MTL783: Theory of Computation	1st Semester, 2025-2026

Lecture 6 — 11/08/2025

Lecturer: Prof Minati De *Scribe: Team 6*

Scribed by:

1. Rohan Chaturvedi (2022MT11262)
2. Anany Mishra (2022MT61965)
3. Dev Elvis Kannath (2022MT61961)
4. Shamil Mohammed (2022MT11938)
5. Rudranil Naskar (2022MT11287)

1 Overview

The lecture introduced *regular expressions* and the *languages associated* with them, describing their basic building blocks, the formal rules used to combine these components, and how each expression corresponds to a specific set of strings. It also discussed the formal definition of the language represented by a regular expression and stated the result that every such language can be recognized by some non-deterministic finite automaton (NFA), making it a regular language.

2 Main Section

2.1 Definitions

Definition: Regular Expression Let Σ be an alphabet. A *regular expression* over Σ is defined inductively as follows:

1. The following are primitive regular expression
   (a) $\emptyset$ is a regular expression, denoting the empty set.
   (b) λ is a regular expression, denoting the language $\{\lambda\}$ containing only the empty string.
   (c) For every $a \in \Sigma$, a is a regular expression, denoting the language $\{a\}$.
2. If r and s are regular expressions, then the following are also regular expressions:
   (a) $(r + s)$ — denoting the union of the languages of r and s,
   (b) (rs) — denoting the concatenation of the languages of r and s,
   (c) (r^*) — denoting the Kleene star of the language of r.
3. A string is a regular expression if and only if it can be obtained from the primitive regular expressions using a finite number of applications of the rule (2)

Example: Let $\Sigma = \{a, b\}$.

- (a^*b) is a regular expression, where $\Sigma = \{a, b\}$.
- $(a^*b)^*c$ is a regular expression, where $\Sigma = \{a, b, c\}$.
- a^*b+ is *not* a regular expression, since $+$ requires a valid regular expression on both sides.

Definition: Language Associated with a Regular Expression For a regular expression r, the *language* $L(r)$ is the language associated with r. The language $L(r)$ is defined inductively as follows, where r and s are regular expressions :

1. $L(\emptyset) = \emptyset$
2. $L(\lambda) = \{\lambda\}$
3. $L(a) = \{a\}$ for each $a \in \Sigma$
4. $L(r + s) = L(r) \cup L(s)$
5. $L(rs) = L(r) \cdot L(s) = \{xy \mid x \in L(r), y \in L(s)\}$
6. $L((r)) = (L(r))$
7. $L(r^*) = (L(r))^*$

Example: For $\Sigma = \{a, b\}$:

- $L(a^*) = \{\lambda, a, aa, aaa, \dots\}$
- $L(a^*(a + b)) = \{a, b, aa, ab, aaa, aab, \dots\}$

Note on Operator Precedence: When parentheses are not explicitly provided in a regular expression, the following *precedence order* is assumed:

1. **Kleene star** $(^*)$ has the highest precedence.
2. **Concatenation** comes next.
3. **Union** $(+)$ has the lowest precedence.

For example:

- $ab^* + c$ is interpreted as $(a(b^*)) + c$.
- $a + bc^*$ is interpreted as $a + (b(c^*))$.

2.2 Equivalence of Regular Expressions and NFAs

Regular expressions and non-deterministic finite automata (NFAs) are two distinct but equivalent formalisms for describing the class of regular languages. In this section, we establish one direction of this equivalence by proving that for every regular expression R, there exists an NFA that accepts the language $L(R)$. This demonstrates that all languages defined by regular expressions are regular.

Theorem 1. *Let R be a regular expression. Then there exists a non-deterministic finite automaton (NFA) that accepts $L(R)$. Consequently, $L(R)$ is a regular language.*

Proof. We prove by structural induction on the regular expression R that there exists an NFA N such that $L(N) = L(R)$.

