Lecture 2

Gidon Rosalki

2025-03-30

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

1 Reminder

Σ - the alphabet we may use, Σ^* all the words of the alphabet Σ. A **language** on Σ: $L \subseteq \Sigma^*$. A DFA $= (\Sigma, Q, q_0, F, \delta)$. $L(A) \subseteq \Sigma^*$ is the collection of words that A accepts. We say that A **determines** the language $L(A)$. $L = L(A), \exists A \implies L$ is regular. REG is the collection of all the regular languages. Something to note is that most of the languages are not in REG. Note that $L_1, L_2 \in REG \implies \overline{L_1}, L_1 \cap L_2, L_1 \cup L_2 \in REG$

Given an automaton $A = (\Sigma, Q, q_0, F, \delta)$, what is the meaning $\delta(q, \alpha)$? It represents the state to which we will move from q when we call the letter α. Let us define the word $w = w'\alpha$:

$$\delta^*(q, w) = \delta(\delta^*(q, w'), \alpha)$$
$$\delta^*(q, \varepsilon) = q$$

So $\delta^*(q, w)$ is the state to which we will arrive if we begin at q, and move over all the letters of w. It is important to note that an automaton only defines δ, and δ defines δ^*. We may now write that

$$L(A) = \{w \in \Sigma^* : \delta^*(q_0, w) \in F\}$$

Theorem 1. $L_1, L_2 \in REG \implies L_1 \cup L_2 \in REG$
$L_1, L_2 \in REG \implies L_1 \cap L_2 \in REG$

Proof . Without loss of generality, we may assume that L_1 and L_2 are on the same alphabet Σ. How can this be without loss of generality? If $L_1 \subseteq \Sigma_1^*$ and $L_2 \subseteq \Sigma_2^*$, then we may take $\Sigma = \Sigma_1 \cup \Sigma_2$, and add a state called sink to the DFAs, such that there are no letters that leave the sink, and all the new letters not dealt with by the original DFA send to the sink. Let there be $A = (\Sigma, Q, q_0, F, \delta)$, such that $L(A) = L_1$, and $B = (\Sigma, P, p_0, G, \eta)$, such that $L(B) = L_2$.
The concept is that we will build an automaton with the set of states $Q \times P$. The word w will reach that state $(\delta^*(q_0, w), \eta^*(p_0, w))$.
Given that the concept works, the automaton that we will build is

$$C = (\Sigma, Q \times P, (q_0, p_0), (F \times P) \cup (Q \times G), \xi)$$

and if we're proving intersection then

$$C = (\Sigma, Q \times P, (q_0, p_0), F \times G, \xi)$$

It is sufficient to prove that the concept is true in order to finish. When ξ is defined by

$$\xi((q, p), \alpha) = (\delta(q, \alpha), \eta(p, \alpha))$$

We will finish by showing that for every word w it is true that

$$\xi^*((q_0, p_0), w) = (\delta^*(q_0, w), \eta^*(p_0, w))$$

We will prove this by induction on $|w| = n$.
Basis $n = 0$: In this case $w = \varepsilon$.

$$\begin{aligned}\xi^*((q_0, p_0), \varepsilon) &= (q_0, p_0) \\ &= (\delta^*(q_0, \varepsilon), \eta^*(p_0, \varepsilon))\end{aligned}$$

Step $n \to n+1$: Given w of length $n+1$, we will write $w = w'\alpha$.

$$\begin{aligned}
\xi^*\left((q_0,p_0),w'\alpha\right) &\overset{def}{=} \xi\left(\xi^*\left((q_0,p_0),w'\right),\alpha\right)\\
\text{Induction hypothesis} &= \xi\left(\left(\delta^*\left(q_0,w'\right),\eta^*\left(p_0,w'\right)\right),\alpha\right)\\
&\overset{def}{=} \left(\delta\left(\delta^*\left(q_0,w'\right),\alpha\right),\eta\left(\eta^*\left(p_0,w'\right),\alpha\right)\right)\\
&\overset{def}{=} \left(\delta^*\left(q_0,w'\alpha\right),\eta^*\left(p_0,w'\alpha\right)\right)\\
&= \left(\delta^*\left(q_0,w\right),\eta^*\left(p_0,w\right)\right)
\end{aligned}$$

□

If L_1, L_2 are languages on Σ, then

$$L_1 \cdot L_2 = \{w \cdot z : w \in L_1 \wedge z \in L_2\}$$

For example, if $L_1 = \{aa, ab\}, L_2 = \{ab, ac\}$, then

$$\{abac, aaab, abab\} \subseteq L_1 \cdot L_2$$

We will also define

$$\begin{aligned}
L_1^k &= L_1 \cdot L_1 \cdot \cdots \cdot L_1\\
L_1^0 &= \{\varepsilon\}\\
L^m \cdot L^k &= L^{m+k}\\
L \cdot \emptyset &= \emptyset\\
L^* &= \bigcup_{k=0}^{\infty} L^k
\end{aligned}$$

The final line is called the Kleene closure of L.

