## DFA vs NDFA (Lecture 7)

The following table lists the differences between DFA and NDFA.

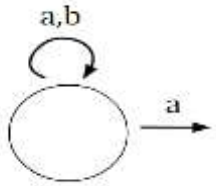
| DFA | NDFA |
|---|---|
| The transition from a state is to a single particular next state for each input symbol. Hence, it is called *deterministic*. | The transition from a state can be to multiple next states for each input symbol.<br>Hence it is called *non-deterministic*. |
| Empty string transitions are not seen in DFA. | NDFA permits empty string transitions. |
| Backtracking is allowed in DFA | In NDFA, backtracking is not always possible. |
| Requires more space. | Requires less space. |
| A string is accepted by a DFA, if it transits to a final state. | A string is accepted by a NDFA, if at least one of all possible transitions ends in a final state. |

## Comparison Table for Automata

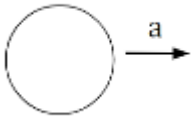
| | FA | TG | NFA |
|---|---|---|---|
| **Start states** | one | One or more | one |
| **Final states** | Some or none | Some or none | Some or none |
| **Edge labels** | Letter from Σ | words from Σ* | Letter from Σ |
| **Number of edges from each state** | One for each letter in Σ | Arbitrary | Arbitrary |
| **Deterministic (every input string has one path)** | Yes | Not necessarily | Not necessarily |
| **Every path represents one word** | Yes | Yes | Yes |

Note: Nondeterminism gives a machine multiple options for its moves.

- In a **nondeterministic** finite automaton (NFA), for each state there can be zero, one, two, or more transitions corresponding to a particular symbol.
- If NFA gets to state with more than one possible transition corresponding to the input symbol, we say it **branches**.
- If NFA gets to a state where there is no valid transition, then that branch **dies**.
- NFA is a state machine consisting of states and transitions that can either accept or reject a finite string.
- Essentially, NFAs have less restriction than DFAs, and can therefore make complicated automata easier to understand and depict in a diagram.
- a DFA can only have one transition for each symbol going outwards from the state. But, an NFA can have multiple transitions for a symbol from the same state.
    - If you see the figure below, the NFA diagram has both **a** and **b** looping back to itself, and also has a second **a** point towards another state. (Note: These diagrams are not functionally equivalent. They are simply created to show differences in transitions)

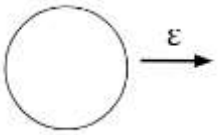



- Another difference is that an NFA is not required to have a transition for each symbol. So if we are creating an NFA for a language with the alphabet **{a, b}**, it is valid to have a state like this:



In this case, if we do happen to get the symbol **b**, we would still remain in the state. This is because this state is only interested in getting the symbol **a** so that it can continue to the next state.

- The last difference between an NFA and a DFA is that an NFA can have a transition for an empty string. A DFA cannot transition on an empty string, as this is an invalid transition, but an NFA is allowed to do this. See below:

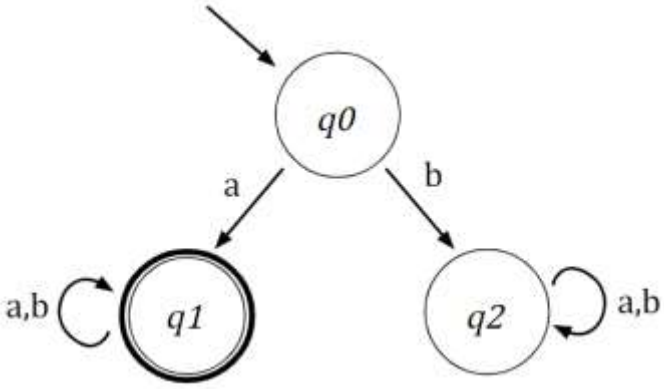



- An important component of an NFA to note is that **an NFA can have multiple outcomes with the same language, but if at least one of those outcomes results in an accepting state, then the diagram and language accept the string.**

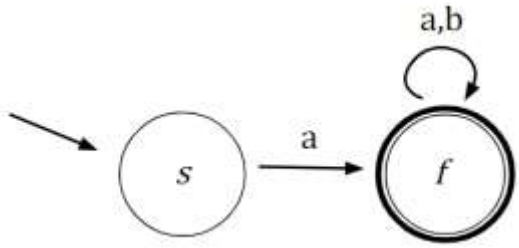
## How can we build an NFA?

Let's use the same example that we used in the DFA article. That is, we will again be using the language: ***{ax | x ∈ {a, b}*}***, which states that we want all strings that begin with the letter **a**.

For this language, our DFA looks like this:

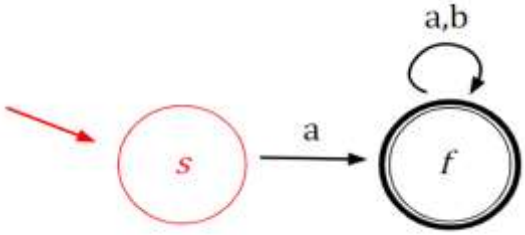


Here is the NFA equivalent for the language…

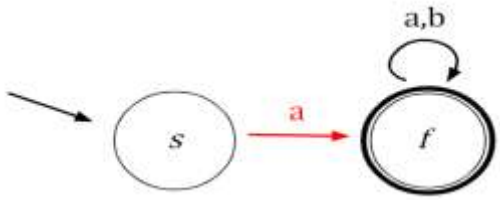



Now, let's break each component down and explain how this diagram works.

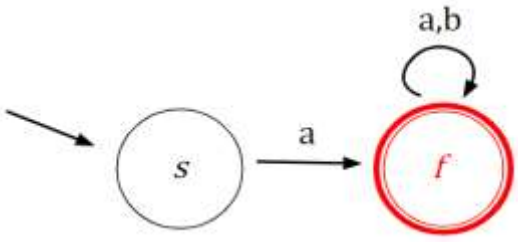
- The first thing we do is begin with the start arrow pointing to the start state. This is where we begin our processing of the string. Because we need at least an **a** to be in the string, our language cannot accept an empty string. Therefore, the start state, *S*, cannot be an accepting state. See the figure below:



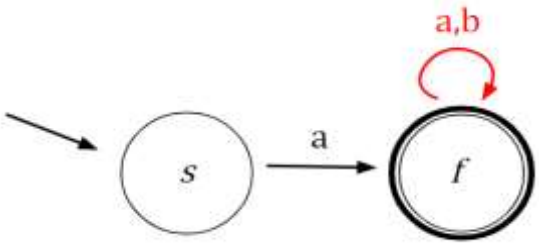

- Now, we must handle the rule of our language that the string **must begin with the symbol a**. We only care if our machine gets the symbol **a** as it's first symbol, and we want the machine to crash/fail if the first symbol is anything other than **a.**
- So, with the power of an NFA, we can simply specify that if we get an **a**, jump to the next state to continue with the string processing. However, if we get anything other than an **a,** remain in the state and reject the string. See the figure below:




- Now that we have an **a** in the first position, we can now move into an accepting state for our string. See below:



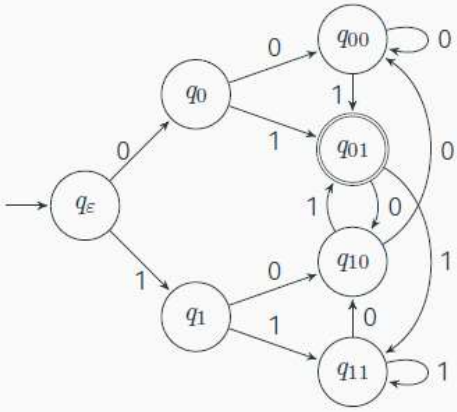

- Finally, our language ***{ax | x ∈ {a, b}*}*** states that our string can be followed by any symbol in any order in the set **{a, b}** by using the **x** in **ax**. In other words, after we get the initial **a** in our string, anything can happen after that. We don't care.
- That means we must have **a** and **b** looping back to itself in the accepting state. See below:



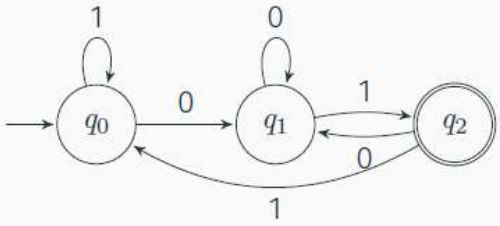

- And that is how we create the NFA for our language!

