Automata, Computability & Complexity

Author: Daria Shutina

23-02-06

Org stuff

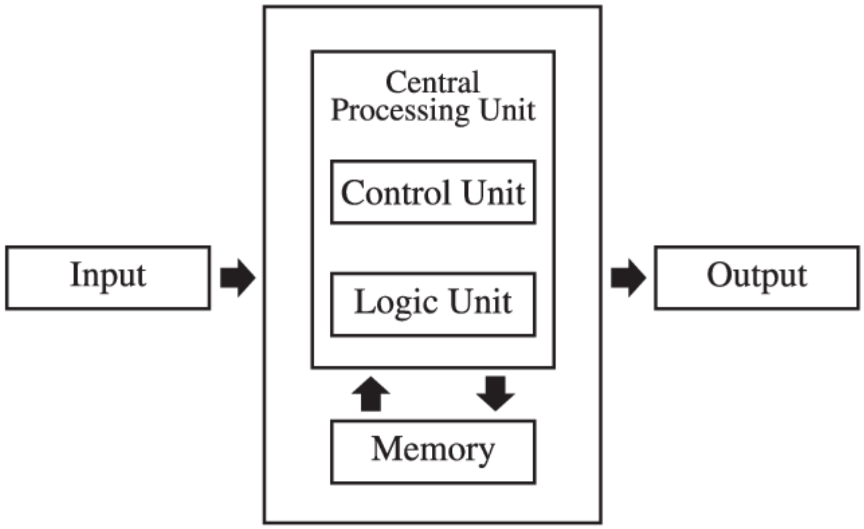
Timetable:

- Mondays 14:15 lectures offline. Moodle quiz in the beginning of each lecture
- Wednesdays 14:15 lectures offline
- Thursdays 17:15 consultations online. Need to book

Grade:

- 100% exam
- bonus 0.33 (5%) if 50% of quizes and 50% of homeworks are done

von Neumann architecture



Our nowadays computers are based on the von Neumann computer architecture.

Finite automata

For theory achievements, we will use a simpler compute model which gets the input as a string of symbols, has no memory, and generates an output that is either `accept` or `reject`. This machine is called *finite automaton.*

Deterministic finite automata

Deterministic means that an input symbol a leads from state q to exactly one possible state q'.

The vizualization of the input computation is a chain.

Definitions

Definition 1.1 (Finite automaton) A **finite automaton** (FA) M is a 5-tuple $M = (Q, \Sigma, \delta, q_0, F)$ where

1. Q is a finite set called the **states**,
2. Σ is a finite set called the **alphabet**,
3. $\delta : Q \times \Sigma \to Q$ is the **transition function**,
4. $q_0 \in Q$ is the **start state**, and
5. $F \subseteq Q$ is the **set of accept states / final states**.

A transition function can be described via *state transition diagram* (STD) or *state transition table*.

If a state is a *start state*, it has an arrow pointing from nowhere.

What is an *accept state*? Imagine we have an input. We go from one state to another, as the input says. Then we finish in a definite state. This state can be a final/accept state. In other words, it is where an input ends. Final states are shown as double-circled states in STD.

FA accepts a string if it starts in a start state, uses only transitions of δ and finishes in one of final states.

Definition 1.2 (Strings accepted by M) Let $M = (Q, \Sigma, \delta, q_0, F)$ be a finite automaton and $w = w_1 w_2 \cdots w_n$ be a string over alphabet Σ.
M **accepts** w if there exists a sequence of states $r_0, r_1, \ldots, r_n$, such that all following three conditions hold:

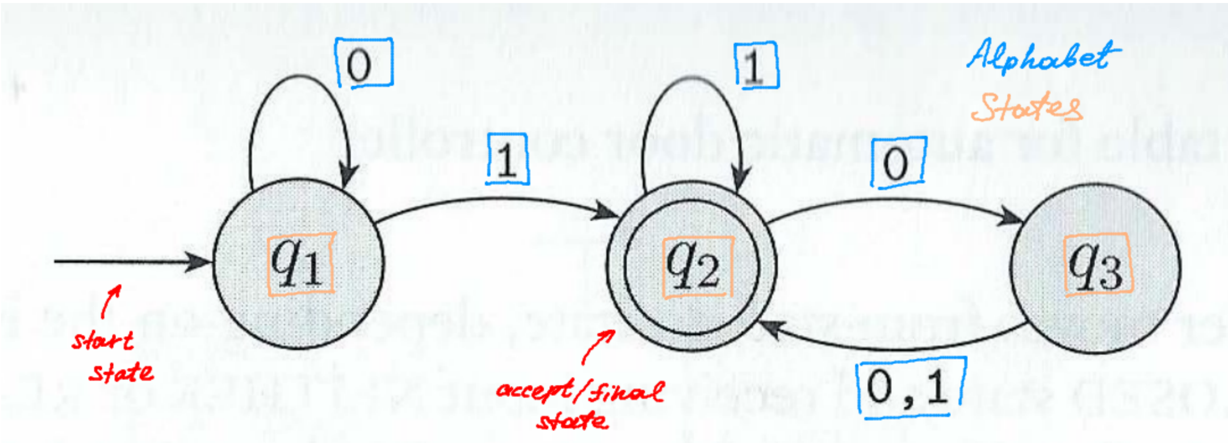
1. $r_0 = q_0$ *(M starts in start state.)*
2. $\delta(r_i, w_{i+1}) = r_{i+1}$, for $i = 0, \ldots n-1$ *(State change follows transition function.)*
3. $r_n \in F$ *(M ends up in accept state)*

If M does not accept w, it **rejects** it.

A *computation of FA on a string* is a sequence of states such that it starts in a start state and uses only transitions of δ.

$L(M)$ -- *the language of machine M* -- is the set of <u>all</u> strings that are accepted by M. Every FA stil recognizes and empty language $\varnothing$.

Example 1: a STD



$Q = \{q_1, q_2, q_3\}$; $\Sigma = \{0, 1\}$; $F = \{q_2\}$.

q_1 is a start state.

δ can be described with a table:

	0	1
q_1	q_1	q_2
q_2	q_3	q_2
q_3	q_2	q_2

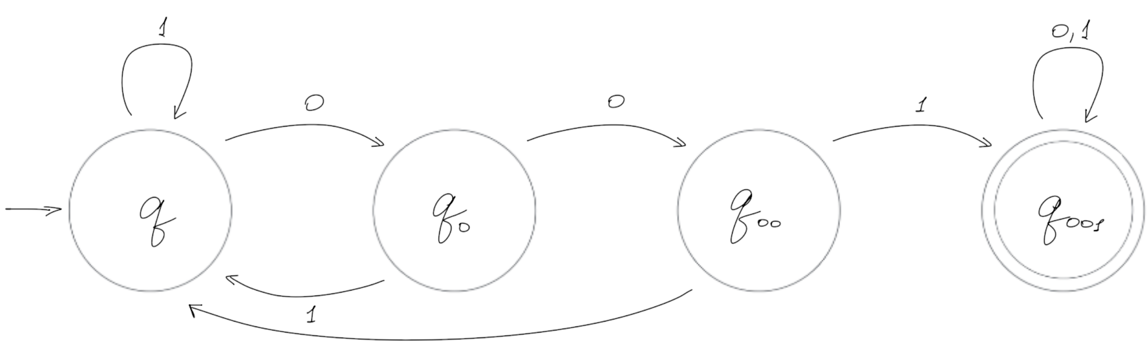
FA accepts a string 1101, for example, since it starts in q_1 and finishes in q_2.

Example 2: finding FA

We consider a language $L = \{w \mid w\ contains\ 001\ as\ substring\}$. The alphabet is $\{0, 1\}$.

The idea is that we will have 4 states:

- q -- no subsequence
- q_0 -- we have 0
- q_{00} -- we have 00
- q_{001} -- we have 001 and we neep to stop.



