# CSL253 - Theory of Computation

Tutorial 2


**Team Members**

1. Kartikeya Nainakhwal - 12341090
2. Paritosh Lahre - 12341550
3. Rahul Dev Reddy - 12342390


**Question 2**

Give state diagrams of NFAs with the specified number of states recognizing each of the following languages. In all parts the alphabet is {0,1}.

1. The language {$\omega$ | $\omega$ ends with 001 with three states}
2. The language {$\omega$ | $\omega$ contains the substring 0101, i.e., w = x0101y for some x and y} with five states
3. The language {$\omega$ | $\omega$ contains an even number of 0s, or contains exactly two 1s} with six states
4. The language {0} with two states
5. The language 0*1*0+ with three states
6. The language 1*(001+)* with three states
7. The language {e} with one state
8. The language 0* with one state

# Solution

## 1. Accepts strings ends with 001

### Language

$A_1 = \{\omega \mid \omega \text{ ends with 001 with three states}\}$

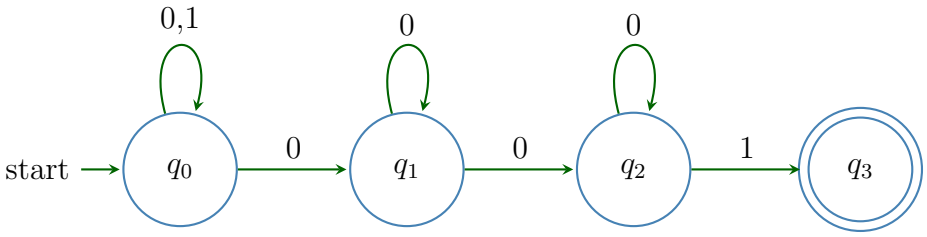
## Issue with the Problem Statement

The problem statement requires constructing an NFA with only three states to recognize strings ending in "001." However, a three-state NFA cannot correctly enforce this condition. The minimal correct NFA for this language requires at least four states:

- One state for tracking the start.
- One state for detecting the first '0'.
- One state for detecting '00'.
- One state for accepting upon encountering '001'.

If we restrict the NFA to three states, it would either fail to distinguish between required patterns or accept incorrect strings, such as "01." Thus, the problem should either specify four states or redefine the accepted strings.

## Corrected NFA Diagram (Four States)




## Defining the NFA

We define the NFA as a five-tuple $M_1 = (Q_1, \Sigma, \delta_1, q_0, F_1)$ where:

- **States:** $Q_1 = \{q_0, q_1, q_2, q_3\}$, where:
  - $q_0$: Initial state
  - $q_1$: After first '0'
  - $q_2$: After '00'

- $q_3$: Accepting state (after '001')

- **Alphabet:** $\Sigma = \{0, 1\}$
- **Transition Function:** $\delta_1 : Q_1 \times \Sigma \to Q_1$

$$\delta(\textit{current state}, \textit{input}) = \textit{next state}$$

The transitions are defined as follows:

$$\begin{aligned}
\delta_1(q_0, 0) &= q_0, q_1 \\
\delta_1(q_0, 1) &= q_0 \\
\delta_1(q_1, 0) &= q_1, q_2 \\
\delta_1(q_2, 0) &= q_3 \\
\delta_1(q_2, 1) &= q_3
\end{aligned}$$

- **Initial State:** $q_0$
- **Acceptance States:** $F_1 = \{q_3\}$

## Transition Table

| **Current State** | **Input '0'** | **Input '1'** |
|---|---|---|
| $q_0$ | $q_0, q_1$ | $q_0$ |
| $q_1$ | $q_1, q_2$ | $-$ |
| $q_2$ | $q_2$ | $q_3$ |

## 2. Accepts strings which contains substring 0101 using five states

### Language

$A_2 = \{\omega \mid \omega$ contains the substring 0101, i.e., w = x0101y for some x and y$\}$

