What is a Non-Deterministic Finite Automaton (NFA)?

A **Non-Deterministic Finite Automaton (NFA)** is a type of finite automaton where:

✅ A **single input** can lead to **multiple possible states**.

✅ It allows **empty (ε) transitions**, meaning it can move between states **without any input**.

✅ It is **more flexible** than **Deterministic Finite Automata (DFA)** but requires extra computational steps.

Analogy: Imagine a **maze** where, at each junction, you have **multiple possible paths** instead of just one. Unlike DFA (where each step leads to exactly one next point), NFA allows **multiple transitions for the same input**.

Elements of NFA

An NFA is defined using a **5-tuple (Q, Σ, δ, q_0, F)**:

1️⃣**Q** → Set of finite states.

2️⃣**Σ** → Set of input symbols (Alphabet).

3️⃣**δ** → Transition function (Rules for moving between states).

4️⃣**q_0** → Initial state (Starting point). ($q0 \in Q$) (*NFA have only one initial state*) 5️⃣**F** → Set of final states (Acceptable end points). *(NFA have one final state, but it also possible that it have more than 1 final states)*

- ❖ NFA allow **empty transition**, it means that such transition which read epsilon / null (λ), due to that it is called **epsilon NFA / λ-NFA**.
- ❖ More than one transition from any state along with same input symbol is allowed.

Key Difference from DFA:

- ❖ In DFA, **each input has exactly one next state**.
- ❖ In NFA, **each input may have multiple next states or even ε-transitions**.

Construction of NFA

To construct an NFA:

✅ **Define the states and alphabet** (possible inputs).

✅ **Create transition rules**, including multiple paths for the same input.

✅ **Define the start state (q_0)** where input processing begins.

✅ **Specify final states (F)** where accepted inputs lead.

Example: Consider an NFA that recognizes **words starting with "a" or "b"**.

- **State q_0** → Start state.
- **"a" or "b" leads to state q_1** (valid words start).
- If another "a" or "b" appears, transition remains in q_1.
- Final state **q_1** → Accepts input.

This NFA allows multiple transitions, unlike a strict DFA.

Conversion of Regular Expression (RE) to NFA

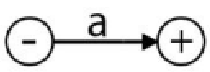
Regular Expressions (RE) can be converted into an **NFA** step by step:

1. **Each symbol** in RE becomes a state in NFA.
2. **Concatenation (ab)** → Connects states sequentially.
3. **Union (a | b)** → Creates multiple paths between states.
4. *Kleene Star (a)** → Allows looping transitions.
5. **ε-transitions** handle empty sequences.

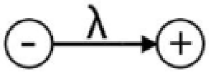
Examples of NFA:

1. Let’s here we take some regular expression and see there NFA.

R.E = a



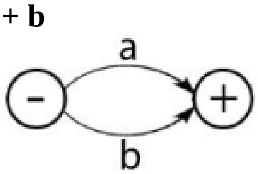
R.E = λ



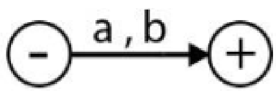
2. Let’s we take another example.

