

Mohsin Iban Hossain

AIUB, Compiler Final-Term Notes

COMPILER DESIGN

Table of Contents

REGULAR
EXPRESSION

Regular Expression

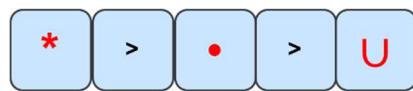
» Regular expressions are used to represent certain sets of strings in an **algebraic** fashion, moving beyond simple English representations of strings.

- **Structure:**

Built using alphabets and **regular operations**:

- ***** (star/**Closure**) – Represents zero or more occurrences of a string.
- **.** (Concatenation) – joining two patterns
- **U** (Union) – either/or (can also be written as )

- **Precedence Order:**



» Examples:

1. **(0U1)0***

- Means: Starts with 0 or 1, followed by any number of 0s
- Language: {w | w starts with 0 or 1, followed by 0s}

2. **(0U1)***

- Means: Any combination of 0s and 1s
- Language: {all binary strings}

3. **(0U1)1**

- Means: Either 0 or 1, followed by 1
- Language: {w | w ends with 1, and starts with 0 or 1}

Designing Regular Expressions

 **Example1:** Write the Regular Expression of the following languages, where $\Sigma = \{a,b\}$, “Language Accepting Strings of Length Exactly Two”

Step 1: Understand the Alphabet	Step 2: Understand the Language Requirement
<p>We are working with:</p> $\Sigma = \{a, b\}$ <p>This means only the characters a and b can appear in our strings.</p>	<p>» The language accepts only strings of length exactly two.</p> <p>This means strings like:</p> <ul style="list-style-type: none"> • aa • ab • ba • bb <p>Only these four strings are valid. Any string with length $\neq 2$ is not allowed.</p>
Step 3: List All Valid Strings	Step 4: Write the Regular Expression
<p>» As we saw above, valid strings of length 2 over $\{a, b\}$ are:</p> <ul style="list-style-type: none"> • aa • ab • ba • bb 	<p>$\therefore R = (aa \cup ab \cup ba \cup bb)$ Or $\therefore R = (aa ab ba bb)$</p>

 **Example2:** Write the Regular Expression of the following languages, where $\Sigma = \{a,b\}$, “which accepts all strings of length at least two.”

Step 1: Understand the Alphabet	Step 2: Understand the Language Requirement
<p>We are working with:</p> $\Sigma = \{a, b\}$ <p>This means only the characters a and b can appear in our strings.</p>	<p>» The language accepts all strings of length at least two</p> <p>This means:</p> <ul style="list-style-type: none"> • The string must be two or more characters long. • This includes all strings of length 2, 3, 4, and so on.
Step 3: List All Valid Strings	Step 4: Write the Regular Expression
<p>» For strings of length at least two:</p> <ul style="list-style-type: none"> • Valid examples: aa, ab, ba, bb, aaa, aab, abb, baa, bab, bba, bbb, etc. • Invalid examples (strings of length less than two): a, b 	$\therefore R = (a b)(a b)(a b)^*$

Explanation:

- $(a \mid b)$: The first character can be either a or b .
- $(a \mid b)$: The second character can also be either a or b .
- $(a \mid b)^*$: After the second character, any number (including zero) of a or b characters can follow. This makes the string longer than two characters if needed.

 **Example3:** Write the Regular Expression of the following languages, where $\Sigma = \{a,b\}$, “which accepts **Strings of Length At Most Two**.”

Step 1: Understand the Alphabet	Step 2: Understand the Language Requirement
<p>We are working with:</p> $\Sigma = \{a, b\}$ <p>This means only the characters a and b can appear in our strings.</p>	<p>» The language accepts all strings of length at Most two.</p> <p>This means:</p> <ul style="list-style-type: none"> • Length 0 (the empty string, denoted by ϵ) • Length 1 (like a, b) • Length 2 (like aa, ab, ba, bb)
Step 3: List All Valid Strings	Step 4: Write the Regular Expression
<p>» For strings of length at Most two:</p> <ul style="list-style-type: none"> • ϵ • a • b • aa • ab • ba • bb 	$\therefore R = \epsilon \mid a \mid b \mid (a \mid b) (a \mid b)$

Practice Problem

 **Problem1:** Write the Regular Expression of the following languages, where $\Sigma = \{a,b\}$,
“Language Accepting Strings of Length Exactly Two”

→ Here,

» $\Sigma = \{a,b\}$,

» $L(M) = \{aa, ab, ba, bb\}$

$$\therefore R = (a \cup b)(a \cup b)$$

Or

$$\therefore R = (a | b)(a | b)$$

 **Problem2:** Write the Regular Expression of the following languages, where $\Sigma = \{a,b\}$,
“which accepts all strings of length at least two.”

→ Here,

» $\Sigma = \{a,b\}$,

» $L(M) = \{aa, ab, ba, bb, aaa, aab, abab..., bbaa..., \}$

$$\therefore R = (a | b)(a | b)(a | b)^*$$

 **Problem3:** Write the Regular Expression of the following languages, where $\Sigma = \{a,b\}$,
“which accepts all strings of length at Most two.”

→ Here,

» $\Sigma = \{a,b\}$,

» $L(M) = \{\epsilon, a, b, aa, bb, ab, ba\}$

$$\therefore R = \epsilon | a | b | (a | b)(a | b)$$

 **Problem4:** Write the Regular Expression of the following languages, where $\Sigma = \{a,b\}$,
 “ $L=\{w| \text{ each } a \text{ in } w \text{ is followed by at least two } b\}$ ”

 **Here,**

» $\Sigma = \{a,b\}$,

» $L(M) = \{abb, b...abb, abbb..., abbabb..., abbbabb...\}$

$$\therefore R = b^* abb b^* (abb b^*)^*$$

Let's check whether the string “abbabb” matches the regular expression for the language:

1. **Breakdown of abbabb:**

- o The string can be grouped as: a b b a b b
 (Each a is followed by exactly two bs, which satisfies the condition.)

2. **Matching with $b^* a b b b^* (a b b b^*)^*$:**

- o b^* → Matches 0 leading bs (empty).
- o $a b b b^*$ → Matches a b b (the first b^* matches 0 trailing bs here).
- o $(a b b b^*)^*$ → Matches the remaining a b b
 (again, b^* matches 0 trailing bs).
- o **Result:** The entire string abbabb is matched.

 **Problem5:** Write the Regular Expression of the following languages, where $\Sigma = \{a,b\}$,
 “ $L=\{w| w \text{ contains odd number of } b\}$ ”

 **Here,**

» $\Sigma = \{a,b\}$,

» $L(M) = \{b, a...b, ba..., bababa..., ba...ba...ba...\}$

$$\therefore R = a^*b (a^*b a^*b a^*)^*$$

 **Problem5:** Write the Regular Expression of the following languages, where $\Sigma = \{a,b\}$,
“ $L=\{w \mid w \text{ contains } abbab \text{ substring}\}$ ”

 **Here,**

» $\Sigma = \{a,b\}$,

» $L(M) = \{abbab, a..abbab, b..abbab, abbaba..., abbabb...\}$

$$\therefore R = (a \cup b)^* abbab (a \cup b)^*$$

 **Problem6:** Write the Regular Expression of the following languages, where $\Sigma = \{m,n\}$

- i. “ $\{w \mid \text{each } m \text{ in } w \text{ is followed by at least three } n\}$ ”
- ii. “ $\{w \mid w \text{ has odd length and starts with } m\}$ ”
- iii. “ $\{w \mid w \text{ contains 'mnnmmn' substring}\}$ ”

(i)

 **Here,**

» $\Sigma = \{m, n\}$,

» $L(M) = \{mnnn, n...mnnn, mnnnn..., mnmmnn... \}$

$$\therefore R = n^* mnnn n^* (mnnn n^*)^*$$

(ii)

 **Here,**

» $\Sigma = \{m, n\}$,

» $L(M) = \{m, mmn, mnn, mnnnm, ... \}$

$$\therefore R = m((m \cup n)(m \cup n))^*$$

(iii)

→ Here,

$$\gg \Sigma = \{m, n\},$$

$$\gg L(M) = \{ mnnmmmn, mnm...mnnmmmn, mnnmmmn...mnnnm, \dots \}$$

$$\therefore R = (m|n)^* mnnmmn (m|n)^*$$

Problem7: Give the **Description** of the following Regular Expressions.

Consider $\Sigma = \{x, y\}$

- i. “ $(x \cup y)^* xyyxxy$ ”
- ii. “ $x^* y x^* y x^* y y^*$ ”
- iii. “ $(x \cup y). ((x \cup y). (x \cup y))^*$ ”

(i)

→ $(x \cup y)^* xyyxxy$:

- o Language: $\{w \mid w \text{ is a string over } \Sigma = \{x, y\} \text{ that ends with "xyyxx"}\}$.

(ii)

→ $x^* y x^* y x^* y y^*$:

- o Language: $\{w \mid w \text{ is a string over } \Sigma = \{x, y\} \text{ with at least three 'y's, and the third 'y' is followed by zero or more 'y's}\}$.

(iii)

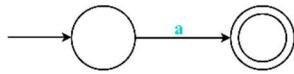
→ $(x \cup y). ((x \cup y). (x \cup y))^*$

- o Language: $\{w \mid w \text{ string starts with } x \text{ or } y, \text{ string has odd number of length}\}$

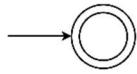
CONVERSION OF
RE TO NFA

Formal procedure of Regular Language

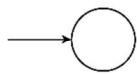
1. $R = a$; $a \in \Sigma$



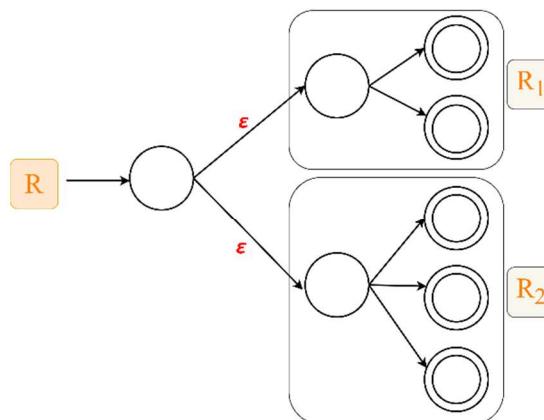
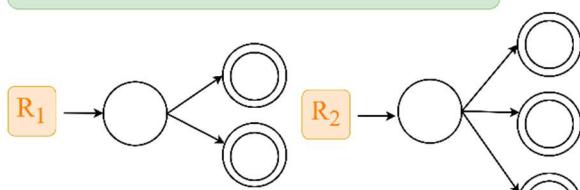
2. $R = \epsilon$; initial state is the final state



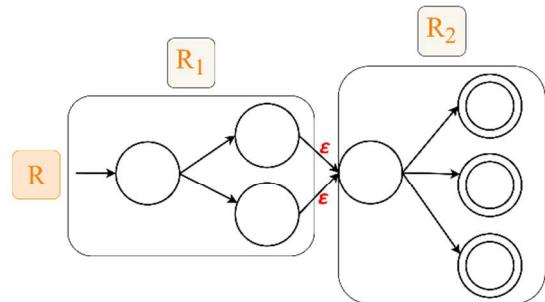
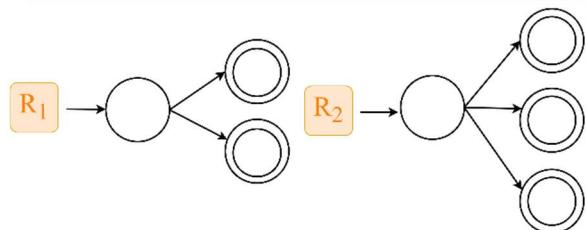
3. $R = \emptyset$



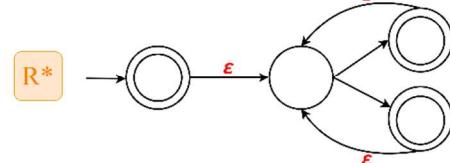
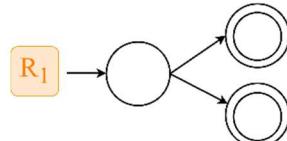
4. $R = R_1 \cup R_2$ or $R = R_1|R_2$ or $R = R_1$ or $R = R_2$



5. $R = R_1 \circ R_2$ or $R = R_1 R_2$ or $R = R_1$ Followed by R_2

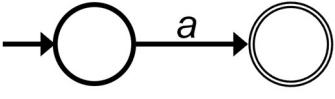
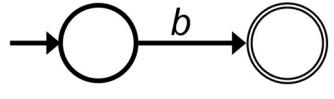
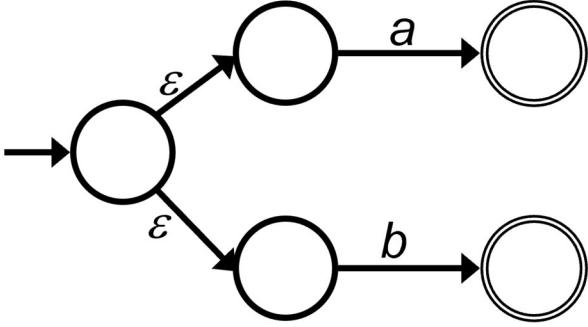
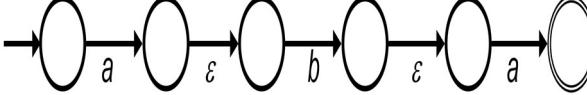
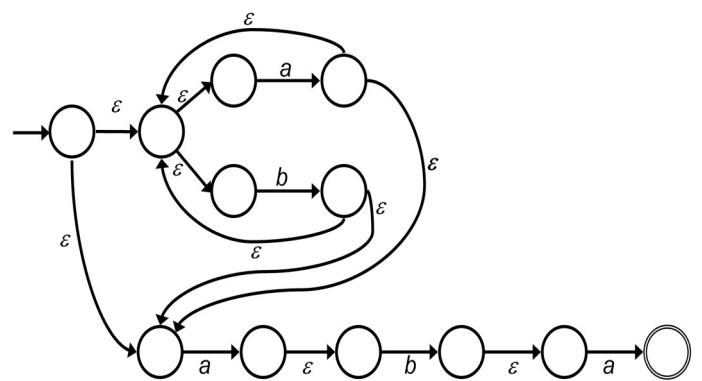


6. $R^* = R_1^*$

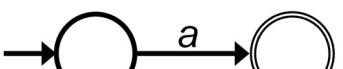
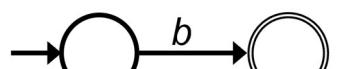
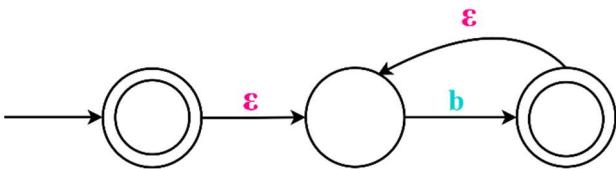
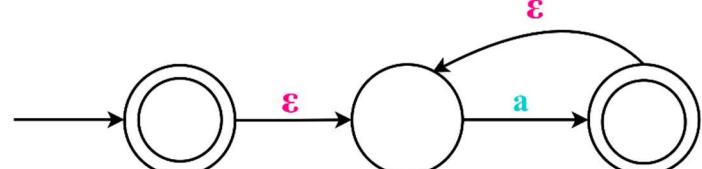
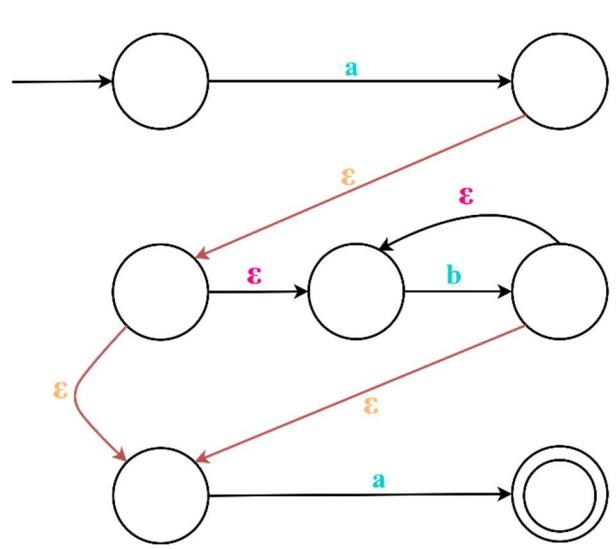
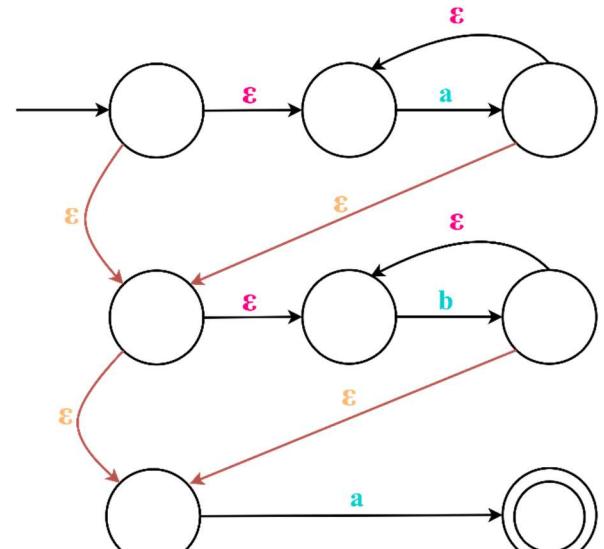