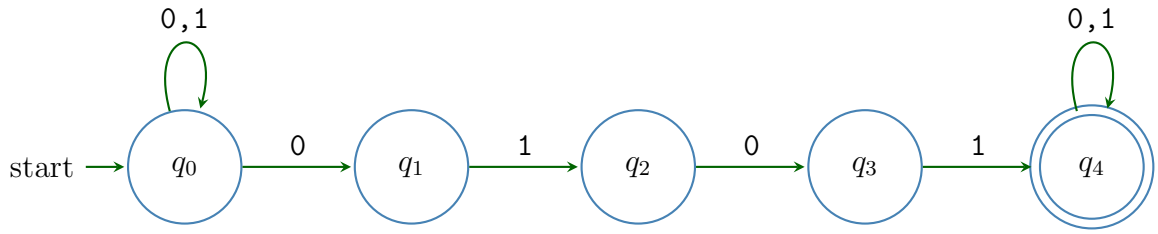
### Explanation

This NFA is designed to recognize binary strings that contain the substring '0101'. It consists of five states:

- The automaton starts at $q_0$ and moves through states $q_1$, $q_2$, $q_3$, and finally $q_4$ as it reads the pattern '0101'.
- If the pattern '0101' appears anywhere in the input, the automaton reaches the accepting state $q_4$.
- Once in $q_4$, the automaton remains there for all further inputs, ensuring acceptance.
- Any other sequence of '0's and '1's keeps the automaton in its non-accepting states.

This automaton effectively captures the required substring using only five states, demonstrating an efficient finite state approach.

### NFA Diagram




### Defining the NFA

We define the NFA as a five-tuple $M_1 = (Q_1, \Sigma, \delta_1, q_0, F_1)$ where:

- **States:** $Q_1 = \{q_0, q_1, q_2, q_3, q_4\}$, where:
  - $q_0$: Initial state
  - $q_1$: Recognizing '0'
  - $q_2$: Recognizing '01'
  - $q_3$: Recognizing '010'
  - $q_4$: Recognizing '0101' (accepting state)
- **Alphabet:** $\Sigma = \{0, 1\}$

- **Transition Function:** $\delta_1 : Q_1 \times \Sigma \to Q_1$

$$\boxed{\delta(\text{current state}, \text{input}) = \text{next state}}$$

The transitions are defined as follows:

$$\begin{aligned}
\delta_1(q_0, 0) &= q_1 \\
\delta_1(q_0, 1) &= q_0 \\
\delta_1(q_1, 1) &= q_2 \\
\delta_1(q_2, 0) &= q_3 \\
\delta_1(q_3, 1) &= q_4 \\
\delta_1(q_4, 0) &= q_4 \\
\delta_1(q_4, 1) &= q_4
\end{aligned}$$

**Transition Table**

| **Current State** | **Input '0'** | **Input '1'** |
|---|---|---|
| $q_0$ | $q_1$ | $q_0$ |
| $q_1$ | - | $q_2$ |
| $q_2$ | $q_3$ | - |
| $q_3$ | - | $q_4$ |
| $q_4$ | $q_4$ | $q_4$ |

## 3. Accepts strings which contains an even number of 0s or exactly two 1s

### Language

$A_1 = \{\omega \mid \omega$ contains an even number of 0s, or contains exactly two 1s$\}$

### Explanation

The given problem requires constructing an NFA that accepts strings satisfying one of the two conditions:

- The string contains an even number of 0s.
- The string contains exactly two 1s.

To solve this, we break the problem into two individual NFAs:

1. An NFA that accepts strings with an even number of 0s.
2. An NFA that accepts strings containing exactly two 1s.

Finally, we construct the union of these two NFAs using epsilon transitions, forming a new NFA that accepts a string if it satisfies either of the two conditions.

### Even Number of 0s

This NFA maintains a state to track the parity of the number of 0s seen so far:

- The initial state is accepting, as a string with zero 0s is valid.
- Each occurrence of a 0 toggles the state between even and odd.
- The NFA remains in its current state when encountering a 1.

