# 📘 Lecture Summary

## Regular Expressions, NFA, and DFA

*(Compiler Design – Lexical Analysis)*

---

## 1️⃣ Regular Language

A **regular language** is a class of languages that can be described **equivalently** by:
**Regular Expressions (RE)**
**Nondeterministic Finite Automata (NFA)**
**Deterministic Finite Automata (DFA)**

### Key Theoretical Facts:

. There exists an algorithm to convert **any Regular Expression (RE)** into an **NFA**.
. There exists an algorithm to convert **any NFA** into a **DFA**.
. There exists an algorithm to convert **any DFA** into a **Regular Expression (RE)**.

### ✅ Conclusion:

All three formalisms have **equivalent expressive power**.

**RE = NFA = DFA**

This equivalence defines the class of **Regular Languages**.

---

## 2️⃣ Deterministic Finite Automata (DFA)

### Definition:

A **Deterministic Finite Automaton (DFA)** is a finite machine in which:
For **each state** and **each input symbol**, there is **exactly one transition** to a next state.

### Why "Deterministic"?

The computation is **unique**.
Each input symbol leads to **one and only one state**.
**No ε (empty string) transitions** are allowed.

### Important Notes:

A pattern can be represented by **many different DFAs**.
The **minimal DFA** (with the smallest number of states) is preferred.

---

## 3️⃣ Formal Definition of a DFA

A DFA is formally defined as a **5-tuple**:

$$(Q, \Sigma, \delta, q_0, F)$$

Where:
**Q** → Finite set of states
**Σ** → Finite input alphabet
**δ** → Transition function

$$\delta : Q \times \Sigma \rightarrow Q$$

**$q_0$** → Initial state ($q_0 \in Q$)
**F** → Set of final (accepting) states, $F \subseteq Q$

---

## 4️⃣ Graphical Representation of DFA

A DFA is represented using a **state diagram** (directed graph):
**Vertices** → States
**Directed edges (arcs)** → Transitions labeled with input symbols
**Initial state** → Indicated by an arrow with no source
**Final states** → Indicated by **double circles**

---

## 5️⃣ How Does a DFA Work?

. The DFA starts at the **initial state $q_0$**.
. It reads the input string **symbol by symbol**.
. Transitions occur according to the transition function δ.
. After the entire input is read:
✅ If the DFA ends in a **final state**, the string is **accepted**.
❌ If it ends in a **non-final state**, the string is **rejected**.

### Language of a DFA:

The **language of a DFA** is the set of **all strings accepted by that DFA**.

---

## 6️⃣ DFA and NFA Relationship

DFA and NFA have **the same expressive power**.
Every **NFA can be converted into an equivalent DFA**.
Both DFA and NFA may have **multiple final states**.
**NFA** is mainly **theoretical**.
**DFA** is used in **lexical analysis** in compilers.

### State Explosion:

If an NFA has **N states**, the equivalent DFA can have up to:

$$2^N \text{ states}$$

---

## 7️⃣ DFA vs NFA (NDFA)

| DFA | NFA |
|---|---|
| One transition per input symbol | Multiple transitions per input symbol |
| No ε-transitions | ε-transitions allowed |
| Deterministic behavior | Nondeterministic behavior |
| Requires more space | Requires less space |
| A string is accepted if it reaches a final state | Accepted if **at least one path** reaches a final state |
| Harder to construct | Easier to construct |
| Practical implementation feasible | Must be converted to DFA for implementation |

---

## 8️⃣ Regular Expressions (RE)

### Definition:

Regular expressions are strings over:

$$\Sigma \cup \{(,), |, *\}$$

They describe **patterns** for regular languages.

### Basic Regular Expressions:

∅ → Regular expression for the **empty set**
**ε** → Regular expression denoting the set $\{\varepsilon\}$
**a** → Regular expression denoting $\{a\}$, for any a ∈ Σ

---

## 9️⃣ Converting Regular Expressions to NFAs

### General Idea:

To convert a Regular Expression to an NFA, we use a **bottom-up construction** that:
Mimics the structure of the regular expression
Produces an NFA with:
**One start state**
**One final state**

### Algorithm 1 (Bottom-Up Construction):

Recursively builds a finite state automaton (FSA)
Suitable for **machines**
Considered **bad for humans** (complex manually)

---

## 🔟 RE to NFA Constructions

### 🔹 Union (P | Q)

If:
P has NFA **Np**
Q has NFA **Nq**
Construction:
Add a new start state
ε-transitions to the start states of Np and Nq
ε-transitions from final states of Np and Nq to a new final state

---

### 🔹 Concatenation (PQ)

Construction:
Connect the final state of Np to the start state of Nq using an ε-transition
Start state = start of Np
Final state = final of Nq

---

### 🔹 Closure (Kleene Star) (Q*)

Construction:
Add a new start state and a new final state
ε-transition from new start to:
old start
new final
ε-transition from old final to:
old start (loop)
new final

This allows:
Zero or more repetitions of Q

---

## 🔚 Final Takeaway (Very Important for Exams)

**RE, NFA, and DFA are equivalent**
They all describe **regular languages**
**NFA** simplifies construction
**DFA** enables implementation
**Regular Expressions** describe patterns
**Lexical analyzers** in compilers rely on DFA

$$\boxed{\textbf{RE = NFA = DFA}}$$