Practice Problem

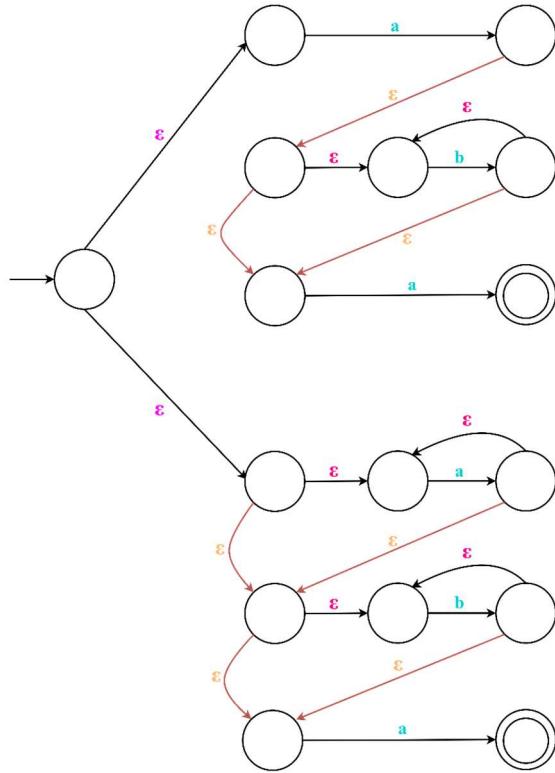
 **Problem1:** Convert the following Regular Expression into equivalent NFA using formal procedure. “ $(a \cup b)^* aba$ ”.

Step 1: Parse the RE into Subexpressions	Step 2: Build NFAs for Basic Symbols	
» The RE “$(a b)^*aba$” can be decomposed as: <ol style="list-style-type: none"> 1. $(a b)^*$ 2. a 3. b 4. a <p>We'll build NFAs for each component and then concatenate them.</p>	NFA for a	NFA for b
		
Step 4: NFA for $(a b)^*$ (Star)	Step 3: NFA for $(a b)$ (Union) 	
Step 5: NFA for “aba” (Concatenation)	Step 6: Combine $(a b)^*$ and aba (Concatenation)  	

 **Problem2:** Convert the following Regular Expression into equivalent NFA using formal procedure. “ $ab^*a \cup a^*b^*a$ ”.

Step 1: Parse the RE into Subexpressions	Step 2: Build NFAs for Basic Symbols		
» The RE “$ab^*a \cup a^*b^*a$” can be decomposed as: <ol style="list-style-type: none"> 1. ab^*a 2. a^*b^*a 3. a 5. a^* 4. b 6. b^* 	NFA for a 		
We'll build NFAs for each component and then union them.		NFA for b 	
Step 4: NFA for b^* (Star)	Step 3: NFA for a^* (Star)		
			
Step 5: NFA for ab^*a (Concatenation)	Step 6: NFA for a^*b^*a (Concatenation)		
			

Step 7: Final NFA for $ab^*a \cup a^*b^*a$ (Union)



Problem3: Convert the following Regular Expression into equivalent NFA using formal procedure. “ $((a \mid b)^*c)^*$ ”.

Step 1: Parse the RE into Subexpressions	Step 2: Build NFAs for Basic Symbols
<p>» The RE “$((a b)^*c)^*$” can be decomposed as:</p> <ol style="list-style-type: none"> 1. $(a b)^*$ 2. a 3. b 4. c <p>We'll build NFAs for each component and then add them.</p>	<p>NFA for a NFA for b NFA for c</p> <pre> graph LR S1(()) -- a --> FA1((())) S2(()) -- b --> FA2((())) S3(()) -- c --> FA3((())) </pre>

Step 4: NFA for $(a b)^*$ (Star)	Step 3: NFA for $(a b)$ (Union)
Step 5: NFA for "$(a b)^*c$" (Concatenation)	Step 6: Combine "$((a \mid b)^*c)^*$". (Star)

Problem4: Convert the following Regular Expression into equivalent NFA using formal procedure. “ $a (b \mid c)^*d$ ”

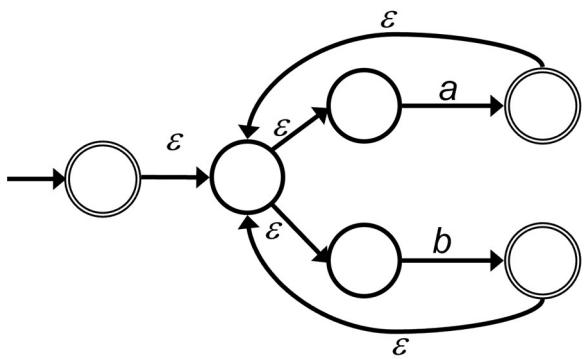
Step 1: Parse the RE into Subexpressions	Step 2: Build NFAs for Basic Symbols	
» The RE “$a (b \mid c)^*d$” can be decomposed as: <ol style="list-style-type: none"> 1. $(b c)$ 2. a 3. b 4. c 5. d <p>We'll build NFAs for each component and then concatenate them.</p>	NFA for a NFA for b	NFA for c NFA for d

Step 4: NFA for $(b c)^*$ (Star)	Step 3: NFA for $(b c)$ (Union)
Step 5: NFA for "$(b c)^*d$" (Concatenation)	Step 6: Combine "$a(b c)^*d$" (Concatenation)

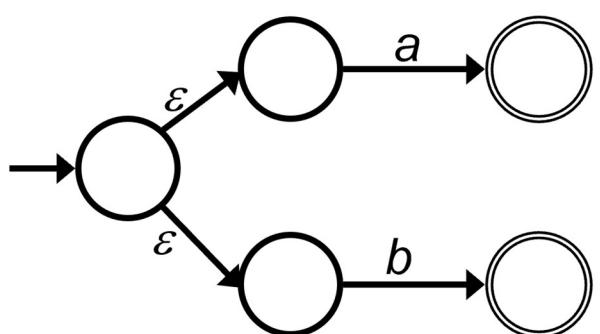
Problem 5: Convert the following Regular Expression into equivalent NFA using formal procedure. " $(a | b)^*abc$ ".

Step 1: Parse the RE into Subexpressions	Step 2: Build NFAs for Basic Symbols
» The RE "$(a b)^*abc$" can be decomposed as: 1. $(a b)^*$ 2. a 3. b 4. c	NFA for a NFA for b NFA for c
<u>We'll build NFAs for each component and then concatenate them.</u>	

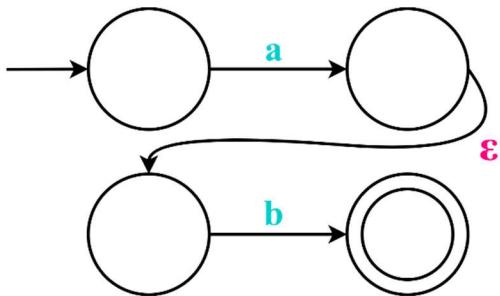
Step 4: NFA for $(a|b)^*$ (Star)



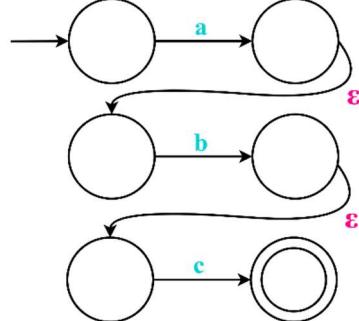
Step 3: NFA for $(a|b)$ (Union)



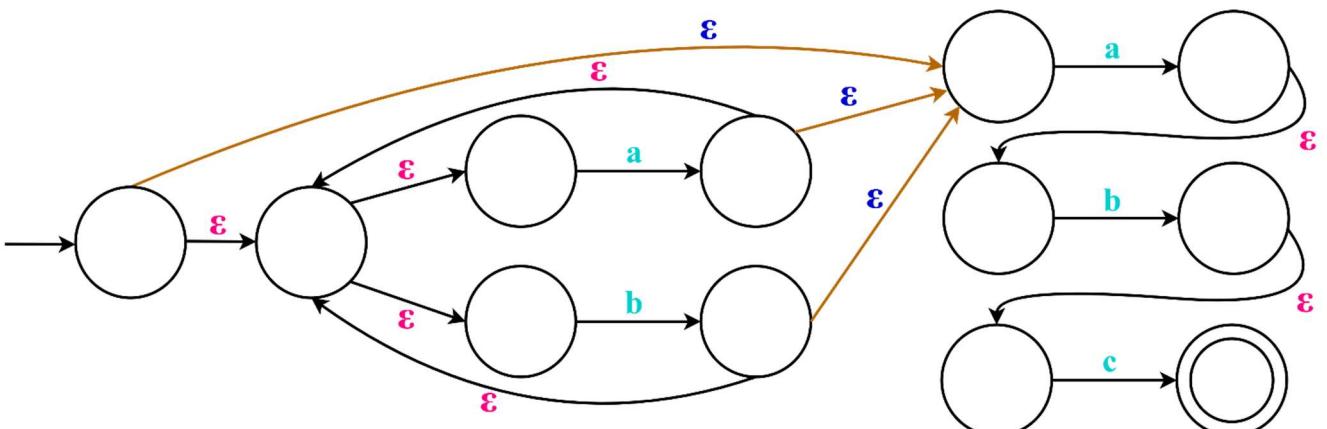
Step 5: NFA for "ab" (Concatenation)



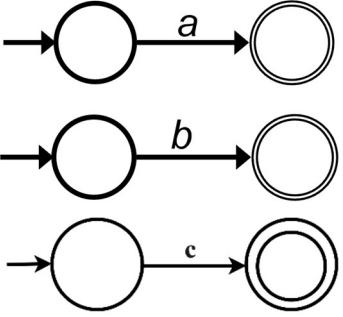
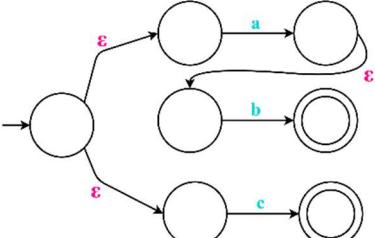
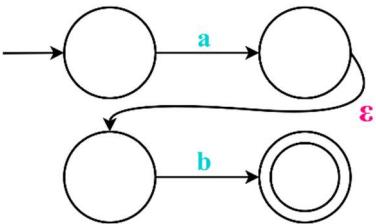
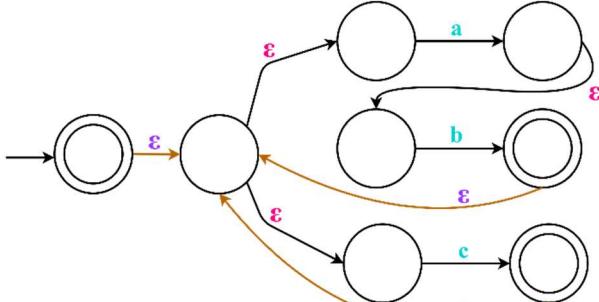
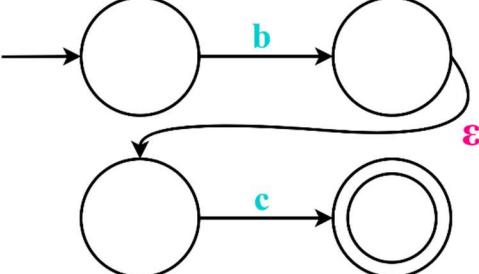
Step 6: Combine "abc". (Concatenation)



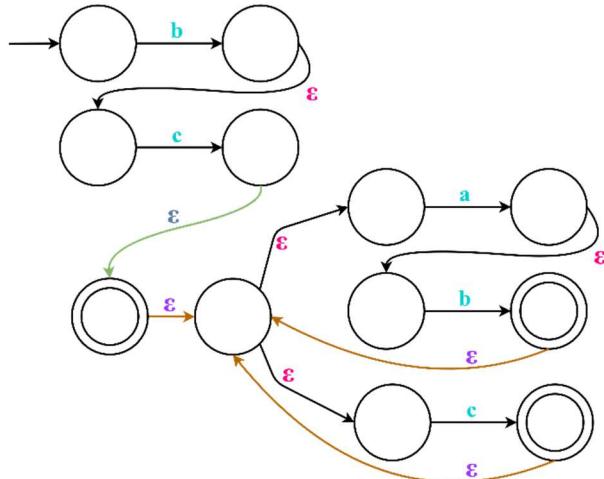
Step 7: Combine " $(a \mid b)^*abc$ ". (Concatenation)



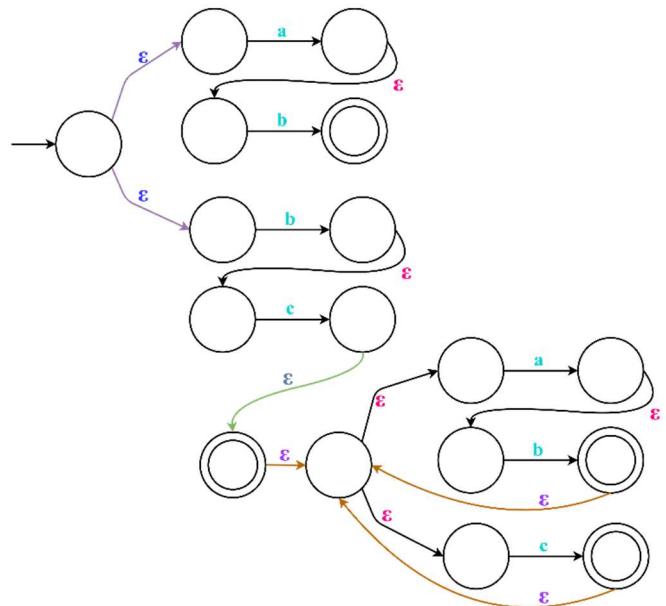
 **Problem6:** Convert the following Regular Expression into equivalent NFA using formal procedure. “**(abUbc(abUc)*)***”.