### Exactly Two 1s

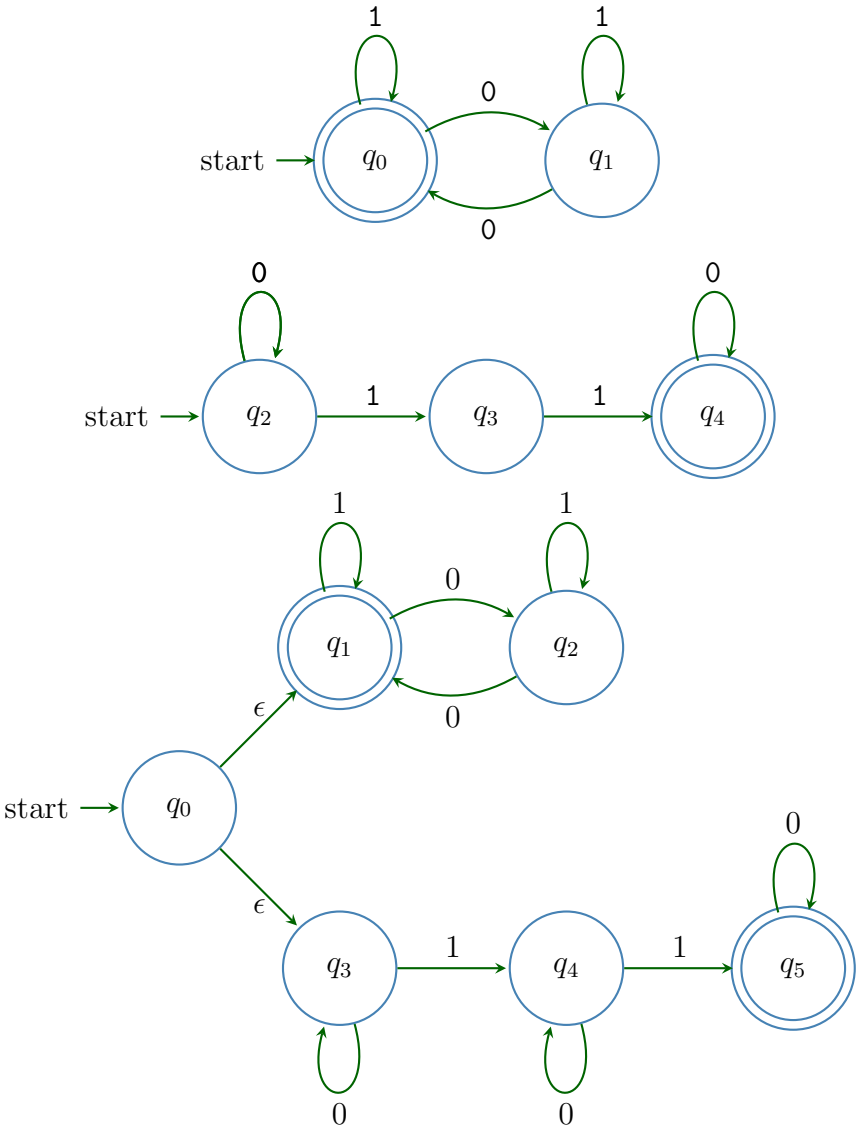
This NFA tracks the number of 1s in the input:

- It starts in an initial state that transitions upon encountering a 1.
- After two 1s are received, the NFA moves to an accepting state.
- Any additional 1s push the NFA into a dead state.
- The NFA remains in its current state for 0s.

## Union of the Two NFAs

To combine the two NFAs into a single NFA, we introduce a new initial state with epsilon transitions to the start states of both NFAs. This allows the new NFA to nondeterministically choose which condition to track for any given input string. The final NFA correctly recognizes any string that meets either of the conditions.