Base cases: We first handle the *primitive regular expressions*, and directly construct NFAs for each of them. These serve as the starting point for the inductive proof.

1. $R = \emptyset$.
   Let $N = (\{q_0\}, \Sigma, \delta, q_0, \varnothing)$ where $\delta(q_0, a) = \varnothing \ \forall\, a \in \Sigma$; then $L(N) = \varnothing$.

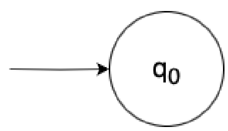


Figure 1: NFA for $R = \emptyset$

2. $R = \lambda$.
   Take $N = (\{q_0\}, \Sigma, \delta, q_0, \{q_0\})$ where $\delta(q_0, a) = \varnothing \ \forall\, a \in \Sigma$. Then $L(N) = \{\lambda\}$.

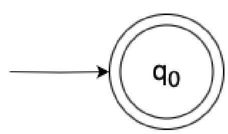


Figure 2: NFA for $R = \lambda$

3. $R = a$ for some $a \in \Sigma$.
   Take $N = (\{q_0, q_1\}, \Sigma, \delta, \{q_0\}, \{q_1\})$ with $\delta(q, w) = \begin{cases} \{q_1\}, & \text{if } w = a \textit{ and } q = q_0 \\ \varnothing, & \text{otherwise.} \end{cases}$. Then $L(N) = \{a\}$.

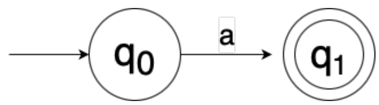


Figure 3: NFA for $R = a$

Inductive step. Assume for regular expressions r_1 and r_2 there exist NFAs $N_1 = (Q_1, \Sigma, \delta_1, q_1, F_1)$ and $N_2 = (Q_2, \Sigma, \delta_2, q_2, F_2)$ with $L(N_1) = L(r_1)$ and $L(N_2) = L(r_2)$. We construct NFAs for r_1+r_2, r_1r_2, and r_1^*.

- **Union:** $R = (r_1 + r_2)$.

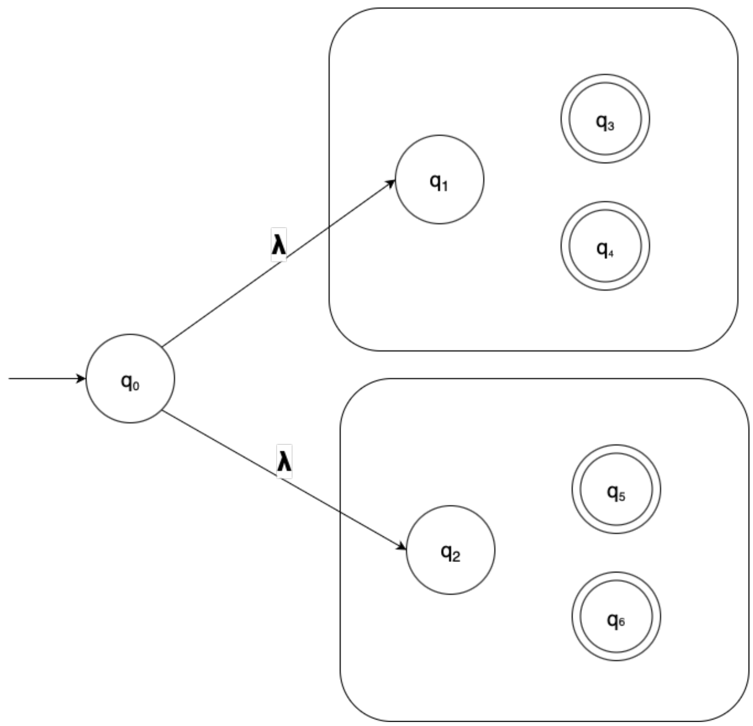


Figure 4: NFA for $R = r_1 + r_2$

Construct $N = (Q, \Sigma, \delta, \{q_0\}, F)$ where

$$Q = Q_r \cup Q_s \cup \{q_0\}, \qquad F = F_1 \cup F_2,$$

and

$$\delta(q, a) = \begin{cases} \delta_1(q, a) & \text{if } q \in Q_1, \\ \delta_2(q, a) & \text{if } q \in Q_2, \\ \{q_1, q_2\} & \text{if } q = q_0 \text{ and } a = \lambda, \\ \varnothing & \text{if } q = q_0 \text{ and } a \neq \lambda. \end{cases}$$

