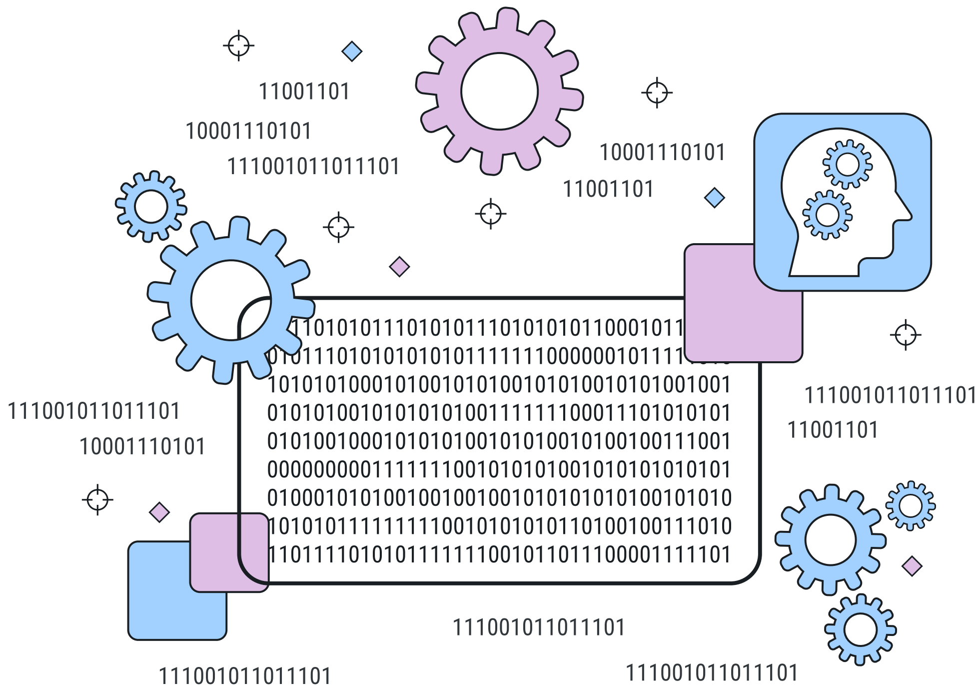



# Theory of Computation

# &

# Compiler Design

# CONTENTS

# Symbol

### Defination

A symbol (also called a character) is the smallest building block in a system, which can be any alphabet, letter, number, or picture used to represent information or construct larger structures like strings or languages.

### Examples of Symbols:

i. Alphabets (a to z): a,b,.....,z

ii. Digits (0 to 9): 0,1,2,....,9

### Why uppercase letters aren't symbol?

# Alphabets

### Defination

An alphabet (also called an input in some contexts) is a finite set of distinct symbols, typically denoted by Σ (Greek letter sigma), used to construct strings and define languages. These symbols can be letters, digits, or special characters, and they serve as the basic building blocks for representing information in computational systems. The symbols in the alphabet are elements of the set.

### Examples of Alphabets:

1. Σ={a,b,c}
   This is an alphabet containing three symbols: a,b and c.

2. Σ={0,1}
   This is an alphabet containing three symbols: 0 and 1.

3. Σ={a,b,c,0,1}
   This is an alphabet containing three symbols: a,b,c,0 and 1.

4. Σ={a,a,b,c}
   An alphabet must contain distinct symbols. In this case, adding a twice is redundant, and the set should be reduced to:

   Σ={a,b,c}

   This is an alphabet containing three symbols: a,b and c.

# String

### Defination

A string is a finite sequence of symbols (or characters) taken from an alphabet Σ. It can include any combination of symbols from the alphabet, and its length can vary from zero (an empty string) to any finite number of symbols.

### Notation:

1. A string is typically denoted by a sequence of characters inside quotation marks, like w = "abc"
2. If Σ={a,b,c}, then w="abc" is a string made from those symbols.

### Key Points:

1. Finite Sequence: A string is a sequence of symbols, and the length of the string is finite.
2. From an Alphabet: The symbols in a string come from a predefined alphabet Σ.
3. Empty String: The empty string, denoted by ε, is a string that contains no symbols. It has a length of zero.
4. Length of a String: The length of a string w, denoted as |w|, is the number of symbols it contains. For example, |"abc"|=3.

# Language

### Defination
A language is a set of strings formed from an alphabet that satisfies specific rules or properties.

Languages are two types: 1. Finite language 2. Infinite Language

A language is often denoted by L

### Examples:

$\Sigma$ = {a, b}
L1 = {W is a set of string where the length of each string must be 2}
= {aa, ab, ba, bb}
L2 = {W is a set of string where the length of each string must be 3}
= {aaa, aab, bab, bba, abb, aba, bbb, baa}
L3 = {W is a set of string where each string starts with 'a'}
= {a, ab, aa, aaa,,,,,,,,,,,,,,,,,,,}

Here, L1 and L2 is called the finite language and L3 is infinite language.

# Power of Sigma

Suppose, $\Sigma$ = {a, b}

$\Sigma^1$= Set of all string over this $\Sigma$ of the length is '1'
= {a, b}

$\Sigma^2$= Set of all string over this $\Sigma$ of the length is '2'
= {aa, ab, ba, bb}

Note: $\Sigma^0$= empty string where the length of string is 0

$\Sigma^* = \Sigma^0 \cup \Sigma^1 \cup \Sigma^2 \cup \ldots \ldots \ldots$
$\Sigma^*$ is called mother of all string.

# Finite Automata Model

## Defination

A Finite Automata Model (FA Model) is a mathematical model used to recognize patterns in input, such as strings.