## NFA Diagram




## Defining the NFA

We define the NFA as a five-tuple $M_1 = (Q_1, \Sigma, \delta_1, q_0, F_1)$ where:

- **States:** $Q_1 = \{q_0, q_1, q_2, q_3, q_4, q_5\}$, where:
  - $q_0$: Initial state

- $q_1$: '1' state
- $q_2$: Intermediate state
- $q_3$: '0' state
- $q_4$: Intermediate state
- $q_5$: Accepting state

- **Alphabet:** $\Sigma = \{0, 1\}$
- **Transition Function:** $\delta_1 : Q_1 \times (\Sigma \cup \{\varepsilon\}) \to 2^{Q_1}$

$$\delta_1(\text{current state}, \text{input}) = \text{next state(s)}$$

The transitions are defined as follows:

$$\begin{aligned}
\delta_1(q_0, \varepsilon) &= \{q_1, q_3\} \\
\delta_1(q_1, 0) &= \{q_2\} \\
\delta_1(q_1, 1) &= \{q_1\} \\
\delta_1(q_2, 0) &= \{q_1\} \\
\delta_1(q_2, 1) &= \{q_2\} \\
\delta_1(q_3, 0) &= \{q_3\} \\
\delta_1(q_3, 1) &= \{q_4\} \\
\delta_1(q_4, 0) &= \{q_4\} \\
\delta_1(q_4, 1) &= \{q_5\} \\
\delta_1(q_5, 0) &= \{q_5\} \\
\delta_1(q_5, 1) &= \{q_5\}
\end{aligned}$$

**Transition Table**

| **Current State** | **Input '0'** | **Input '1'** |
|---|---|---|
| $q_0$ | $q_1, q_3$ ($\varepsilon$-transition) | - |
| $q_1$ | $q_2$ | $q_1$ |
| $q_2$ | $q_1$ | $q_2$ |
| $q_3$ | $q_3$ | $q_4$ |
| $q_4$ | $q_4$ | $q_5$ |
| $q_5$ | $q_5$ | $q_5$ |

## 4. Accepts string 0 only using two states

### Regular Language

$A_1 = \{\omega \mid \omega \text{ is a string which contain only } 0\}$

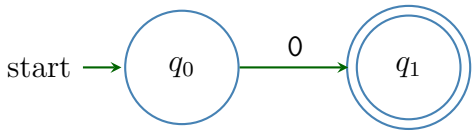
### Explanation

This finite automaton is designed to accept only the string "0" and reject all other strings, including the empty string and any string containing "1". The automaton consists of two states:

- $q_0$: The initial state.
- $q_1$: The accepting state, reached only when the input is exactly "0".

If the automaton reads the input "0", it transitions from $q_0$ to $q_1$, which is an accepting state. Any other input (including "1" or multiple characters) is not handled by a transition, effectively rejecting them.

### NFA Diagram




### Defining the NFA

We define the NFA as a five-tuple $M_1 = (Q_1, \Sigma, \delta_1, q_0, F_1)$ where:

- **States:** $Q_1 = \{q_0, q_1\}$, where:
  - $q_0$: Initial state
  - $q_1$: '0' state
- **Alphabet:** $\Sigma = \{0, 1\}$
- **Transition Function:** $\delta_1 : Q_1 \times \Sigma \to Q_1$

$$\delta(\textit{current state}, \textit{input}) = \textit{next state}$$

The transitions are defined as follows:

$$\delta_1(q_0, 0) = q_1$$

- **Initial State:** $q_0$
- **Acceptance States:** $F_1 = \{q_1\}$

**Transition Table**

| Current State | Input '0' | Input '1' |
|---|---|---|
| $q_0$ | $q_1$ | - |

## 5. The language $\{0^*1^*0+\}$ with three states

### Language

$A_1 = 0^*1^*0+$ = This language consists of strings that start with any number of '0's, followed by any number of '1's, and end with at least one '0'.