1.0.1 Examples

$$\begin{aligned}
L = \{a^n : n \geq 0 \wedge 2 \mid n\} &\implies L^* = L\\
L = \{a\} &\implies L^* = \{a^n : n \geq 0\}\\
L = \{a, b\} &\implies L^* = \{a, b\}^*
\end{aligned}$$

Theorem 2. *1. $\emptyset, \{\varepsilon\}, \{a\} \in REG$, which is to say, languages of size 1, or the empty language*

2. REG is closed to union, concatenation, and a Kleene closure

3. Every language in REG may be built from languages of type 1 through operation of type 2

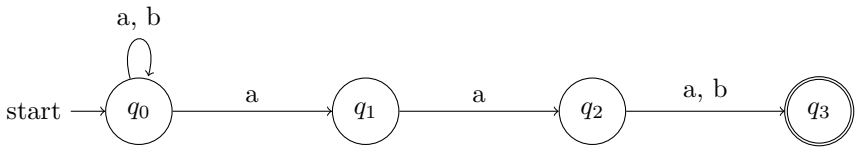
2 NFAs

In this section we will define

1. Non-deterministic Finite Automaton
2. We will define NREG, all the languages that characterise NFAs
3. We will show that NREG is closed to concatenation and Kleene closures
4. We will show that NREG is equivalent to REG

Given a word, the automaton has a set of runs that finish. The automaton A accepts a word w if there is a correct run on A that finishes in the accepting state.
Example: $L_1 = \{w \in \{a, b\} : w \text{ finishes with } aaa \vee aab\}$.

The concept is that we will leave the starting state only on the third to last letter:

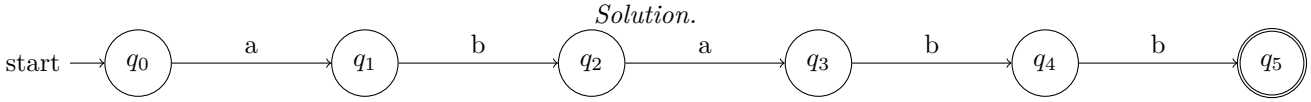


Using the above NFA, on the word *abaab*, we may have the runs $q_0, q_0, q_0, q_0, q_0,\ q_0, q_1, x, x, x, x,\ q_0, q_0, q_0, q_1, q_2, q_3$, and of these 3, only the last run is accepted. Let us call this NFA A. We will say that A identifies $L(A)$, rather than determines, since it is non deterministic.

Let there be A, B to NFAs, such that A identifies L, and B identifies L'. We want to show that $L, L' \in NREG \implies L \cup L' \in NREG$. We can do this by simply pretending A and B to be the same automaton, all their start states are the new NFA's start states. If the word is located in one of the languages L or L', then the same run that identifies it in the original NFA will also identify it in our new NFA.

Example 1. *Given the language $L_2 = \{ababb\}$, write an NFA that identifies L_2*

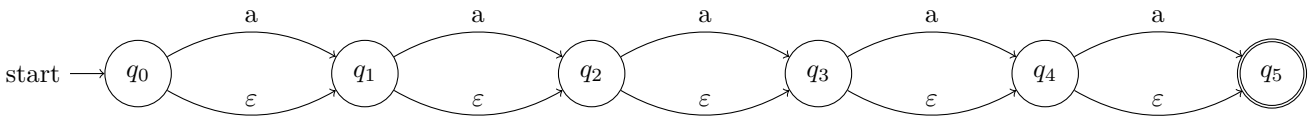
Solution.



□

Example 2. *$L_3 = \{$All the words that may be made fromababb by deleting some or all of the letters$\}$. For example, $aa \in L_3$.*

Solution. To achieve this we will add the epsilon path to each of the nodes, which allows us to start from any given node, and skip any given node:



□