## Types

1. Deterministic Finite Automaton (DFA)
2. Nondeterministic Finite Automaton (NFA)

## Why FA Model is used?

To give a finite representation of an infinite language

### Example:

L(M) = {W is a set of string where each string starts with 'a'} and Σ={a,b}
W = {a, ab, aa, aaa,,,,,,,,,,,,,,,,,,}

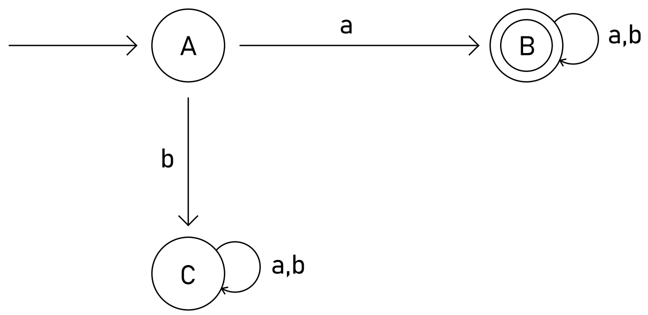



Figure: State Transition Diagram

This diagram accepts only those strings that start with 'a' and rejects those that start with 'b'. It is impossible to allocate infinite memory space for a language set. Therefore, a finite memory slot is used. For this reason, Finite Automata is implemented, as it requires very little memory and can efficiently determine whether a string is accepted or not.

# States Finite Automata Model

States mean memory, which is denoted by a circle (◯) = {A, B, C}

## 1. Initial State

i. Start scanning or traveling.

ii. A DFA (Deterministic Finite Automaton) and an NFA (Nondeterministic Finite Automaton) both contain one initial state.

iii. This arrow signifies the starting point of the automaton. Additionally, there is no label (value) on the edge leading to the initial state

## 2. Final State

i. Stop scanning or traveling.

ii. Denoted by double circle (◎)

iii. DFA (Deterministic Finite Automaton) and NFA (Nondeterministic Finite Automaton), there can be one or more final states.

### 3. Trap State (or Dead State)

i. A trap state (or dead state) always has a self-loop for all input symbols.
ii. We cannot go to the final state from a trap state
iii. It is not mandatory to use a trap state in a DFA or NFA
iii. When a rejected string is processed, the automaton eventually reaches the trap state (also called the dead state).

### 4. Normal State

i. A state that handles input symbols and is not a final or trap state.

# DFA Model

A DFA is a type of Finite Automaton (FA) that consists of five tuples:

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

### Example:

Let $\Sigma = \{a, b\}$

Define the language L(M) as the set of all strings that start with the letter 'a'.
Now, consider the state transition diagram for L(M):

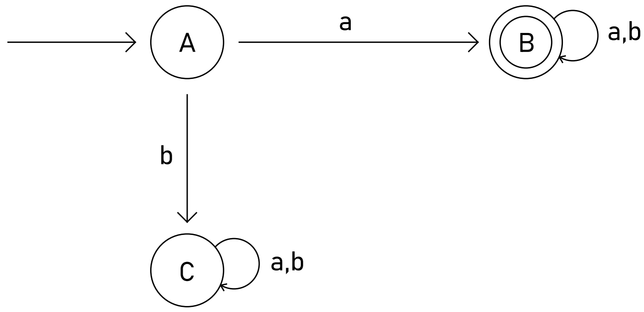



Fig: DFA Diagram

### State Transition Diagram for L(M)

The five components of the DFA are described as follows:

**Q** - The set of all states: $Q = \{A, B, C\}$

**Σ** - The set of input alphabets: $\Sigma = \{a, b\}$

**$q_0$** - The start (initial) state: $q_0 = A$

**F** - The set of final states: $F = \{B\}$

**δ** - The transition function

The transition function δ is defined as:

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

## The Transition Function (δ) in a DFA can be represented in two ways:

1. Transition Table (Tabular Form)
2. Using Equation

### 1. Transition Table (Tabular Form)

This method uses a table where rows represent states and columns represent input symbols. Each cell shows the next state for a given state-input pair.

| State ↓ \ Input → | a | b |
|---|---|---|
| A | B | C |
| B | B | B |
| C | C | C |

Fig: DFA Transition Table

### 2. Using Equation

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

$$A \times a \rightarrow B$$
$$A \times b \rightarrow C$$
$$B \times a \rightarrow B$$
$$B \times b \rightarrow B$$
$$C \times a \rightarrow C$$
$$C \times b \rightarrow C$$

### Characteristics of DFA:

1. For each input symbol, there is exactly one transition to a state.
2. A DFA has only one initial state.
3. A DFA can have more than one final state.

# How To Construct A DFA

## Problem 1

Construct a DFA which accepts the set of all strings over the Σ={a,b} where each string starts with an 'a'.

### Solution

Alphabet: Σ={a,b}

Language: L(M)={a,ac,ab,aaa,aab,aba,abb,...}

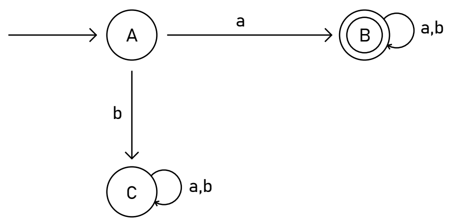



Fig: DFA Diagram

## Problem 2

L(M)={w | w starts with 'aa' }

### Solution

Alphabet: Σ={a,b}

Language: L(M)={aa,aaa,aab,aaab,aabb,...}

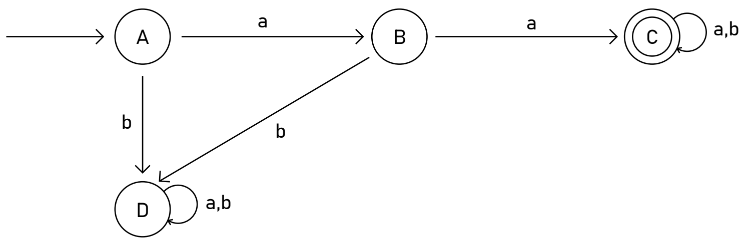



Fig: DFA Diagram

## Problem 3

Construct a DFA which accepts the set of all strings over Σ={0,1} where each string starts with 10

