Introduction to Automata Theory

Introduction to Finite Automata: Structural Representations, Automata and Complexity, the Central Concepts of Automata Theory – Alphabets, Strings, Languages, Problems. Nondeterministic Finite Automata: Formal Definition, an application, Text Search, Finite Automata with Epsilon-Transitions. Deterministic Finite Automata: Definition of DFA, How A DFA Process Strings, The language of DFA, Conversion of NFA with €-transitions to NFA without €-transitions. Conversion of NFA to DFA, Moore and Melay machines

What is Automata Theory?

- Theory of automata is a theoretical branch of computer science and mathematical. It is the study of abstract machines and the computation problems that can be solved using these machines. The abstract machine is called the automata. The main motivation behind developing the automata theory was to develop methods to describe and analyse the dynamic behaviour of discrete systems.

Why Study Automata Theory?

Finite automata are a useful model for many important kinds of software and hardware:

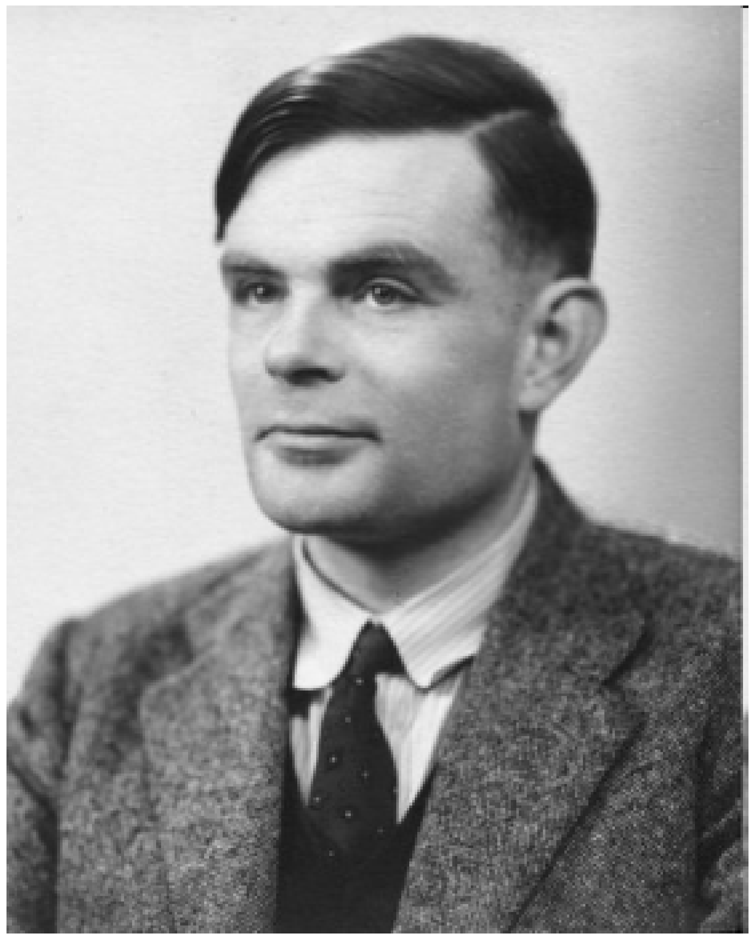
1. Software for designing and checking the behavior of digital circuits
2. The lexical analyzer of a typical compiler, that is, the compiler component that breaks the input text into logical units
3. Software for scanning large bodies of text, such as collections of Web pages, to find occurrences of words, phrases or other patterns
4. Software for verifying systems of all types that have a finite number of distinct states, such as communications protocols of protocols for secure exchange information

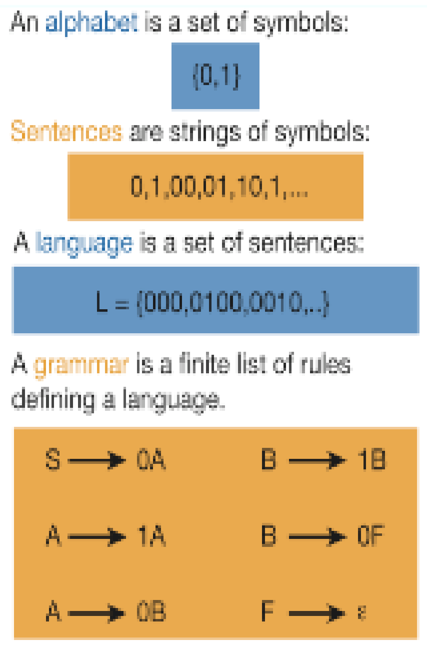
(A pioneer of automata theory)

Alan Turing (1912-1954)

- Father of Modern Computer Science
- English mathematician
- Studied abstract machines called *Turing machines* even before computers existed

Heard of the Turing test?





An alphabet is a set of symbols:

{0,1}

Sentences are strings of symbols:

0,1,00,01,10,1,...

A language is a set of sentences:

L = {000,0100,0010,..}

A grammar is a finite list of rules defining a language.

$S \longrightarrow 0A$ $\quad B \longrightarrow 1B$

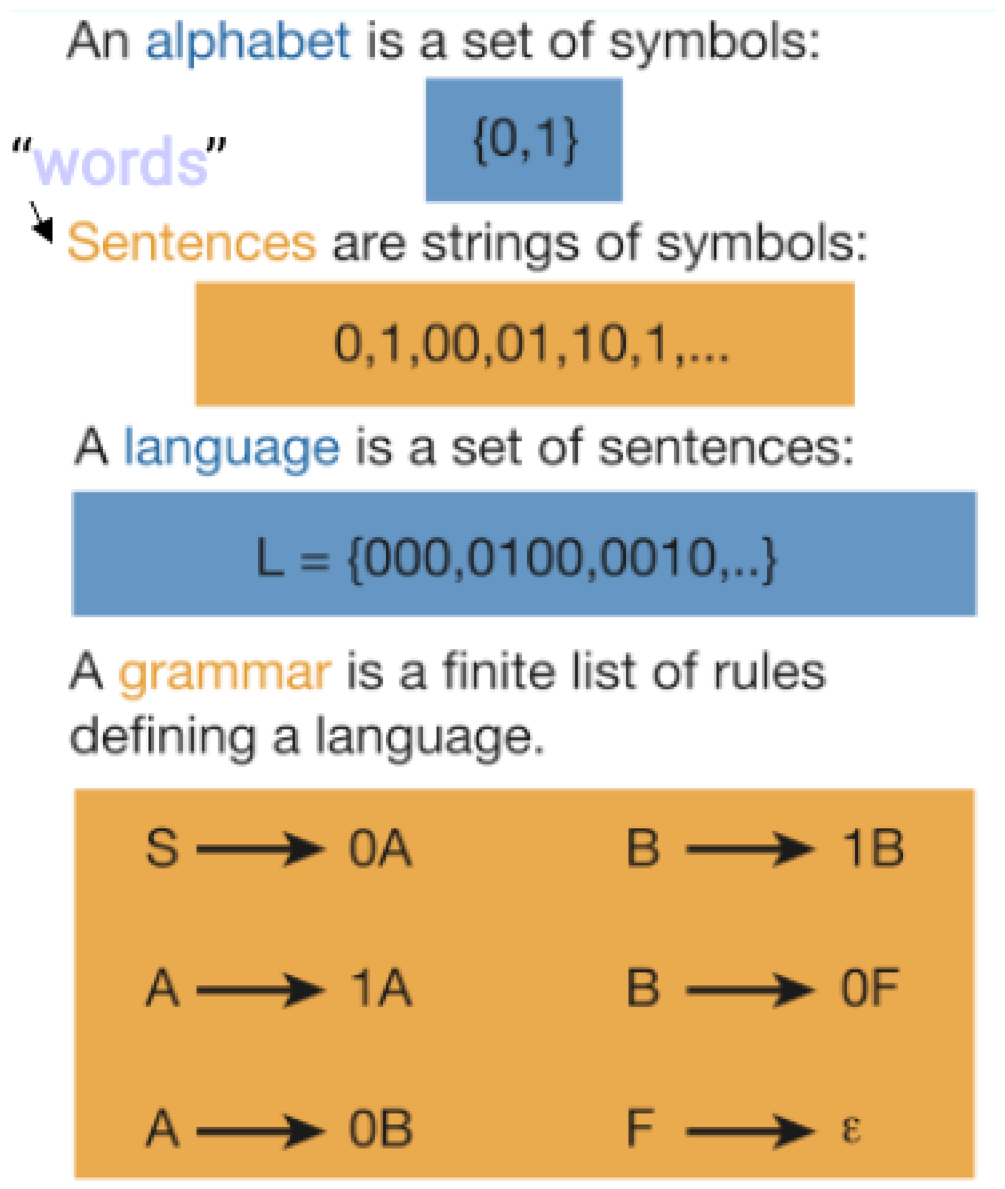
$A \longrightarrow 1A$ $\quad B \longrightarrow 0F$