R.E = a + b

NFA =



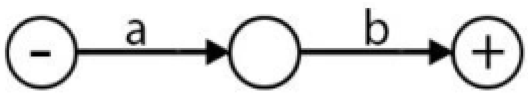
It can also draw like this one



3. Let’s we take another example.

R.E = ab

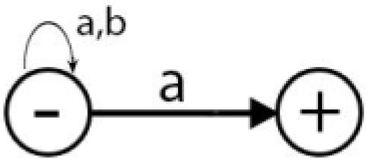
NFA =



4. All strings that end with **a** if ∑ {a , b}

R.E = (a+b)* a

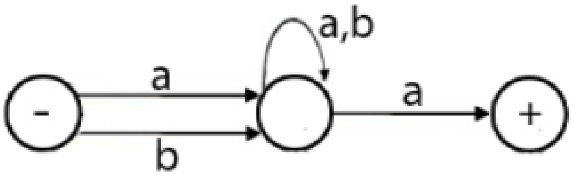
NFA =



5. All strings that end with **a** if ∑ {a , b}

R.E = $(a+b)^+$ a

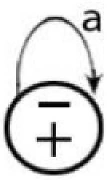
NFA =



6. Draw NFA for all strings, if ∑ {a}.

RE = a*

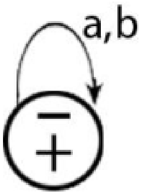
NFA=



7. Draw NFA for all strings, if ∑ {a,b}.

R.E = (a+b)*

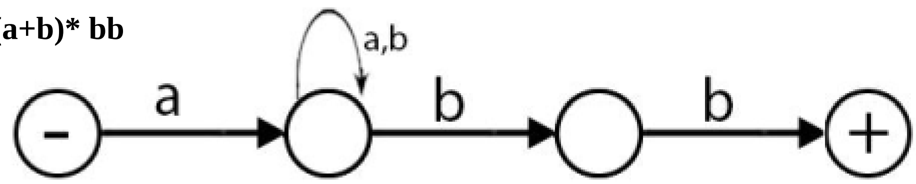
NFA=



8. Draw NFA for all strings that starts with a & end with bb, if ∑ {a,b}.

R.E = a (a+b)* bb

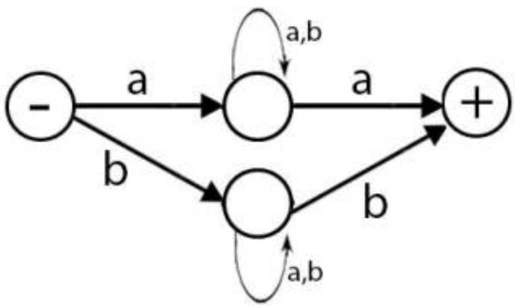
NFA=



9. Draw NFA for all strings that starts & end with same letter, if ∑ {a,b}.

RE = a (a+b)* a + b (a+b)* b

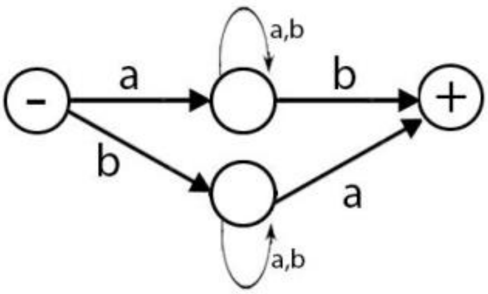
NFA=



10. Draw NFA for all strings that starts & end with different letter, if ∑ {a,b}.

RE = a (a+b)* b + b (a+b)* a

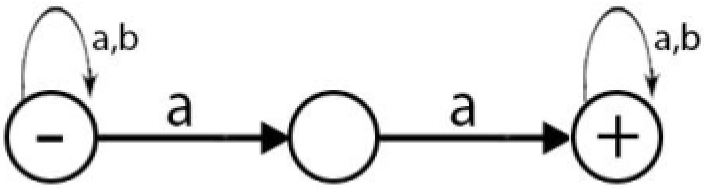
NFA=



11. Draw NFA for all strings that contains double **aa** , if ∑ {a,b}.

RE = (a+b)* aa (a+b)*

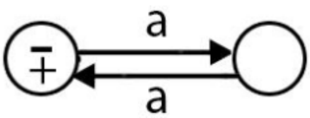
NFA=



12. Draw NFA for all strings of even length, if ∑ {a}.

RE = (aa)*

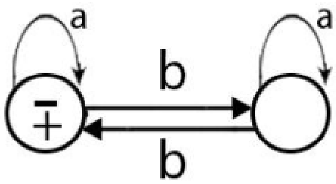
NFA=



13. Draw NFA for all strings of even no. of **b’s**, if ∑ {a,b}.

RE = a* (ba*b)* a*

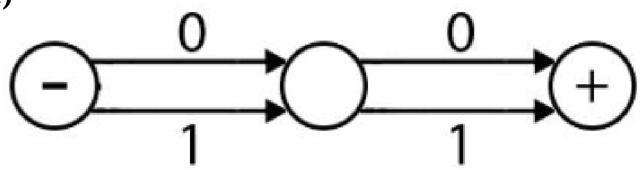
NFA=



14. Draw NFA for all strings of length 2, if ∑ {0 ,1}.

RE = (0+1) (0+1)

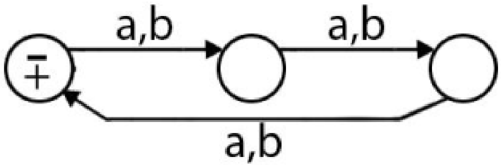
NFA=



15. Draw NFA for all strings of multiple 3 , if ∑ {a ,b}.

RE = ((a+b) (a+b) (a+b))*

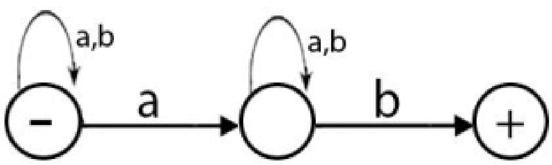
NFA=



16. Draw NFA for all strings that contain at-least one **a** & end with **b**, if ∑ {a ,b}.

RE = (a+b)* a (a+b)* b

NFA=



Transition Table of NFA

A **transition table** is a structured way to represent state transitions in an **automaton**. It shows how an automaton moves from one state to another based on input symbols. It is widely used in **Finite Automata (FA), Pushdown Automata (PDA), and Turing Machines (TM)**.

Structure of a Transition Table

A **transition table** consists of:

- **Rows** for states in the automaton.
- **Columns** for input symbols.
- **Cells** indicating the next state for a given state and input.

Nondeterministic Finite Automaton (NFA)

- An input symbol **may lead to multiple states**.
- Transition table allows **multiple entries** per input.

Current State	Input 0	Input 1	Next States
q_0	$\{q_1, q_2\}$	$\{q_0\}$	Multiple transitions possible
q_1	$\{q_2\}$	$\{q_0, q_1\}$	
q_2	$\{q_1\}$	$\{q_2\}$	

Unlike DFA, **NFA allows multiple possible states** for the same input.

Applications of Transition Tables

- **Language Recognition**: DFA/NFA transition tables validate if a string belongs to a formal language.
- **Compiler Design**: Lexical analyzers use transition tables for token recognition.
- **Game Development**: AI decision-making can use state transitions.
- **Network Protocols**: Transition tables help define communication protocols.

Key Points:

- ❖ Inputs can lead to **multiple states**.
- ❖ **Empty transitions (ε)** allow **jumping** between states without any input.

Summary:

✅ **NFA is more flexible** than DFA.

✅ **It can have multiple transitions per input**, unlike DFA.

✅ **Construction involves defining states, inputs, and transitions.**

✅ **Regular expressions can be converted into NFA using transition rules.**

✅ **Transition tables** help visualize how inputs affect state changes.