Example 3: finding a FA for a union of RLs

$M_1 = (S, \Sigma_1, \delta_1, q_1^0, , F_1)$

$M_2 = (T, \Sigma_2, \delta_2, q_2^0, F_2)$

We can finds a new FA for $M = (Q, \Sigma, \delta, q_0, F)$ where

$Q = S \times T$

$\Sigma = \Sigma_1 \cap \Sigma_2$

$\delta((s,t),a) = (\delta_s(s,a), \delta_t(t,a))$

$q_0 = (s_0, t_0)$

$F = F_1 \times F_2$

Regular language

A language is called a *regular language* if $\exists$ FA that recognizes it.

To prove that a language is regular, we need to build a FA that will recognize it. If we are able to build a STD, then the language is regular.

Example of a non-regular lenguage:

$L = \{0^n 1^n \mid n \geqslant 1\}$.

n is not fixed, so we have a problem with choosing a transition function.

Regular operations

Definition 1.6 (Regular operations) Let A and B be languages. We define the regular operations **union**, **concatenation** and **star** as follows:

- **Union**: $A \cup B := \{x | x \in A \text{ or } x \in B\}$
- **Concatenation**: $A \circ B := \{xy | x \in A \text{ and } y \in B\}$.
- **Star**: $A^* := \{x_1 x_2 \dots x_k | k \geq 0 \text{ and each } x_i \in A\}$.

Th 1.1 (unoin operation of RLs)

The class of regular languages is closed under the union operation:

A_1, A_2 are RLs $\Rightarrow A_1 \cup A_2$ is a RL.

Proof:

There are regular languages A_1 and A_2. W

There exists FAs M_1 and M_2 such that

$$\begin{cases} L(M_1) = A_1,\ M_1 = (S, \Sigma, \delta_1, s_0, F_1) \\ L(M_2) = A_2,\ M_2 = (T, \Sigma, \delta_2, t_0, F_2) \end{cases}$$

If M_1 and M_2 have different alphabets, then Σ will be the union of their alphabets.

To show that $A_1 \cup A_2$ is a RL, we construct a FA M such that $M = (Q, \Sigma, \delta, q_0, F)$

$Q = S \times T$

Σ is the same

$\delta((s,t),a) = (\delta_s(s,a), \delta_t(t,a))$

$q_0 = (s_0, t_0)$

$F = F_1 \times F_2$

Th. 1.2 (concatenation operation of RLs)

The class of regular languages is closed under the concatenation operation.

A_1, A_2 are RLs $\Rightarrow A_1 \circ A_2$ is a RL.

Proof: его нет

23-02-13

Nondeterministic finite automata

In contrast to deterministic FAs, nondeterministic FAs allow several successor states (or even none) for a given fixed input. An NFA adds more flexibility to the computation.

- There can be several transitions with the same symbol
- There can also be no transition for some symbol
- There can be additional label ϵ. It is like a special symbol (i.e., an empty string). Every alphabel has its own ϵ (provided that it has special symbols),

The vizualization of the computation is a tree.

Definitions

Definition 1.7 (Nondeterministic finite automaton) A **nondeterministic finite automaton** (NFA) is a 5-tuple $(Q, \Sigma, \delta, q_0, F)$, where

1. Q is a finite set of states,
2. Σ is a finite alphabet,
3. $\delta : Q \times \Sigma_\varepsilon \to \mathcal{P}(Q)$ is the transition function, with $\Sigma_\varepsilon := \Sigma \cup \{\varepsilon\}$,
4. $q_0 \in Q$ is the start state, and
5. $F \subseteq Q$ is the set of accept states.

In the transition functin definition, $P(Q)$ means a set of states, since in NFA we can get from one state to several (or zero) states.

NFA *accepts* a given input string, if there exists a computation branch in a tree that ends in an accept state; otherwise it *rejects* it.

Definition 1.8 (Strings accepted by NFA N)
Let $N = (Q, \Sigma, \delta, q_0, F)$ be a nondeterministic finite automaton and w be a string over alphabet Σ.