$A \longrightarrow 0B$ $\quad F \longrightarrow \varepsilon$

Theory of Computation: A Historical Perspective

1930s	• Alan Turing studies Turing machines • Decidability • Halting problem
1940-1950s	• "Finite automata" machines studied • Noam Chomsky proposes the "Chomsky Hierarchy" for formal languages
1969	Cook introduces "intractable" problems or "NP-Hard" problems
1970-	Modern computer science: compilers, computational & complexity theory evolv

Languages & Grammars



An alphabet is a set of symbols:

{0,1}

Or "words"

Sentences are strings of symbols:

0,1,00,01,10,1,...

A language is a set of sentences:

L = {000,0100,0010,..}

A grammar is a finite list of rules defining a language.

$S \longrightarrow 0A$ $\quad B \longrightarrow 1B$

$A \longrightarrow 1A$ $\quad B \longrightarrow 0F$

$A \longrightarrow 0B$ $\quad F \longrightarrow \varepsilon$

Image source: Nowak et al. Nature, vol 417, 2002

- Languages: "*A language is a collection of sentences of finite length all constructed from a finite alphabet of symbols*"
- Grammars: "*A grammar can be regarded as a device that enumerates the sentences of a language*" - nothing more, nothing less

- *N. Chomsky, Information and Control, Vol 2, 1959*

Finite Automata

- Finite automata are used to recognize patterns.
- It takes the string of symbol as input and changes its state accordingly. When the desired symbol is found, then the transition occurs.
- At the time of transition, the automata can either move to the next state or stay in the same state.
- Finite automata have two states, **Accept state** or **Reject state**. When the input string is processed successfully, and the automata reached its final state, then it will accept.

Automata and Complexity,

- **Automata theory:**

  Automata theory deals with the theory with which these machines work.. An automaton can be in various states. One state is the state the automaton starts in and the others are the final or exit states.

- Some of the models are:

  i. Finite Automata: These are used in text processing, compilers, and hardware design.

  ii. Context-Free Grammars: There are used to define programming languages and in Artificial Intelligence

  iii. Turing machines: Abstract models of the simple PCs we have at home.

Finite Automata Model Structural Representations

finite automata can be represented by input tape and finite control.

- **Input tape:** It is a linear tape having some number of cells. Each input symbol is placed in each cell.
- **Finite control:** The finite control decides the next state on receiving particular input from input tape. The tape reader reads the cells one by one from left to right, and at a time only one input symbol is read.

Automata and Complexity

Computational Complexity Theory:

● Computational Complexity Theory deals with the efficiency with which a computer can solve a problem. Time and Space are considered to be the two vectors responsible for a problems efficiency.

● This theory mainly asks a general question about possible algorithm to solve a problem, whereas analysis of algorithms involves analyzing the amount of resources needed to solve a problem.

The Central Concepts of Automata Theory

Alphabet:

An alphabet is a finite, non-empty set of symbols

- We use the symbol ∑ (sigma) to denote an alphabet
- Examples:
  - Binary: ∑ = {0,1}
  - All lower case letters: ∑ = {a,b,c,..z}
  - Alphanumeric: ∑ = {a-z, A-Z, 0-9}
  - DNA molecule letters: ∑ = {a,c,g,t}

Strings

A string or word is a finite sequence of symbols chosen from ∑

- ***Empty string is ε (or "epsilon")***

Length of a String:

- Length of a string *w*, denoted by "|*w*|", is equal to the *number of (non-ε) characters in the string*
  - *E.g., x = 010100 |x| = 6*
  - *x = 01ε0ε1ε00ε |x| = ?*
  - *xy =* concatentation of two strings *x* and *y*

Powers of an alphabet

Kleene Star

Definition: The Kleene star, Σ * , is a unary operator on a set of symbols or strings, Σ, that gives the infinite set of all possible strings of all possible lengths over Σ including λ.

Representation: $\Sigma^* = \Sigma^0 \cup \Sigma^1 \cup \Sigma^2 \cup \ldots \Sigma^p$ where Σ^p is the set of all possible strings of length p.

Example: If Σ = {a, b}, Σ *= {λ, a, b, aa, ab, ba, bb,……….}

Powers of an alphabet

Kleene Closure / Plus

Definition: The set Σ + is the infinite set of all possible strings of all possible lengths over Σ excluding λ.

Representation: $\Sigma^+ = \Sigma^1 \cup \Sigma^2 \cup \Sigma^3$

$$\Sigma^+ = \Sigma^* - \{ \lambda \}$$

Example: If Σ = { a, b } , Σ+ ={ a, b, aa, ab, ba, bb,……….}

Languages

*L is a said to be a language over alphabet ∑, only if L ▯ ∑**

▯ this is because ∑* is the set of all strings (of all possible length including 0) over the given alphabet ∑

Examples:

– Let L be *the* language of <u>all strings consisting of *n* 0's followed by *n* 1's</u>:
  L = {▯, 01, 0011, 000111,…}

– Let L be *the* language of <u>all strings of with equal number of 0's and 1's</u>:
  L = {▯, 01, 10, 0011, 1100, 0101, 1010, 1001,…} $\longrightarrow$

Canonical ordering of strings in the language

Definition: Ø denotes the Empty language

- Let L = {▯}; Is L=Ø? $\boxed{\text{NO}}$

The Membership Problem

Given a string $w \in \Sigma^*$ and a language L over Σ, decide whether or not $w \in L$.

