Complexity Theory

Written by HADIOUCHE Azouaou.

Disclaimer

This course is one I write from some sources I read. Please be careful about the information given here.

> To seperate the contents of the course to actual additions or out of context information, a black band will be added by its side like the one englobing this comment.

Contents

Chapter 1

Introduction

Computational complexity theory, or complexity theory in short, is a theory made for classifying algorithms and computational problems by their time, memory, or other resources usage throughout their runtime which allows a clear objective comparison between algorithms performance in general.

It is easy to notice that for different kind of machines, the same algorithm will lead to different runtimes, and even in the same machine it can lead to multiple different runtimes. To be objective, we will start by making a model for a universal machine, the Turing machine, that allows us to compare algorithms independent of the hardware they run on.

Notation: *Let Σ be a set of alphabets. We define the following:*

- $\mathbb{N} = \{0, 1, 2, \ldots\}$ *and for* $n < m$, $[\![n, m]\!] = \{n, n+1, \ldots, m-1, m\}$.
- $\Sigma^n = \Sigma \times \cdots \times \Sigma$ *the set of all words written with* n *alphabets of* Σ, *for* $n = 0$, *we get* $\Sigma^n = \{\varepsilon\}$ *where* ε *is the empty word.*
- *If* $\Sigma = \{a\}$ *then* $\Sigma^n = \{a\}^n$ *and we denote it* a^n.
- $\Sigma^* = \cup_{n\in\mathbb{N}} \Sigma^n$ *the set of all possible words written with alphabets of* Σ.
- *Instead of using the notation* $x = (x_1, x_2, \ldots, x_n)$ *we will use the word notation* $x = x_1 x_2 \ldots x_n$.
- *If* $x = x_1 x_2 \ldots x_n \in \Sigma^n, y = y_1 y_2 \ldots y_m \in \Sigma^m$, *we define the concatenation operation* xy *such that* $xy = x_1 x_2 \ldots x_n y_1 y_2 \ldots y_m \in \Sigma^{n+m}$ *and for* S_1, S_2 *two sets of words, we define* $S_1 S_2 = \{xy \mid x \in S_1, y \in S_2\}$.

1. *Define the length* $|\cdot| : \Sigma^\star \to \mathbb{N}, \forall n \in \mathbb{N}, S \in \Sigma^n \Leftrightarrow |S| = n$.

1.1. Machines & Automata

We will present briefly some machines and their workflow, to give a general idea of how they work, and how they model a modern computation machine.

1.1.1. Deterministic Automaton

Definition (DFA): *Let* $M = (\Sigma, Q, \delta, q_0, F)$ *a* deterministic finite automaton *(DFA), it satisfies:*

- Σ *an alphabet.*
- Q *a finite set of states.*
- $\delta : Q \times \Sigma \to Q$ *a transition function.*
- $q_0 \in Q$ *a starting state.*
- $F \subseteq Q$ *a set of accepted/final states.*

A computation in M *is done as follows: let* $S = s_1 s_2 \ldots s_m \in \Sigma^*$, *define the sequence* $\{D_i\}_{i\in[\![1,m]\!]} \subseteq Q$ *that satisfies the recursion*

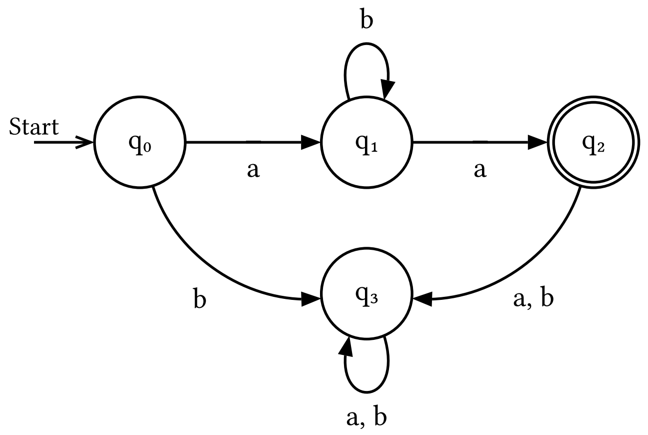
$$\begin{cases} D_0 = q_0 \\ D_i = \delta(D_{i-1}, s_i) \text{ with } i \in [\![1, m]\!] \end{cases}$$