N **accepts** w if we can write w as $w = y_1 y_2 \dots y_m$, $y_i \in \Sigma_\varepsilon$ and if there exists a sequence of states $r_0, r_1, \dots, r_m$ (in Q), such that all following three conditions hold:

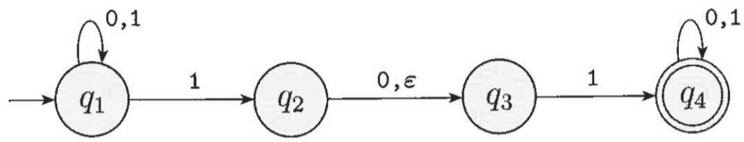
1. $r_0 = q_0$ *(N starts in start state.)*
2. $r_{i+1} \in \delta(r_i, y_{i+1})$, for $i = 0, \dots m-1$
   (State change follows transition function.)
3. $r_m \in F$ *(N ends up in accept state)*

If N does not accept w, it **rejects** it.

A *computation brance of NFA on a string* is a sequence of states such that it starts in a start state and uses only transitions of δ.

A computation branch is *accepting* if the last state after all transitions is an element of F; otherwise, it is a *rejecting* branch.

Example of NFA



Given the definition of an NFA as 5-tuple, N_1 is defined by

$$N_1 = (Q, \Sigma, \delta, q_1, F), \quad Q = \{q_1, q_2, q_3, q_4\}, \quad \Sigma = \{0, 1\}, \quad F = \{q_4\}\,,$$

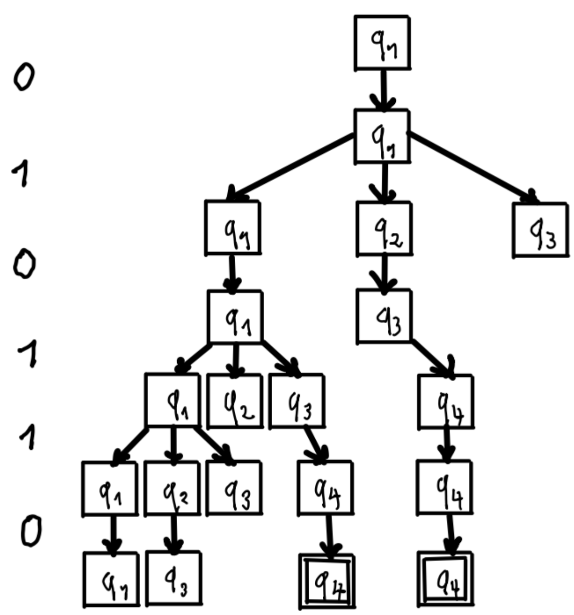
with transition function

δ	0	1	ε
q_1	$\{q_1\}$	$\{q_1, q_2\}$	$\emptyset$
q_2	$\{q_3\}$	$\emptyset$	$\{q_3\}$
q_3	$\emptyset$	$\{q_4\}$	$\emptyset$
q_4	$\{q_4\}$	$\{q_4\}$	$\emptyset$

Finally, we can to the conclusion that $L(N_1) = \{\, w \mid w \text{ contains either 101 or 11 as a substring} \,\}$.

Creating a tree for an input

Let us consider an input 010110 for the STD from the example above. The tree will be:



$q_1 \to q_3$ is $1 + \epsilon$ that is equal to 1.

Th 2.1 (equivalence between NFA and FA)

$\forall NFA\ \ N\ \ \exists FA\ \ M\ \ :\ \ L(M) = L(N)$.

In other words, all languages that can be recognized by an NFA, can also be recognized by some FA.

Proof:

We have an NFA $N = (Q, \Sigma, \delta, q_0, F)$ and we want to construct a deterministic FA $M = (Q', \Sigma, \delta', q'_0, F')$.