The NFA N recognizes $L(r_1) \cup L(r_2)$ as follows. Any accepting run of N begins at the new start state q_0, from which it nondeterministically moves via a λ-transition to either the start state of N_1 or N_2. Once inside N_1 or N_2, the run proceeds according to the transitions of that NFA and ends in one of its original final states, which are also final in N.

First, consider a word w that is accepted by N. Since the run must pass through either N_1 or N_2, as the first λ transition takes you to their starting , it follows that w is accepted by one of these NFAs. Therefore, $w \in L(r_1) \cup L(r_2)$.

Conversely, consider a word $w \in L(r_1) \cup L(r_2)$. Then w is accepted by either N_1 or N_2. In N, starting at q_0 and taking the λ-transition to the corresponding start state ensures that w follows the same accepting run as in N_1 or N_2, reaching one of the final states of N. Hence, w is accepted by N.

Combining these two observations, we conclude that $L(N) = L(r_1) \cup L(r_2)$.

- **Concatenation:** $R = (r_1 r_2)$.

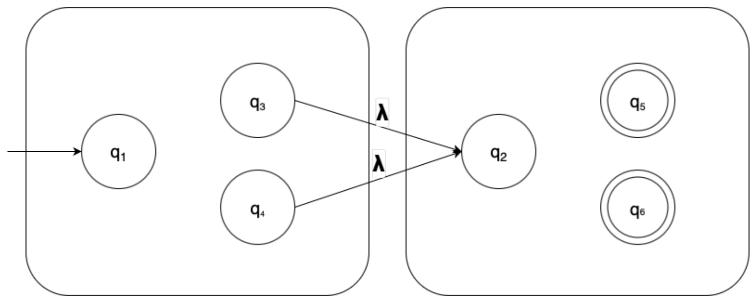


Figure 5: NFA for $R = r_1 r_2$

Construct $N = (Q, \Sigma, \delta, \{q_1\}, F)$ where

$$Q = Q_r \cup Q_s, \qquad F = F_{r_2},$$

and

$$\delta(q,a) = \begin{cases} \delta_1(q,a) & \text{if } q \in Q_1 \text{ and } q \notin F_1, \\ \delta_2(q,a) & \text{if } q \in Q_2, \\ q_2 \cup \delta_1(q,a) & \text{if } q \in F_1 \text{ and } a = \lambda, \\ \delta_1(q,a) & \text{if } q \in F_1 \text{ and } a \neq \lambda. \end{cases}$$

The NFA N constructed for $R = r_1 r_2$ recognizes the language $L(r_1) \cdot L(r_2)$ as follows. Any accepting run of N starts at the start state of N_1 and proceeds according to the transitions of N_1 until it reaches a final state in F_1. From each final state of N_1, a λ-transition leads to the start state of N_2, after which the run continues according to the transitions of N_2 and ends in one of the final states of F_2, which are the final states of N.

First, consider a word $w \in L(N)$. By construction, any accepting run of N must pass through N_1 followed by N_2, as the initial state is in N_1 and final in N_2 with no transition from N_2 to N_1. Therefore, w can be split as $w = w_1 w_2$, where w_1 is accepted by N_1 and w_2 is accepted by N_2. Hence, $w \in L(r_1) \cdot L(r_2)$.