**Example**

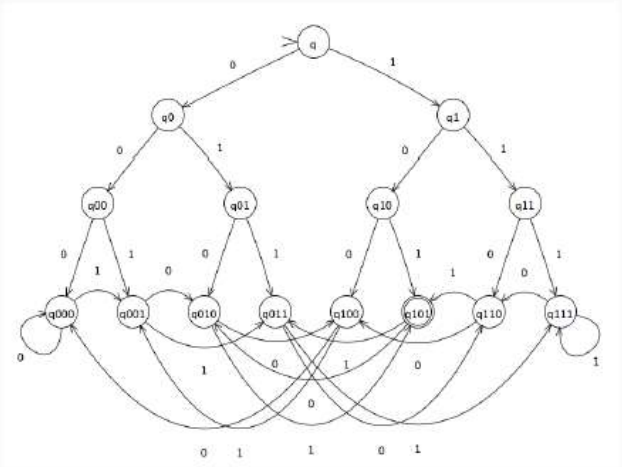
Construct a DFA over $\{0, 1\}$ that accepts all strings ending in 01



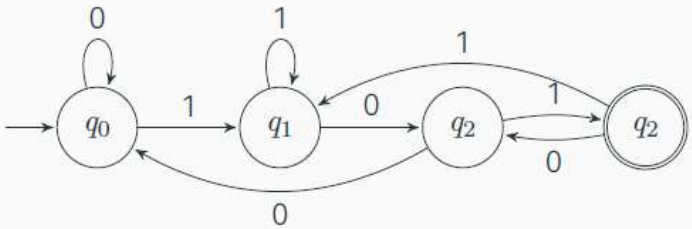

or



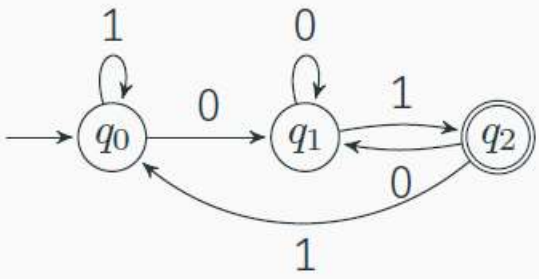

Construct a DFA over $\{0, 1\}$ that accepts all strings ending in 101



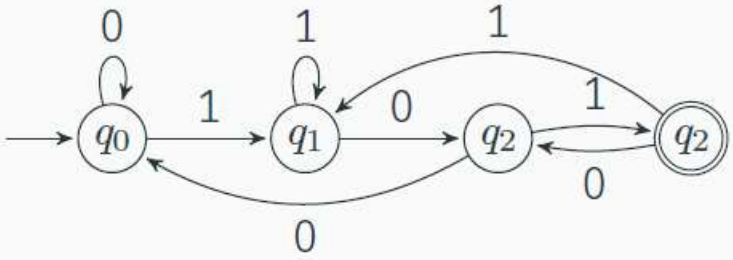

or



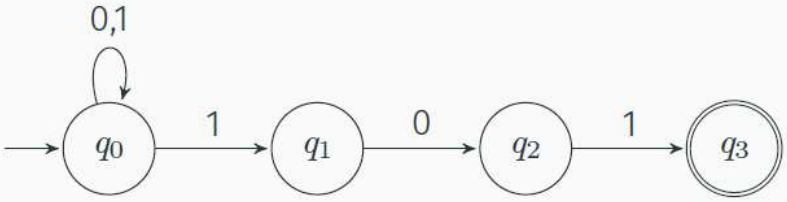
Ending in 01



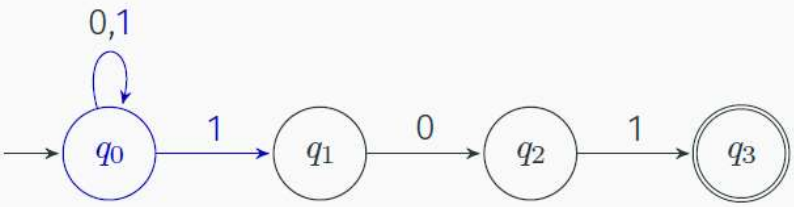

Ending in 101




A machine that is nondeterministic (and effectively making guesses)

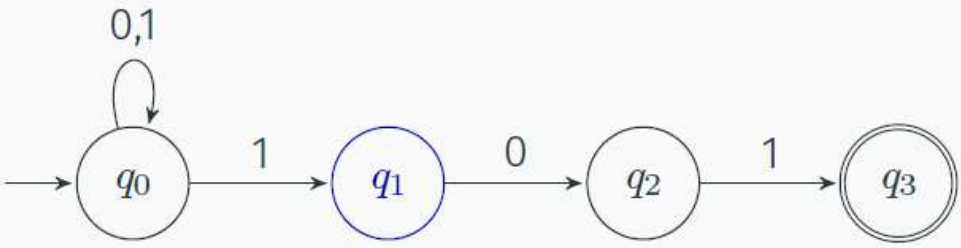



Each state can have zero, one, or more outgoing transitions labeled by the same symbol



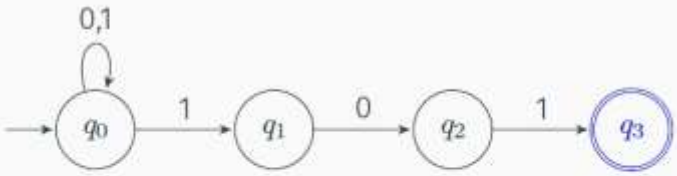

State $q_0$ has two transitions labeled 1

Upon reading 1, we have the choice of staying at $q_0$ or moving to $q_1$



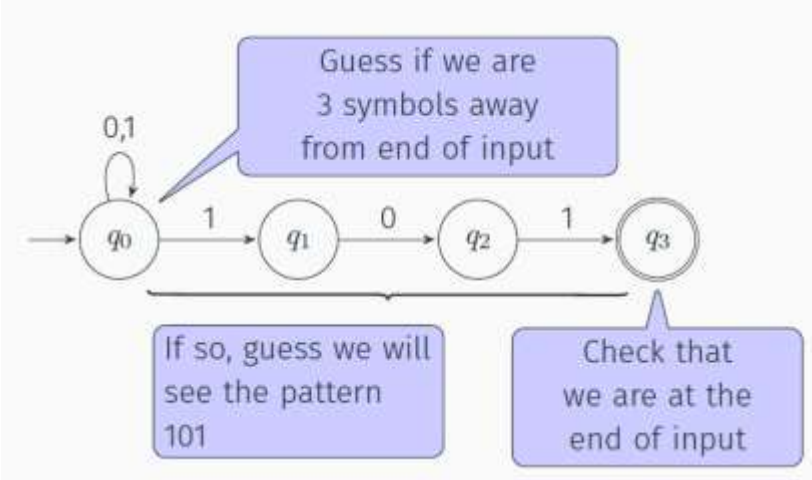

State $q_1$ has no transition labeled 1

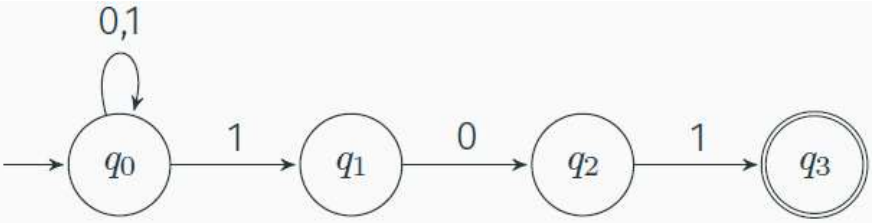
Upon reading 1 at $q_1$, die; upon reading 0, continue to $q_2$