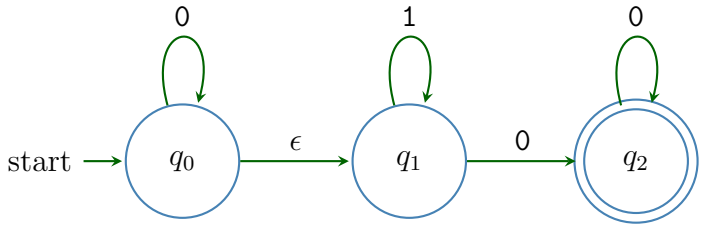
### Explanation

The given language consists of:

- Any number of '0's at the beginning (including none at all).
- Followed by any number of '1's (including none at all).
- Ending with at least one '0', making it a necessary condition.

The purpose of this NFA is to efficiently recognize this pattern with a minimal number of states.

### NFA Diagram




### Defining the NFA

We define the NFA as a five-tuple $M = (Q, \Sigma, \delta, q_0, F)$ where:

- **States:** $Q = \{q_0, q_1, q_2\}$, where:
  - $q_0$: Initial state (handles leading '0's)
  - $q_1$: Intermediate state (handles '1's)
  - $q_2$: Accepting state (ensures at least one 0 at the end)
- **Alphabet:** $\Sigma = \{0, 1\}$
- **Transition Function:** $\delta : Q \times \Sigma \to Q$

$$\delta(current\ state, input) = next\ state$$

The transitions are defined as follows:

$$\delta(q_0, 0) = q_0$$
$$\delta(q_0, \epsilon) = q_1$$
$$\delta(q_1, 1) = q_1$$
$$\delta(q_1, 0) = q_2$$
$$\delta(q_2, 0) = q_2$$

- **Initial State:** $q_0$
- **Acceptance States:** $F = \{q_2\}$

**Transition Table**

| **Current State** | **Input '0'** | **Input '1'** |
|---|---|---|
| $q_0$ | $q_0$ | - |
| $q_1$ | $q_2$ | $q_1$ |
| $q_2$ | $q_2$ | - |

## 6. The language $1^*(001+)^*$ with three states

### Regular Language

$A_1 = 1^*(001+)^*$ = The language consists of strings that may contain any number of 1's followed by at least one occurrence of '001', which can repeat any number of times.

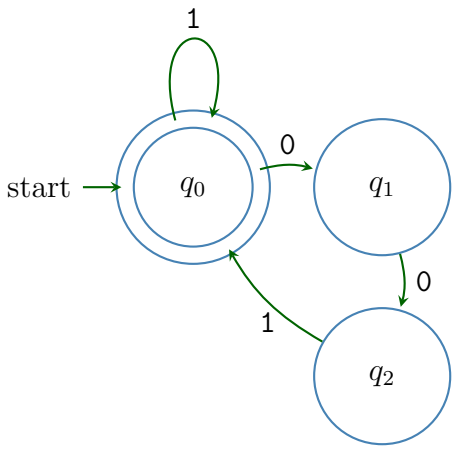
### Explanation

This NFA accepts strings that consist of any number of 1s followed by at least one occurrence of '001', which can repeat. Here's how it works:

- The automaton starts in state $q_0$.
- While reading '1', it remains in $q_0$.
- Upon encountering a '0', it transitions to $q_1$.
- A second '0' moves it to $q_2$.
- A '1' from $q_2$ sends it back to $q_0$, completing one cycle of '001'.
- The accepting state is $q_0$, ensuring at least one full '001' pattern is read for a valid string.

This minimal NFA with three states efficiently captures the essence of the given language.