Step 1: Parse the RE into Subexpressions	Step 2: Build NFAs for Basic Symbols
<p>» The RE “(abUbc(abUc)*)*” can be decomposed as:</p> <ol style="list-style-type: none"> 1. $(ab c)^*$ 2. ab 3. bc 4. a 5. b 6. c 	<p>NFA for a NFA for b NFA for c</p>
<p>We'll build NFAs for each component and then add them.</p>	
Step 4: NFA for $(ab c)$ (Union)	Step 3: NFA for (ab) (Concatenation)
	
Step 5: NFA for $“(ab c)^*”$ (Star)	Step 6: Combine “bc”. (Concatenation)
	

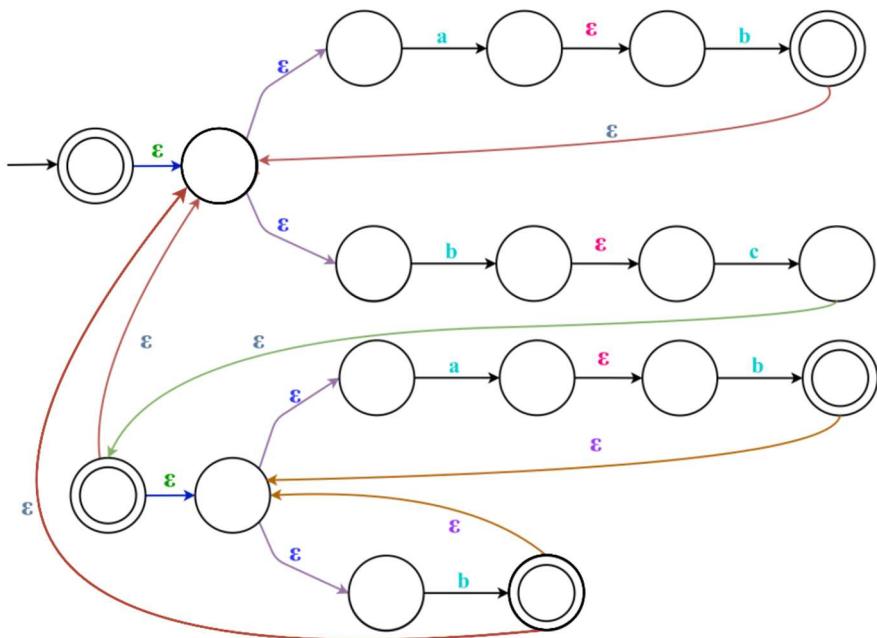
Step 7: Combine “ $bc(abUc)^*$ ”. (Concatenation)



Step 8: Combine “ $abUbc(abUc)^*$ ”. (union)



Step 9: Combine “ $(abUbc(abUc)^*)^*$ ”. (Star)



CONVERSION OF NFA TO DFA

Conversion from NFA with ϵ to DFA

→ Non-deterministic finite automata (NFA) is a finite automata where for some cases when a **specific input is given to the current state**, the machine **goes to multiple states** or **more than 1 states**. It can **contain ϵ move**.

It can be represented as **M = {Q, Σ , δ , q_0 , F}**.

Were,

1. **Q: finite set of states**
2. **Σ : finite set of the input symbol**
3. **q_0 : initial state**
4. **F: final state**
5. **δ : Transition function**

NFA with ϵ move: If any FA **contains ϵ transaction or move**, the finite automata is called NFA with ϵ move.

ϵ -closure: ϵ -closure for a given **state A means a set of states** which can be **reached from the state A with only ϵ (null) move including the state A itself**.

Steps for converting NFA with ϵ to DFA

Step 1: We will take the ϵ -closure for the starting state of NFA as a starting state of DFA.

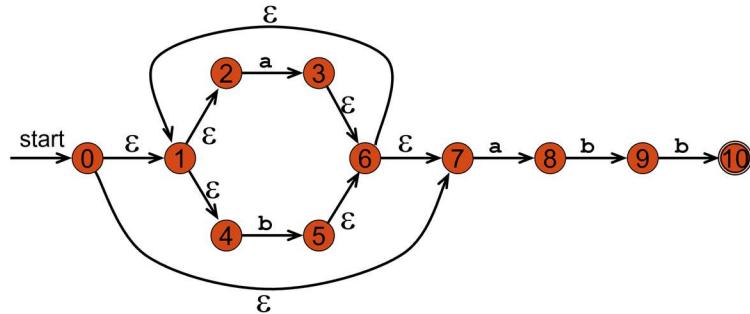
Step 2: Find the states for each input symbol that can be traversed from the present. That means the union of transition value and their closures for each state of NFA present in the current state of DFA.

Step 3: If we found a new state, take it as current state and repeat step 2.

Step 4: Repeat Step 2 and Step 3 until there is no new state present in the transition table of DFA.

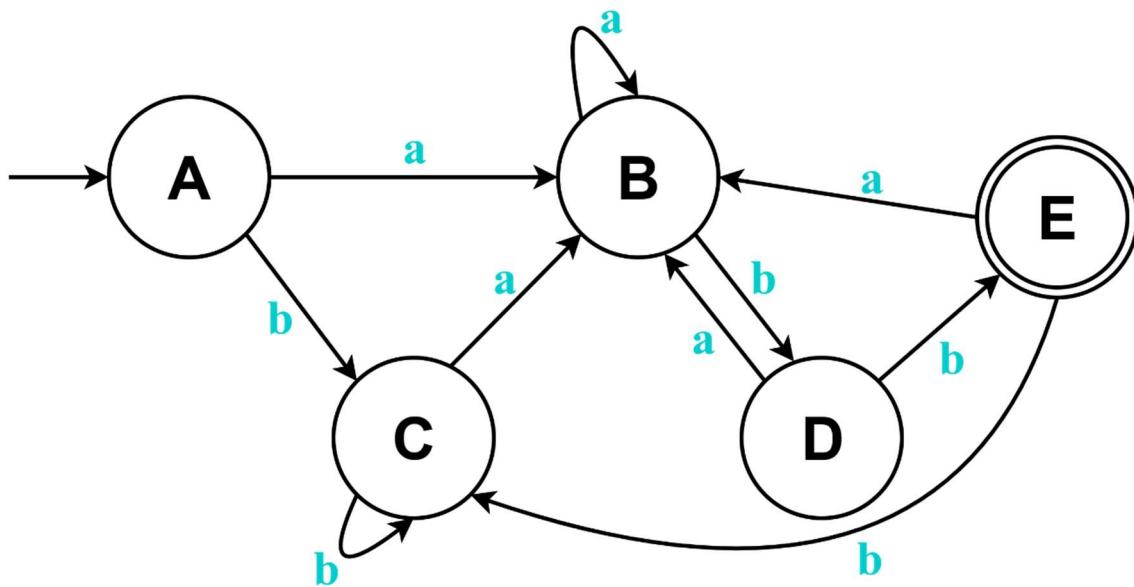
Step 5: Mark the states of DFA as a final state which contains the final state of NFA.

Problem 1: Convert the following NFA with ϵ into equivalent DFA using Subset Construction Method



ε-Closure Table for DFA States			DFA Transition Table Constructed from NFA			
DFA State	ε -closure of	ε -closure outcome states	NFA States	DFA State	a	b
A	ε-Closure of (<{0})	0,1,2,4,7	0,1,2,4,7	→ A	B	C
B	ε-Closure of (<{3,8})	3,6,7,1,2,4,8	3,6,7,1,2,4,8	B	B	D
C	ε-Closure of (<{5})	5,6,7,1,2,4	5,6,7,1,2,4	C	B	C
D	ε-Closure of (<{5,9})	5,6,7,1,2,4,9	5,6,7,1,2,4,9	D	B	E
E	ε-Closure of (<{5,10})	5,6,7,1,2,4,10	5,6,7,1,2,4,10	E	B	C

Now, Construct the DFA from [transition table](#):



"An Alternative Approach to Solving It"

Step1: Write the transition table of NFA

δ	a	b
0	ϕ	ϕ
1	ϕ	ϕ
2	3	ϕ
3	ϕ	ϕ
4	ϕ	5
5	ϕ	ϕ
6	ϕ	ϕ
7	8	ϕ
8	ϕ	9
9	ϕ	10
10	ϕ	ϕ

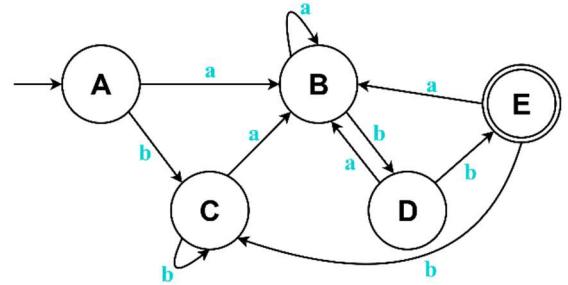
Step2: Find the Epsilon (ϵ) Closure of all the state

- ϵ -Closure of $(\{0\})=\{0,1,2,4,7\}$
- ϵ -Closure of $(\{1\})=\{1,2,4\}$
- ϵ -Closure of $(\{2\})=\{2\}$
- ϵ -Closure of $(\{3\})=\{3,6,7,1,2,4\}$
- ϵ -Closure of $(\{4\})=\{4\}$
- ϵ -Closure of $(\{5\})=\{5,6,7,1,2,4\}$
- ϵ -Closure of $(\{6\})=\{6,7,1,2,4\}$
- ϵ -Closure of $(\{7\})=\{7\}$
- ϵ -Closure of $(\{8\})=\{8\}$
- ϵ -Closure of $(\{9\})=\{9\}$
- ϵ -Closure of $(\{10\})=\{10\}$

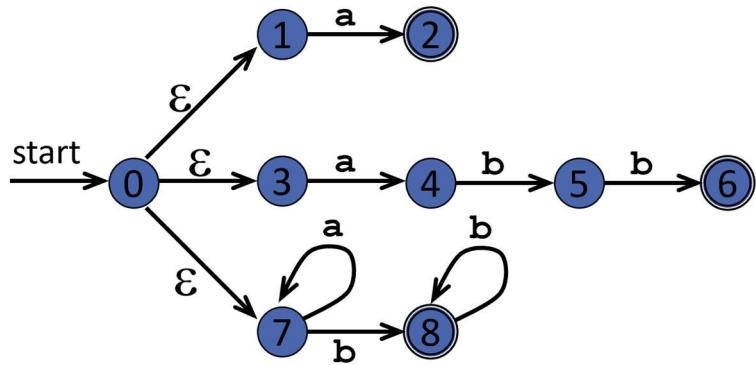
Step3: Convert this NFA transition table to its equivalent DFA transition table.

ϵ -closure of	NFA	DFA	a	b
ϵ -Closure of $(\{0\})$	0,1,2,4,7	$\rightarrow A$	B	C
ϵ -Closure of $(\{3,8\})$	3,6,7,1,2,4,8	B	B	D
ϵ -Closure of $(\{5\})$	5,6,7,1,2,4	C	B	C
ϵ -Closure of $(\{5,9\})$	5,6,7,1,2,4,9	D	B	E
ϵ -Closure of $(\{5,10\})$	5,6,7,1,2,4,10	E	B	C

Step4: Now, Construct the DFA from transition table.



Problem2: Convert the following NFA with ϵ into equivalent DFA using Subset Construction Method



Step1: Write the transition table of NFA

δ	a	b
0	ϕ	ϕ
1	2	ϕ
2	ϕ	ϕ
3	4	ϕ
4	ϕ	5
5	ϕ	6
6	ϕ	ϕ
7	7	8
8	ϕ	8

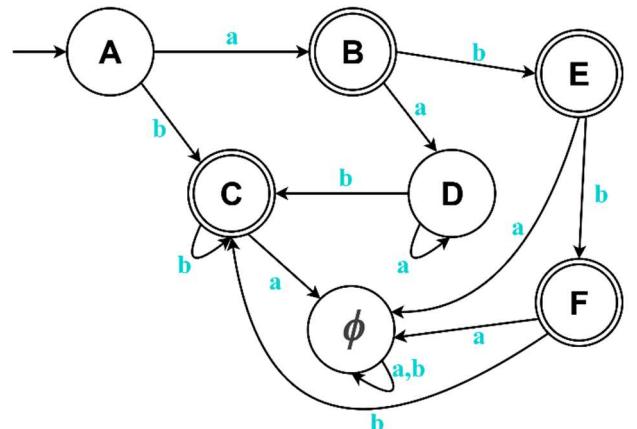
Step2: Find the Epsilon (ϵ) Closure of all the state

- ϵ -Closure of $(\{0\})=\{0,1,3,7\}$
- ϵ -Closure of $(\{1\})=\{1\}$
- ϵ -Closure of $(\{2\})=\{2\}$
- ϵ -Closure of $(\{3\})=\{3\}$
- ϵ -Closure of $(\{4\})=\{4\}$
- ϵ -Closure of $(\{5\})=\{5\}$
- ϵ -Closure of $(\{6\})=\{6\}$
- ϵ -Closure of $(\{7\})=\{7\}$
- ϵ -Closure of $(\{8\})=\{8\}$

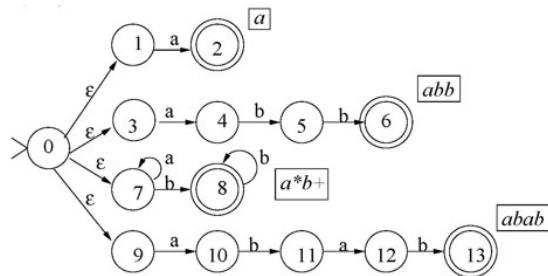
Step3: Convert this NFA transition table to its equivalent DFA transition table.

ϵ -closure of	NFA	DFA	a	b
ϵ -Closure of $(\{0\})$	$0,1,3,7$	A	B	C
ϵ -Closure of $(\{2,4,7\})$	$2,4,7$	B	D	E
ϵ -Closure of $(\{8\})$	8	C	ϕ	C
ϵ -Closure of $(\{7\})$	7	D	D	C
ϵ -Closure of $(\{5,8\})$	5,8	E	ϕ	F
ϵ -Closure of $(\{6,8\})$	6,8	F	ϕ	C
		ϕ	ϕ	ϕ

Step4: Now, Construct the DFA from transition table.



Problem2: Convert the following NFA with ϵ into equivalent DFA using Subset Construction Method



Step1: Write the transition table of NFA

δ	a	b
0	ϕ	ϕ
1	2	ϕ
2	ϕ	ϕ
3	4	ϕ
4	ϕ	5
5	ϕ	6
6	ϕ	ϕ
7	7	8
8	ϕ	8
9	10	ϕ
10	ϕ	11
11	12	ϕ
12	ϕ	13
13	ϕ	ϕ

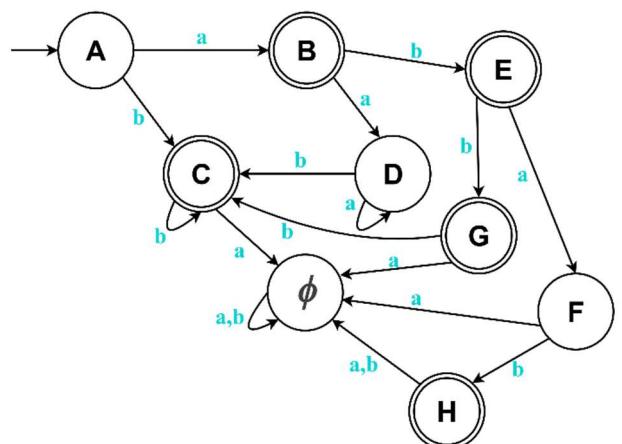
Step2: Find the Epsilon (ϵ) Closure of all the state

- ϵ -Closure of $\{0\} = \{0, 1, 3, 7, 9\}$
- ϵ -Closure of $\{1\} = \{1\}$
- ϵ -Closure of $\{2\} = \{2\}$
- ϵ -Closure of $\{3\} = \{3\}$
- ϵ -Closure of $\{4\} = \{4\}$
- ϵ -Closure of $\{5\} = \{5\}$
- ϵ -Closure of $\{6\} = \{6\}$
- ϵ -Closure of $\{7\} = \{7\}$
- ϵ -Closure of $\{8\} = \{8\}$
- ϵ -Closure of $\{9\} = \{9\}$
- ϵ -Closure of $\{10\} = \{10\}$
- ϵ -Closure of $\{11\} = \{11\}$
- ϵ -Closure of $\{12\} = \{12\}$
- ϵ -Closure of $\{13\} = \{13\}$

Step3: Convert this NFA transition table to its equivalent DFA transition table.

Step4: Now, Construct the DFA from transition table.