### Solution

Alphabet: Σ={0,1}

Language: L(M)={10,100,101,1001,1011,10001,...}

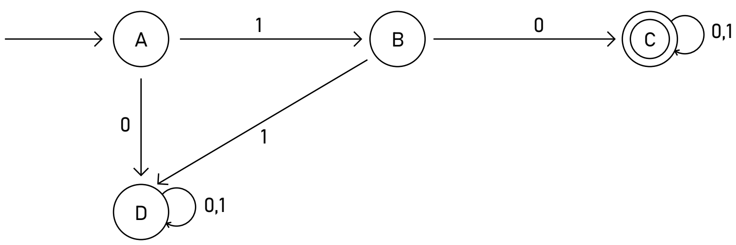



Fig: DFA Diagram

# Minimization of DFA

## Example 01

Given the language:
L = {w | w has an odd number of a's and ends with 'a' or 'b'}

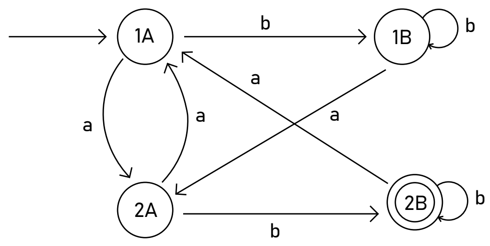



Fig: DFA Diagram

## Steps for Minimizing a DFA

**1. Remove Unreachable States:**
Identify and eliminate any states that cannot be reached from the initial state.

**2. Construct the State Transition Table:**
Create a table representing state transitions for each input symbol.

**3. Mark Initial and Final States:**
Clearly indicate which states are initial and final.

**4. Find Equivalent States:**
Identify and merge states that belong to the same equivalence class.

### Now, proceed with minimizing the DFA of Example 01 using these steps.

**Step 1: Remove unreachable states**

Remove those states which are unreachable from the initial state. No states are removed here because states 1B, 2A, and 2B are reachable from the initial state.

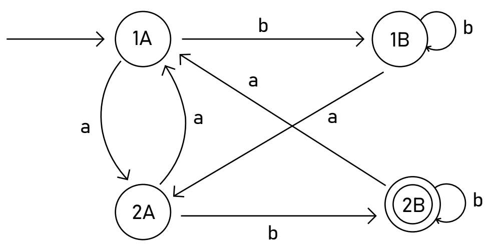



Fig: DFA Diagram

**2. Construct the State Transition Table:**

| State ↓ / Input → | a | b |
|---|---|---|
| 1A | 2A | 1B |
| 1B | 2A | 1B |
| 2A | 1A | 2B |
| 2B | 1A | 2B |

Fig: DFA Transition Table

**3. Mark Initial and Final States:**
Clearly indicate which states are initial and final.

| State ↓ \ Input → | a | b |
|---|---|---|
| → 1A | 2A | 1B |
| 1B | 2A | 1B |
| 2A | 1A | 2B |
| * 2B | 1A | 2B |

**4. Find Equivalent States:**
Identify and merge states that belong to the same equivalence class.

0-equivalent = [Set of non-final states] [Set of final state]

= [ 1A 1B 2A ] [ 2B ]

1-equivalent = [ 1A 1B ] [ 2A ] [ 2B ]

2-equivalent = [ 1A 1B ] [ 2A ] [ 2B ]

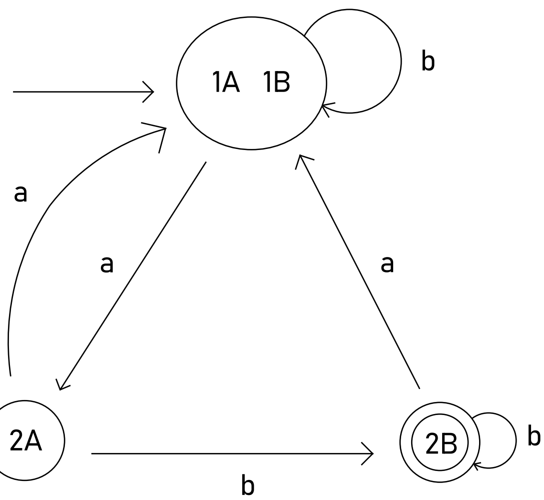



Fig: Minimized DFA Diagram

# NFA Figure Construction

## Rules for Each Input

**1. No Transition:**

**1.1** Imagine you're walking on a path, but suddenly, the road ends. You have no way to move forward.

**1.2** In an NFA, if there is no transition for a given input, the machine cannot continue from that state.

**2. Single Transition:**

**2.1** Imagine you're at a door, and there is only one way to go forward.

**2.2** In an NFA (or DFA), if a state has exactly one transition for a symbol, it moves to only one specific next state.

**3. Multiple Transitions:**

**3.1** Imagine you're at a junction where you can go in two or more different directions at the same time.

**3.2** In an NFA, a state can have multiple paths for the same input symbol, meaning the machine can follow multiple options at once.

