Lecture 3

Gidon Rosalki

2025-04-06

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

1 Reminder

A legal run on $w \in \Sigma^*$ from $q \in Q$: For the letters $w = w_1 \dots w_k$ is a series of states $q^0, \dots, q^k$, such that $q^0 = q$, and $q^i \in \delta\left(q^{i-1}, w_i\right)$.
A run on w is a correct run on w that starts at a state within Q_0.
The extended transition function of A: $\delta^* : 2^Q \times \Sigma^* \to 2^Q$ where $\delta^*(S, w)$ is

1. The collection of states that we may get to from a correct run on w that starts in some state within S.
2. S when $w = \varepsilon$, otherwise when $w = w'\alpha$.

Theorem 1. *1 and 2 are equivalent definitions for δ on w*

Proof. This is not a proof, but the proof is by induction.

□

Theorem 2. $NREG = REG$

Proof .

- $REG \subseteq NREG$: This is true since a DFA is turned into an NFA through a very simple change in minor definitions.
- $NREG \subseteq REG$: Let there be $L \in NREG$, and $A = (\Sigma, Q, Q_0, F, \delta)$, an NFA that recognises it. We will build an automaton A_d that decides L through A_d. We know that A_d has the same alphabet, but what is its states? The concept: A_d will have a state for every subset $S \subseteq Q$, when running A_d on w, A_d will get to state $\delta^*(Q_0, w)$. Considering this, let us define $A_d = \left(\Sigma, 2^Q, Q_0, F_d, \delta_d\right)$, where

$$\begin{aligned} F_d &= \{S \subseteq Q : S \cap F \neq \emptyset\} \\ \delta_d(S, \alpha) &= \delta^*(S, \alpha) \end{aligned}$$

  Our theorem is that $L(A_d) = L(A)$, and it is sufficient to show that $\delta^*(Q_0, w) = \delta_d^*(Q_0, w)$, since if we do show this then we get $w \in L(A_d) \Leftrightarrow \delta_d^*(Q_0, w) \in F_d \Leftrightarrow \delta^*(Q_0, w) \in F_d \Leftrightarrow \delta^*(Q_0, w) \cap F \neq \emptyset \Leftrightarrow w \in L(A)$.
  By induction on $|w|$:
  Basis: $\delta^*(Q_0, \varepsilon) = Q_0 = \delta^*(Q_0, \varepsilon)$.
  $w = w'\alpha$:

$$\begin{aligned} \delta_d^*(Q_0, w'\alpha) &= \delta_d\left(\delta_d^*(Q_0, w'), \alpha\right) \\ &= \delta_d\left(\delta^*(Q_0, w'), \alpha\right) \\ &= \delta^*\left(\delta^*(Q_0, w'), \alpha\right) \\ &= \delta^*(Q_0, w'\alpha) \end{aligned}$$

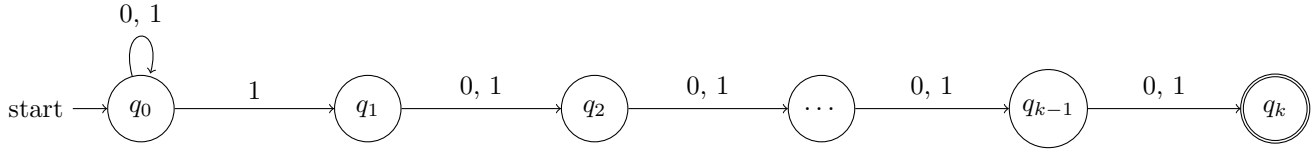
  as required.

□

Do we absolutely need the exponential growth of states? Yes. Example: Let there be $\Sigma = \{0, 1\}$, for $k \in \mathbb{N}$, we will define

$$L_k := \Sigma^* \cdot \{1\} \cdot \Sigma^{k-1}$$

The NFA



recognises the alphabet through $k+1$ states.

Definition 1.1 (Myhill-Nerode (MN)). *If L is a language on Σ, and $x, y \in \Sigma^*$. We will say that x and y are not MN-equivalent with respect to L if there exists a word $z \in \Sigma^*$ such that $xz \in L \wedge yz \notin L \vee xz \notin L \wedge yz \in L$. In this case we will write $x \not\sim_L y$ and we will write that z is the differentiating suffix. If there is no differentiation suffix, then we will write that $x \sim_L y$, and say that they are equivalent with respect to L.*

Benefits:

1. We will assume that $x \sim_L y$ and $x \in L$, then $y \in L$
2. We will assume that $x \not\sim_L y$, and $L = L(A)$ where $A \in DFA$, then when running on x and on y, A will reach different states, since otherwise A would reach the same state on xz, and on yz for every suffix z.
3. If there are n words in Σ^*, that are not equivalent, then A has at least n states. Or If in the ratio $\sim_L$ there are k equivalency sets, then in the DFA that decides L, there are at least k states.

Conclusion If there are ∞ equivalency states $\sim_L$, then $L \notin REG$.
To sum up, if $\sim_L$ has at least k sets, then in the automaton that decides L, there are at least k states, therefore if there is A that decides L, then the number of sets in $\sim_L$ is at most n

For every 2 different words $x, y \in \{0,1\}^*$, $x \not\sim_{L_k} y$: For words such as these, there exists $1 \leq i \leq k$ such that $x_i \neq y_i$. We will define $z = 0^{k-i+1}$. This way the ith letter is k before the end of both words. This is to say that in xz, the kth letter before the end is x_i, and is y_i in yz. So only one of xz and yz will be in the language L_k, and so $y \not\sim_{L_k} x$, and so the automaton that decides L_k has at least 2^k states.

$\Sigma = \{a, b\}$, $L = \{a^n b^n : n \geq 0\}$. For every $m \neq n$, $a^n \not\sim_L a^m$. This comes from $a^n b^n \in L, a^m b^n \notin L$, and so $L \notin REG$.

Symbol: $[w]_L$ the equivalency set of w with respect to L (as defined in discrete maths)

Theorem 3. *If L is a language on Σ, and in $\sim_L$ there are $k < \infty$ equivalency sets, then $L \in REG$. and there is a DFA for L with k states.*

Proof . Concept: We will symbolise the set of equivalency sets in $\sim_L$ as Q. These will be the states, and thus $\delta^*(q_0, w) = [w]_L$.
Building A: $A = (\Sigma, Q, [\varepsilon]_L, F, \delta)$ where

$$F = \{[w]_L : w \in L\}$$
$$\delta([w]_L, \alpha) = [w\alpha]_L$$

We need to verify that if $[w]_L = [y]_L$, then $[w\alpha]_L = [y\alpha]_L$:

$$\begin{aligned}
[w]_L = [y]_L &\implies w \sim_L y \\
&\implies \forall z\ wz \in L \Leftrightarrow yz \in L \\
&\implies \forall z'\ w\alpha z' \in L \Leftrightarrow y\alpha z' \in L \\
&\implies w\alpha \sim_L y\alpha \\
&\implies [w\alpha]_L = [y\alpha]_L
\end{aligned}$$

□