ϵ -closure of	NFA	DFA	a	b
ϵ -Closure of $\{0\}$	0,1,3,7,9 \rightarrow A	B	C	
ϵ -Closure of $\{2,4,7,10\}$	2,4,7,10	B	D	E
ϵ -Closure of $\{8\}$	8	C	ϕ	C
ϵ -Closure of $\{7\}$	7	D	D	C
ϵ -Closure of $\{5,8,11\}$	5,8,11	E	F	G
ϵ -Closure of $\{12\}$	12	F	ϕ	H
ϵ -Closure of $\{6,8\}$	6,8	G	ϕ	C
ϵ -Closure of $\{13\}$	13	H	ϕ	ϕ

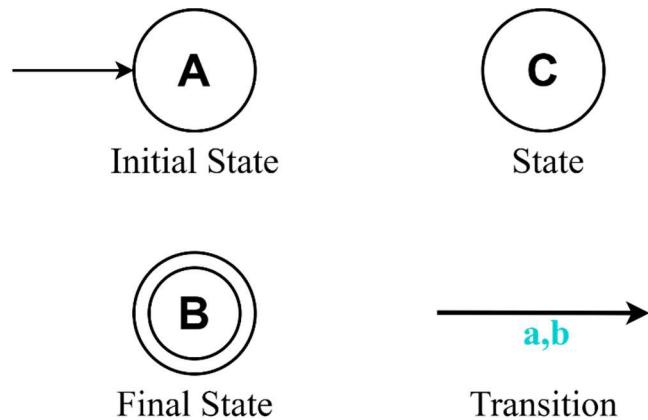


DETERMINISTIC FINITE AUTOMATON (DFA)

Finite Representation (FR) / Finite Automaton (FA)

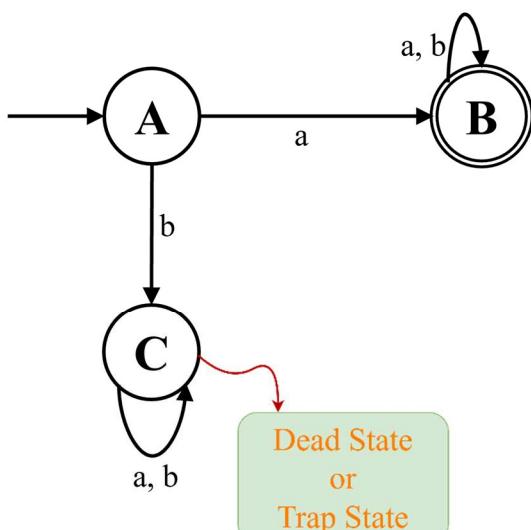
- **Definition:** If a language can be recognized with a finite number of states, it's called Finite Representation (FR) or Finite Automaton (FA).
- **Key Point:** FA uses a finite number of states to represent and accept or reject strings from a language.

→ Finite Representation:

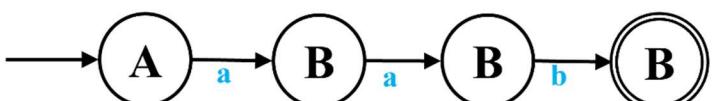


∅ Example 1:

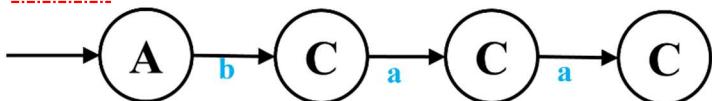
- ➔ $\Sigma = \{a, b\}$
- ➔ L1 = Set of all strings starting with 'a'
- ➔ $L1 = \{a, aa, ab, abb, aba, \dots\}$



ACCEPT:



REJECT:

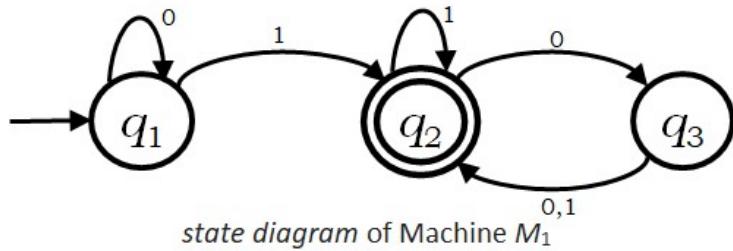


Formal Definition - DFA

→ A DFA is a **5-tuple**: $(Q, \Sigma, \delta, q_0, F)$

1. $Q \rightarrow$ Finite set of states.
2. $\Sigma \rightarrow$ Finite input alphabet.
3. $\delta: Q \times \Sigma \rightarrow Q \rightarrow$ Transition function (maps state & input to next state).
4. $q_0 \in Q \rightarrow$ Start state.
5. $F \subseteq Q \rightarrow$ Set of accept (final) states.

↗ Example 1:



→ Here,

➤ $M_1 = (Q, \Sigma, \delta, q_0, F),$

➤ Where:

$$Q = \{q_1, q_2, q_3\},$$

$$\Sigma = \{0, 1\},$$

$$F = \{q_2\},$$

$$q_0 = \{q_1\},$$

δ is describe as:

$\delta(q_1, 0) = q_1, \quad \delta(q_1, 1) = q_2,$
$\delta(q_2, 0) = q_3, \quad \delta(q_2, 1) = q_2,$
$\delta(q_3, 0) = q_2, \quad \delta(q_3, 1) = q_2.$

Transition Function

Or

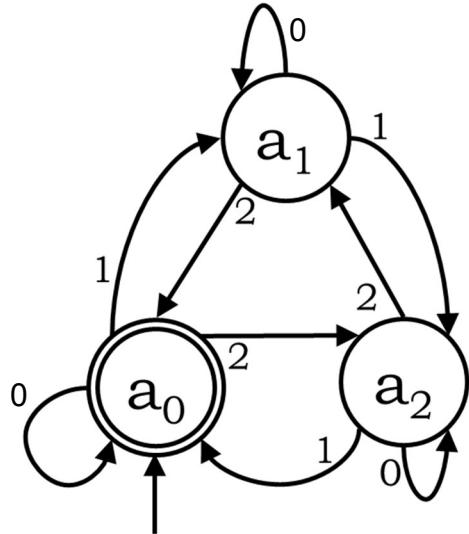
δ	0	1
q ₁	q ₁	q ₂
q ₂	q ₃	q ₂
q ₃	q ₂	q ₂

Transition Table

Designing DFA

Example 1:

→ Design a DFA that accepts strings where the sum of all symbols (0,1,2) is a multiple of 3.



Input example: 01120101

Input symbol: Accepted

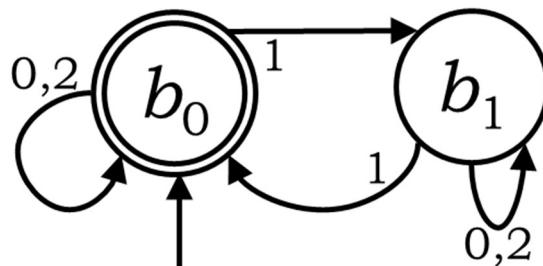
→ DFA for Sum of Symbols $\equiv 0,1,2 \pmod{3}$:

- **Alphabet:** $\Sigma = \{0,1,2\}$
- **Language:** Strings where the sum of symbols is a multiple of 3.
- **States:** $Q = \{a_0, a_1, a_2\}$, where a_i represents $S \bmod 3 = i$.
- **Start State:** $q_0 = a_0$
- **Final State:** $F = \{a_0\}$
- **Transition Table (δ_1):**

δ	0	1	2
a_0	a_0	a_1	a_2
a_1	a_1	a_2	a_0
a_2	a_2	a_0	a_1

Example 2:

→ Design a DFA that accepts strings where the sum of all symbols (0,1,2) is an even number.



Input example: 01120101

Input symbol: Accepted

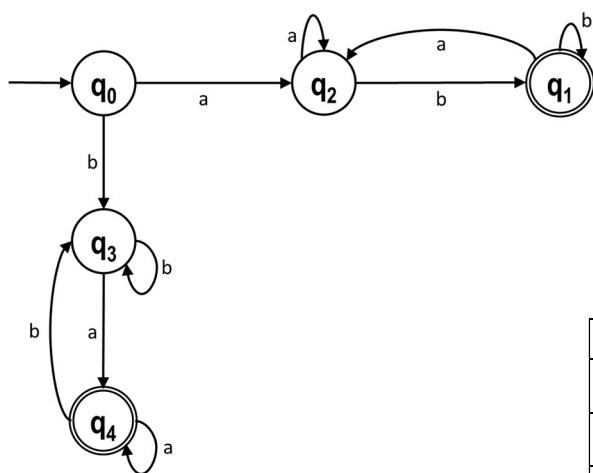
DFA for Sum of Symbols $\equiv 0,1,2 \pmod{2}$ (Even Sum):

- **Alphabet:** $\Sigma = \{0,1,2\}$
- **Language:** Strings where the sum of symbols is an even number.
- **States:** $Q = \{b_0, b_1\}$, where b_i represents $S \bmod 2 = i$.
- **Start State:** $q_0 = b_0$
- **Final State:** $F = \{b_0\}$
- **Transition Table (δ_2):**

δ	0	1	2
b_0	b_0	b_1	b_0
b_1	b_1	b_0	b_1

Example 3:

→ Identify the Components of the DFA



Components of the DFA:

- **Alphabet:** $\Sigma = \{a, b\}$
- **States:** $Q = \{q_0, q_1, q_2, q_3, q_4\}$,
- **Start State:** $q_0 = q_0$
- **Final State:** $F = \{q_1, q_4\}$
- **Transition Table (δ):**

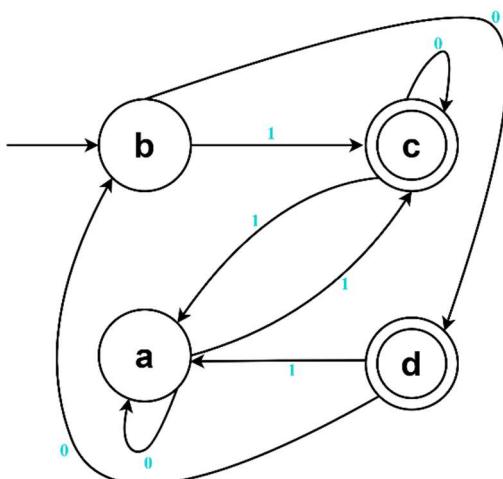
δ	a	b
q_0	q_2	q_3
q_1	q_2	q_1
q_2	q_2	q_1
q_3	q_4	q_3
q_4	q_4	q_3

Example 4:

→ Design a DFA that $Q = \{a, b, c, d\}$, $\Sigma = \{0, 1\}$, $q_0 = b$, $F = \{c, d\}$,

δ	0	1
a	a	c
b	d	c
c	c	a
d	b	a

→ Solution:



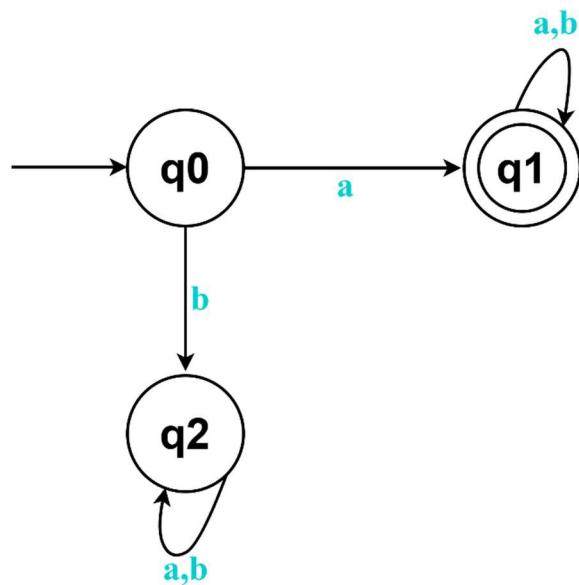
Example 5:

→ Given the state diagram of the machine ($\{q_0, q_1, q_2\}, \{a, b\}, \delta, q_0, \{q_1\}$) Where δ is given in the following table

δ	a	b
q_0	q_1	q_2
q_1	q_1	q_1
q_2	q_2	q_2

→ Design The DFA

→ Solution:

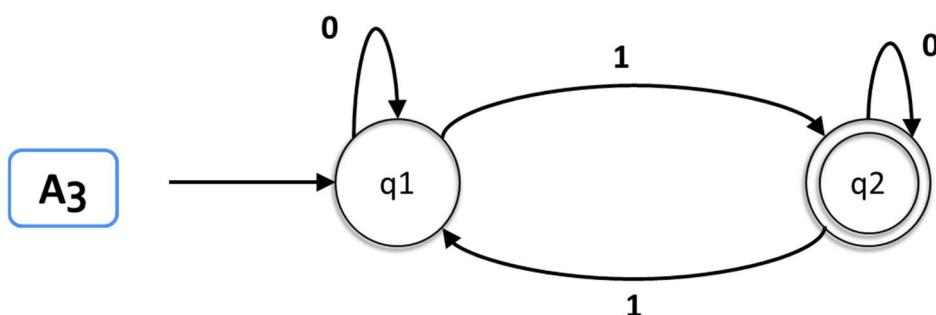


Example 6:

→ $A_3 = \{w : w \text{ is a binary string containing an odd number of } 1s\}$. $\Sigma = \{0,1\}$

→ So, $w_1 = 1$, $w_2 = 10$, $w_3 = 10101$, $w_4 = 10101\dots$

Now, $L = \{1, 100, 10101, 11010, 11100\dots\}$



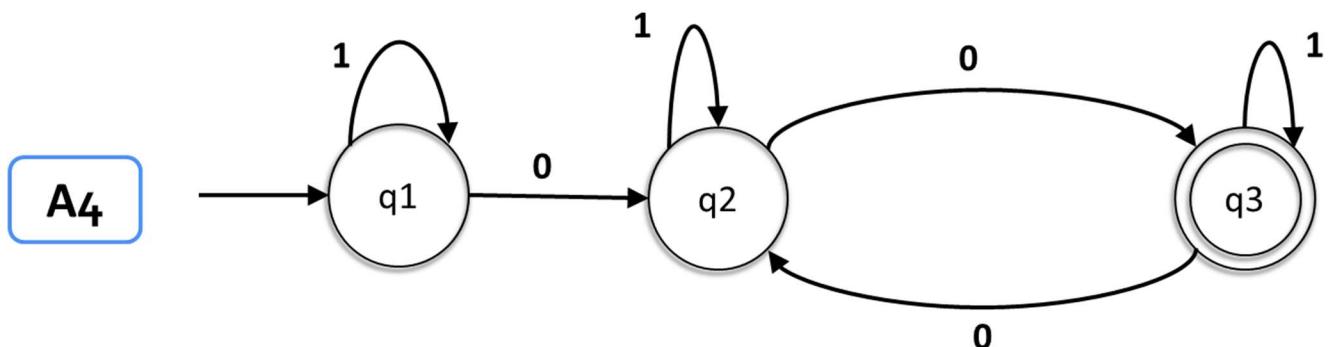
Accepted

Example 7:

→ A4 = {w : w is a binary string containing an even number of 0s}. $\Sigma = \{0,1\}$

→ So, $w_1 = 00, w_2 = 001, w_3 = 0101, w_4 = 0101010\dots$

Now, $L = \{00, 100, 1010, 11010, 11100\dots\}$

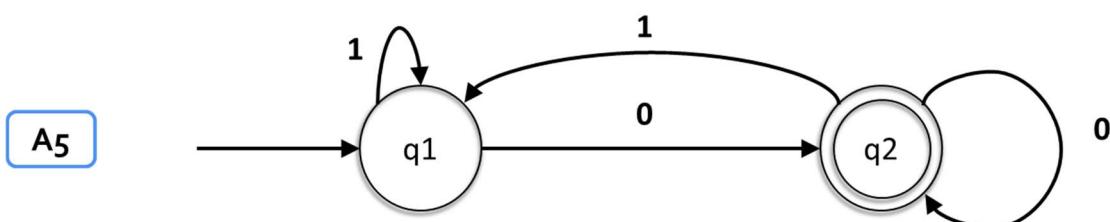


Example 8:

→ A5 = {w | w ends with a 0}. $\Sigma = \{0,1\}$

→ So, $w_1 = 010, w_2 = 000, w_3 = 10, w_4 = 0101010\dots$

Now, $L = \{0, 10, 100, 11010, 11100\dots\}$

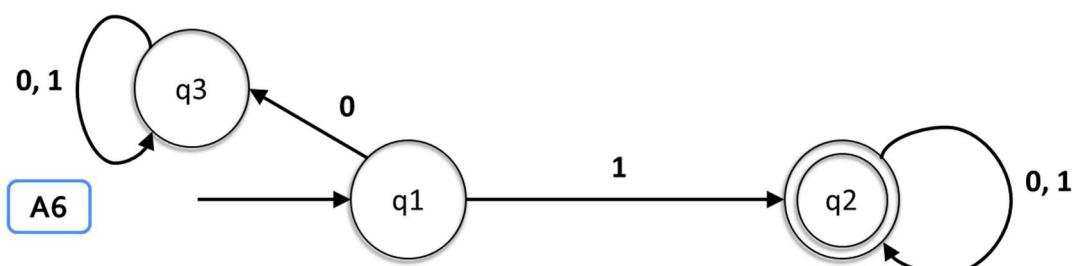


Example 9:

→ A6 = {w | w begins with a 1}. $\Sigma = \{0,1\}$

→ So, $w_1 = 100, w_3 = 10, w_5 = 101010\dots$

Now, $L = \{1, 11, 101, 1001, 11100\dots\}$

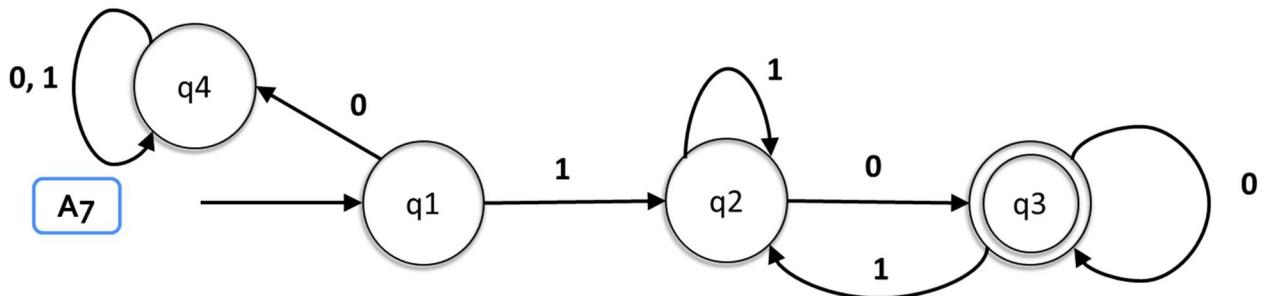


Example 10:

→ A7 = {w | w begins with a 1 and ends with a 0}. $\Sigma = \{0,1\}$

→ So, $w_1 = 100$, $w_2 = 10$, $w_3 = 101010\dots$

Now, $L = \{1, 100, 1010, 11010, 11100\dots\}$

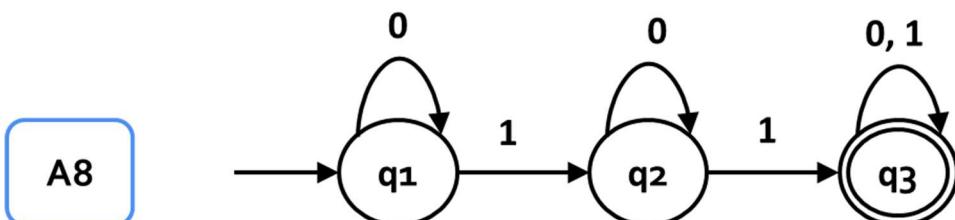


Example 11:

→ A8 = {w | w has at least two 1s}. $\Sigma = \{0,1\}$

→ So, $w_1 = 1010$, $w_2 = 101$, $w_3 = 01010$, $w_4 = 101010\dots$

Now, $L = \{11, 1100, 1010, 11010, 11100\dots\}$

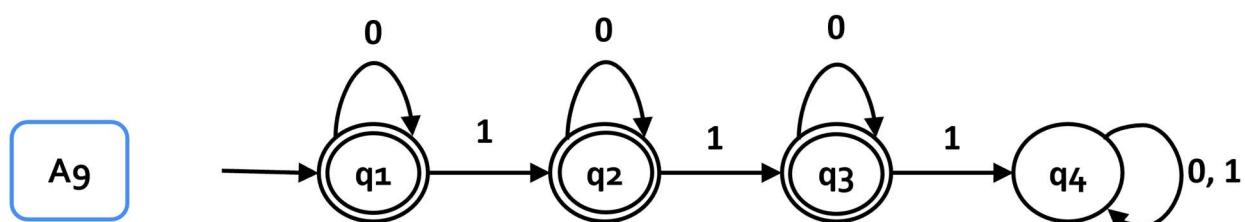


Example 12:

→ A9 = {w | w has at most two 1s}. $\Sigma = \{0,1\}$

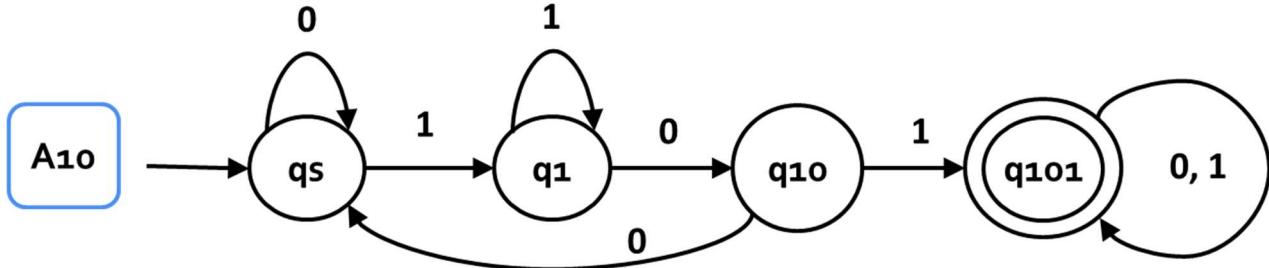
→ So, $w_1 = 1010$, $w_2 = 001$, $w_3 = 000$, $w_4 = 10100\dots$

Now, $L = \{11, 1100, 1010, 11010, 11100\dots\}$



∅ Example 13:
→ $A_{10} = \{w \mid w \text{ has substring } 101\}$. $\Sigma = \{0,1\}$

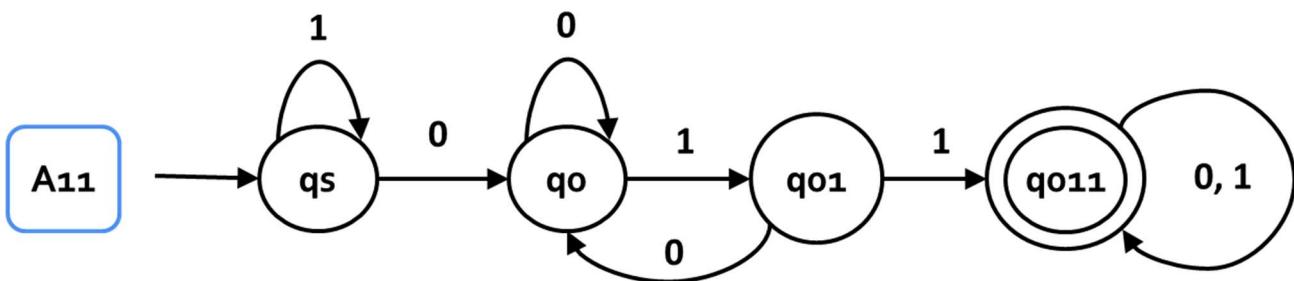
→ So, $w_1 = 1010$, $w_2 = 001$, $w_3 = 10100\dots$
Now, $L = \{101, 1101, 10101, 11010, 111010\dots\}$



∅ Example 14:
→ $A_{11} = \{w \mid w \text{ has substring } 011\}$. $\Sigma = \{0,1\}$

→ So, $w_1 = 10110$, $w_2 = 000110$, $w_3 = 101100\dots$

Now, $L = \{011, 11011, 10101011, 110110, 1011010\dots\}$



→ What happens for the language, $A_{11} = \{w \mid w \text{ does not have substring } 011\}$?

→ For $A_{11} = \{w \mid w \text{ has substring } "011"\}$, the DFA **accepts** strings containing "011".

→ For $A'_{11} = \{w \mid w \text{ does not have substring } "011"\}$, the DFA **rejects** any string with "011" and stays in states ensuring "011" never appears.

→ A **dead state** is used for rejection.

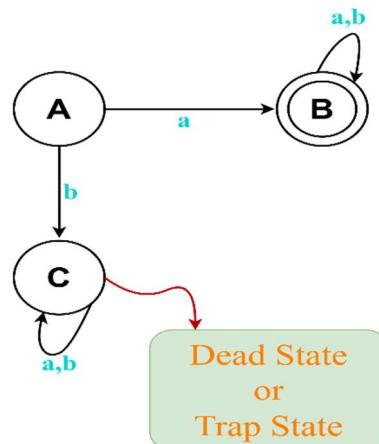
❖ Problem 1:

» Construct a DFA which accepts set of all string over the $\Sigma = \{a,b\}$, where each string starts with an “a”

Or $L(M) = \{w \mid w \text{ Start with an } "a"\}$

→ Is Given,

$$\begin{aligned}\Sigma &= \{a,b\}, \\ L(M) &= \{a, aa, aba, aabba, \dots\}\end{aligned}$$



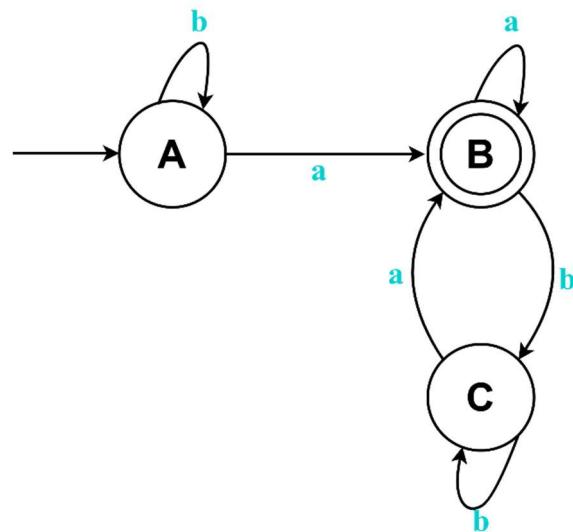
❖ Problem 2:

» Construct a DFA which accepts set of all string over the $\Sigma = \{a,b\}$, where each string ends with an “a”

Or $L(M) = \{w \mid w \text{ End with an } "a"\}$

→ Is Given,

$$\begin{aligned}\Sigma &= \{a,b\}, \\ L(M) &= \{a, aa, aba, aabba, abababa, bba, \dots\}\end{aligned}$$

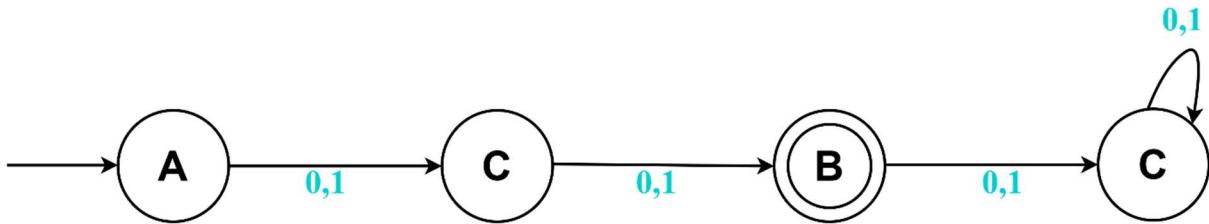


❖ Problem 3:

» Construct a DFA that accepts set of all string over the $\Sigma = \{0,1\}$, of length 2.

→ Is Given,

$$\begin{aligned}\Sigma &= \{0,1\}, \\ L(M) &= \{00, 01, 10, 11\}\end{aligned}$$



❖ Problem 4:

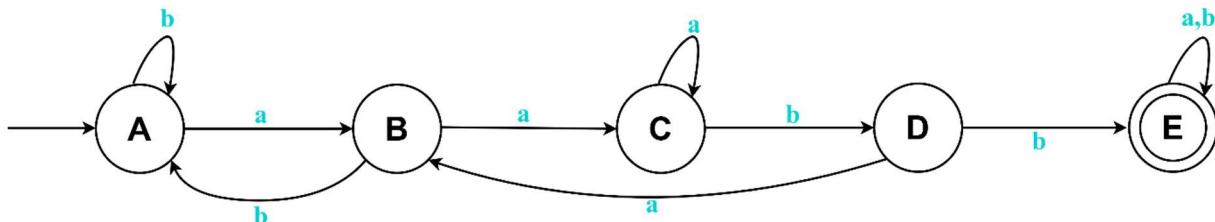
» Construct a DFA that accepts any string over the $\Sigma = \{a,b\}$, that does not contain the string “aabb” in it.

→ Is Given,

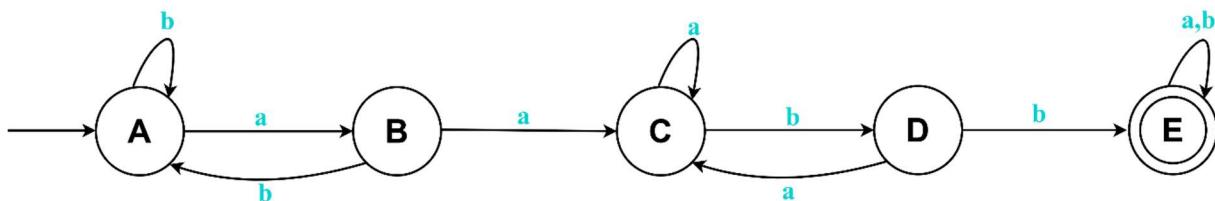
$$\Sigma = \{a,b\},$$

→ Let, Construct a DFA that accepts any string over the $\Sigma = \{a,b\}$, that contains the string “aabb” in it.

$$L(M) = \{aabb, baabb, babaabb, babaaabb, aabaabb, aababaabb, \dots\}$$



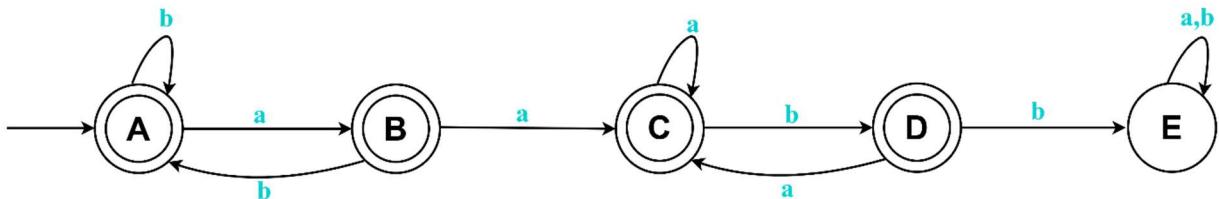
OR



→ Okay, This is not our desired output since it does contain the string "aabb."

→ So, To have our desired output, we just need to:

- Flip the states.
- Make the final states non-final state.
- Make the non-final states final state.