Conversely, consider a word $w \in L(r_1) \cdot L(r_2)$. Then w can be written as $w = w_1 w_2$, where $w_1 \in L(r_1)$ and $w_2 \in L(r_2)$. In N, starting from the start state of N_1, the NFA processes w_1 according to N_1, reaches a state in F_1, takes the λ-transition to the start of N_2, and then processes w_2 according to N_2, ending in a state of F_2. Therefore, w is accepted by N.

Combining these two observations, we conclude that $L(N) = L(r_1) \cdot L(r_2)$.

- **Kleene star:** $R = (r_1^*)$.

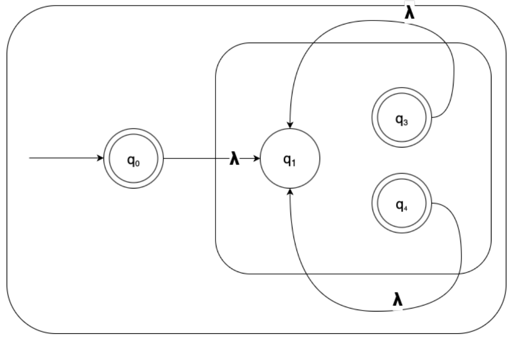


Figure 6: NFA for $R = r_1^*$

Construct $N = (Q, \Sigma, \delta, q_0, F)$ where

$$Q = Q_1 \cup \{q_0\}, \quad F = F_1 \cup \{q_0\}.$$

The transition function δ is defined as:

$$\delta(q,a) = \begin{cases} \delta_1(q,a) & \text{if } q \in Q_1 \text{ and } q \notin F_1, \\ \{q_1\} \cup \delta_1(q,a) & \text{if } q \in F_1 \text{ and } a = \lambda, \\ \delta_1(q,a) & \text{if } q \in F_1 \text{ and } a \neq \lambda, \\ \{q_1\} & \text{if } q = q_0 \text{ and } a = \lambda, \\ \varnothing & \text{if } q = q_0 \text{ and } a \neq \lambda. \end{cases}$$

The NFA N constructed for $R = r_1^*$ recognizes the language $L(r_1)^*$ as follows. Any accepting run of N begins at the new start state q_0, which is also a final state. From q_0, the NFA can either accept the empty string immediately or take a λ-transition to the start state of N_1 to process a word in $L(r_1)$. After processing a word in N_1 and reaching a final state in F_1, the NFA can take a λ-transition back to the start state of N_1 to repeat the pattern, or stop and accept, since q_0 and all states in F_1 are considered final states in N.

First, consider a word $w \in L(N)$. Any accepting run of N consists of zero or more iterations of runs through N_1, starting at q_0 and ending at F_1, so each transition from the final state to q_0 starts a new iteration. This implies that w can be written as a concatenation $w = w_1 w_2 \cdots w_k$ where each $w_i \in L(r_1)$, and $k \geq 0$. Therefore, $w \in L(r_1)^*$.

Conversely, let $w \in L(r_1)^*$. Then w can be decomposed as $w = w_1 w_2 \cdots w_k$ with each $w_i \in L(r_1)$ and $k \geq 0$. Starting at q_0, the NFA N can take a λ-transition to the start state of N_1 to process w_1, then after reaching a final state in F_1, either take a λ-transition back to the start of N_1 for w_2, and so on, until all w_i are processed. Since q_0 and all F_1 states are final, the run ends in a final state, and thus w is accepted by N.

Combining both directions, we conclude that $L(N) = L(r_1)^*$.

Since every regular expression is built from the base cases using a finite number of the above operations, repeated application of the constructions yields an NFA accepting $L(R)$. Hence every language denoted by a regular expression is accepted by some NFA, and so is a regular language.