State $q_1$ has no transition going out

Upon reading 0 or 1 at $q_3$, die



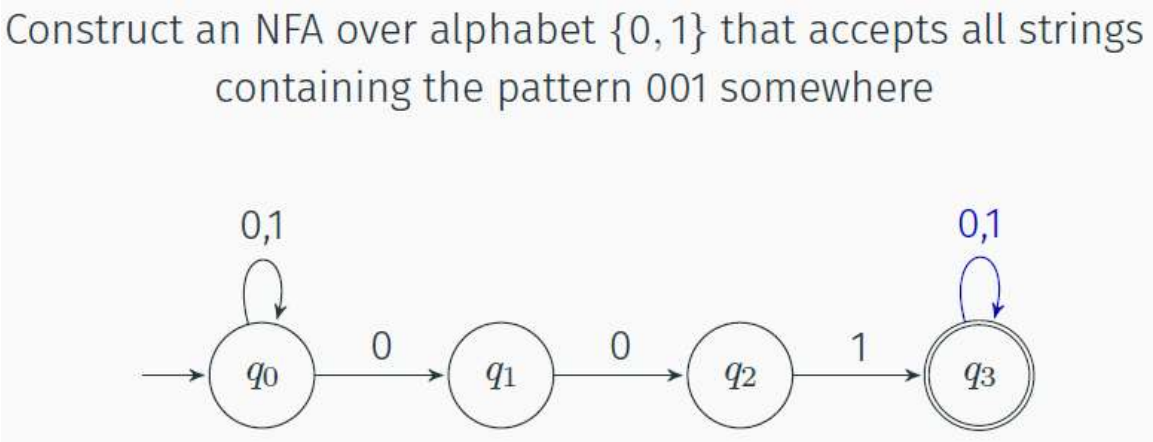

input: 01101

The NFA can have several active states at the same time

NFA accepts if at the end, one of its active states is accepting

Construct an NFA over alphabet $\{0, 1\}$ that accepts all strings containing the pattern 001 somewhere

| | |
|---|---|
| 11001010, 001001, 111001 | should be accepted |
| $\varepsilon$, 000, 010101 | should not |



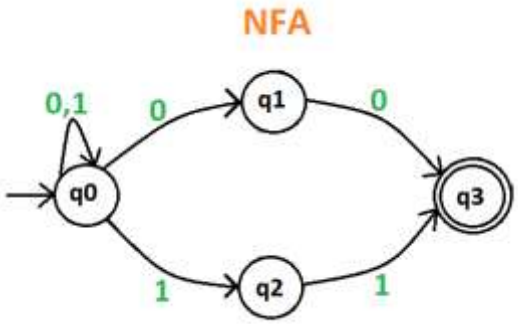

**The NFA accepts string *x* if there is some path that, starting from *q*0, ends at an accepting state as *x* is read from left to right.**

**The language of an NFA is the set of all strings accepted by the NFA.**

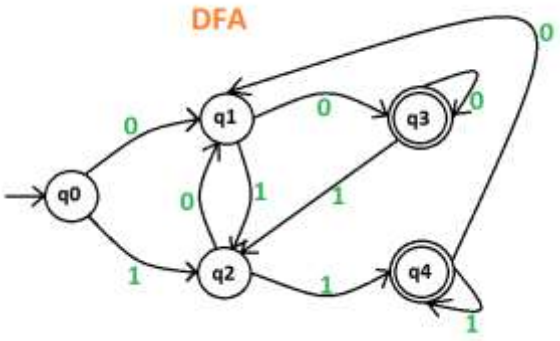
## Example

Draw a deterministic and non-deterministic finite automate which accept 00 and 11 at the end of a string containing 0, 1 in it, e.g., 01010100 but not 000111010.

**Explanation –** Design a DFA and NFA of a same string if input value reaches the final state then it is acceptable otherwise it is not acceptable. NFA of the given string is as follows:




DFA of the given string is as follows:



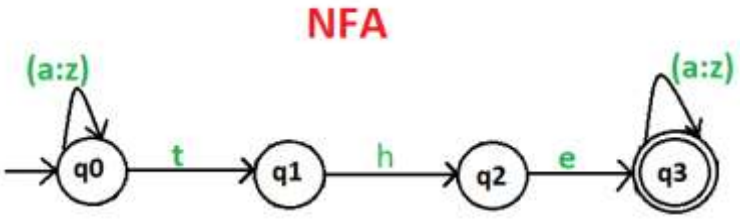

Here, **q0** shows the initial state, **q1** and **q2** are the transition states, and **q3** and **q4** are the final states.

**Note –** NFA and DFA both have same power that means if NFA can recognized an language L then DFA can also be defined to do so and if DFA can recognized an language L then NFA can also be defined to do so.

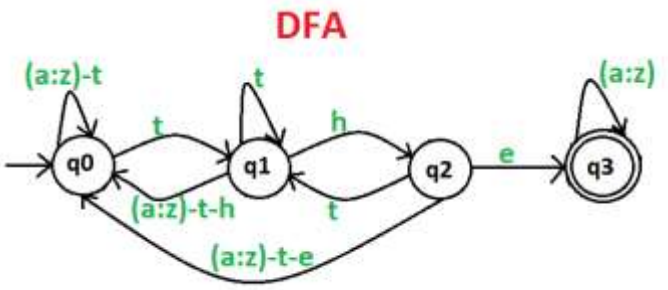
## Example

Draw a deterministic and non-deterministic finite automata which accept a string containing "the" anywhere in a string of {a-z}, e.g., "there" but not "those".

**Explanation –** Design a DFA and NFA of a same string if input value reaches the final state then it is acceptable otherwise, it is not acceptable. It is applicable for all the DFA and NFA. Since, NFA is quit easier than DFA, so first make its NFA and then go through the DFA. NFA of the given string is as follows:




DFA of the given string is as follows:

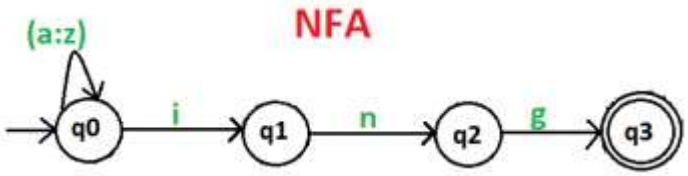



Here, **q0** shows the initial state, **q1** and **q2** are the transition states, and **q3** is the final state.