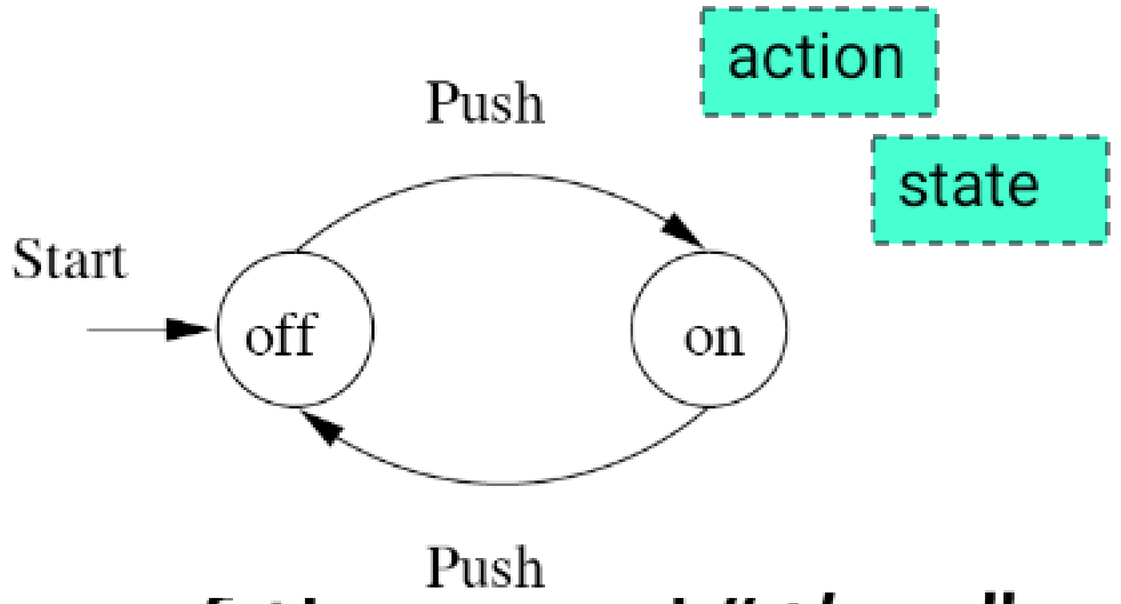
<u>Example:</u>

Let w = 100011

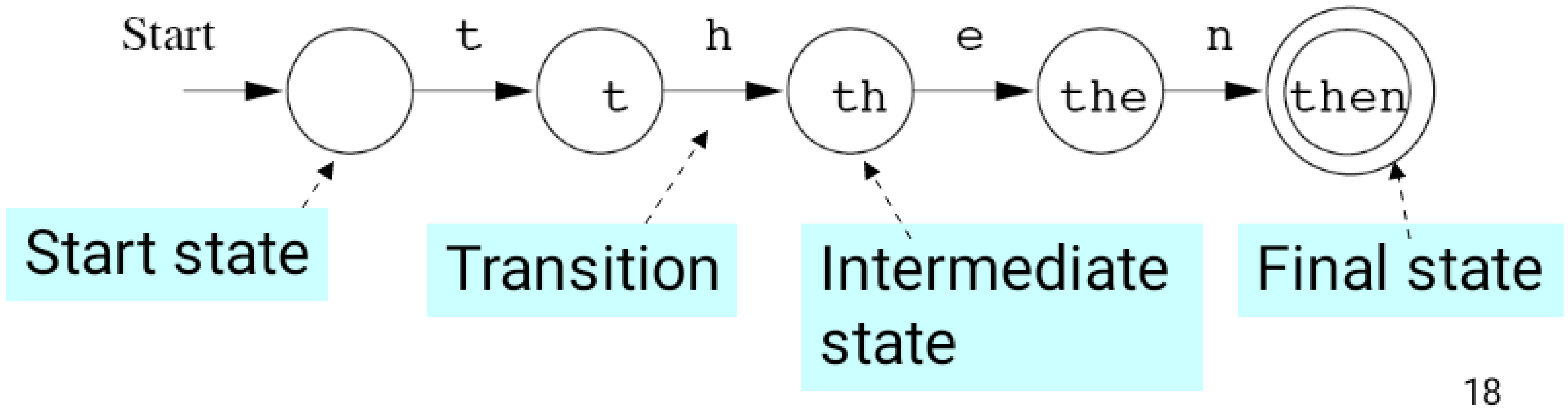
Q) Is w ∈ the language of strings with equal number of 0s and 1s?

Finite Automata : Examples

- On/Off switch



- Modeling recognition of the word "*then*"



Types of Finite Automata

- Deterministic Finite Automaton (DFA)
- Nondeterministic Finite Automaton (NFA)

Formal Definition of a DFA

- A DFA is a five-tuple:

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

Q A <u>finite</u> set of states
Σ A <u>finite</u> input alphabet
q_0 The initial/starting state, q_0 is in Q
F A set of final/accepting states, which is a subset of Q
δ A transition function, which is a total function from Q x Σ to Q

$\delta: (Q \times \Sigma) \rightarrow Q$ δ is defined for any q in Q and s in Σ, and
$\delta(q,s) = q'$ is equal to some state q' in Q, could be q'=q

Intuitively, δ(q,s) is the state entered by M after reading symbol s while in state q.

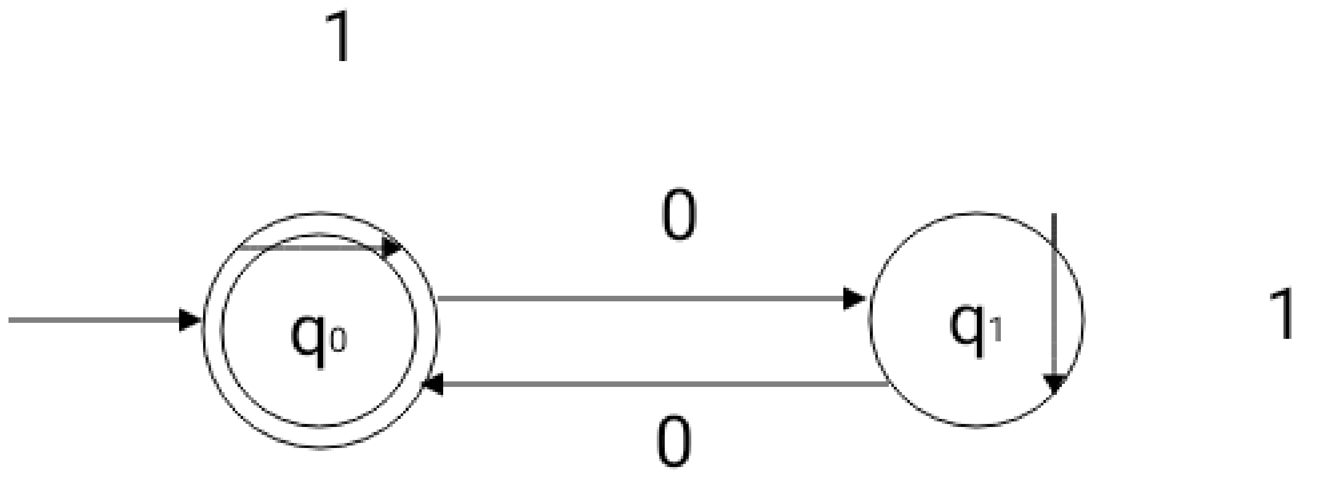
- Revisit example #1:

$Q = \{q_0, q_1\}$

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

Start state is q_0

$F = \{q_0\}$

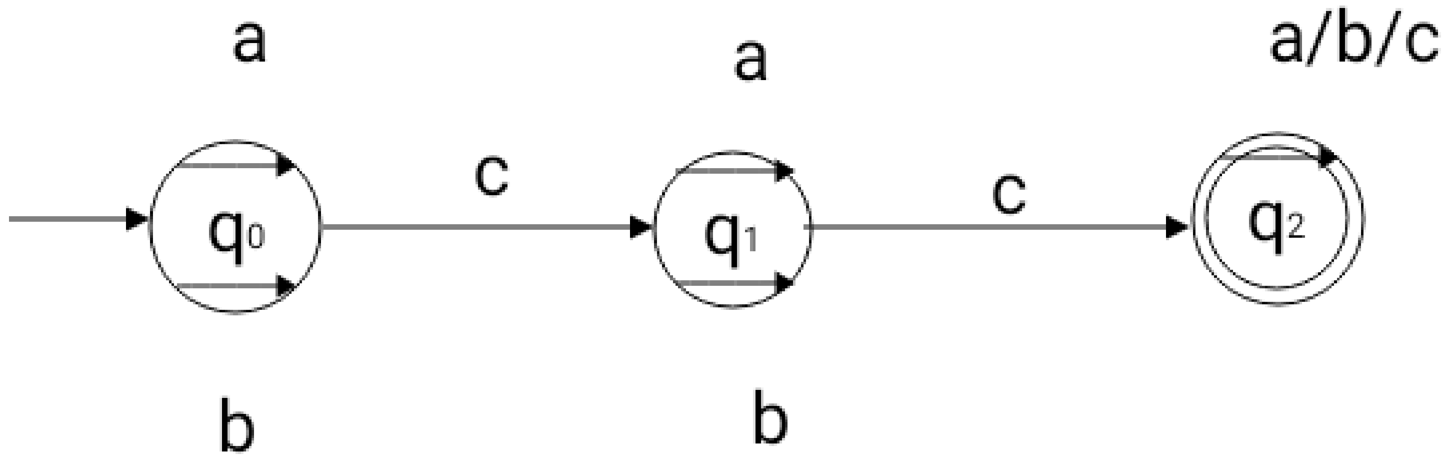


δ:

	0	1
q_0	q_1	q_0
q_1	q_0	q_1

- Revisit example #2:

$Q = \{q_0, q_1, q_2\}$
$\Sigma = \{a, b, c\}$
Start state is q_0
$F = \{q_2\}$



δ:	a	b	c
q_0	q_0	q_0	q_1
q_1	q_1	q_1	q_2
q_2	q_2	q_2	q_2

- Since δ is a function, at each step M has exactly one option.
- It follows that for a given string, there is exactly one computation.

Extension of δ to Strings

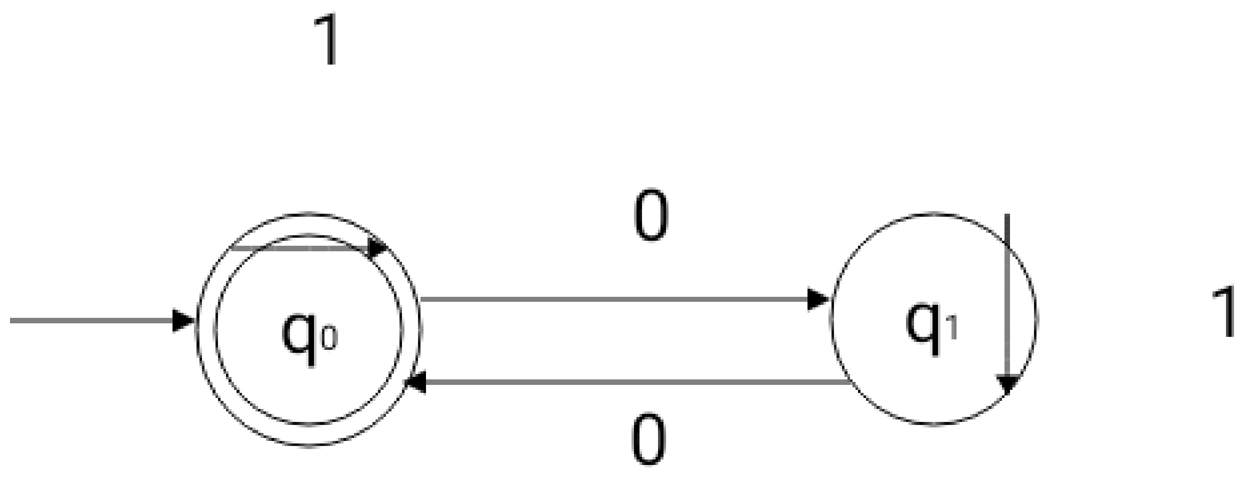
$\hat{\delta} : (Q \times \Sigma^*) \rightarrow Q$

$\hat{\delta}(q,w)$ – The state entered after reading string *w* having started in state *q*.

Formally:

1) $\hat{\delta}(q, \varepsilon) = q$, and
2) For all w in Σ^* and a in Σ
   $\hat{\delta}(q,wa) = \delta(\hat{\delta}(q,w), a)$

- Recall Example #1:



- What is $\hat{\delta}(q_0, 011)$? Informally, it is the state entered by M after processing 011 having started in state q_0.
- Formally:

$\hat{\delta}(q_0, 011) = \delta(\hat{\delta}(q_0,01), 1)$ by rule #2
$= \delta(\delta(\hat{\delta}(q_0,0), 1), 1)$ by rule #2
$= \delta(\delta(\delta(\hat{\delta}(q_0, \lambda), 0), 1), 1)$ by rule #2
$= \delta(\delta(\delta(q_0,0), 1), 1)$ by rule #1
$= \delta(\delta(q_1, 1), 1)$ by definition of δ
$= \delta(q_1, 1)$ by definition of δ
$= q_1$ by definition of δ

- Is 011 accepted? No, since $\hat{\delta}(q_0, 011) = q_1$ is not a final state.

- Note that:

$\hat{\delta}(q,a) = \delta(\hat{\delta}(q, \varepsilon), a)$ by definition of $\hat{\delta}$, rule #2

$= \delta(q, a)$ by definition of $\hat{\delta}$, rule #1

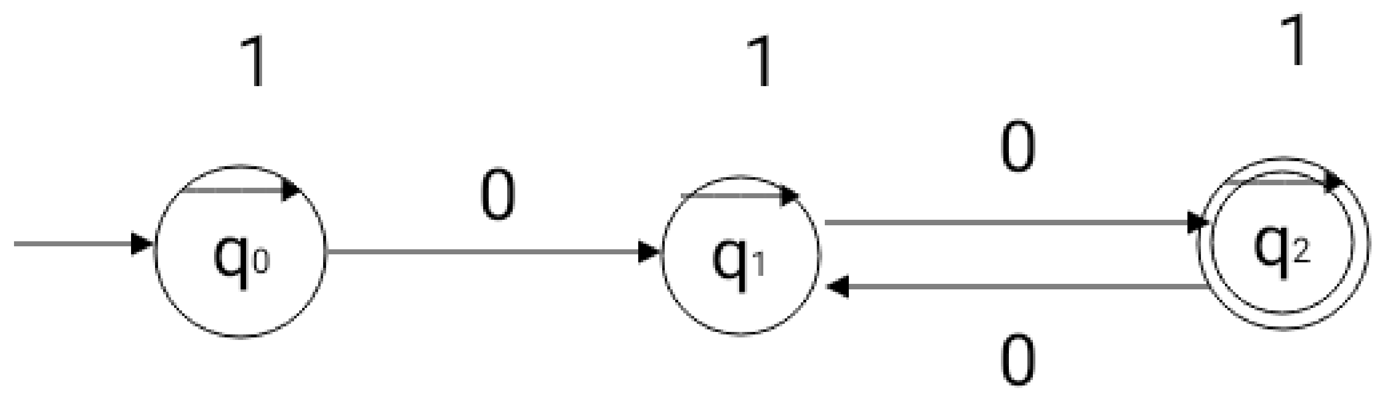
- Therefore:

$\hat{\delta}(q, a_1a_2\ldots a_n) = \delta(\delta(\ldots\delta(\delta(q, a1), a2)\ldots), a_n)$

- However, we will abuse notations, and use δ in place of $\hat{\delta}$:

$\hat{\delta}(q, a_1a_2\ldots a_n) = \delta(q, a_1a_2\ldots a_n)$

- Example #3:


$$\begin{tikzcd} \arrow[r] & q_0 \arrow[loop above, "1"] \arrow[r, "0"] & q_1 \arrow[loop above, "1"] \arrow[r, bend left, "0"] & q_2 \arrow[loop above, "1"] \arrow[l, bend left, "0"] \end{tikzcd}$$

- What is $\delta(q_0, 011)$? Informally, it is the state entered by M after processing 011 having started in state q_0.
- Formally:

$\delta(q_0, 011) = \delta(\delta(q_0,01), 1)$ by rule #2

$= \delta(\delta(\delta(q_0,0), 1), 1)$ by rule #2

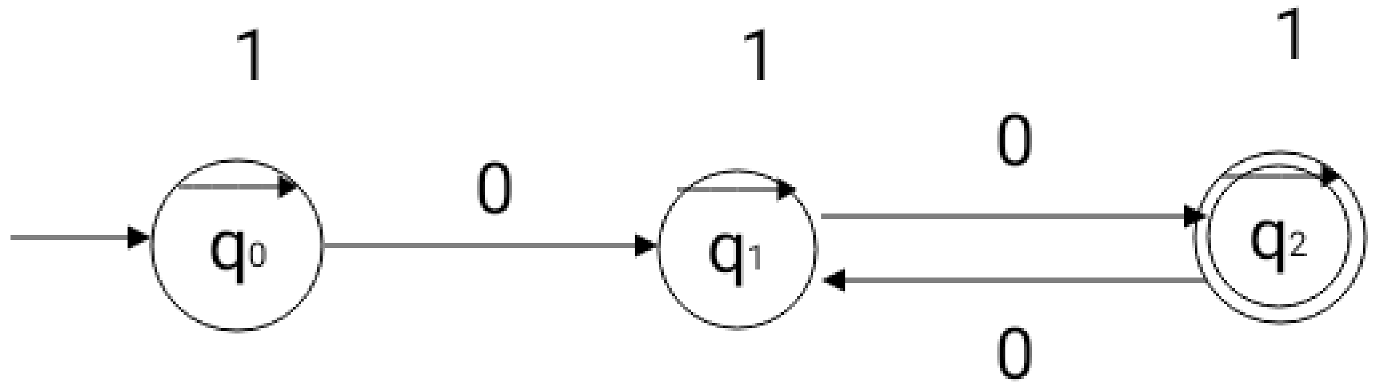
$= \delta(\delta(q_1, 1), 1)$ by definition of δ

$= \delta(q_1, 1)$ by definition of δ

$= q_1$ by definition of δ

- Is 011 accepted? No, since $\delta(q_0, 011) = q_1$ is not a final state.
- *Language?*
- L ={ all strings over {0,1} that has 2 or more *0* symbols}

- Recall Example #3:



- What is $\delta(q_1, 10)$?

$\delta(q_1, 10) = \delta(\delta(q_1,1), 0)$ by rule #2

$= \delta(q_1, 0)$ by definition of δ

$= q_2$ by definition of δ

- Is 10 accepted? No, since $\delta(q_0, 10) = q_1$ is not a final state. The fact that $\delta(q_1, 10) = q_2$ is irrelevant, q1 is not the start state!

Definitions related to DFAs

- Let $M = (Q, \Sigma, \delta, q_0, F)$ be a DFA and let w be in Σ^*. Then w is *accepted* by M iff $\delta(q_0,w) = p$ for some state p in F.

- Let $M = (Q, \Sigma, \delta, q_0, F)$ be a DFA. Then the *language accepted* by M is the set:

$L(M) = \{w \mid w \text{ is in } \Sigma^* \text{ and } \delta(q_0,w) \text{ is in } F\}$

- Another equivalent definition:

$L(M) = \{w \mid w \text{ is in } \Sigma^* \text{ and } w \text{ is accepted by } M\}$

- Let L be a language. Then L is a ***regular language*** iff there exists a DFA M such that $L = L(M)$.

- Let $M_1 = (Q_1, \Sigma_1, \delta_1, q_0, F_1)$ and $M_2 = (Q_2, \Sigma_2, \delta_2, p_0, F_2)$ be DFAs. Then M_1 and M_2 are *equivalent* iff $L(M_1) = L(M_2)$.

- Notes:
  - A DFA $M = (Q, \Sigma, \delta, q_0, F)$ partitions the set Σ^* into two sets: $L(M)$ and Σ^* - $L(M)$.
  - If $L = L(M)$ then L is a subset of $L(M)$ and $L(M)$ is a subset of L (def. of set equality).
  - Similarly, if $L(M_1) = L(M_2)$ then $L(M_1)$ is a subset of $L(M_2)$ and $L(M_2)$ is a subset of $L(M_1)$.
  - Some languages are regular, others are not. For example, if

Regular: L_1 = {x | x is a string of 0's and 1's containing an even number of 1's} and

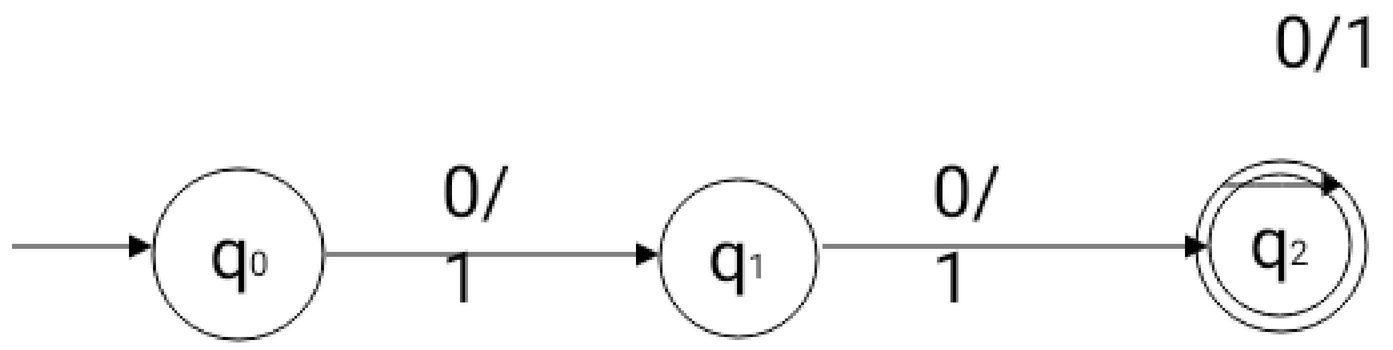
Not-regular: $L_2 = \{x \mid x = 0^n 1^n$ for some $n >= 0\}$

- *Can you write a program to "simulate" a given DFA, or any arbitrary input DFA?*

- Question we will address later:
  - How do we determine whether or not a given language is regular?