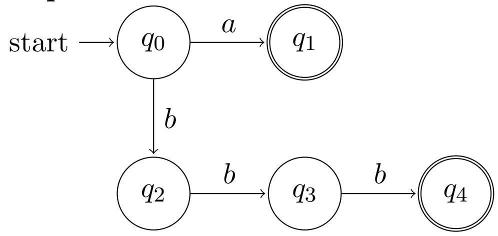
□

Question: Find an NFA which accepts $L(R)$, where

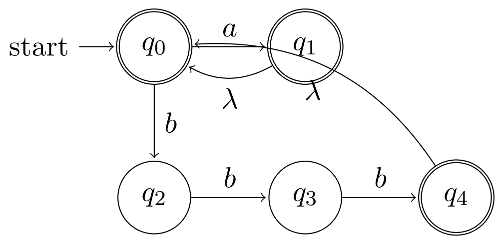
$$r = (a + bbb)^*(ba^* + \lambda)$$

Answer: The following are the steps of the formation of the NFA:

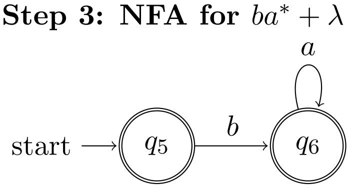
Step 1: NFA for $a + bbb$



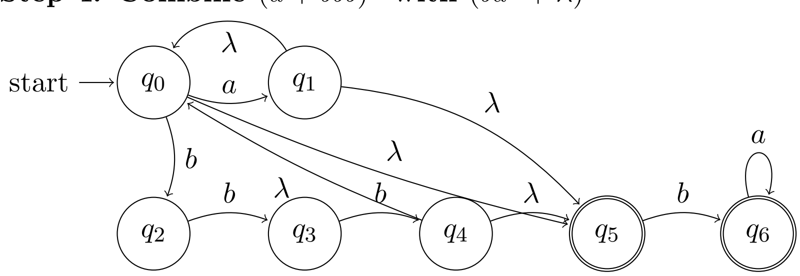
Step 2: NFA for $(a + bbb)^*$ We can construct a compact NFA without introducing a new start state because q_0 did not have any incoming transitions in the original NFA for $a + bbb$. By adding λ-transitions from the final states q_1 and q_4 back to q_0, we achieve the Kleene star behavior.



Step 3: NFA for $ba^* + \lambda$



Step 4: Combine $(a + bbb)^*$ **with** $(ba^* + \lambda)$



Homework Questions: Find NFA of regular expression r associated with

1. $L_1(r) = \{\, v \in \{0, 1\}^* \mid v \text{ has at least one pair of consecutive zeroes}\}$
2. $L_2(r) = \{\, v \in \{0, 1\}^* \mid v \text{ has no pair of consecutive zeroes}\}$
3. $L_3(r) = \{\, v \in \{0, 1\}^* \mid v \text{ has exactly one pair of consecutive zeroes}\}$

Solutions:

1. *Regular expression*: $r_1 = (0+1)^*00(0+1)^*$
   Explanation: Any string that contains at least one "00" can have any combination of 0s and 1s before and after the first occurrence of "00". Hence, $(0+1)^*$ before and after, and "00" in the middle.

2. *Regular expression*: $r_2 = (1+01)^*(\lambda+0)$
   Explanation: To avoid consecutive zeroes:

   - Each 0 must be immediately followed by 1 or be at the end.
   - 1s can appear anywhere.

   The pattern $(1+01)^*$ ensures no "00" appears, and $(\lambda+0)$ allows for a single trailing 0 at the end if needed.

3. *Regular expression*: $r_3 = (1+01)^*00(1+01)^*$
   Explanation: The string is split into three parts:

   - Anything before the unique "00" without forming another "00" $\rightarrow (1+01)^*$
   - The exact "00" in the middle
   - Anything after the "00" without creating another "00" $\rightarrow (1+01)^*$

   This guarantees that only one "00" appears in the entire string.

3 Conclusion

In this lecture, we studied regular expressions and the languages they define. We showed how any regular expression can be converted into an equivalent NFA, illustrating the construction through examples of concatenation, union, and Kleene star operations.

References

[1] Peter Linz, *An Introduction to Formal Languages and Automata*, 6th ed., Jones & Bartlett Learning, Burlington, MA, 2016.