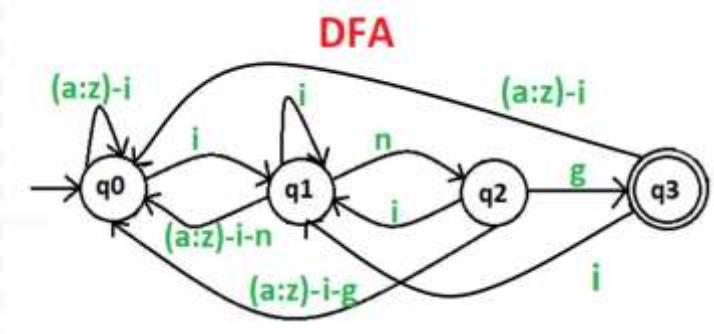
## Example

Draw a deterministic and non-deterministic finite automata which accept a string containing "ing" at the end of a string in a string of {a-z}, e.g., "anything" but not "anywhere".

**Explanation –** Design a DFA and NFA of a same string if input value reaches the final state then it is acceptable otherwise it is not acceptable. It is applicable for all the DFA and NFA.
NFA of the given string is as follows:




DFA of the given string is as follows:




Here, **q0** shows the initial state, **q1** and **q2** are the transition states, and **q3** is the final state.

## Example

1. Q = {q0, q1, q2}
2. ∑ = {0, 1}
3. q0 = {q0}
4. F = {q2}

**Solution:**

Transition diagram:

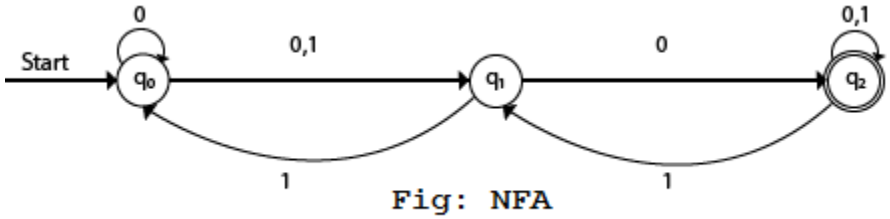



Fig: NFA

Transition Table:

| Present State | Next state for Input 0 | Next State of Input 1 |
|---|---|---|
| →$q_0$ | $q_0$, $q_1$ | $q_1$ |
| $q_1$ | $q_2$ | $q_0$ |
| *$q_2$ | $q_2$ | $q_1$, $q_2$ |

In the above diagram, we can see that when the current state is $q_0$, on input 0, the next state will be q0 or $q_1$, and on 1 input the next state will be $q_1$. When the current state is $q_1$, on input 0 the next state will be $q_2$ and on 1

input, the next state will be $q_0$. When the current state is q2, on 0 input the next state is $q_2$, and on 1 input the next state will be $q_1$ or $q_2$.

**Example**

NFA with $\sum = \{0, 1\}$ accepts all strings with 01.

**Solution:**


Fig: NFA

Transition Table:

| Present State | Next state for Input 0 | Next State of Input 1 |
|---|---|---|
| $\rightarrow q_0$ | $q_1$ | ε |
| $q_1$ | ε | $q_2$ |
| $*q_2$ | $q_2$ | $q_2$ |

**Example:**

NFA with $\sum = \{0, 1\}$ and accept all string of length at least 2.

**Solution:**

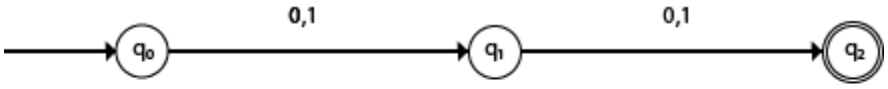



Fig: NFA

Transition Table:

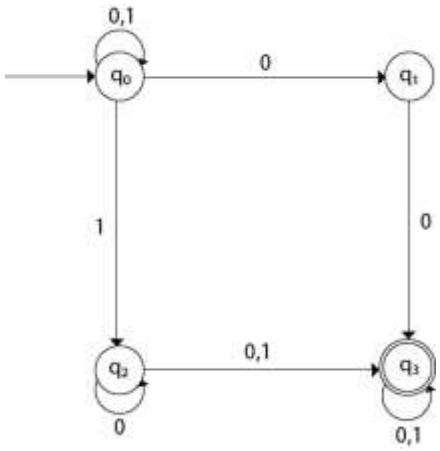
| Present State | Next state for Input 0 | Next State of Input 1 |
|---|---|---|
| $\rightarrow q_0$ | $q_1$ | $q_1$ |
| $q_1$ | $q_2$ | $q_2$ |
| $*q_2$ | ε | ε |

**Example**

Design a NFA for the transition table as given below:

| Present State | 0 | 1 |
|---|---|---|
| $\rightarrow q_0$ | $q_0, q_1$ | $q_0, q_2$ |
| $q_1$ | $q_3$ | ε |
| $q_2$ | $q_2, q_3$ | $q_3$ |
| $\rightarrow q_3$ | $q_3$ | $q_3$ |

**Solution:**
The transition diagram can be drawn by using the mapping function as given in the table.



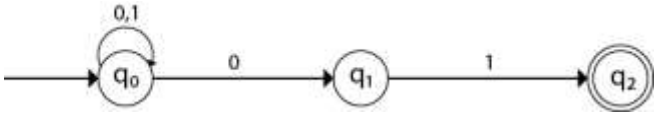

Here,

δ(q0, 0) = {q0, q1}
δ(q0, 1) = {q0, q2}
Then, δ(q1, 0) = {q3}
Then, δ(q2, 0) = {q2, q3}
δ(q2, 1) = {q3}
Then, δ(q3, 0) = {q3}
δ(q3, 1) = {q3}

**Example**
Design an NFA with ∑ = {0, 1} accepts all string ending with 01.
**Solution:**

Anything either 0 or 1 0 1

Hence, NFA would be:

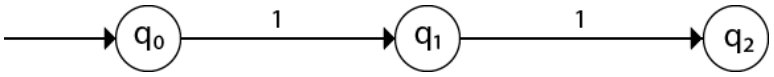



**Example**

Design an NFA with ∑ = {0, 1} in which double '1' is followed by double '0'.

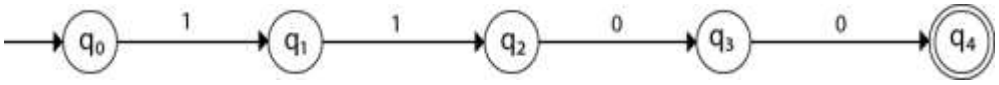
**Solution:**

The FA with double 1 is as follows:



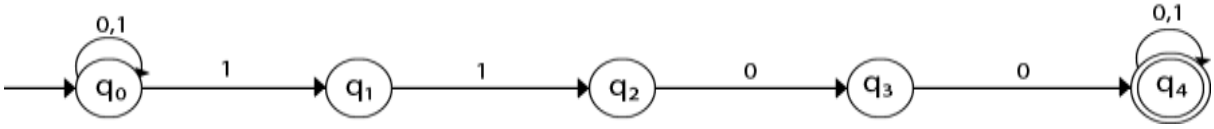

It should be immediately followed by double 0.

Then,




Now before double 1, there can be any string of 0 and 1. Similarly, after double 0, there can be any string of 0 and 1.