- Give a DFA M such that:

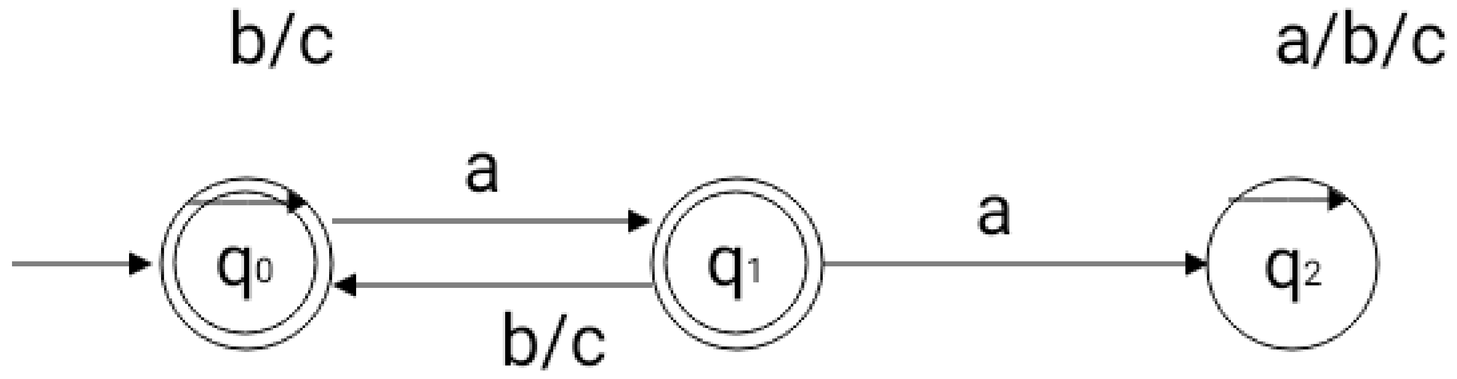
L(M) = {x | x is a string of 0's and 1's and |x| >= 2}



Prove this by induction

- Give a DFA M such that:

L(M) = {x | x is a string of (zero or more) a's, b's and c's such that x does *not* contain the substring *aa*}



Logic:

In Start state (q0): b's and c's: ignore – stay in same state

q0 is also "accept" state

First 'a' appears: get ready (q1) to reject

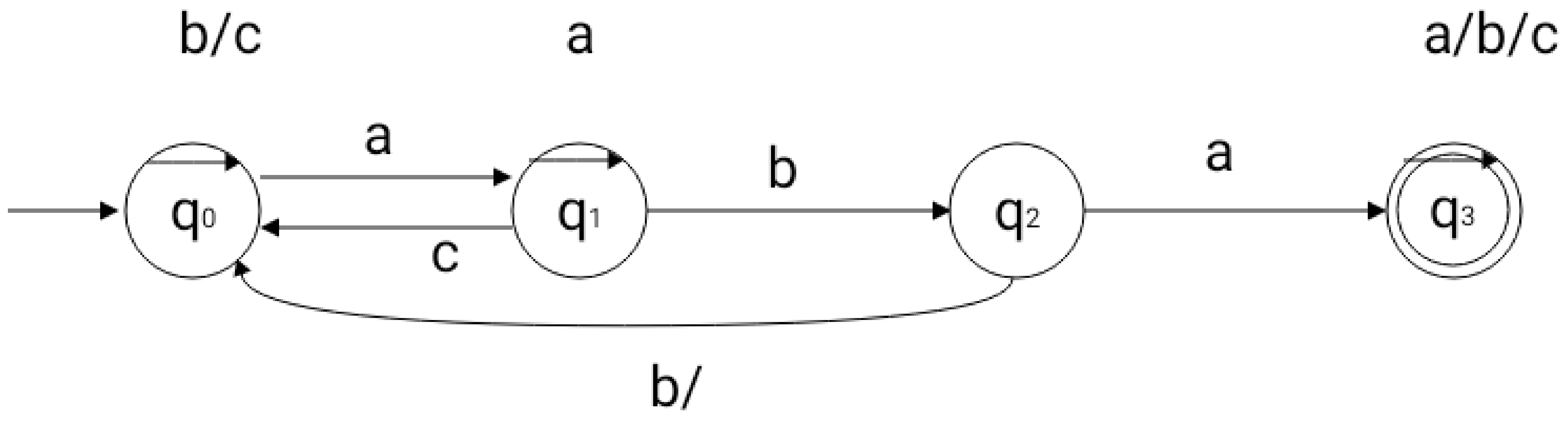
But followed by a 'b' or 'c': go back to start state q0

When second 'a' appears after the "ready" state: go to reject state q2

Ignore everything after getting to the "reject" state q2

- Give a DFA M such that:

  L(M) = {x | x is a string of a's, b's and c's such that x contains the substring *aba*}



Logic: acceptance is straight forward, progressing on each expected symbol

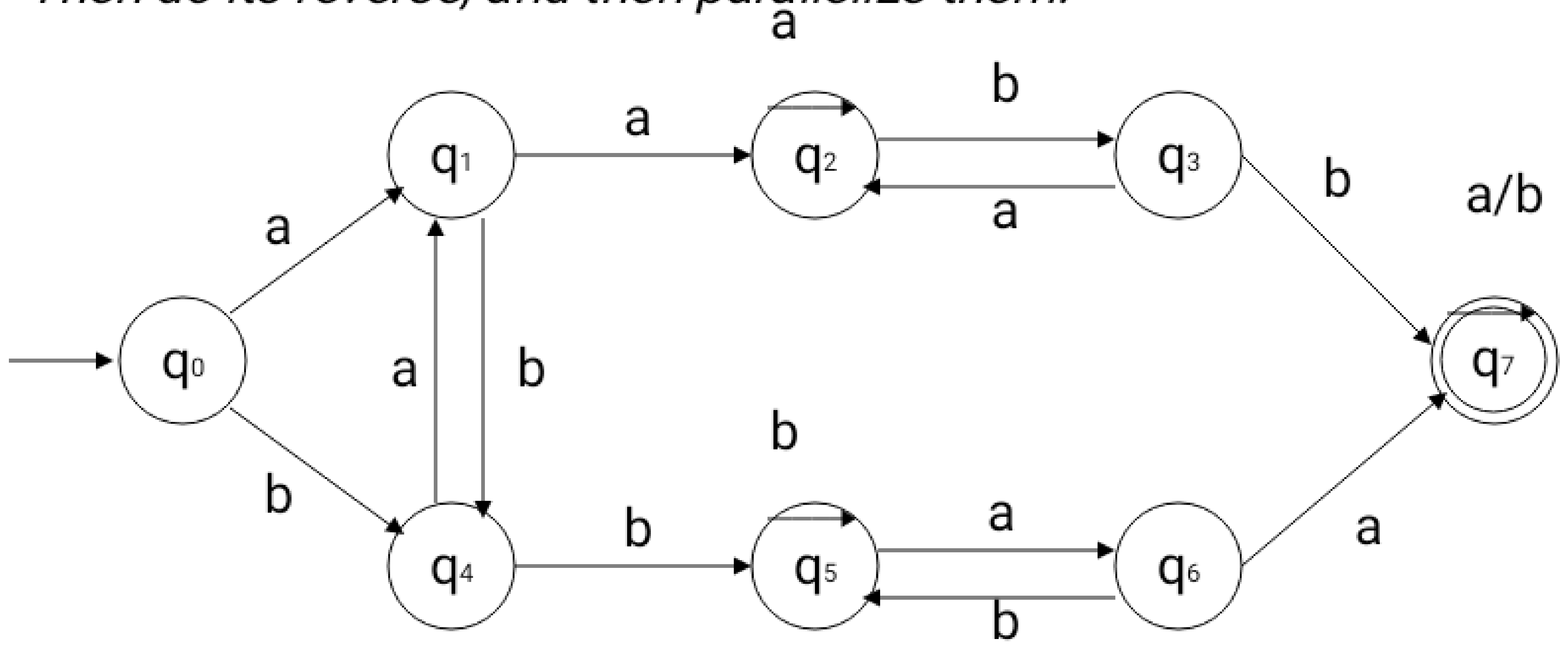
However, rejection needs special care, in each state (for DFA, we will see this becomes easier in NFA, non-deterministic machine)

- Give a DFA M such that:

L(M) = {x | x is a string of a's and b's such that x contains both *aa* and *bb*}

First do, for a language where 'aa' comes before 'bb'

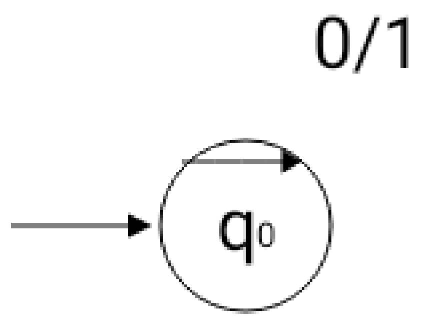
Then do its reverse; and then parallelize them.