M is said to accept S if and only if $D_m \in F$, it is said to be rejected otherwise. We define the language *of M denoted $\mathcal{L}(M)$ as the set of all strings in Σ^* that the machine M accepts.*

Note that the finite in DFA comes from the fact that the set of states is finite. We have a natural way to represent DFAs as a graph which is similar to discrete Markov chains. We take the following example: Let $M = (\Sigma, Q, \delta, q_0, F)$ with $\Sigma = \{a, b\}$, $Q = \{q_0, q_1, q_2, q_3\}$ and $F = \{q_3\}$ and the transition function is

$q \in Q$	$c \in \Sigma$	$\delta(q, c)$
q_0	a	q_1
q_0	b	q_2
q_1	a	q_2
q_1	b	q_1
q_2	a, b	q_3
q_3	a, b	q_2

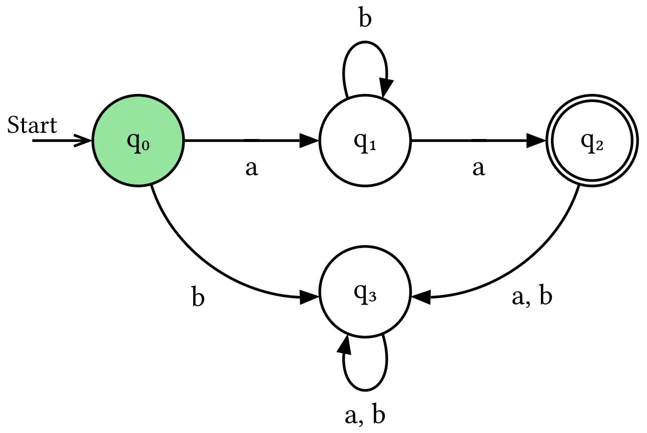
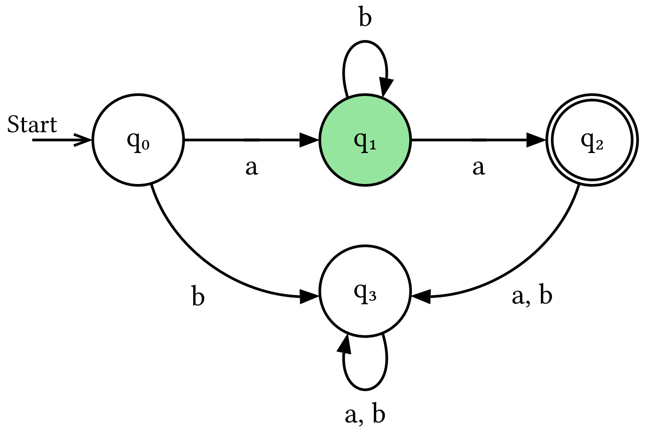
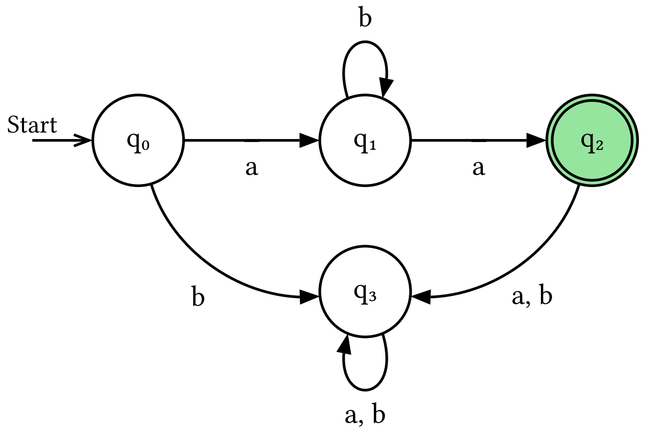
We represent the machine in the previous diagram, the starting state has an arrow “ start“ pointing to it, the final states having a double circle around them, the states are represented using circles with the name of the state inside, and the transitions are represented going from the state to the one it has to go through with the input given.



