), the correct output words for the input (some set of output words)*

2.1 Decision problems

These are computation problems where the output alphabet is of the type $\Gamma = \{T, F\}$, and where the input of the problem is matched to the set of legal outputs that only contains one word, of length 1.

Example 1 (Decision)**.** *Given a graph G, and the nodes s, t, and a number k, return T if there is a path of length $\leq k$ from s to t in G*

2.2 We will prove

- There is no python program that solves the stopping problem.
- There is no realistic computational model that solves the stopping problem.
- There are some decision problems that are not solvable through python programs (python programs are not defined, nor will they be defined, as it is beyond the scope)

Definition 2.4 (Language)**.** *A set of words*

2.2.1 Clarification

There is a natural one to one and on matching between decision problems over Σ and languages over Σ. We will match to the decision problem defined over f the language L_f, which contains the words over Σ for which f returns true.

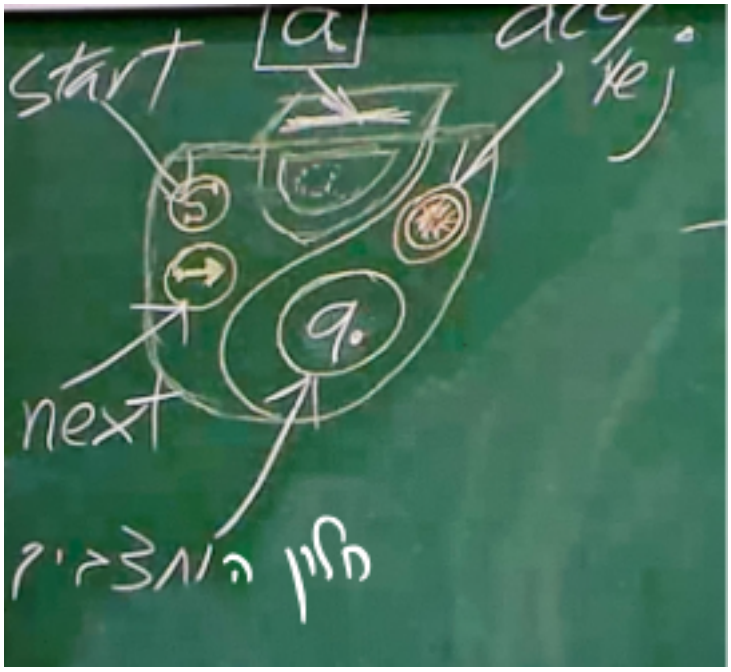
2.2.2 Counting

Given Σ, the number of words of length k over Σ is $1 \leq |\Sigma|^k < \aleph_0$.
The number of words of every length over Σ is $\aleph_0$.
The number of languages over Σ is $2^{\aleph_0} = \aleph > \aleph_0$.

There are $\aleph_0$ different python programs (python is a language, so it is just an ordering of letters, each program is a single word). There are $\aleph$ decision problems over $\Sigma = \{a, b\}$ or any other alphabet you may want. Since $\aleph > \aleph_0$, there are too many decision problems, and there exists decision problems over $\Sigma = \{a, b\}$ that do not have solutions in python programs. This is a decidedly uninteresting thing to know, since there are more decision problems than we can define.

3 Toy model

3.1 Deterministic Finite Automaton (DFA)



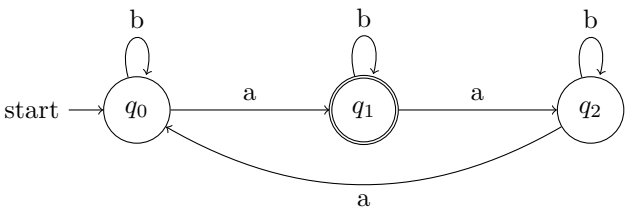
The automaton has a finite number of states, which are shown in this diagram at the window q. We press S to start, and the window shows q_0. When we press the right arrow, it uses the current letter from the input word to move to the next state q_1. Let us consider the input word aba.

1. $a \to q_1$
2. $b \to q_2$
3. $a \to q_3$

All these states q_i may be equivalent, they are numbered differently for ease of communication. The state of the automaton will dictate whether the light is lit or not to indicate T, F respectively.