## Example 01

L(M) = { w | w starts with 'a' }

Σ = {a, b}

W = { a, ab, aba, abb, ... }


Fig: NFA Diagram

## Example 02

L(M) = { w | w is a string that ends with 'a' }

Σ = {a, b}

W = { a, ba, aa, aba, bba, abba, ... }

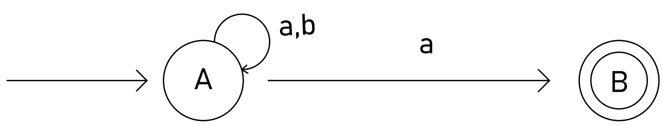



Fig: NFA Diagram

# NFA to DFA Construction

## Language

L(M) = { w | w ends with "ab" } where Σ = {a, b}

## Solution

### 1. Construct the NFA

Create an NFA for the given language. The transition diagram consists of states A, B, and C, with the following transitions:

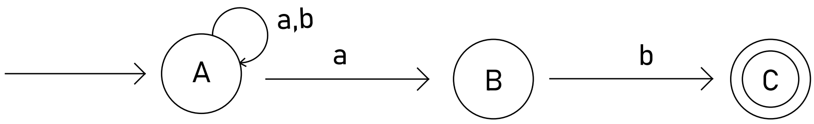



Fig: NFA Diagram

### 2. State Transition Table for NFA

| Input → / State ↓ | a | b |
|---|---|---|
| A | {A, B} | {A} |
| B | { } | {C} |
| C | { } | { } |

Fig: NFA Transition Table

### 3. Construct the DFA Transition Table

| Input → / State ↓ | a | b |
|---|---|---|
| A | [A B] | [A] |
| B | [A B] | [A C] |
| C | [A B] | [A] |

Fig: DFA Transition Table

**5. Construct the DFA Figure**

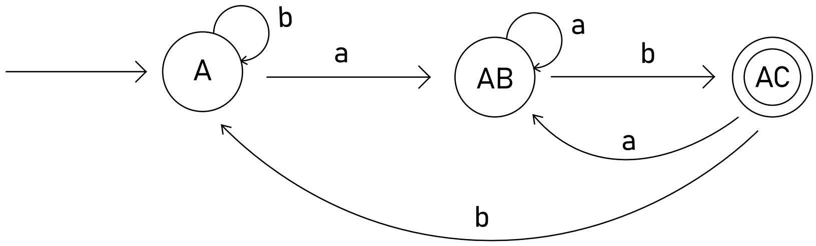



Fig: DFA Diagram

# Regular Expression

A regular expression is a method or representation technique used to match patterns on strings. It is widely used in compiler design, is easy to understand, and has efficient implementation.

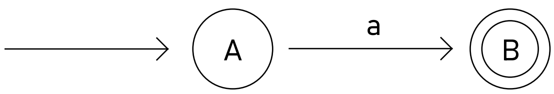
## Formal Definition (6 Cases):

**Case 1**

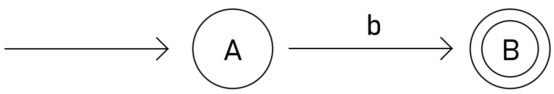
R = a or R = b where a, b ∈ Σ

A language that accepts only one single character: either 'a' or 'b'.

If R = a, then the language is: L = { "a" }




If R = b, then the language is: L = { "b" }

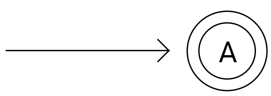



**Case 2**

R = ε

The empty string ε is a string with length 0 — it contains no characters.

The language is: L = { ε }
That means the NFA should accept the empty string, and only the empty string.




**Case 3**

R = ∅

∅ represents the empty set — a language that contains no strings at all, not even ε. So the NFA should reject everything.

**Case 4**

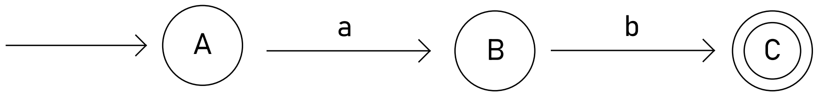
Union ($R = R_1 \cup R_2$ or $R_1 \mid R_2$ or $R_1 + R_2$)

Let's take:

$R_1$ = ab

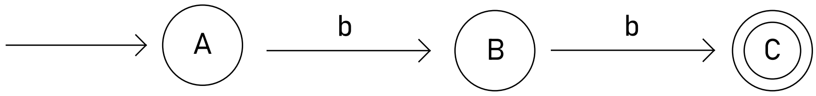
$R_2$ = bb

Build NFA for $R_1$ = ab



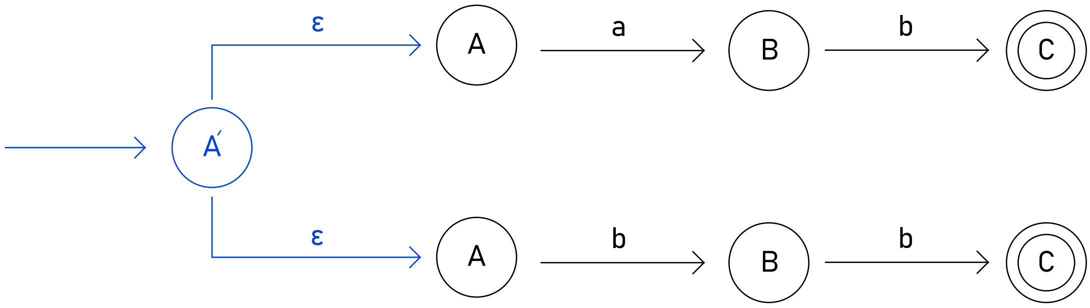

N1

Build NFA for $R_2$ = bb




N2

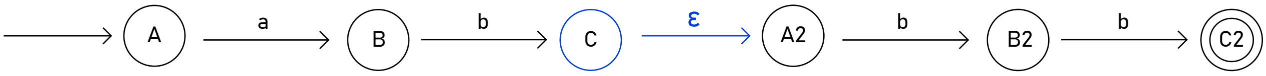
Now we have to union N1 and N2




When we perform a union, we create an extra state and connect it to the initial states of both NFAs using ε.

**Case 5**

$R = R_1 \cdot R_2$




Why ε-transition?

Because after finishing $R_1$, the machine jumps to $R_2$ without reading any input — and continues reading.

When we concatenate two NFA models, the second NFA remains unchanged. We make the final state of the first NFA non-final and connect them using an ε.