We will run it on multiple examples,

- $S = abba$
  1. $D_0 = q_0, S \leftarrow abba$
  2. $D_1 = \delta(D_0, s_1) = \delta(q_0, a) = q_1, S \leftarrow bba$
  3. $D_2 = \delta(D_1, s_2) = \delta(q_1, b) = q_1, S \leftarrow ba$
  4. $D_3 = \delta(D_2, s_3) = \delta(q_1, b) = q_1, S \leftarrow a$
  5. $D_4 = \delta(D_3, s_4) = \delta(q_1, a) = q_2, S \leftarrow \varepsilon$

Notice that $D_4 = q_2 \in F$ thus $S \in \mathcal{L}(M)$, we will represent these transitions using the machine and we will get the same result.

It is easy to notice that $\mathcal{L}(M) = \{ab^k a \mid k \in \mathbb{N}\}$, we will prove it

Proposition: *Let M the defined automaton in the previous example, then $\mathcal{L}(M) = \{ab^k a \mid k \in \mathbb{N}\}$.*

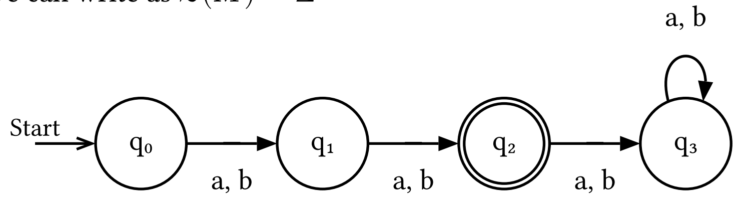
Proof. Let $S = s_1 s_2 \cdots s_n \in \mathcal{L}(M)$ and D_i the associated sequence of steps.

- $s_1 = a$: suppose by contradiction that $s_1 = b$, then $D_1 = \delta(D_0, b) = q_3$ thus we get $\forall i \in [\![1, n]\!], D_i = q_3$ but $q_3 \notin F$, so $s_1 = a$.
- $\forall i \in [\![2, n-1]\!], s_i = b$: since $s_1 = a$ then $D_2 = q_1$, if for some $i \in [\![2, n-1]\!], s_i = a$ then $D_i = q_2$ thus $D_{i+1} = q_3$ and using the same argument as before we get that $q_3 \notin F$ which is a contradiction, thus $\forall i \in [\![2, n-1]\!], s_i = b$.
- $s_n = a$: notice that $D_{n-1} = q_1$ thus if $s_n = b$, $D_n = q_1 \notin F$ hence $s_n = a$.

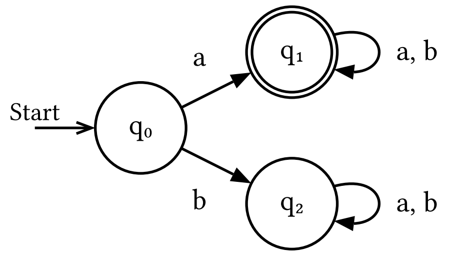
□

We will go through a series of examples of automata and their languages. We consider $\Sigma = \{a, b\}$.

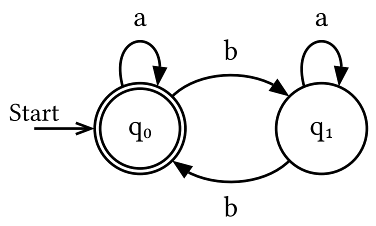
1. The language of the automaton below is the set of words with length exactly 2 which we can write as $\mathcal{L}(M) = \Sigma^2$



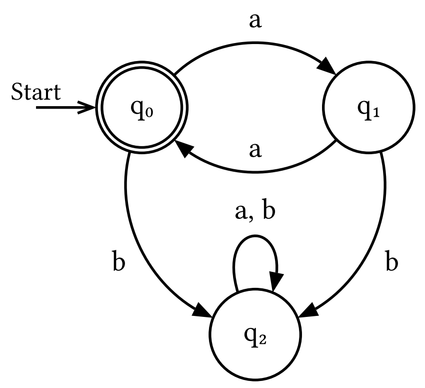
2. The language of the automaton below is the set of words starting with a, written as $\mathcal{L}(M) = a\Sigma^* = \{aw \mid w \in \Sigma^*\}$.