### NFA Diagram




### Defining the NFA

We define the NFA as a five-tuple $M = (Q, \Sigma, \delta, q_0, F)$ where:

- **States:** $Q = \{q_0, q_1, q_2\}$

- $q_0$: Initial and accepting state
- $q_1$: Intermediate state after reading 0
- $q_2$: Intermediate state after reading 00

- **Alphabet:** $\Sigma = \{0, 1\}$
- **Transition Function:** $\delta : Q \times \Sigma \to Q$

$$\delta(\textit{current state}, \textit{input}) = \textit{next state}$$

The transitions are defined as follows:

$$\begin{aligned}
\delta(q_0, 1) &= q_0 \\
\delta(q_0, 0) &= q_1 \\
\delta(q_1, 0) &= q_2 \\
\delta(q_2, 1) &= q_0
\end{aligned}$$

- **Initial State:** $q_0$
- **Acceptance States:** $F = \{q_0\}$

**Transition Table**

| **Current State** | **Input '0'** | **Input '1'** |
|---|---|---|
| $q_0$ | $q_1$ | $q_0$ |
| $q_1$ | $q_2$ | - |
| $q_2$ | - | $q_0$ |

## 7. The language $\{\epsilon\}$ with one state

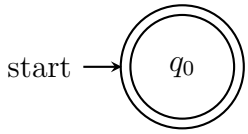
### Regular Language

$A_1 = \{\epsilon\}$ — This language consists of only the empty string $\epsilon$, meaning it contains no actual symbols.

### Explanation

This NFA represents the language $\{\epsilon\}$, which contains only the empty string. The automaton consists of a single state, $q_0$, which serves as both the initial and accepting state. Since the language contains no actual symbols, the alphabet is empty $\Sigma = \emptyset$, meaning there are no transitions. The automaton accepts the empty string by default because the initial state is also an accepting state. Any input other than the empty string is not part of this language, making this the simplest possible NFA.

### NFA Diagram




### Defining the NFA

We define the NFA as a five-tuple $M = (Q, \Sigma, \delta, q_0, F)$ where:

- **States:** $Q = \{q_0\}$, where:
  - $q_0$: Initial and accepting state
- **Alphabet:** $\Sigma = \emptyset$ (since no input symbols are required)
- **Transition Function:** $\delta : Q \times \Sigma \to Q$ (no transitions needed)
- **Initial State:** $q_0$
- **Acceptance States:** $F = \{q_0\}$

### Transition Table

| **Current State** |
|---|
| $q_0$ (accepting) |

## 8. The language $0^*$ with one state

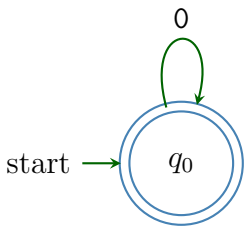
### Regular Language

$A_1 = 0^* = \{\omega \mid \omega$ is language consists of all strings made up of zero or more occurrences of 0, including the empty string.$\}$

### Explanation

This NFA represents the language $0^*$, which consists of any number of '0's, including the empty string. The automaton has only one state, $q_0$, which serves as both the initial and accepting state. The transition function ensures that any input '0' leads back to $q_0$, allowing for repetition of '0's indefinitely while remaining in an accepting state. Since the initial state is also an accepting state, the empty string is accepted by default.

### NFA Diagram




### Defining the NFA

We define the NFA as a five-tuple $M = (Q, \Sigma, \delta, q_0, F)$ where:

- **States:** $Q = \{q_0\}$, where:
  - $q_0$: Initial and accepting state
- **Alphabet:** $\Sigma = \{0\}$
- **Transition Function:** $\delta : Q \times \Sigma \to Q$

$$\delta(q_0, 0) = q_0$$

- **Initial State:** $q_0$
- **Acceptance States:** $F = \{q_0\}$

### Transition Table

| Current State | Input '0' |
|---|---|
| $q_0$ | $q_0$ |

**Question 4**

Construct the state diagrams of NFAs recognizing the union of the languages described in:

a. $\{w \mid w \text{ contains the substring } 0101, \text{ i.e., } w = x0101y \text{ for some } x \text{ and } y\}$