Hence the NFA becomes:




Now considering the string 01100011

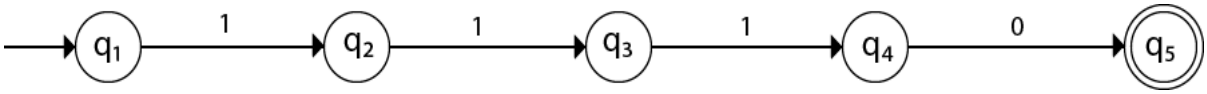
**$q_0 \rightarrow q_1 \rightarrow q_2 \rightarrow q_3 \rightarrow q_4 \rightarrow q_4 \rightarrow q_4 \rightarrow q_4$**

**Example**

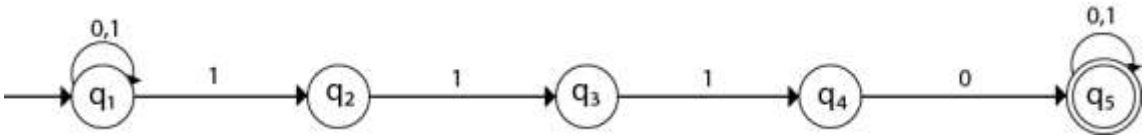
Design an NFA in which all the string contain a substring 1110.

**Solution:**

The language consists of all the string containing substring 1010. The partial transition diagram can be:



Now as 1010 could be the substring. Hence we will add the inputs 0's and 1's so that the substring 1010 of the language can be maintained. Hence the NFA becomes:




Transition table for the above transition diagram can be given below:

| **Present State** | **0** | **1** |
|---|---|---|
| $\rightarrow q_1$ | $q_1$ | $q_1, q_2$ |
| $q_2$ | | $q_3$ |
| $q_3$ | | $q_4$ |
| $q_4$ | $q_5$ | |
| $*q_5$ | $q_5$ | $q_5$ |

Consider a string 111010,

$$\begin{aligned}\delta(q1, 111010) &= \delta(q_1, 1100)\\ &= \delta(q_1, 100)\\ &= \delta(q_2, 00)\end{aligned}$$

Got stuck! As there is no path from $q_2$ for input symbol 0. We can process string 111010 in another way.

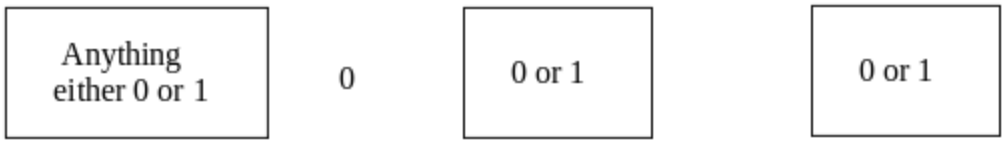
$$\begin{aligned}\delta(q1, 111010) &= \delta(q_2, 1100)\\ &= \delta(q_3, 100)\\ &= \delta(q_4, 00)\\ &= \delta(q_5, 0)\\ &= \delta(q_5, \varepsilon)\end{aligned}$$

As state $q_5$ is the accept state. We get the complete scanned, and we reached to the final state.

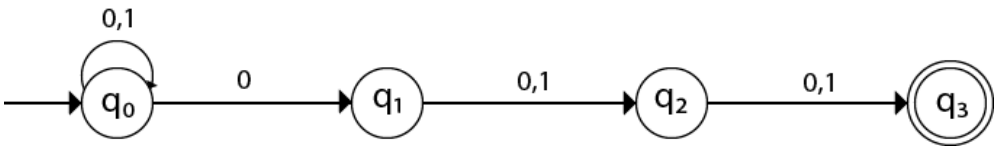
**Example**

Design an NFA with $\sum = \{0, 1\}$ accepts all string in which the third symbol from the right end is always 0.

**Solution:**




Thus we get the third symbol from the right end as '0' always. The NFA can be:




The above image is an NFA because in state $q_0$ with input 0, we can either go to state $q_0$ or $q_1$.

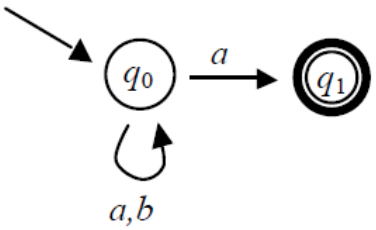
## Notes:

From DFA to NFA

•A DFA has exactly one transition from every state on every symbol in the alphabet.

•By relaxing this requirement we get a related but more flexible kind of automaton: the nondeterministic finite automaton or NFA.

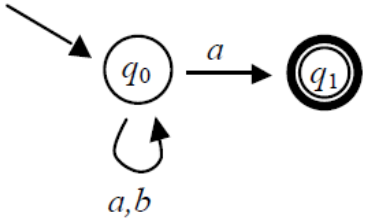
**Not A DFA**




•Does not have exactly one transition from every state on every symbol: –Two transitions from q0 on **a** –No transition from q1 (on either a or b).

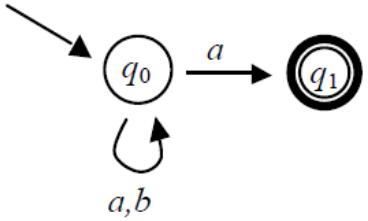
•Though not a DFA, this can be taken as defining a language, in a slightly different way.

**Possible Sequences of Moves**




• We'll consider all possible sequences of moves the machine might make for a given string.
• For example, on the string *aa* there are three:
– From *q*0 to *q*0 to *q*0, rejecting.
– From *q*0 to *q*0 to *q*1, accepting.
– From *q*0 to *q*1, getting stuck on the last *a*.
• Our convention for this new kind of machine: a string is in *L*(*M*) if there is *at least one* accepting sequence.

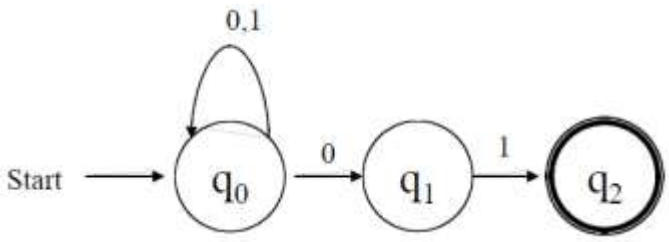
**Nondeterministic Finite Automaton (NFA)**



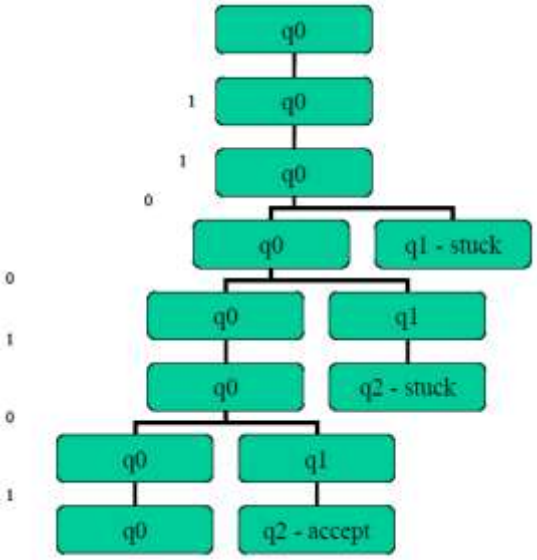

• *L*(*M*) = the set of strings that have *at least one* accepting sequence
• In the example above, $L(M) = \{xa \mid x \in \{a,b\}^*\}$
• A DFA is a special case of an NFA:
– An NFA that happens to be deterministic: there is exactly one transition from every state on every symbol.
– So there is exactly one possible sequence for every string
• NFA is *not necessarily* deterministic

## NFA Example

- This NFA accepts only those strings that end in 01



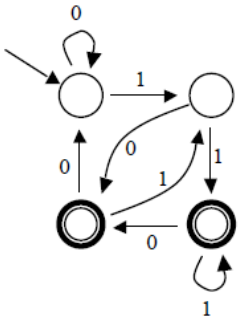

• Running in “parallel threads” for string 1100101.