**Case 6: Kleene Star**

$R = R_1^*$ This means: 0 or more repetitions of $R_1$

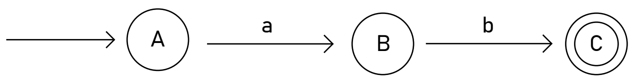
Let's say

$R_1$ = ab, so R = (ab)*

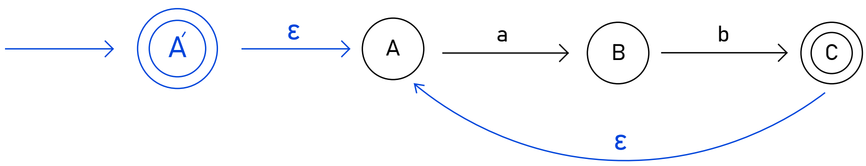
That means:

Accept: "", "ab", "abab", "ababab", etc.

Build NFA for $R_1$ = ab




Build NFA for $R_1$ = (ab)*




When we apply the Kleene Star, we create an extra final state, but the previous final state remains unchanged. We connect the new final state to the NFA using ε-transitions. Additionally, we draw an ε-transition from the old final state back to the initial state.

# Regular Expression To NFA

## Example 01: (ab U a)*

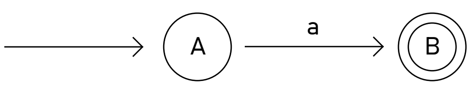
a* = 0 or more

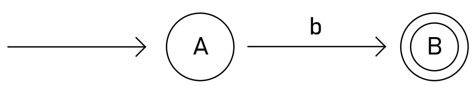
= {ε, a, aa, aaa, ...}

a+ = 1 or more

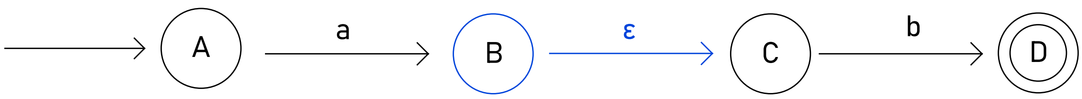
= {a, aa, aaa, ...}

Plot NFA for each symbol. Here, symbol is a and b.

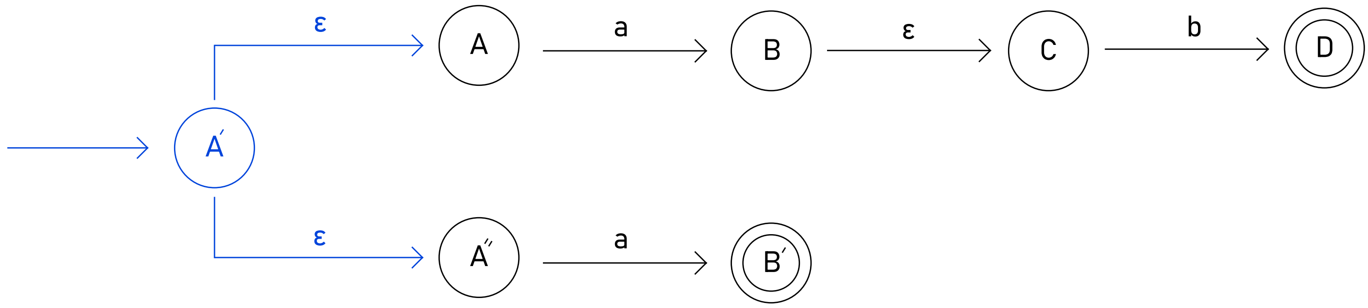


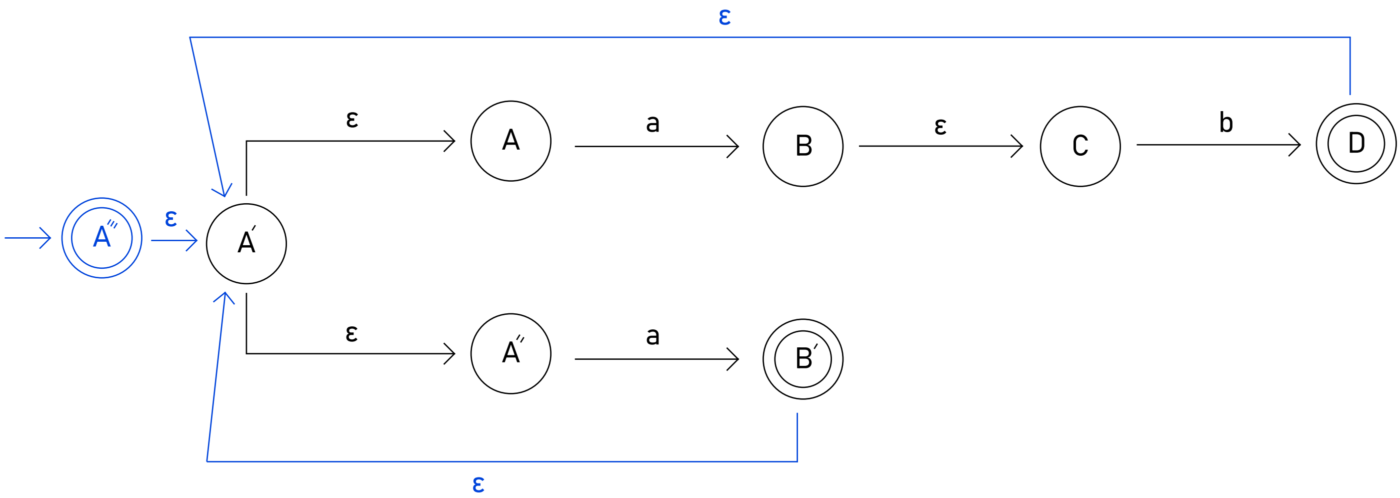

here a and b are concatenated, so **Case 5** is used here



(ab U a)