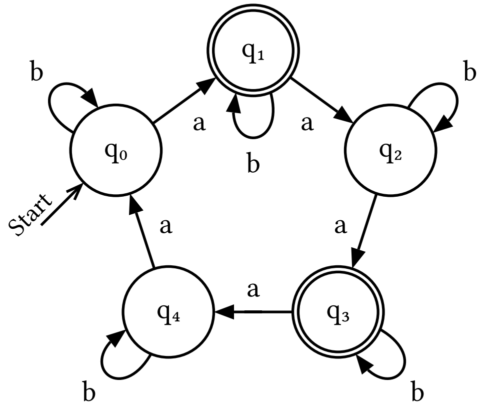
3. The language of the automaton below is the set of words that have an even number of bs, $\mathcal{L}(M) = \{S \in \Sigma^* \mid \# b(S) \text{ is even}\}$, notice in this case the $\varepsilon \in \mathcal{L}(M)$ since $\# b(\varepsilon) = 0$ is even too.



4. The language of the automaton below is the set of words that are of the form aa, $aaaa$, $aaaaaa$, ..., a^{2n}, $\mathcal{L}(M) = \{a^{2n} \mid n \in \mathbb{N}\}$.



5. The language of the automaton below is the set of words that have either 1 or 3 mod 5 as, $\mathcal{L}(M) = \{S \in \Sigma^* \mid (\# a(S) \bmod 5) \in \{1, 3\}\}$.



The expression of such a machine is really limited, that is, if the set of languages $\mathcal{L} \subseteq \Sigma^*$ that deterministic finite automatas can determine is really limited, the set of languages that the deterministic finite automata can detect is called regular languages. An example of a language that is not regular is $\mathcal{L} = \{a^n b^n \mid n \in \mathbb{N}\}$ where $\Sigma = \{a, b\}$.

1.1.2. Non-Deterministic Finite Automaton

A non-deterministic automaton is a generalization of the automaton, where we give it more abilities to be stronger than the deterministic finite automata.

Definition (NFA): *Let $M = (\Sigma, Q, \delta, q_0, F)$ a* non-deterministic finite automaton *(NDFA or NFA), it satisfies:*

- *Σ an alphabet.*
- *Q a finite set of states.*
- *$\delta : Q \times (\Sigma \cup \{\varepsilon\}) \to Q$.*
- *q_0 a starting state.*
- *$F \subseteq Q$ a set of accepted/final states.*

A computation in M is done as follows: let $S = s_1 s_2 ... s_m \in \Sigma^*$, S is said to be accepted by S if there exists a sequence $\{D_i\}_{i \in [\![1,m]\!]} \subseteq Q$ that satisfies the recursion.

$$\begin{cases} D_0 = q_0 \\ D_i \in \delta(D_{i-1}, s_i) \end{cases}$$

and $D_m \in F$, it is said to reject otherwise.

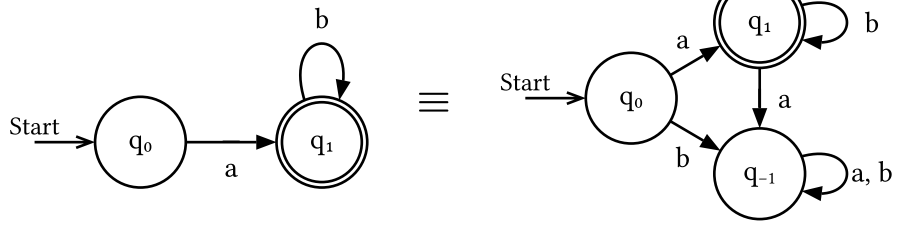
The definition above is taken directly from Wikipedia, I have a doubt about it so here is how I formalized it, even though it is a mouthful.

$$\begin{cases} D_0 = \{(q_0, S)\} \\ D_i = \bigcup\limits_{\substack{(q,S) \in D_{i-1} \\ S = s_1 s_2 ... s_n}} \left(\cup_{q' \in \delta(q, s_1)} \{(q', s_2 s_3 ... s_n)\}\right) \cup \left(\cup_{q' \in \delta(q, \varepsilon)} \{(q', S)\}\right) \end{cases}$$

M is said to accept S if and only if $\exists (q, \varepsilon) \in D_m, q \in F$ and it is said to reject otherwise.