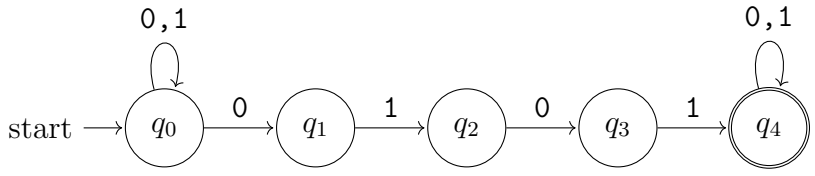
b. $\{w \mid w \text{ does not contain the substring } 1101\}$

# Solution

## Step 1: Construct NFA for Language (a)

Language (a) is $\{w \mid w \text{ contains the substring } 0101\}$.

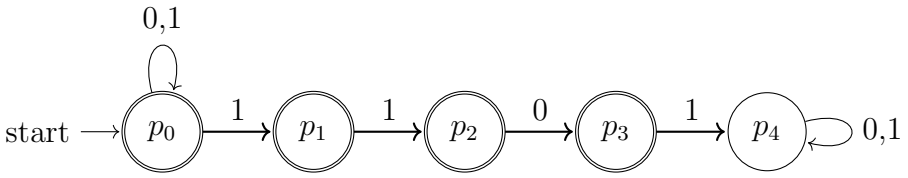
- **States**:
  - $q_0$: Start state (no part of 0101 recognized yet).
  - $q_1$: Recognized '0'.
  - $q_2$: Recognized '01'.
  - $q_3$: Recognized '010'.
  - $q_4$: Recognized '0101' (accepting state).
- **Transitions**:
  - From $q_0$, on input '0', go to $q_0$,$q_1$.
  - From $q_0$, on input '1', stay at $q_0$.
  - From $q_1$, on input '1', go to $q_2$.
  - From $q_2$, on input '0', go to $q_3$.
  - From $q_3$, on input '1', go to $q_4$.
  - From $q_4$, on any input, stay in $q_4$ (accepting state).
- **Accepting State**: $q_4$.



## Step 2: Construct NFA for Language (b)

Language (b) is $\{w \mid w \text{ does not contain the substring } 1101\}$.

- **States**:
  - $p_0$: Start state (no part of 1101 recognized yet).
  - $p_1$: Recognized '1'.
  - $p_2$: Recognized '11'.
  - $p_3$: Recognized '110'.
  - $p_4$: Recognized '1101' (trap state, non-accepting).
- **Transitions**:
  - From $p_0$, on input '1', go to $p_0$,$p_1$.
  - From $p_0$, on input '0', stay at $p_0$.
  - From $p_1$, on input '1', go to $p_2$.
  - From $p_1$, on input '0', return to $p_0$ (sequence broken).
  - From $p_2$, on input '0', go to $p_3$.
  - From $p_2$, on input '1', stay at $p_2$.
  - From $p_3$, on input '1', go to $p_4$ (trap state).
  - From $p_3$, on input '0', return to $p_0$ (sequence broken).
  - From $p_4$, on any input, stay in $p_4$ (trap state).
- **Accepting States**: All states except $p_4$.

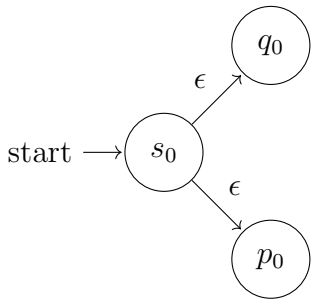



## Step 3: Construct NFA for the Union of Languages (a) and (b)

To recognize the union of the two languages, we use the standard construction for the union of two NFAs.

- **New Start State**: Create a new start state $s_0$.
- **Transitions from $s_0$**:
  - Add $\epsilon$-transitions from $s_0$ to the start states of the two NFAs ($q_0$ and $p_0$).
- **States**: Combine the states of the two NFAs.

- **Transitions**: Keep all transitions from both NFAs.
- **Accepting States**: Any state that is an accepting state in either of the two NFAs is an accepting state in the new NFA.