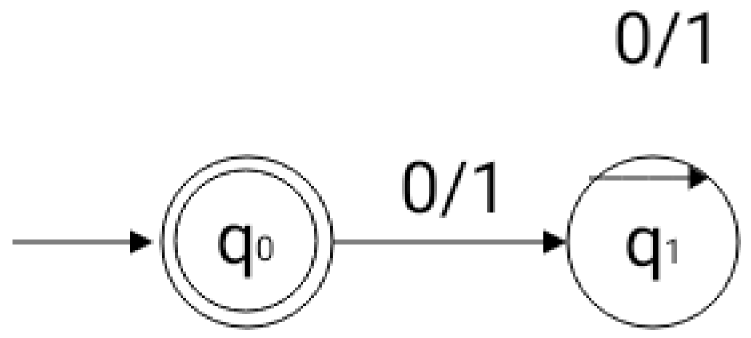


Remember, you may have multiple "final" states, but only one "start" state

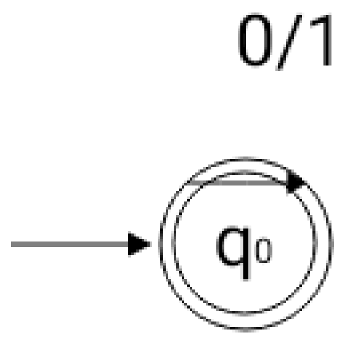
- Let $\Sigma = \{0, 1\}$. Give DFAs for $\{\}$, $\{\varepsilon\}$, Σ^*, and Σ^+.

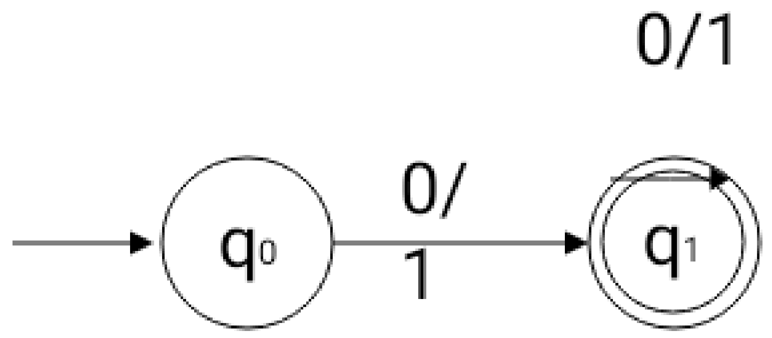
For $\{\}$:   For $\{\varepsilon\}$:



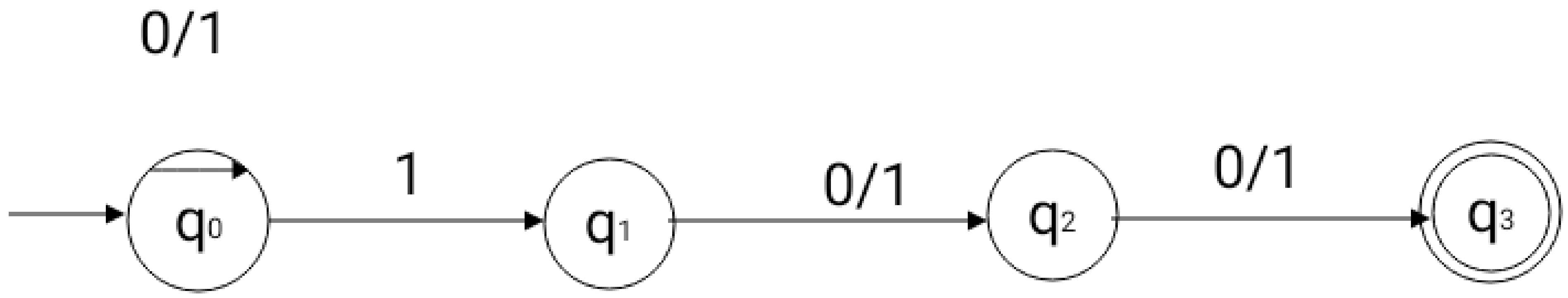


For Σ^*:   For Σ^+:





- Problem: Third symbol from last is *1*



$$\begin{tikzcd}
\arrow[r] & q_0 \arrow[loop above, "0/1"] \arrow[r, "1"] & q_1 \arrow[r, "0/1"] & q_2 \arrow[r, "0/1"] & q_3
\end{tikzcd}$$

Is this a DFA?

No, but it is a *Non-deterministic Finite Automaton*

Nondeterministic Finite State Automata (NFA)

- An NFA is a five-tuple:

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

Q A <u>finite</u> set of states
Σ A <u>finite</u> input alphabet
q_0 The initial/starting state, q_0 is in Q
F A set of final/accepting states, which is a subset of Q
δ A transition function, which is a total function from Q x Σ to 2^Q

$\delta: (Q \times \Sigma) \rightarrow$ **2^Q** :2^Q is the power set of Q, the set of *all subsets* of Q
δ(q,s) :The set of all states p such that there is a transition labeled s from q to p

δ(q,s) is a function from Q x S to 2^Q (but not only to Q)

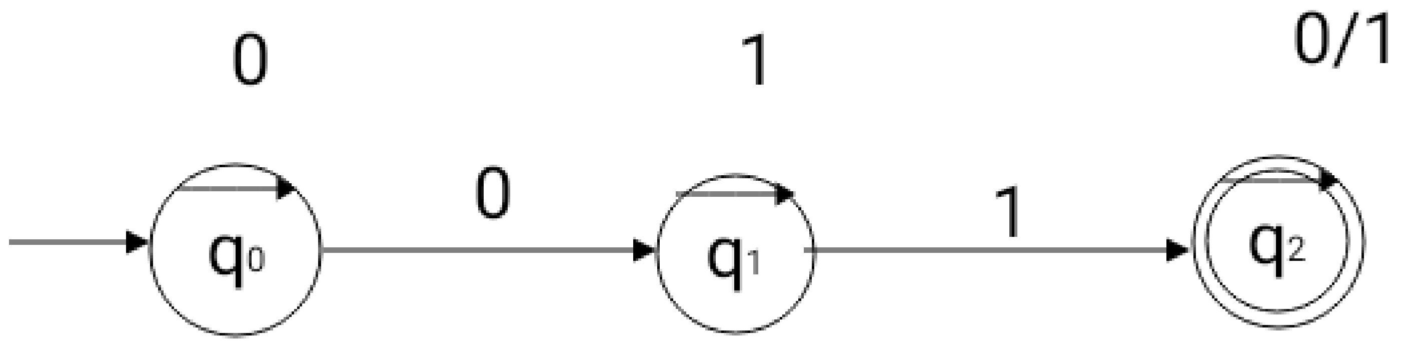
- Example #1: one or more 0's followed by one or more 1's