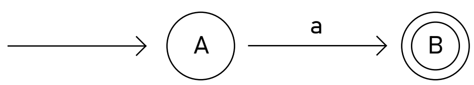


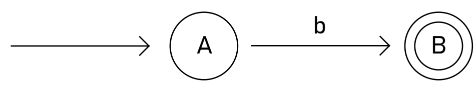

(ab U a)*



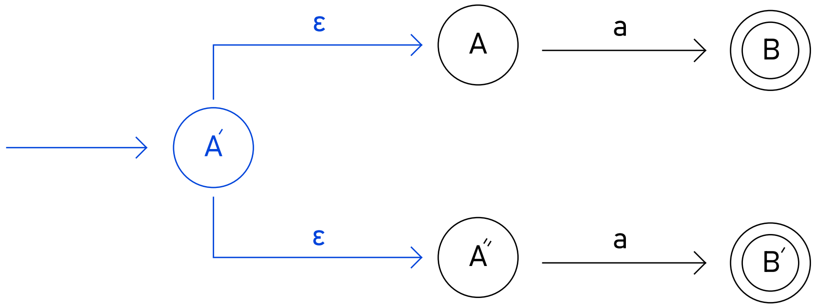

## Example 02: (a U b )* aba

plot NFA for each symbol. Here, symbol is a and b.

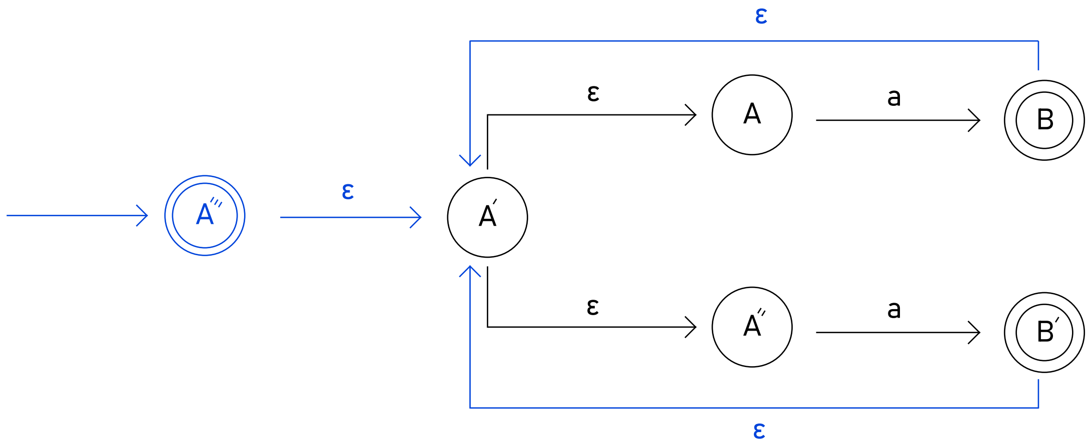


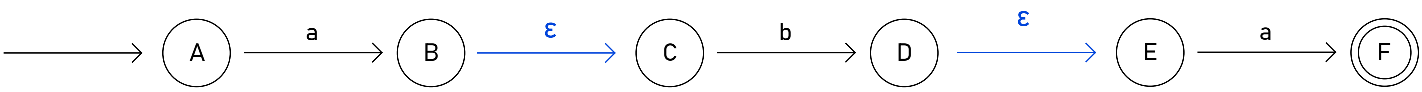

(a U b )