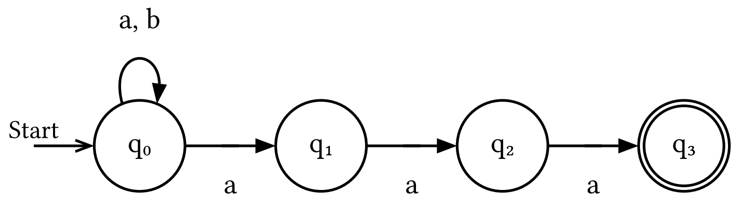
In this new model, we have 3 additional features:

1. The ability to jump without consuming any character from the string with the ε-transitions, that is, the transitions that have ε as the character.
2. The ability to jump to multiple states using just one letter of the string, this comes from the fact that $\delta(q, c)$ is a set of possible paths to take.
3. The implicit transitions to trap states, that is, if $\delta(q, c)$ is not defined, then we assume implicitly that it jumps to a state $q_\emptyset$ and does not transition to anything else. To do this, we just create a new state q_{-1}, and whenever there is no transition from a state q_i with letter c then we define $\delta(q_i, c) = \{q_{-1}\}$ and we define $\forall c \in \Sigma \cup \{\varepsilon\}, \delta(q_{-1}, c) = \{q_{-1}\}$.

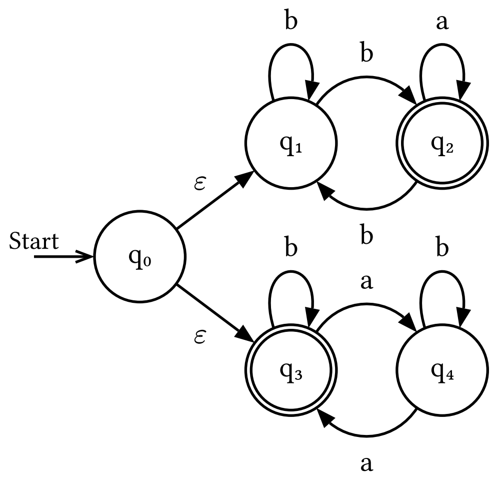


The non-determinism comes from the fact that we do not have a deterministic path we walk along, but each step may lead to many paths. We will go through a series of examples of NFAs to see how those features add expression.

1. The language of the automaton below is the set of words that end with 3 as, which we can write as $\mathcal{L}(M) = \Sigma^* aaa$. Notice that there are two transitions with a that go from q_0.



2. The language of the automaton below is the set of words that either have an odd number of bs or have an even number of as, which can be written as follows $\mathcal{L}(M) = \{S \in \Sigma \mid \# a(S) \text{ is even} \lor \# b(S) \text{ is odd}\}$.



From the previous examples, we can assume that an NFA will be stronger than an NFA, which is wrong. The set of languages that can be determined with an NFA is exactly the one that can be determined by a DFA. It is clear that any DFA is an NFA. Now we will prove that the languages from NFA can be determined using DFA too.

> **Theorem:** *Let M be a non-deterministic automaton, then there exists a deterministic automaton M' such that $\mathcal{L}(M) = \mathcal{L}(M')$.*

Proof. Let $M = (\Sigma, Q, \delta, q_0, F)$ be an NFA and define the function $r : Q \to \mathcal{P}(Q)$ such that $r(q) = \left\{q' \in Q \mid \exists (q_j)_{j \in [\![1,k]\!]}, q_1 = q, q_k = q', q_j \in \delta(q_{j-1}, \varepsilon)\right\}$, that is, $r(q)$ is the set of states that q can reach with using only ε-transitions. We define the DFA $M' = (\Sigma, Q', \delta', q_{0'}, F')$ where $Q' = \mathcal{P}(Q)$, $\delta'(\{q_1, ..., q_k\}, c) = \cup_{i=1}^{k} r(\delta(q_i), a)$, $q_{0'} = r(q_0)$, $F' = \{f \subseteq Q \mid f \cap F \neq \emptyset\}$. □

> **Note:** *The proof is incomplete, given that we have to prove that the languages are the same. I am finding a difficulty to prove it given that the definition given by Wikipedia seems to not make sense with the ε-transitions, and my definition is not usable in this context.*

We have proved the equivalence of expression of both models, but there is a difference. A DFA explores a single, unique path of computation for each step and goes step by step while the NFA can explore multiple paths in parallel. So even though in theory they express the same languages, in practice using the method given in the proof results in an exponential increase of time for the simulation of the same language using the DFA instead of NFA.