## Final NFA

The final NFA will have:

- A new start state $s_0$ with $\epsilon$-transitions to $q_0$ and $p_0$.
- All states and transitions from both NFAs.
- Accepting states are $q_4$ and all states from the second NFA except $p_4$.

This NFA will accept any string that is in language (a) or language (b).

**Question 5**

Prove that every NFA can be converted to an equivalent one that has a single accept state.

# 1 Introduction

A Nondeterministic Finite Automaton (NFA) is a theoretical computational model used in formal language theory. It is defined by a tuple $(Q, \Sigma, \delta, q_0, F)$ where:

- $Q$ is a finite set of states,
- $\Sigma$ is a finite set of input symbols (the alphabet),
- $\delta$ is the transition function: $\delta : Q \times \Sigma \to 2^Q$,
- $q_0 \in Q$ is the initial state, and
- $F \subseteq Q$ is the set of accept states.

An NFA accepts a string if there exists a sequence of transitions that lead to an accept state after consuming the entire input.

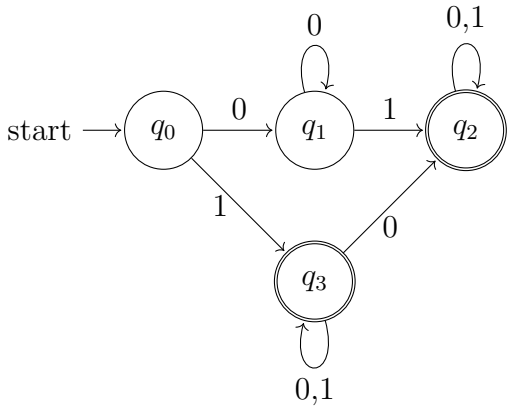
# 2 Conversion to an NFA with a Single Accept State

Given an NFA $(Q, \Sigma, \delta, q_0, F)$, we can construct an equivalent NFA $(Q_{\text{new}}, \Sigma, \delta_{\text{new}}, q_0, F_{\text{new}})$ with a single accept state as follows:

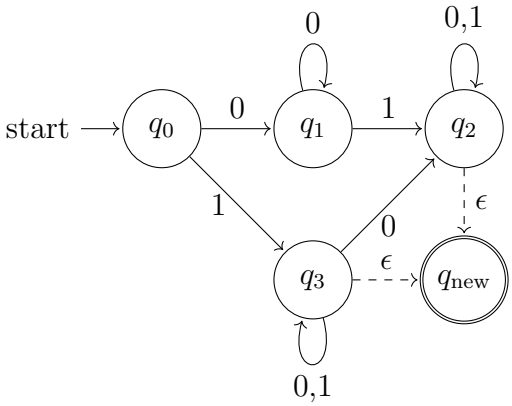
1. Introduce a new accept state $q_{\text{new}}$.
2. Modify the transition function by adding an $\epsilon$-transition from each state in $F$ to $q_{\text{new}}$.
3. Define the new accept state as $F_{\text{new}} = \{q_{\text{new}}\}$.

# 3 Example: NFA with Two Accepting States

Consider an NFA with two accept states, $q_2$ and $q_3$, which accepts strings ending in either 01 or 10:




Now, we convert this NFA to an equivalent one with a single accept state:




# 4 Proof of Equivalence

To show that both NFAs accept the same language:

- If the original NFA accepts a string, it must end in either $q_2$ or $q_3$. Since both states now have an $\epsilon$-transition to $q_{\text{new}}$, the new NFA will also accept the string.
- If the new NFA accepts a string, it must have reached state $q_{\text{new}}$, which can only occur through $q_2$ or $q_3$. Therefore, the original NFA must also have accepted the string.

Since both NFAs accept the same set of strings, they are equivalent.

# 5 Conclusion

We have successfully constructed an NFA with a single accept state that is equivalent to any given NFA, even one with multiple accept states. This proves that every NFA can be transformed into an equivalent one with exactly one accept state.