- Q' includes subsets of Q that are outcomes of δ: $Q' = P(Q)$.
- δ' get a success if one of considered gates gets a success: $\delta' = \bigcup\limits_{r \in R} \delta(r, a),\ \forall R \in Q', a \in \Sigma$.
- $q'_0 = \{q_0\}$. The start state becomes a set, because Q' consists of sets.
- $F' = \{\ R \in Q' \mid R \textbf{ contains an accept state of } N\ \}$

We do not have ϵ in FA. What to do with ϵ-transitions?

Let us create a set $E(R) = \{\ q \in Q \mid \exists \{s_i\}_{0..m}\ :\ s_0 \in R\ \wedge\ s_m = q\ \wedge\ s_{i+1} \in \delta(s_i, \epsilon)\ \}$.

A set R is a state for FA: $R \in Q'$.

A set $E(R)$ is a set of states for NFA: $E(R) \in Q$. It consists of such states that can be reached from R only via ϵ-transitions.

It is obvious that $R \in E(R)$: $\delta(r, \epsilon) = r,\ \forall r \in R$. Thus, we can modify our function δ': $\delta'(R, a) = \bigcup\limits_{r \in R} E(\delta(r, a)),\ \forall R \in Q', a \in \Sigma$. As we can see, a set of values for δ' can become only larger.

Also, we need to modify the start state: $q_0 = E(\{q_0\})$.

Corollary 2.1 (relation between RL and NFA)

Language is regular $\Leftrightarrow$ some NFA recognizes it.

Proof:

Language is regular $\Leftrightarrow \exists$ FA that recognizes it $\underset{\text{Th 2.1}}{\Leftrightarrow}$ $\exists$ NFA that recognizes it.

Th 2.2 (star operation of RLs)

The class of regular languages is closed under the star operation:

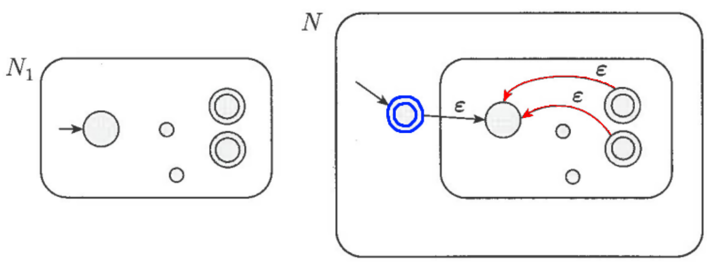
A is a RL $\Rightarrow$ A^* is an RL.

Proof:

Remainder: $A^* = \{\, x_1 x_2 \ldots x_k \ \ : \ k \geqslant 0 \ \wedge \ x_i \in A, \forall i \,\}$.

We have an NFA $N_1 = (Q_1, \Sigma, \delta_1, q_1, F_1)$ that recognizes A. We want to construct an NFA $N = (Q, \Sigma, \delta, q_0, F)$ such that it recognizes A^*.

In the STD we connect final states with the start state via ϵ-transitions, thus we get a concatenation of elements. Also, there is also an empty string ϵ as an input in A^*, so we choose a new start state by adding a new state to the initial start state:



- $Q = \{q_0\} \cup Q_1$
- $F = \{q_0\} \cup F_1$
- $\forall q \in Q, a \in \Sigma_\epsilon$:

$$\delta(q,a) = \begin{cases} \delta_1(q,a) & \text{if } q \in Q_1 \text{ and } q \notin F_1, & \textit{(main part of } N_1\textit{)} \\ \delta_1(q,a) & \text{if } q \in F_1 \text{ and } a \neq \varepsilon, & \\ \delta_1(q,a) \cup \{q_1\} & \text{if } q \in F_1 \text{ and } a = \varepsilon, & \textit{(loop to old start state)} \\ \{q_1\} & \text{if } q = q_0 \text{ and } a = \varepsilon, & \textit{(adding } q_0\textit{)} \\ \emptyset & \text{if } q = q_0 \text{ and } a \neq \varepsilon, \text{ and} & \end{cases}$$