❖ Problem 5:

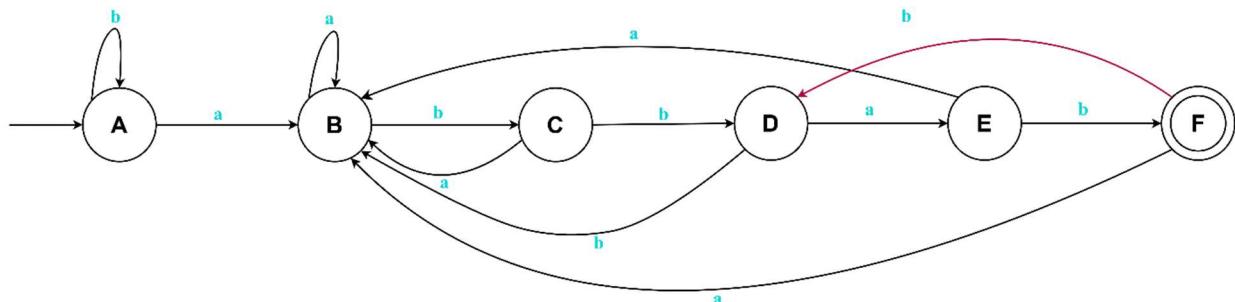
» Design DFA for the following Language. Where $\Sigma = \{a,b\}$ and give the formal definition of your machine. (Any-ONE)

- i. $L = \{w \mid \text{ends with 'abbab' substring}\}$
- ii. $L = \{ w \mid w \text{ contains exactly two a's and odd number of b } \}$ Mode1#

(i)

→ Is Given,

$\Sigma = \{a,b\}$,
 $L(M1) = \{\text{abbab, babbab, aabbab, ababbab, abbbabbab, abbabbab, abbababbab,}\}$

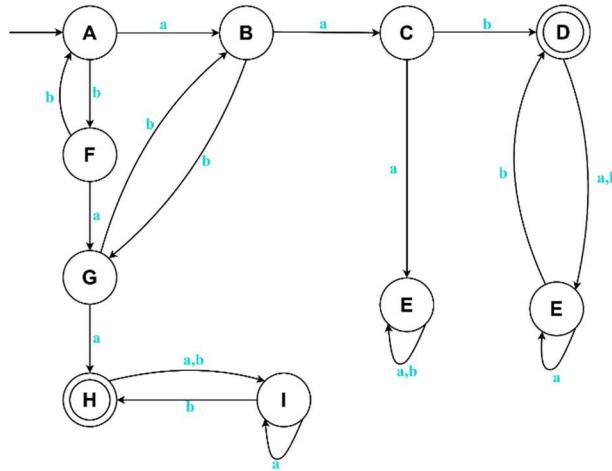


(ii)

→ Is Given,

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

$$L(M1) = \{baa, aab, baabb, babab, bbababb, bbababb, abbab, \dots\}$$



Problem 6:

» Design DFA for the following Language. Consider $\Sigma = \{x, y\}$,

- i. {w | each 'y' in w is followed by at least two 'x' } and simulate your machine using the string: "yxxyxx" and comment whether this string is accepted/ rejected by your machine.

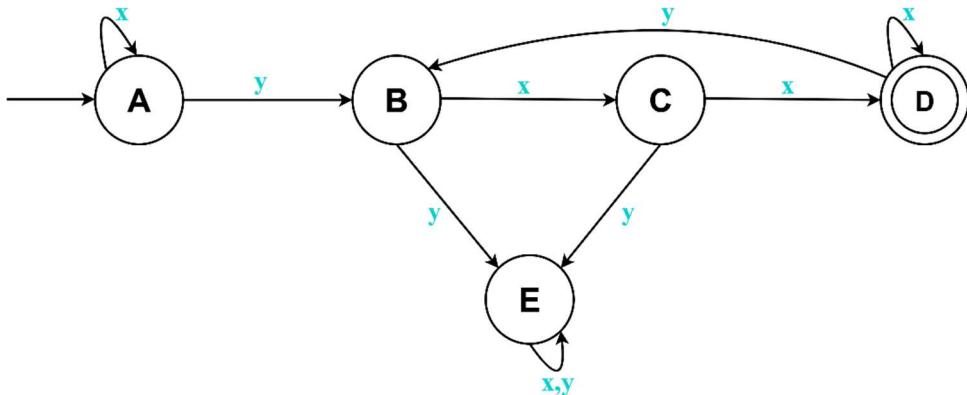
- ii. {w | w starts and ends with same symbol} and simulate your machine using the string: "yyxyxx" and comment whether this string is accepted/ rejected by your machine.

(i)

→ Is Given,

$$\Sigma = \{x, y\},$$

$$L(M1) = \{yxx, yxyxx, yxxxx, \dots\}$$



Eg: xyxyxx

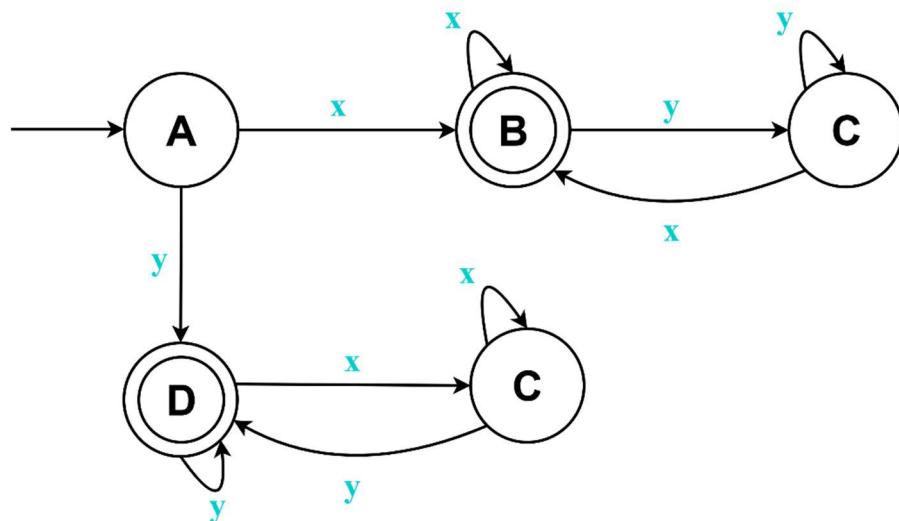


Accepted

(ii)

→ Is Given,

$$\Sigma = \{x, y\}, \\ L(M1) = \{yxy, yxyxy, yxyxyx, \dots\}$$



Eg: yyxyxy



Accepted

FIRST &
FOLLOW

FIRST

- The **FIRST set** of a grammar symbol (terminal or non-terminal) is the set of terminals that can appear **first** in some string derived from that symbol.
- If a non-terminal can derive ϵ (**epsilon**), then ϵ is also included in its FIRST set.

Rules for FIRST Set

Rule1: Terminal → itself	Rule2: $\epsilon \rightarrow \epsilon$
If the symbol is a terminal , its FIRST is the terminal itself.	If the production is epsilon (ϵ) , then FIRST is just { ϵ }.
$\text{FIRST}(a) = \{ a \}$ $\text{FIRST}(+) = \{ + \}$	$A \rightarrow \epsilon$ $\text{FIRST}(A) = \{ \epsilon \}$

Rule3: Non-terminal → check first symbol

If a non-terminal has a production, look at the **first symbol** on the right-hand side:

- If it's a terminal → put it in FIRST.
- If it's a non-terminal → take its FIRST (but **exclude ϵ**).
- If ϵ is in that FIRST, then **look at the next symbol**.
- If all symbols can produce ϵ , then **include ϵ** .

Grammar:	FIRST:
$A \rightarrow a \ B \mid b$ $B \rightarrow c \mid \epsilon$	$\text{FIRST}(A) = \{ a, b \}$ $\text{FIRST}(B) = \{ c, \epsilon \}$

Example 2:

Grammar:	FIRST:
$E \rightarrow T \ E'$ $E' \rightarrow + \ T \ E' \mid \epsilon$ $T \rightarrow id$	$\text{FIRST}(E) = \{ id \}$ $\text{FIRST}(E') = \{ +, \epsilon \}$ $\text{FIRST}(T) = \{ id \}$

Ø Problem 1:

» Calculate the first functions for the given grammar-

$$\begin{aligned} S &\rightarrow aBDh \\ B &\rightarrow cC \\ C &\rightarrow bC \mid \epsilon \\ D &\rightarrow EF \\ E &\rightarrow g \mid \epsilon \\ F &\rightarrow f \mid \epsilon \end{aligned}$$

Solutions :

Grammar:	FIRST:
$S \rightarrow aBDh$	$\text{FIRST}(S) = \{ a \}$
$B \rightarrow cC$	$\text{FIRST}(B) = \{ c \}$
$C \rightarrow bC \mid \epsilon$	$\text{FIRST}(C) = \{ b, \epsilon \}$
$D \rightarrow EF$	$\text{FIRST}(D) = \{ g, f, \epsilon \}$
$E \rightarrow g \mid \epsilon$	$\text{FIRST}(E) = \{ g, \epsilon \}$
$F \rightarrow f \mid \epsilon$	$\text{FIRST}(F) = \{ f, \epsilon \}$

Ø Problem 2:

» Calculate the first functions for the given grammar-

$$\begin{aligned} S &\rightarrow A \\ A &\rightarrow aB \mid dA \\ B &\rightarrow b \\ C &\rightarrow g \end{aligned}$$

Solutions :

Grammar:	FIRST:
$S \rightarrow A$	$\text{FIRST}(S) = \{ a, d \}$
$A \rightarrow aB \mid d$	$\text{FIRST}(A) = \{ a, d \}$
$B \rightarrow b$	$\text{FIRST}(B) = \{ b \}$
$C \rightarrow g$	$\text{FIRST}(C) = \{ g \}$

Ø Problem 3:

» Calculate the first functions for the given grammar-

$$\begin{aligned} S &\rightarrow (L) \mid a \\ L &\rightarrow SL' \\ L' &\rightarrow , SL' \mid \epsilon \end{aligned}$$

Solutions :

Grammar:	FIRST:
$S \rightarrow (L) \mid a$	$\text{FIRST}(S) = \{ (, a \}$
$L \rightarrow SL'$	$\text{FIRST}(L) = \{ (, a \}$
$L' \rightarrow , SL' \mid \epsilon$	$\text{FIRST}(L') = \{ , \epsilon \}$

Ø Problem 4:

» Calculate the first functions for the given grammar-

$$\begin{aligned} S &\rightarrow AaAb \mid BbBa \\ A &\rightarrow \epsilon \\ B &\rightarrow \epsilon \end{aligned}$$

Solutions :

Grammar:	FIRST:
$S \rightarrow AaAb \mid BbBa$ $A \rightarrow \epsilon$ $B \rightarrow \epsilon$	FIRST(S) = { a, b } FIRST(A) = { } FIRST(B) = { }

Ø Problem 5:

» Calculate the first functions for the given grammar-

$$\begin{aligned} E &\rightarrow F + T \mid T \\ T &\rightarrow E \times F \mid F \\ F &\rightarrow (E) \mid id \end{aligned}$$

Solutions :

Grammar:	FIRST:
$E \rightarrow F + T \mid T$ $T \rightarrow E \times F \mid F$ $F \rightarrow (E) \mid id$	FIRST(E) = { (, id } FIRST(T) = { (, id } FIRST(F) = { (, id }

Ø Problem 6:

» Calculate the first functions for the given grammar-

$$\begin{aligned} S &\rightarrow ACB \mid CbB \mid Ba \\ A &\rightarrow da \mid BC \\ B &\rightarrow g \mid \epsilon \\ C &\rightarrow h \mid \epsilon \end{aligned}$$

Solutions :

Grammar:	FIRST:
$S \rightarrow ACB \mid CbB \mid Ba$ $A \rightarrow da \mid BC$ $B \rightarrow g \mid \epsilon$ $C \rightarrow h \mid \epsilon$	FIRST(S) = { d, g, h, \epsilon, b, a } FIRST(A) = { d, g, h, \epsilon } FIRST(B) = { g, \epsilon } FIRST(C) = { h, \epsilon }

FOLLOW

- The FOLLOW(A) of a non-terminal A is the set of terminals that can appear **immediately to the right of A** in some derivation.
- Think of it as: “*what symbols can come after A in the grammar?*”

Rules for FIRST Set

Rule1: Start Symbol Rule	Rule2: $A \rightarrow \alpha B \beta$
For the start symbol s of the grammar:	If a non-terminal B is followed by something (β), then
$\$ \in FOLLOW(s)$	$FIRST(\beta) - \{ \epsilon \} \subseteq FOLLOW(B)$ $A \rightarrow B C$ Here C comes after B , so $FOLLOW(B) = \{ C \}$.

Rule3: $A \rightarrow \alpha B$ or $A \rightarrow \alpha B \beta$ where $\epsilon \in FIRST(\beta)$

- If B is at the **end** of production, or what follows can be ϵ , then:
 $FOLLOW(A) \subseteq FOLLOW(B)$

Grammar:	FOLLOW:
$A \rightarrow a B \mid b$ $B \rightarrow c \mid Ab$	$FOLLOW(A) = \{ \$, b \}$ $FOLLOW(B) = \{ \$, b \}$

Rule4: $A \rightarrow \alpha BC$ where C is a non-terminal.

- If B is followed by $C \rightarrow$
 $FIRST(C) - \{ \epsilon \} \subseteq FOLLOW(B)$

Grammar:	FIRST:
$A \rightarrow B C$ $B \rightarrow b B$ $C \rightarrow c \mid \epsilon$	$FIRST(A) = \{ b \}$ $FIRST(B) = \{ b \}$ $FIRST(C) = \{ c, \epsilon \}$

FOLLOW:

$$\begin{aligned}
 FOLLOW(A) &= \{ \$ \} \\
 FOLLOW(B) &= \{ c, \$ \} \\
 FOLLOW(C) &= \{ \$ \}
 \end{aligned}$$

Ø Problem 1:

» Calculate the first and follow functions for the given grammar-

$$\begin{aligned}
 S &\rightarrow aBdh \\
 B &\rightarrow cC \\
 C &\rightarrow bC \mid \epsilon \\
 D &\rightarrow EF \\
 E &\rightarrow g \mid \epsilon \\
 F &\rightarrow f \mid \epsilon
 \end{aligned}$$

Solutions :

FIRST:	FOLLOW:
$\text{FIRST}(S) = \{ a \}$	$\text{FOLLOW}(S) = \{ \$ \}$
$\text{FIRST}(B) = \{ c \}$	$\text{FOLLOW}(B) = \{ g, f, h \}$
$\text{FIRST}(C) = \{ b, \epsilon \}$	$\text{FOLLOW}(C) = \{ g, f, h \}$
$\text{FIRST}(D) = \{ g, f, \epsilon \}$	$\text{FOLLOW}(D) = \{ h \}$
$\text{FIRST}(E) = \{ g, \epsilon \}$	$\text{FOLLOW}(E) = \{ f, h \}$
$\text{FIRST}(F) = \{ f, \epsilon \}$	$\text{FOLLOW}(F) = \{ h \}$

Ø Problem 2:

» Calculate the first and follow functions for the given grammar-

$$\begin{aligned}
 S &\rightarrow A \\
 A &\rightarrow aBb \mid dA \\
 B &\rightarrow bCc \\
 C &\rightarrow g
 \end{aligned}$$

Solutions :

FIRST:	FOLLOW:
$\text{FIRST}(S) = \{ a, d \}$	$\text{FOLLOW}(S) = \{ \$ \}$
$\text{FIRST}(A) = \{ a, d \}$	$\text{FOLLOW}(A) = \{ \$ \}$
$\text{FIRST}(B) = \{ b \}$	$\text{FOLLOW}(B) = \{ b \}$
$\text{FIRST}(C) = \{ g \}$	$\text{FOLLOW}(C) = \{ c \}$

Ø Problem 3:

» Calculate the first and follow functions for the given grammar-

$$\begin{aligned} S &\rightarrow (L) \mid a \\ L &\rightarrow SL' \\ L' &\rightarrow , SL' \mid \epsilon \end{aligned}$$

Solutions :

FIRST:	FOLLOW:
$\text{FIRST}(S) = \{ (, a \}$	$\text{FOLLOW}(S) = \{ \$, ,) \}$
$\text{FIRST}(L) = \{ (, a \}$	$\text{FOLLOW}(L) = \{) \}$
$\text{FIRST}(L') = \{ , , \epsilon \}$	$\text{FOLLOW}(L') = \{) \}$

Ø Problem 4:

» Calculate the first and follow functions for the given grammar-

$$\begin{aligned} S &\rightarrow AaAb \mid BbBa \\ A &\rightarrow \epsilon \\ B &\rightarrow \epsilon \end{aligned}$$

Solutions :

FIRST:	FOLLOW:
$\text{FIRST}(S) = \{ a, b \}$	$\text{FOLLOW}(S) = \{ \$ \}$
$\text{FIRST}(A) = \{\epsilon\}$	$\text{FOLLOW}(A) = \{ a, b \}$
$\text{FIRST}(B) = \{\epsilon\}$	$\text{FOLLOW}(B) = \{ b, a \}$

Ø Problem 5:

» Calculate the first and follow functions for the given grammar-

$$\begin{aligned} E &\rightarrow F + T \mid T \\ T &\rightarrow E \times F \mid F \\ F &\rightarrow (E) \mid \text{id} \end{aligned}$$

Solutions :