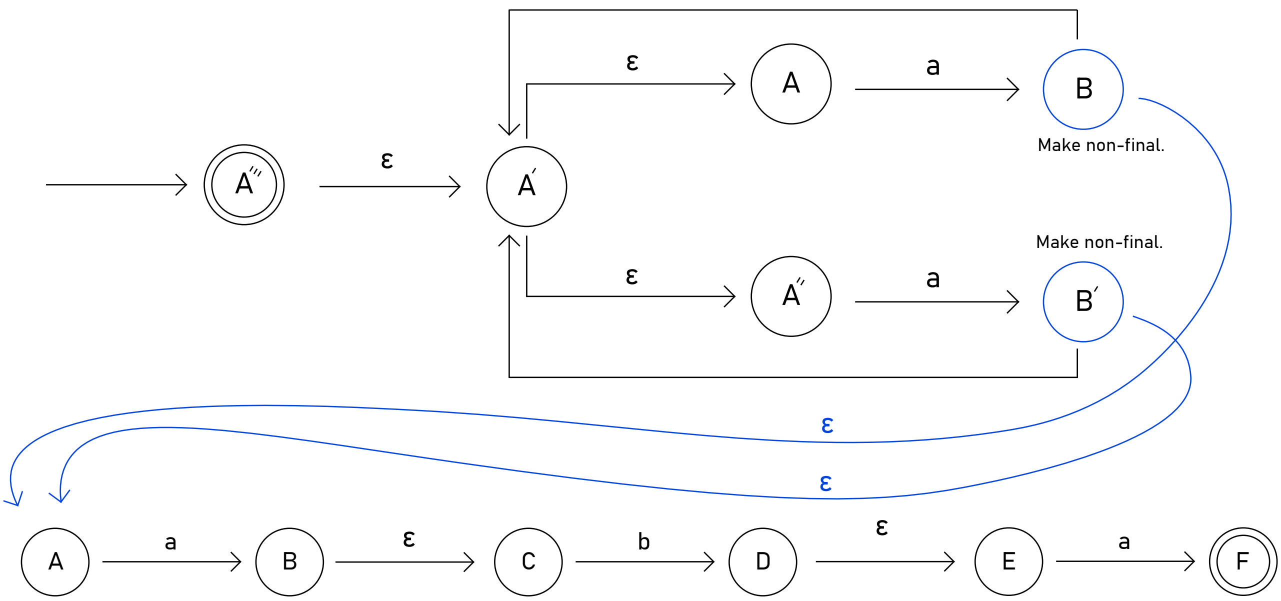
**(a U b )***




**aba**

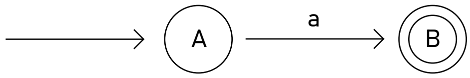



**(a U b )* aba**

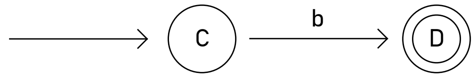


## Example 03: ba* + ε

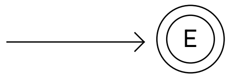
For a



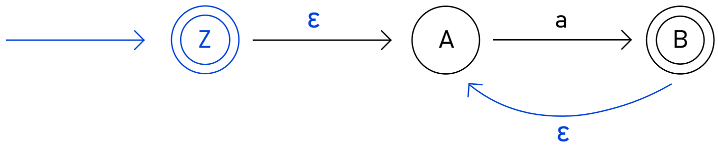

For b



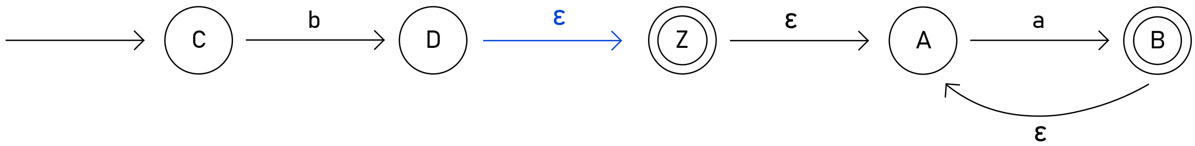

For ε



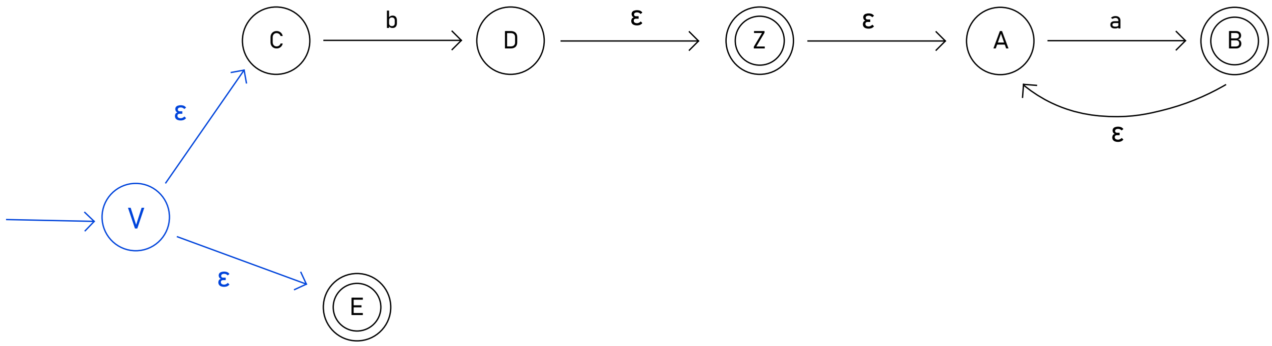

**a***




**ba***



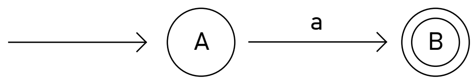

**ba* + ε**



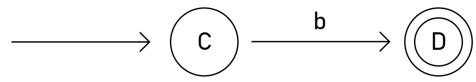
## Example 04: a⁺U (ab)⁺

**aa* U ab(ab)***

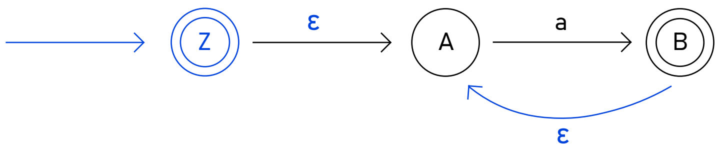
For a



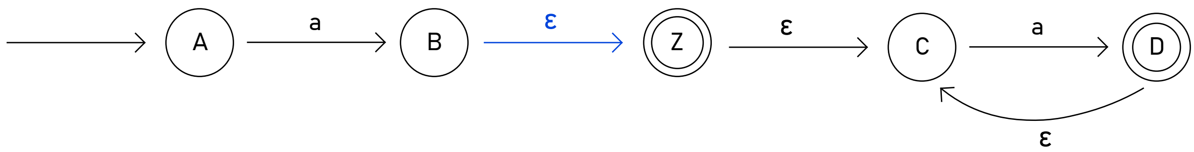

For b



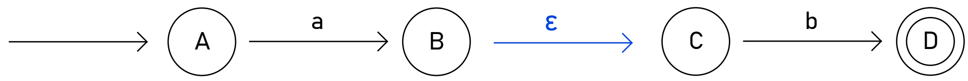

**a***



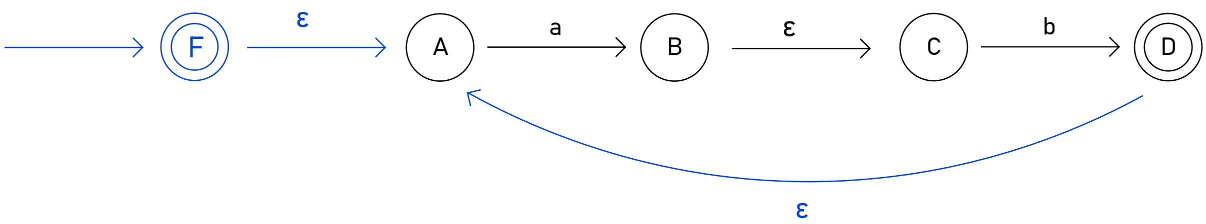

**aa***



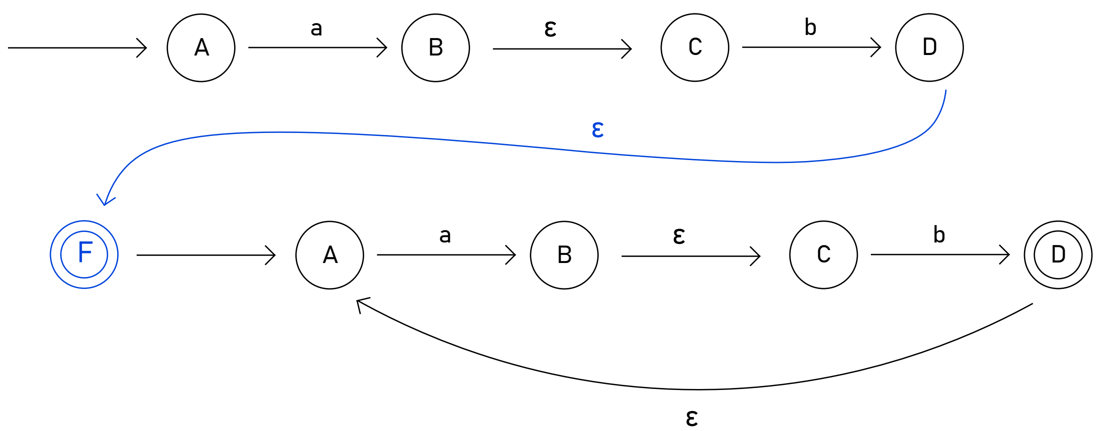

**ab**



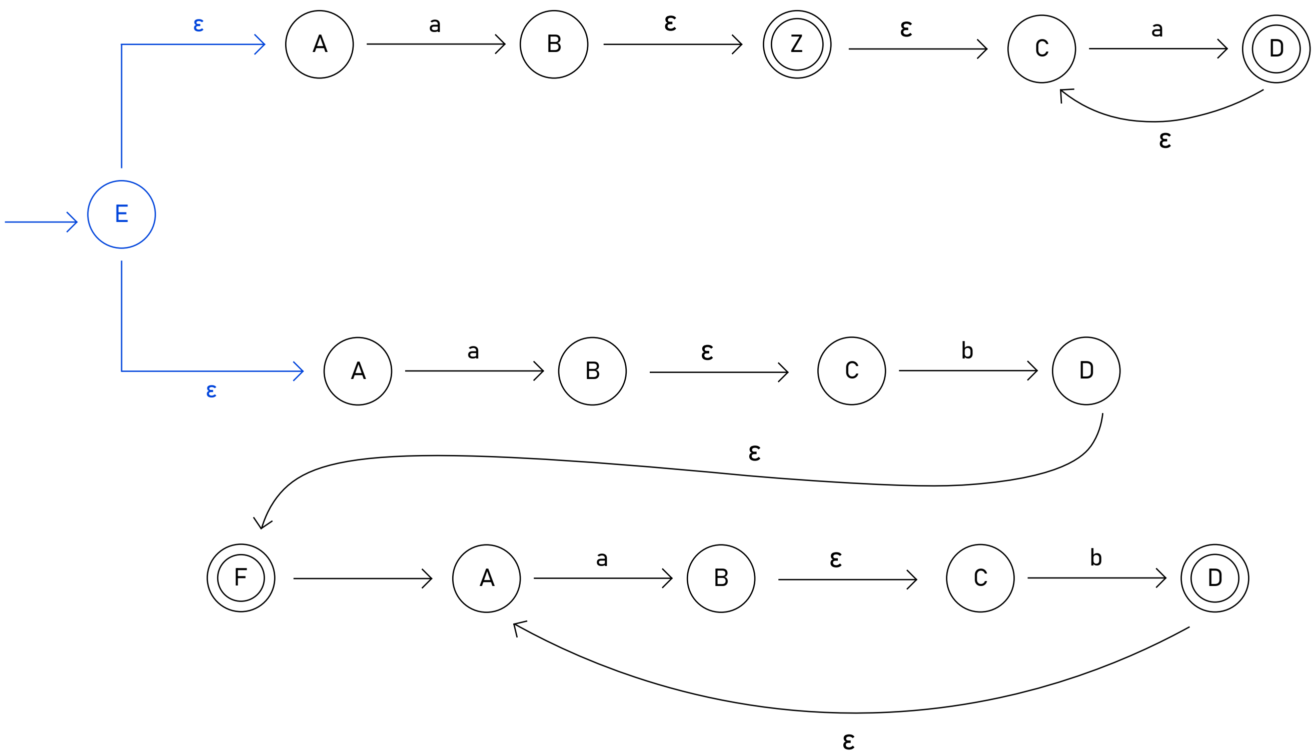

**(ab)***




**ab(ab)***



**aa* U ab(ab)***



# Context-Free Grammar (CFG)

Context Free Grammar is a set of rules/production which is used to generate patterns of string.

### CFG has 4-tuples. These are V, T, P, S

| | Description |
|---|---|
| V | Variable / Non-Terminal |
| T | Set of Terminals |
| P | Production / rules |
| S | Start-Variable |

### Now, from a grammar, we introduce with four-tuples.

**Suppose the grammar is, $S \rightarrow 01 \mid 0S1$**

This grammar is for the language, L(M) = { w | w is a set of string of $0^n 1^n$ where $n \geq 1$ }

$$0^1 \ 1^1 = 01$$

$$0^2 \ 1^2 = 0011$$

The grammar is defined with the production rule: $S \rightarrow 01 \mid 0S1$
The 4-tuples are determined as

V = {S} (set of variables/non-terminals)

T = {0, 1} (set of terminals)

S = S (start variable)

P = {$S \rightarrow 01 \mid 0S1$} (set of production rules)

Note Under Construction