3.1.1 Example

$\Sigma = \{a, b\}$. Our decision problem will be we accept a word $w \Leftrightarrow \#_a(w) = 1 \mod 3$, which is to say the number of a's in the word is $1 \mod 3$. So a, $abaaa$, and $bbbba$ are all correct words, but $ababab$ is not. The respective language is $L = \{w \subseteq \Sigma^n \ \forall n \in \mathbb{N} : \#_a(w) = 1 \mod 3\}$.
The concept: For $i = 0, 1, 2$ there will be the state in the automaton q_i. When there are i instances of $a \mod 3$, the automaton will arrive at the state q_i.
We will represent the automaton as a graph, and each node is a state of the graph.



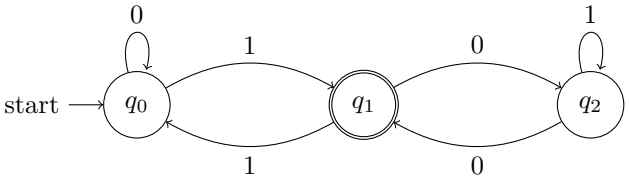
Let us call $L(A)$ the collection of words that the automaton receives. We can now ask if this is a correct solution by asking $L(A) = L$. If so, we say that the automaton **determines** L.

Theorem 1. $L(A) = L$

Proof . We will show that for every word w which has $1 mod 3$ a's within, the automaton will reach the wanted state. We will prove this by induction on the length of the word w.
Basis: $n = 0$: For $w = \varepsilon$ the automaton begins and ends in the starting state, which is q_0, as is needed.
Inductive assumption: we will assume it holds for n
Inductive step: We will prove for $n+1$. Let there be a word w of length $n+1$, and we will write $w = w'\alpha$, where w' is a word of length n, and $\alpha \in \{a, b\}$. Since w' is of length n, the automaton reaches state q_i where $i = \#_a(w') \pmod 3$. If $\alpha = b$, then the automaton will remain at q_i, which is suitable for the theorem. If $\alpha = a$, then the automaton will move from q_i to q_j where $j = i+1 \pmod 3$, and indeed $\#_a(w) = i+1 \pmod 3$. We have thus proven the inductive step, and have proven the theorem. □

Example 2. $\Sigma = \{0,1\}$, $L_b = \{w \subseteq \Sigma^n \ \forall n \in \mathbb{N} : (w)_2 = 1 \pmod 3\}$, *where* $(w)_2$ *is a binary number. So* $110 \notin L_b$, $0111 \in L_b$, $10000 \in L_b$.

Solution . When running the word w the automaton will reach the situation q_i when $i = (w)_2 \pmod 3$. For example $110 \to q_0$



The induction to prove this is incredibly similar to the previous example, and is left as an exercise for the reader. □

This can grant us solutions in number theory, for example this shows us that the number $(101111...1110)_2 = 1 \pmod 3$, where there are n 1's in the middle. This comes from the correctness of the state machine.

3.1.2 Questions on state machines

1. Is it possible to find the binary representation of prime numbers?
2. Is there a general simple characterisation for the collection of decision problems with automata?
3. Can we determine L or L' through a DFA? What about $L \cap L'$? $L \cup L'$? $\overline{L}$?

Definition 3.1 (DFA)**.** *A is a vector of 5 things:* $A = (\Sigma, Q, q_0, F, \delta)$ *where*

- Σ *is an alphabet*
- Q *is the non empty finite set of states*
- $q_0 \in Q$ *is the starting state*
- $F \subseteq Q$ *is the set of accepted finishing states*
- δ *is the transition function* $\delta : Q \times \Sigma \to Q$

Definition 3.2. *The running of an automaton A on the word* $w = \alpha_1\alpha_2 \ldots \alpha_n$ *is the series of states* $q_0, q_1, \ldots, q_n$*, which enables:*

- q_0 *is the starting state*
- $q_i = \delta(q_{i-1}, \alpha_i)$

Definition 3.3 (The extended transition function of A)**.** $\delta^*(q_0, w) = q_n$

So this allows us to write $L(A) = \{wover\Sigma : \delta^*(q_t, w) \in F\}$