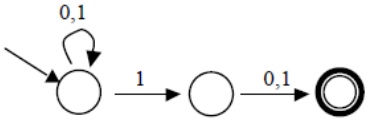

**NFA Advantage**

• An NFA for a language can be smaller and easier to construct than a DFA
• Let L={x ∈ {0,1}*|where x is a string whose next-to-last symbol is 1}
• Construct both a DFA and NFA for recognizing L.

DFA:

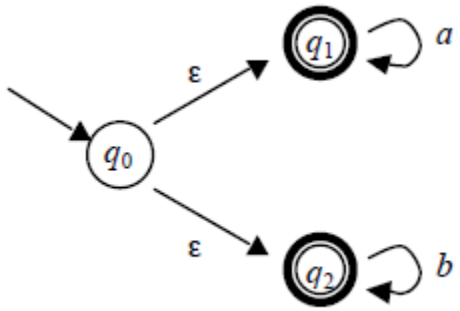



NFA:

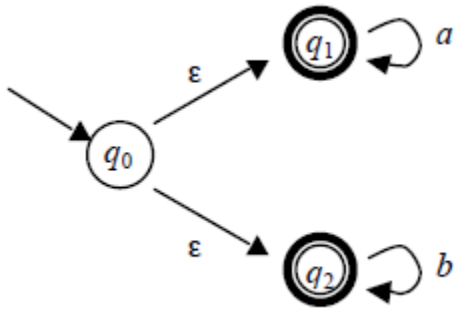



**Spontaneous Transitions**

• An NFA can make a state transition spontaneously, without consuming an input symbol
• Shown as an arrow labeled with ε
• For example, $\{a\}^* \cup \{b\}^*$:

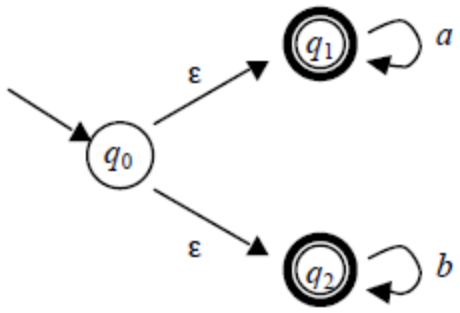



**ε-Transitions To Accepting States**

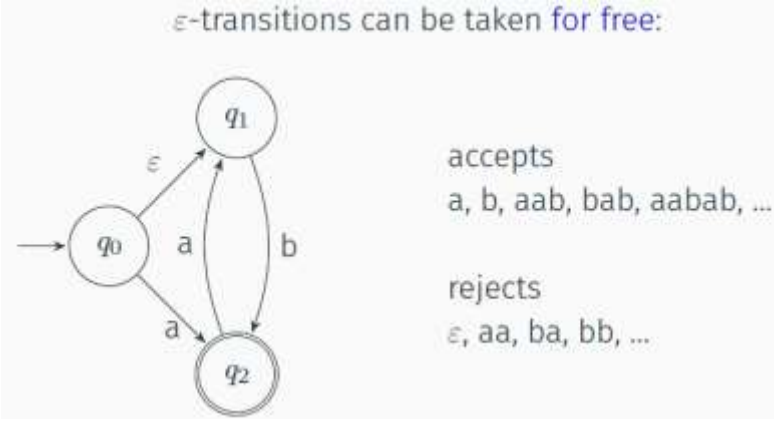


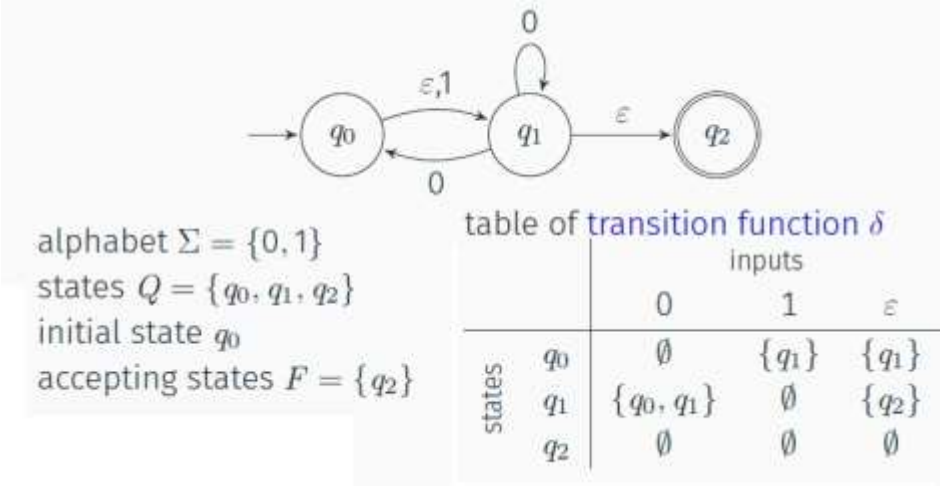

• An ε-transition can be made at any time
• For example, there are three sequences on the empty string
– No moves, ending in $q_0$, rejecting
– From $q_0$ to $q_1$, accepting
– From $q_0$ to $q_2$, accepting
• Any state with an ε-transition to an accepting state ends up working like an accepting state too