FIRST:	FOLLOW:
$\text{FIRST}(E) = \{ (, \text{id} \}$	$\text{FOLLOW}(E) = \{ \$, \times,) \}$
$\text{FIRST}(T) = \{ (, \text{id} \}$	$\text{FOLLOW}(T) = \{ \$, \times,) \}$
$\text{FIRST}(F) = \{ (, \text{id} \}$	$\text{FOLLOW}(F) = \{ \$, \times,) \}$

Ø Problem 6:

» Calculate the first and follow functions for the given grammar-

$$\begin{aligned} E &\rightarrow TE' \\ E' &\rightarrow +TE' \quad | \quad \epsilon \\ T &\rightarrow FT' \\ T' &\rightarrow *FT' \quad | \quad \epsilon \\ F &\rightarrow (E) \quad | \quad id \end{aligned}$$

Solutions:

FIRST:	FOLLOW:
$\text{FIRST}(E) = \{ (, \text{id} \}$	$\text{FOLLOW}(E) = \{ \$,) \}$
$\text{FIRST}(E') = \{ +, \epsilon \}$	$\text{FOLLOW}(E') = \{ \$,) \}$
$\text{FIRST}(T) = \{ (, \text{id} \}$	$\text{FOLLOW}(T) = \{ +, \$,) \}$
$\text{FIRST}(T') = \{ *, \epsilon \}$	$\text{FOLLOW}(T') = \{ +, \$,) \}$
$\text{FIRST}(F) = \{ (, \text{id} \}$	$\text{FOLLOW}(F) = \{ *, +, \$,) \ }$

Ø Problem 7:

» Calculate the first and follow functions for the given grammar-

$$\begin{aligned} S &\rightarrow ACB \quad | \quad CbB \quad | \quad Ba \\ A &\rightarrow da \quad | \quad BC \\ B &\rightarrow g \quad | \quad \epsilon \\ C &\rightarrow h \quad | \quad \epsilon \end{aligned}$$

Solutions:

FIRST:	FOLLOW:
$\text{FIRST}(S) = \{ d, g, h, \epsilon, b, a \}$	$\text{FOLLOW}(S) = \{ \$ \}$
$\text{FIRST}(A) = \{ d, g, \epsilon \}$	$\text{FOLLOW}(A) = \{ h, g, \$ \}$
$\text{FIRST}(B) = \{ g, \epsilon \}$	$\text{FOLLOW}(B) = \{ \$, a, h, g \}$
$\text{FIRST}(C) = \{ h, \epsilon \}$	$\text{FOLLOW}(C) = \{ g, b, h, \$ \}$

Ø Problem 8:

» Calculate the first and follow functions for the given grammar-

$$S \rightarrow ABCDE$$

$$A \rightarrow a \mid \epsilon$$

$$B \rightarrow b \mid \epsilon$$

$$C \rightarrow c$$

$$D \rightarrow d \mid \epsilon$$

$$E \rightarrow e \mid \epsilon$$

Solutions:

FIRST:	FOLLOW:
$\text{FIRST}(S) = \{ a, b, c \}$	$\text{FOLLOW}(S) = \{ \$ \}$
$\text{FIRST}(A) = \{ a, \epsilon \}$	$\text{FOLLOW}(A) = \{ b, c \}$
$\text{FIRST}(B) = \{ b, \epsilon \}$	$\text{FOLLOW}(B) = \{ c \}$
$\text{FIRST}(C) = \{ c \}$	$\text{FOLLOW}(C) = \{ d, e, \$ \}$
$\text{FIRST}(D) = \{ d, \epsilon \}$	$\text{FOLLOW}(D) = \{ e, \$ \}$
$\text{FIRST}(E) = \{ e, \epsilon \}$	$\text{FOLLOW}(E) = \{ \$ \}$

Ø Problem 9:

» Calculate the first and follow functions for the given grammar-

$$S \rightarrow ABC \mid DE$$

$$A \rightarrow id \mid (S) \mid \epsilon$$

$$B \rightarrow +B \mid *B \mid \epsilon$$

$$C \rightarrow DA \mid \epsilon$$

$$D \rightarrow \text{num}(A) \mid \text{num} \mid \text{num}(E)$$

$$E \rightarrow S \mid \epsilon$$

Solutions:

FIRST:	FOLLOW:
$\text{FIRST}(S) = \{ id, (, +, *, \text{num}, \epsilon \}$	$\text{FOLLOW}(S) = \{ \$,) \}$
$\text{FIRST}(A) = \{ id, (, \epsilon \}$	$\text{FOLLOW}(A) = \{ +, *, \text{num}, \$,) \}$
$\text{FIRST}(B) = \{ +, *, \epsilon \}$	$\text{FOLLOW}(B) = \{ \text{num}, \$,) \}$
$\text{FIRST}(C) = \{ \text{num}, \epsilon \}$	$\text{FOLLOW}(C) = \{ \$,) \}$
$\text{FIRST}(D) = \{ \text{num} \}$	$\text{FOLLOW}(D) = \{ id, (, +, *, \text{num}, \$,) \}$
$\text{FIRST}(E) = \{ id, (, +, *, \text{num}, \epsilon \}$	$\text{FOLLOW}(E) = \{ \$,) \}$

Left Recursion in Grammar

⇒ A grammar is **left recursive** if there exists a non-terminal A such that:

$$A \Rightarrow A \alpha$$

A can derive a string starting with itself. This causes **infinite recursion** in **top-down parsers** (like recursive descent or LL(1)).

Types of Left Recursion

Rule1: (a) Immediate Left Recursion	
➤ When a production refers to itself directly:	
Example Grammar:	Here:
$A \rightarrow A \alpha \mid \beta$	α = a sequence of grammar symbols (must not be ϵ) β = some alternative that does not begin with A
Elimination Rule:	
Before Elimination:	After Elimination:
$A \rightarrow A \alpha \mid \beta$	$A \rightarrow \beta A'$ $A' \rightarrow \alpha A' \mid \epsilon$

Example 1: Immediate Left Recursion

Grammar:	After Elimination:
$E \rightarrow E + T \mid T$	$E \rightarrow T E'$ $E' \rightarrow + T E' \mid \epsilon$

Rule2: (b) Indirect Left Recursion

- When recursion happens through another non-terminal:

Example Grammar:	Here:
$A \rightarrow B \alpha$ $B \rightarrow A \beta$	This means $A \Rightarrow B \alpha \Rightarrow A \beta \alpha \rightarrow$ left recursion indirectly.

Elimination Rule:

- First, reorder productions so non-terminals appear in a proper sequence.
- Substitute productions to make recursion direct.
- Then remove using the **immediate left recursion rule**.

Before Elimination: Gramma	Step 1: Substitute B in $A \rightarrow B a$
$A \rightarrow B a \mid b$ $B \rightarrow A c \mid d$	$A \rightarrow (A c \mid d) a \mid b$ $A \rightarrow A c a \mid d a \mid b$

Step 2: Now we see immediate left recursion $A \rightarrow A c a$.

Fix it like before:

$$\begin{aligned} A &\rightarrow d a A' \mid b A' \\ A' &\rightarrow c a A' \mid \epsilon \end{aligned}$$

Example 1: Indirect Left Recursion

Grammar:	Step 1: Substitute B in $A \rightarrow B a$
$A \rightarrow B a \mid c$ $B \rightarrow C b \mid d$ $C \rightarrow A c \mid e$	$A \rightarrow C b a \mid c$ $B \rightarrow C b \mid d$ $C \rightarrow A c \mid e$
Step2: Substitute C in $A \rightarrow C b a$	After Elimination:
$A \rightarrow A c b a \mid c$ $B \rightarrow C b \mid d$ $C \rightarrow A c \mid e$	$A \rightarrow c A'$ $A' \rightarrow c b a A' \mid \epsilon$ $B \rightarrow C b \mid d$ $C \rightarrow A c \mid e$

Parsing Table Construction (LL(1))

- An LL(1) parsing table is a table used in **predictive parsing** (top-down).
- “LL(1)” means:
 - L: Left-to-right scanning of input
 - L: Leftmost derivation
 - 1: One token lookahead
- The table helps the parser decide **which production rule to apply** based only on:
 - the **current non-terminal (row)**, and
 - the **next input symbol (column)**.

Requirements

- Grammar must be **free of left recursion**.
- Grammar must be **left factored**.
- Otherwise, conflicts appear (two entries in the same cell).

Construction Rules

⇒ For each production $A \rightarrow \alpha$

Rule1: If $\epsilon \notin FIRST(\alpha)$:	Rule2: If $\epsilon \in FIRST(\alpha)$:
For each terminal $a \in FIRST(\alpha)$, add:	For each terminal $b \in FOLLOW(A)$, add:
$M[A, a] = A \rightarrow \alpha$	$M[A, b] = A \rightarrow \alpha$
Rule3: If $\$ \in FOLLOW(A)$, also add:	
$M[A, \$] = A \rightarrow \alpha$	

Ø Problem 1:

» Calculate the first and follow functions for the given grammar- also construct parsing table.

$$\begin{aligned}
 E &\rightarrow T \ E' \\
 E' &\rightarrow + \ T \ E' \mid \epsilon \\
 T &\rightarrow F \ T' \\
 T' &\rightarrow * \ F \ T' \mid \epsilon \\
 F &\rightarrow (\ E) \mid id
 \end{aligned}$$

Solutions :

FIRST:		FOLLOW:						
$FIRST(E) = \{(, id\}$		$FOLLOW(E) = \{\$,)\}$						
$FIRST(E') = \{+, \epsilon\}$		$FOLLOW(E') = \{\$,)\}$						
$FIRST(T) = \{(, id\}$		$FOLLOW(T) = \{+, \$,)\}$						
$FIRST(T') = \{*, \epsilon\}$		$FOLLOW(T') = \{+, \$,)\}$						
$FIRST(F) = \{(, id\}$		$FOLLOW(F) = \{*, +, \$,)\}$						

Parsing Table.								
FIRST	FOLLOW	Nonterminal	+	*	()	id	\$
{(, id }	{\$,)}	E			$E \rightarrow TE'$		$E \rightarrow TE'$	
{+, ε }	{\$,)}	E'	$E' \rightarrow + TE'$			$E' \rightarrow \epsilon$		$E' \rightarrow \epsilon$
{(, id }	{+, \$,)}	T			$T \rightarrow FT'$		$T \rightarrow FT'$	
{*, ε }	{+, \$,)}	T'	$T' \rightarrow \epsilon$	$T' \rightarrow * FT'$		$T' \rightarrow \epsilon$		$T' \rightarrow \epsilon$
{(, id }	{*, +, \$,)}	F			$F \rightarrow (E)$		$F \rightarrow id$	

Ø Problem 2:

» Calculate the first and follow functions for the given grammar- also construct parsing table.

$$\begin{aligned}
 S &\rightarrow A \\
 A &\rightarrow aB \mid Ad \\
 B &\rightarrow bBC \mid f \\
 C &\rightarrow g
 \end{aligned}$$

Solutions :

Eliminate the left recursive for the given grammar	
Before Elimination:	After Elimination:
$S \rightarrow A$ $A \rightarrow aB \mid Ad$ $B \rightarrow bBC \mid f$ $C \rightarrow g$	$S \rightarrow A$ $A \rightarrow aBA'$ $A' \rightarrow dA' \mid \epsilon$ $B \rightarrow bBC \mid f$ $C \rightarrow g$

FIRST:		FOLLOW:						
$\text{FIRST}(S) = \{a\}$		$\text{FOLLOW}(S) = \{\$\}$						
$\text{FIRST}(A) = \{a\}$		$\text{FOLLOW}(A) = \{\$\}$						
$\text{FIRST}(A') = \{d, \in\}$		$\text{FOLLOW}(A') = \{\$\}$						
$\text{FIRST}(B) = \{b, f\}$		$\text{FOLLOW}(B) = \{d, \$, g\}$						
$\text{FIRST}(C) = \{g\}$		$\text{FOLLOW}(C) = \{d, \$, g\}$						
Parsing Table.								
FIRST	FOLLOW	Nonterminal	a	b	d	f	g	\$
{a}	{\\$}	S	$S \rightarrow A$					
{a}	{\\$}	A	$A \rightarrow aBA'$					
{d, ∈}	{\\$}	A'			$A' \rightarrow dA'$			$A' \rightarrow \in$
{b,f}	{d, \$, g}	B		$B \rightarrow bBC$		$B \rightarrow f$		
{g}	{d, \$, g}	C					$C \rightarrow g$	

Problem 3:

➤ Calculate the first and follow functions for the given grammar- also construct parsing table.

$$\begin{aligned}
 E &\rightarrow TE' \\
 E' &\rightarrow +E \mid \epsilon \\
 T &\rightarrow FT' \\
 T' &\rightarrow T \mid \epsilon \\
 F &\rightarrow PF' \\
 F' &\rightarrow *F' \mid \epsilon \\
 P &\rightarrow (E) \mid a \mid b \mid \text{ep}
 \end{aligned}$$

Solutions :

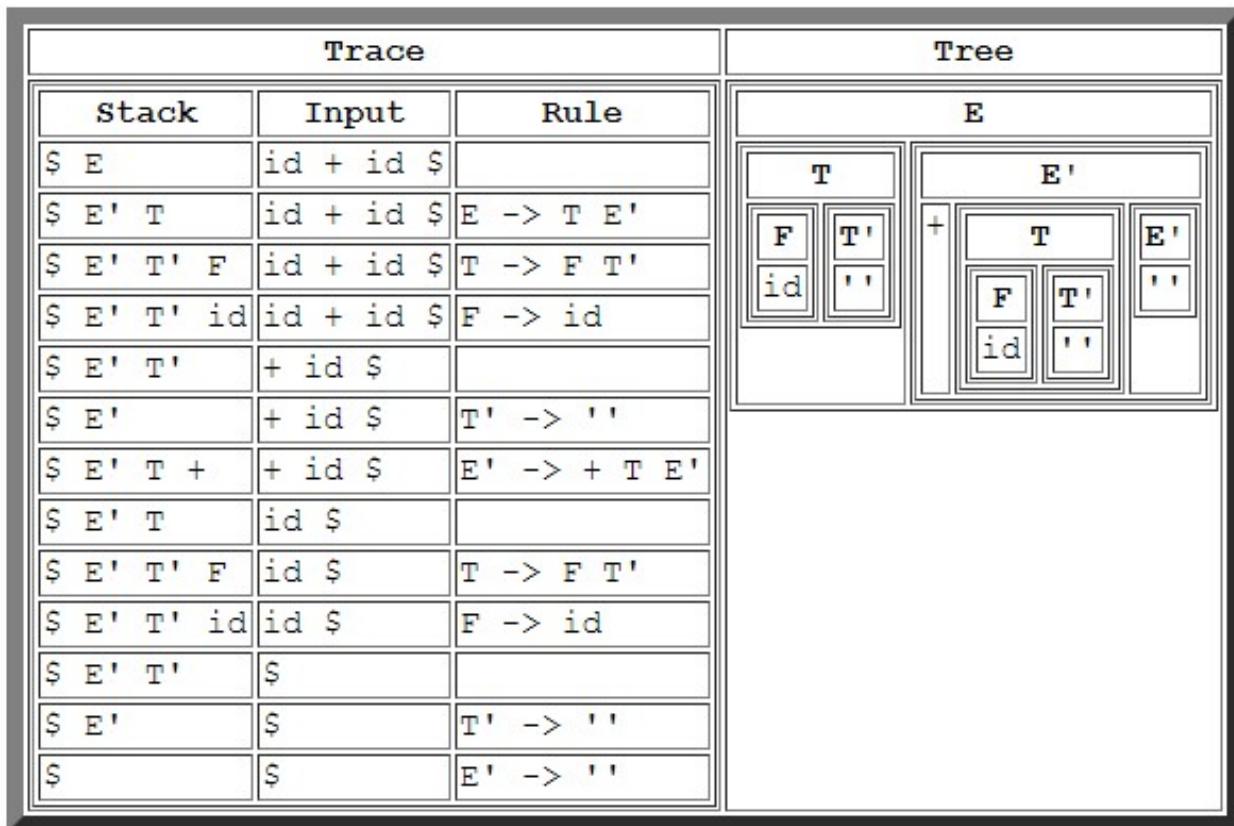
FIRST:		FOLLOW:	
$\text{FIRST}(E) = \{(\, , a, b, \text{ep}\}$		$\text{FOLLOW}(E) = \{\$, \,)\}$	
$\text{FIRST}(E') = \{+, \in\}$		$\text{FOLLOW}(E') = \{\$, \,)\}$	
$\text{FIRST}(T) = \{(\, , a, b, \text{ep}\}$		$\text{FOLLOW}(T) = \{+, \$, \,)\}$	
$\text{FIRST}(T') = \{(\, , a, b, \text{ep}, \in\}$		$\text{FOLLOW}(T') = \{+, \$, \,)\}$	
$\text{FIRST}(F) = \{(\, , a, b, \text{ep}\}$		$\text{FOLLOW}(F) = \{(\, , a, b, \text{ep}, +, \$, \,)\}$	
$\text{FIRST}(F') = \{*, \in\}$		$\text{FOLLOW}(F') = \{(\, , a, b, \text{ep}, +, \$, \,)\}$	
$\text{FIRST}(P) = \{(\, , a, b, \text{ep}\}$		$\text{FOLLOW}(P) = \{*, (\, , a, b, \text{ep}, +, \$, \,)\}$	

Parsing Table.