## ε-transitions For NFA Combining

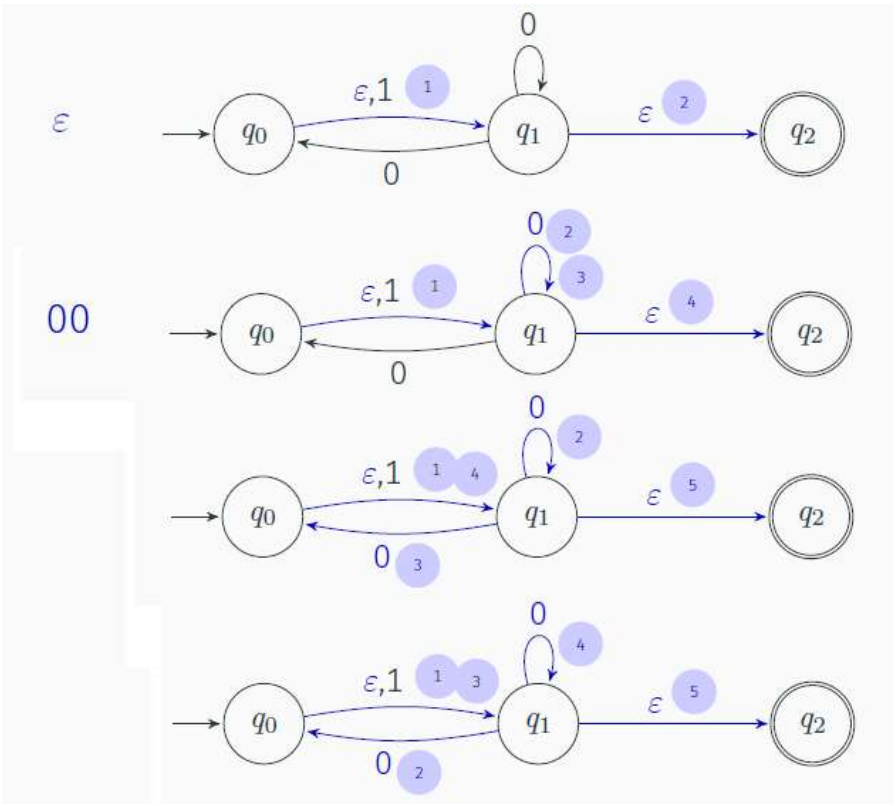


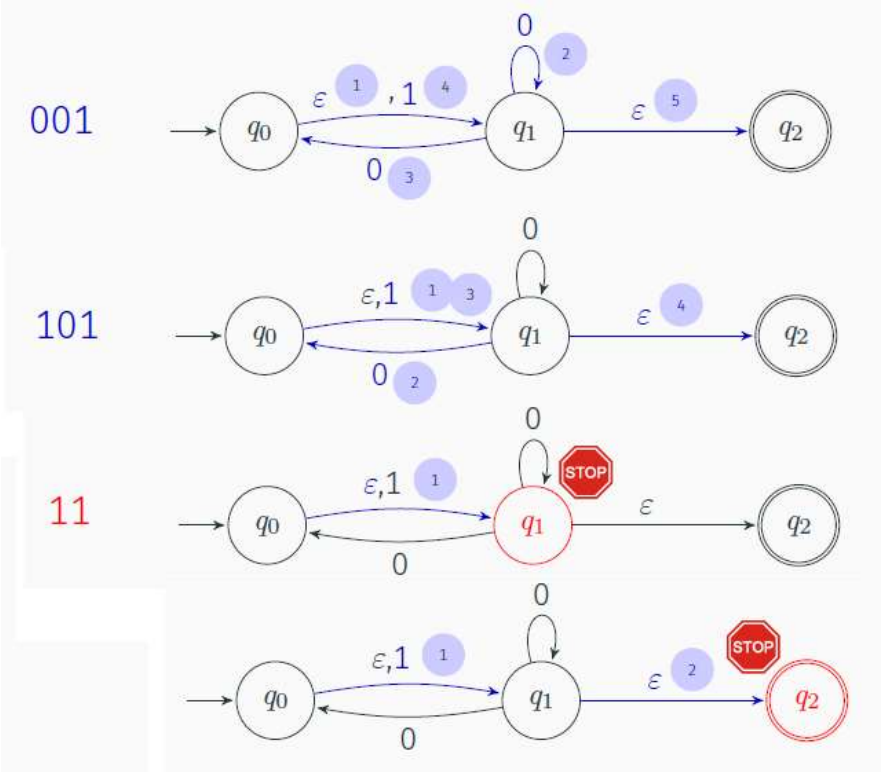

• ε-transitions are useful for combining smaller automata into larger ones.
• This machine is combines a machine for $\{a\}^*$ and a machine for $\{b\}^*$.
• It uses an ε-transition at the start to achieve the union of the two languages.



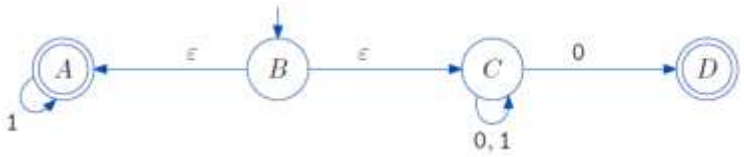

**Some computational paths of the NFA**


ε
0
ε,1
1
ε
2
q0
q1
q2
0
00
0
2
3
ε,1
1
ε
4
q0
q1
q2
0
0
2
ε,1
1
4
ε
5
q0
q1
q2
0
3
0
4
ε,1
1
3
ε
5
q0
q1
q2
0
2


001
0
2
ε
1
,1
4
ε
5
q0
q1
q2
0
3
101
0
ε,1
1
3
ε
4
q0
q1
q2
0
2
11
0
STOP
ε,1
1
ε
q0
q1
q2
0
0
ε,1
1
STOP
ε
2
q0
q1
q2
0

- We also allow ε-transitions: arrows labeled with the empty string. These allow the NFA to change state without consuming an input symbol.

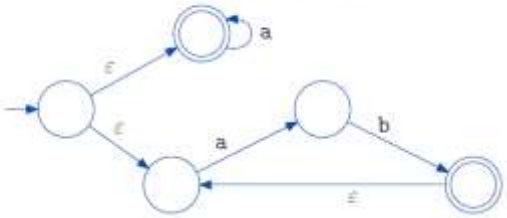
*Example*




Accepts all binary strings where the last symbol is 0 or that contain only 1's.

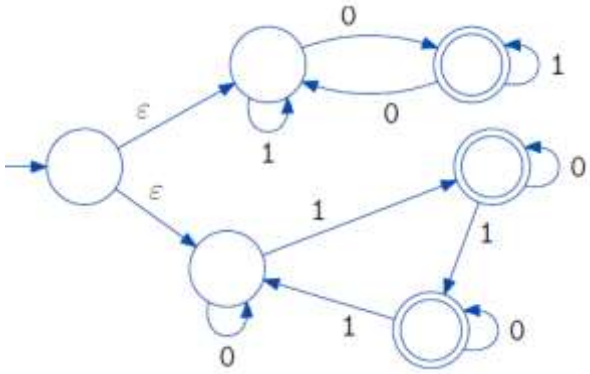
*Another Example*

Here is another NFA for $a^* + (ab)^*$:

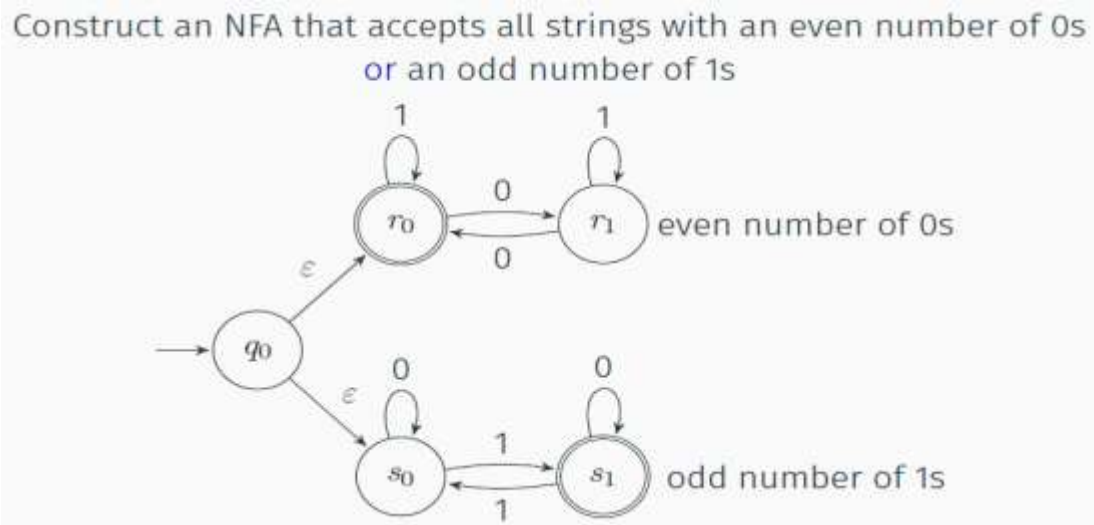



**Example**

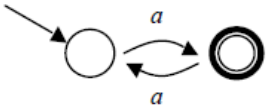
Give an NFA for the set of all binary strings that have either the number of 0's odd, or the number of 1's not a multiple of 3, or both.



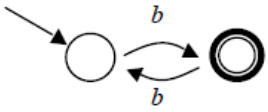
## Example of ε-transitions



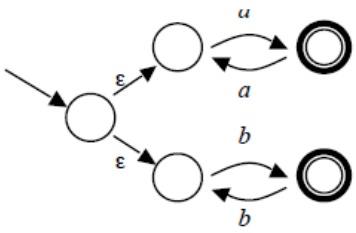

## Revisiting Union




$A = \{a^n \mid n \text{ is odd}\}$

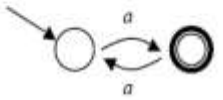



$B = \{b^n \mid n \text{ is odd}\}$

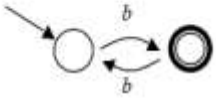



$A \cup B$

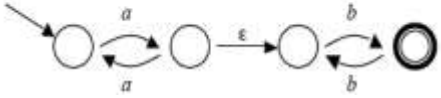
# Concatenation




$A = \{a^n \mid n \text{ is odd}\}$

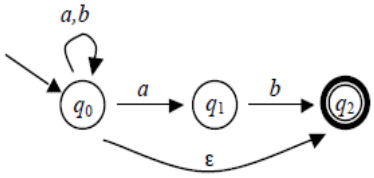



$B = \{b^n \mid n \text{ is odd}\}$




$\{xy \mid x \in A \text{ and } y \in B\}$

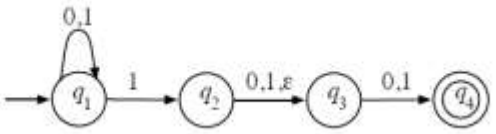
# Example:




- Formally, $M = (Q, \Sigma, \delta, q_0, F)$, where
  - $Q = \{q_0, q_1, q_2\}$
  - $\Sigma = \{a,b\}$ (we assume: it must contain at least $a$ and $b$)
  - $F = \{q_2\}$
  - $\delta(q_0,a) = \{q_0,q_1\}$, $\delta(q_0,b) = \{q_0\}$, $\delta(q_0,\varepsilon) = \{q_2\}$,
    $\delta(q_1,a) = \{\}$, $\delta(q_1,b) = \{q_2\}$, $\delta(q_1,\varepsilon) = \{\}$
    $\delta(q_2,a) = \{\}$, $\delta(q_2,b) = \{\}$, $\delta(q_2,\varepsilon) = \{\}$
- The language defined is $\{a,b\}^*$

## An Example Nondeterministic Finite Automaton

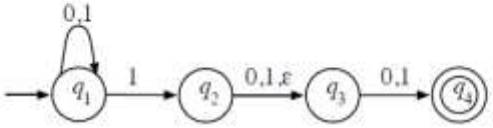
An NFA that accepts all strings over $\{0, 1\}$ that contain a $1$ either at the third position from the end or at the second position from the end.



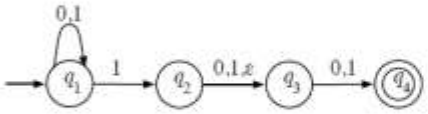

- There are two edges labeled $1$ coming out of $q_1$.
- There are no edges coming out of $q_4$.
- The edge from $q_2$ is labeled with $\epsilon$, in addition to $0$ and $1$.

## What the Diagram Says

- If the node you are in has an outgoing edge labeled $\epsilon$, you may choose to follow it.
- After receiving a symbol,
  - if the node you are in has multiple outgoing edges labeled with the symbol, you nondeterministically choose one of them and cross it;
  - if there are no choices, stop there.




## Transition Function for the Previous Example

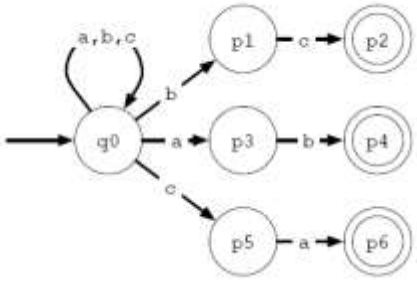



| state | symbol | | |
|---|---|---|---|
| | $0$ | $1$ | $\epsilon$ |
| $q_1$ | $\{q_1\}$ | $\{q_1, q_2\}$ | $\emptyset$ |
| $q_2$ | $\{q_3\}$ | $\{q_3\}$ | $\{\ q_3\ \}$ |
| $q_3$ | $\{q_4\}$ | $\{q_4\}$ | $\emptyset$ |
| $q_4$ | $\emptyset$ | $\emptyset$ | $\emptyset$ |

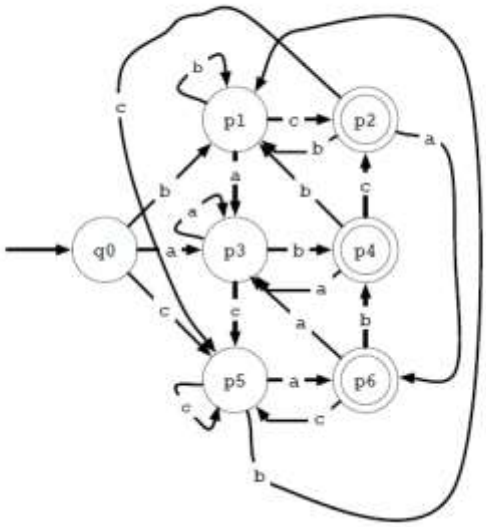
This NFA accepts $y = 11$ with respect to state sequence $(q_1, q_2, q_3, q_4)$ and decomposition $y = 1\epsilon 1$.

## Example

An NFA for the language of all strings over $\{a, b, c\}$ that end with one of $ab$, $bc$, and $ca$.

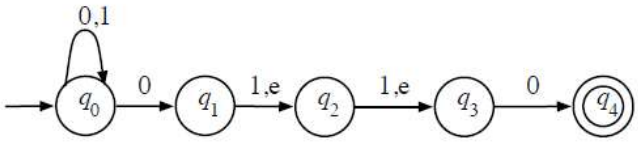



## A DFA Version of the Same Language



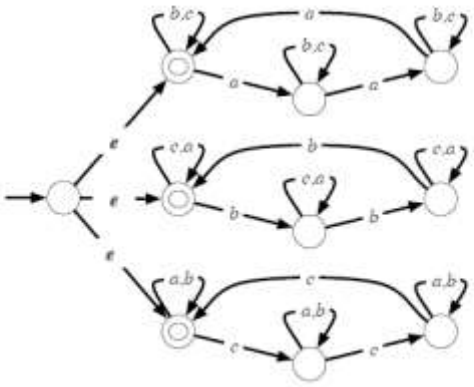

## Example

An NFA for the language of all strings over $\{0, 1\}$ that end with one of $0110$, $010$, and $00$.



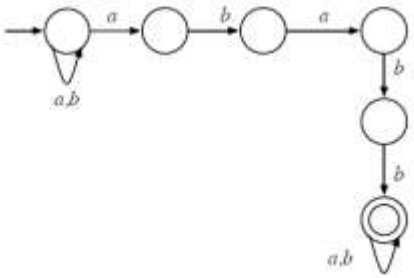
## Example

An NFA for the language of all strings over $\{a, b, c\}$ for which one of (the number of occurrences of $a$), (the number of occurrences of $b$), and (the number of occurrences of $c$) is a multiple of 3.




## Example

An NFA for the language of all strings over $\{a, b\}$ that contain $ababb$.




## Example

An NFA that recognizes the language consisting of all strings over $\{0, 1\}$ that contain a $1$ at either the third to last position or the second to last position.