FIRST	FOLLOW	Non Terminal	()	a	b	ep	+	*	\$
{(, a, b, ep}	{\$,)}	E	$E \rightarrow T E'$		$E \rightarrow T E'$	$E \rightarrow T E'$	$E \rightarrow T E'$			
{+, ε }	{\$,)}	E'		$E' \rightarrow \epsilon$				$E' \rightarrow + E$		$E' \rightarrow \epsilon$
{(, a, b, ep}	{+, \$,)}	T	$T \rightarrow F T'$		$T \rightarrow F T'$	$T \rightarrow F T'$	$T \rightarrow F T'$			
{(, a, b, ep, ε}	{+, \$,)}	T'	$T' \rightarrow T$	$T' \rightarrow \epsilon$	$T' \rightarrow T$	$T' \rightarrow T$	$T' \rightarrow T$	$T' \rightarrow \epsilon$		$T' \rightarrow \epsilon$
{(, a, b, ep}	{(, a, b, ep, +, \$,)}	F	$F \rightarrow PF'$		$F \rightarrow PF'$	$F \rightarrow PF'$	$F \rightarrow PF'$			
{*, ε }	{(, a, b, ep, +, \$,)}	F'	$F' \rightarrow \epsilon$	$F' \rightarrow *F'$	$F' \rightarrow \epsilon$					
{(, a, b, ep}	{*, (, a, b, ep, +, \$,)}	P	$P \rightarrow (E)$		$P \rightarrow a$	$P \rightarrow b$	$P \rightarrow ep$			

Stack Movement a Predictive Parser (Example 1)

Given input String: id + id



Problem 4:

» Check the following context free grammar whether it is LL(1) or not

$$\begin{aligned} S &\rightarrow A \mid B \\ A &\rightarrow A \ a \mid B \ b \\ B &\rightarrow b \mid \epsilon \end{aligned}$$

Solutions :

We know a grammar to be **LL(1)**:

1. If it must **not have left recursion**.
2. FIRST sets of productions must be **disjoint**.
3. If a production has ϵ , **FIRST (non-terminal) \cap FOLLOW (non-terminal) = \emptyset** .

Rule1: Check left recursion	Rule2: Check FIRST sets of multiple productions
$A \rightarrow A \ a$	FIRST(S) = {b, b, ϵ } FIRST(A) = {b} (ignoring recursion for a moment) FIRST(B) = {b, ϵ }
direct left recursion	The FIRST sets intersect
Rule3: If a production has ϵ , FIRST (non-terminal) \cap FOLLOW (non-terminal) = \emptyset	
FOLLOW: $B \rightarrow \epsilon \rightarrow \epsilon$ -production	So , This grammar violates all LL(1) rules:
FOLLOW(B) = {\$, b}	1. Contains left recursion 2. Has intersecting FIRST sets 3. Has ϵ -production conflicting with FOLLOW
FIRST(B) \cap FOLLOW(B) $\neq \emptyset$	So, grammar is not LL(1)

Problem 5:

» Check the following context free grammar whether it is LL(1) or not

$$\begin{aligned} X &\rightarrow aZbXY \mid c \\ Y &\rightarrow dX \mid \epsilon \\ Z &\rightarrow e \end{aligned}$$

Solutions :

We know a grammar to be **LL(1)**:

1. If it must **not have left recursion**.
2. FIRST sets of productions must be **disjoint**.
3. If a production has ϵ , **FIRST (non-terminal) \cap FOLLOW (non-terminal) = \emptyset** .

Rule1: Check left recursion	Rule2: Check FIRST sets of multiple productions
No left recursion.	$\text{FIRST}(X) = \{a, c\}$ $\text{FIRST}(Y) = \{d, \epsilon\}$ $\text{FIRST}(Z) = \{e\}$ FIRST sets of its alternative productions are pairwise disjoint.
Rule3: If a production has ϵ , $\text{FIRST}(\text{non-terminal}) \cap \text{FOLLOW}(\text{non-terminal}) = \emptyset$	
FOLLOW: $Y \rightarrow \epsilon \rightarrow \epsilon\text{-production}$	So, This grammar not justify all LL(1) rules:
$\text{FOLLOW}(Y) = \{\$, d\}$	1. No left recursion. 2. NO intersecting FIRST sets 3. $\epsilon\text{-production conflicting with FOLLOW}$
$\text{FIRST}(A) \cap \text{FOLLOW}(A) = \{d\} \neq \emptyset$	So, grammar is not LL(1)

OR

$$\begin{aligned} X &\rightarrow aZbXY \mid c \\ Y &\rightarrow dX \mid \epsilon \\ Z &\rightarrow e \end{aligned}$$

Solutions :

FIRST:	FOLLOW:
$\text{FIRST}(X) = \{a, c\}$	$\text{FOLLOW}(X) = \{\$, d\}$
$\text{FIRST}(Y) = \{d, \epsilon\}$	$\text{FOLLOW}(Y) = \{\$, d\}$
$\text{FIRST}(Z) = \{e\}$	$\text{FOLLOW}(Z) = \{b\}$

Parsing Table.

FIRST	FOLLOW	Nonterminal	a	b	c	d	e	\$
{a, c}	{\$, d}	X	X \rightarrow aZbXY	X \rightarrow c				
{d, ε}	{\$, d}	Y				Y \rightarrow dX, ε		Y \rightarrow ε
{e}	{b}	Z					Z \rightarrow e	

Since a cell has more than one entry that means there is a parsing conflict, so the given grammar is not LL1

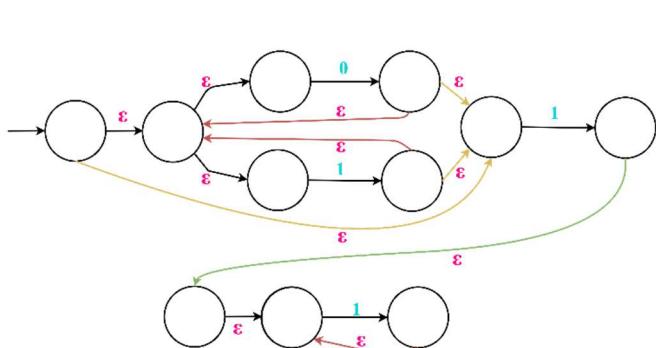
Practice Problem



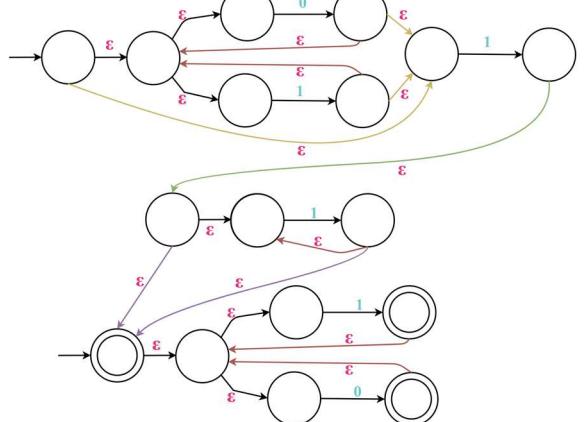
Problem1: Draw the NFA for the following regular expression “ $(0|1)^*11^*(1|0)^*1$ ”.

Step 1: Parse the RE into Subexpressions	Step 2: Build NFAs for Basic Symbols	
» The RE “$(0 1)^*11^*(1 0)^*1$” can be decomposed as: <ol style="list-style-type: none"> 1. $(0 1)^*$ 2. $(1 0)^*$ 3. 1^* 4. 0 5. 1 <p>We'll build NFAs for each component and then add them.</p>	NFA for 0	NFA for 1
Step 4: NFA for $(1 0)$ (Union)		Step 3: NFA for $(0 1)$ (Union)
Step 5: NFA for “$(0 1)^*$” (Star)		Step 6: NFA for “$(1 0)^*$” (Star)
Step 7: NFA for “1^*” (Star)		Step 8: Combine “$(0 1)^*1$”. (Concatenation)

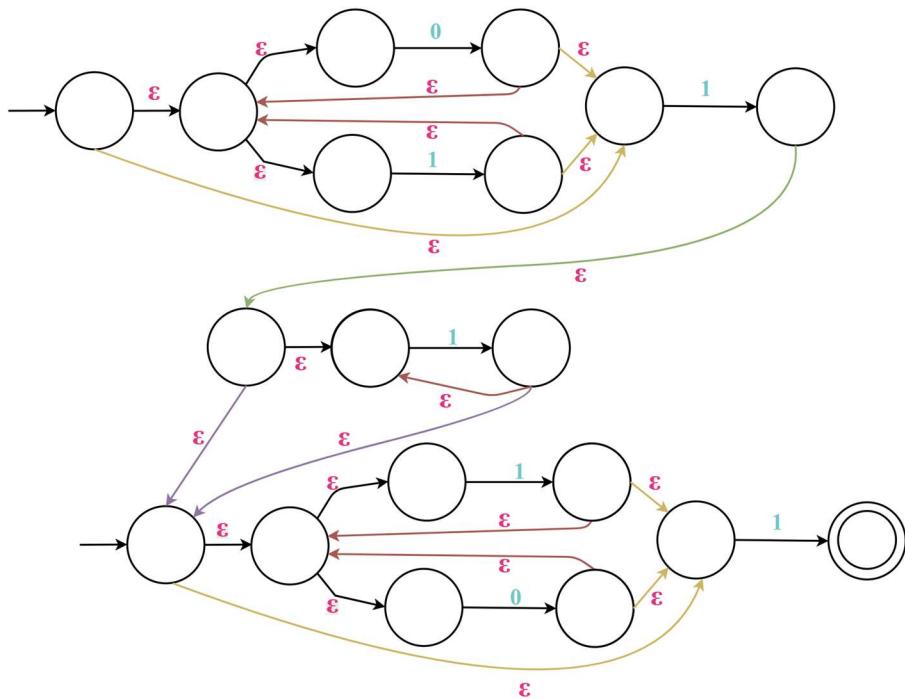
Step 9: Combine “ $(0|1)^*11^*$ ”. (Concatenation)



Step 10: Combine “ $(0|1)^*11^*(1|0)^*$ ”(Concatenation)



Step 11: Combine “ $((0|1)^*11^*(1|0)^*1$ ” (Concatenation)



 **Problem2:** a. Find the FIRST and FOLLOW function from the following grammar.

```

S → FTc
T → id = A*
A → FB
B → +FB | ε
F → int | a | b

```

b. Construct the parsing table for the context free grammar given in problem 2a and show the moves made by the predictive parser for the string: **int id = a + b* c**

(a)

Solutions :

FIRST:		FOLLOW:	
FIRST(S) = {int, a, b }		FOLLOW(S) = { \$ }	
FIRST(T) = {id }		FOLLOW(T) = {c}	
FIRST(A) = {int, a, b }		FOLLOW(A) = { * }	
FIRST(B) = {+, ∈ }		FOLLOW(B) = { * }	
FIRST(F) = {int, a, b }		FOLLOW(F) = {id, +, * }	

(b)

Solutions :

Parsing Table.												
FIRST	FOLLOW	Non Terminal	a	b	c	int	id	=	+	*	\$	
{int, a, b }	{\$}	S	S → FTc	S → FTc		S → FTc						
{id}	{c}	T						T → id = A*				
{int, a, b }	{*}	A	A → FB	A → FB		A → FB						
{+, ∈ }	{*}	B							B → +FB	B → ε		
{int, a, b }	{id, +, * }	F	F → a	F → b		F → int						

Stack Movement a Predictive Parser

Given input String: int id = a + b* c

Trace		
Stack	Input	Rules
\$ S	int id = a + b* c \$	
\$ c T F	int id = a + b* c \$	S → FTc
\$ c T int	int id = a + b* c \$	F → int
\$ c T	id = a + b* c \$	
\$ c * A = id	id = a + b* c \$	T → id = A*
\$ c * A =	= a + b* c \$	
\$ c * A	a + b* c \$	
\$ c * B F	a + b* c \$	A → FB
\$ c * B a	a + b* c \$	F → a
\$ c * B	+ b* c \$	
\$ c * B F +	+ b* c \$	B → +FB
\$ c * B F	b* c \$	
\$ c * B b	b* c \$	F → b
\$ c * B	* c \$	
\$ c *	* c \$	B → ε
\$ c	c \$	
\$	\$	



Problem3: Find the FIRST and FOLLOW set from the following grammar.

$$\begin{aligned}
 E &\rightarrow TX \\
 X &\rightarrow +E \mid \epsilon \\
 T &\rightarrow \text{int } Y \mid (E) \\
 Y &\rightarrow *T \mid \epsilon
 \end{aligned}$$

Solutions:

FIRST:	FOLLOW:
FIRST(E) = {int, (}	FOLLOW(E) = {\$,)}
FIRST(X) = {+, ∈ }	FOLLOW(X) = {\$,)}
FIRST(T) = {int, (}	FOLLOW(T) = {+, \$,)}
FIRST(Y) = {* , ∈ }	FOLLOW(Y) = {+, \$,)}



Problem4: Find the FIRST and FOLLOW set from the following grammar.

$$\begin{aligned}
 S &\rightarrow x \ U \ y \\
 U &\rightarrow M \ V \\
 V &\rightarrow M \ V \mid \epsilon \\
 M &\rightarrow \text{num} \ * \ E \\
 E &\rightarrow H \ W \\
 W &\rightarrow / \ H \ W \mid \epsilon \\
 H &\rightarrow (E) \mid \text{num} \mid x \mid y
 \end{aligned}$$

Solutions:

FIRST:	FOLLOW:
$\text{FIRST}(S) = \{x\}$	$\text{FOLLOW}(S) = \{\$\}$
$\text{FIRST}(U) = \{\text{num}\}$	$\text{FOLLOW}(U) = \{y\}$
$\text{FIRST}(V) = \{\text{num}, \in\}$	$\text{FOLLOW}(V) = \{y\}$
$\text{FIRST}(M) = \{\text{num}\}$	$\text{FOLLOW}(M) = \{\text{num}, y\}$
$\text{FIRST}(E) = \{(), \text{num}, x, y\}$	$\text{FOLLOW}(E) = \{\text{num}, y, ()\}$
$\text{FIRST}(H) = \{/ , \in\}$	$\text{FOLLOW}(W) = \{\text{num}, y, ()\}$
$\text{FIRST}(W) = \{(), \text{num}, x, y\}$	$\text{FOLLOW}(H) = \{(), \text{num}, x, y\}$

Problem 5:

» In the context of compiler design, explain the significance of estimating the FIRST and FOLLOW sets for a given Context-Free Grammar (CFG)

➤ **Significance of FIRST and FOLLOW sets in Compiler Design:**

- **FIRST set:** Shows which terminals can appear first from a symbol/production; used to decide which production rule to apply.
- **FOLLOW set:** Shows which terminals can appear immediately after a non-terminal; important when ϵ -productions are present.
- Together, they are used to **construct LL(1) parsing tables**, detect conflicts, and ensure unambiguous predictive parsing.