$Q = \{q_0, q_1, q_2\}$

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

Start state is q_0

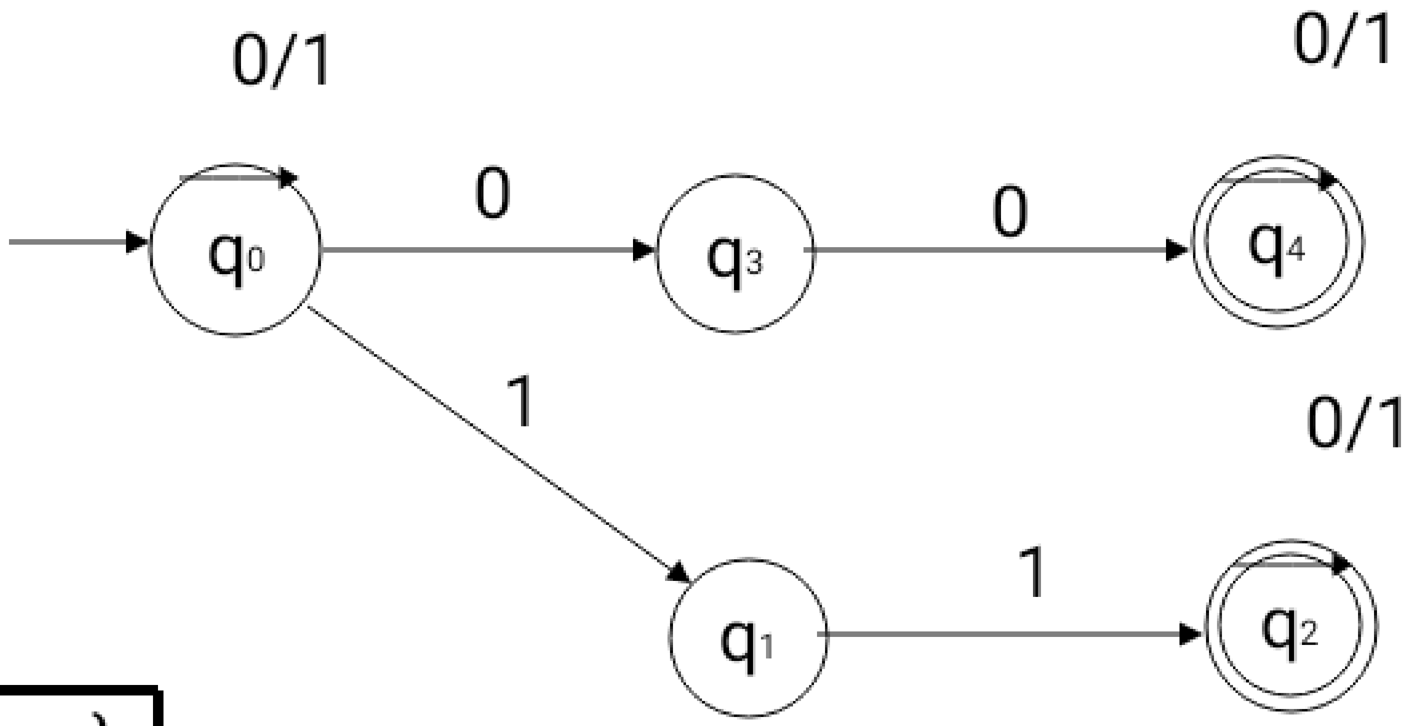
$F = \{q_2\}$



δ:	0	1
q_0	$\{q_0, q_1\}$	$\{\}$
q_1	$\{\}$	$\{q_1, q_2\}$
q_2	$\{q_2\}$	$\{q_2\}$

- Example #2: pair of 0's *or* pair of 1's as substring

$Q = \{q_0, q_1, q_2, q_3, q_4\}$
$\Sigma = \{0, 1\}$
Start state is q_0
$F = \{q_2, q_4\}$


$$\begin{tikzcd}
\arrow[r] & q_0 \arrow[loop above, "0/1"] \arrow[r, "0"] \arrow[dr, "1"] & q_3 \arrow[r, "0"] & q_4 \arrow[loop above, "0/1"] \\
& & q_1 \arrow[r, "1"] & q_2 \arrow[loop above, "0/1"]
\end{tikzcd}$$

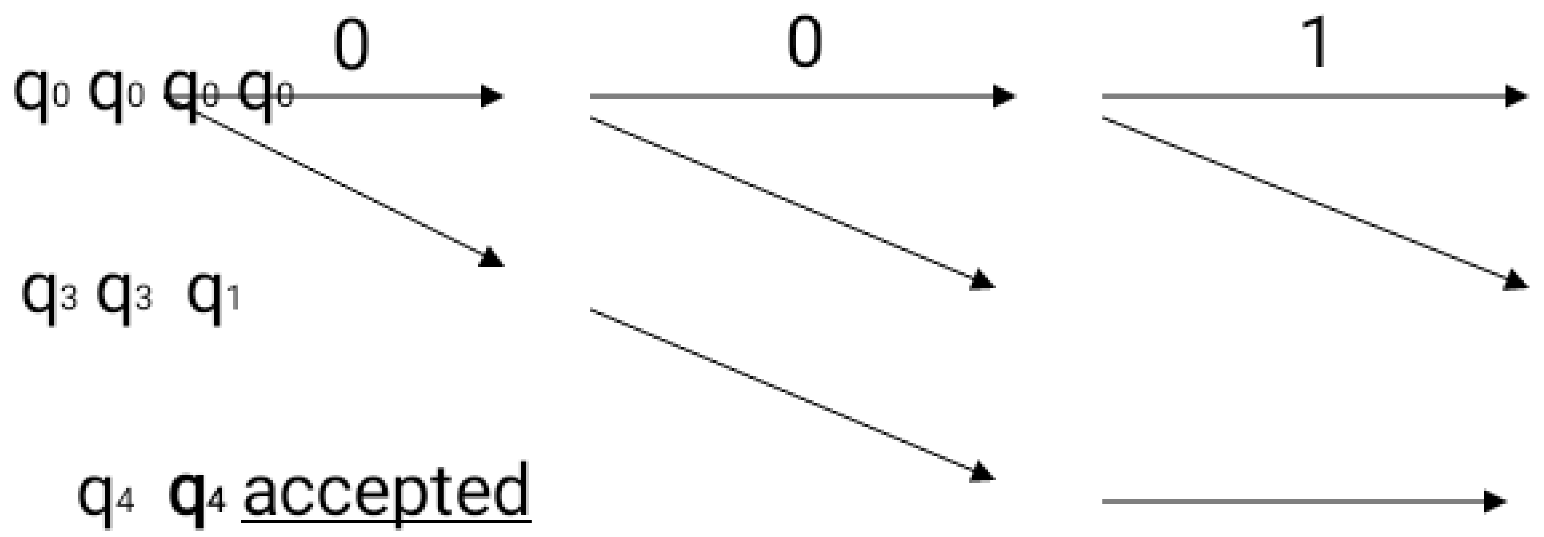
δ: 0 1

q_0	$\{q_0, q_3\}$	$\{q_0, q_1\}$
q_1	$\{\}$	$\{q_2\}$
q_2	$\{q_2\}$	$\{q_2\}$
q_3	$\{q_4\}$	$\{\}$
q_4	$\{q_4\}$	$\{q_4\}$

- Notes:
  - δ(q,s) may not be defined for some *q* and *s* (*what does that mean*?)
  - δ(q,s) may map to multiple *q*'s
  - A string is said to be accepted *if there exists* a path from q_0 to some state in *F*
  - A string is rejected *if there exist NO path* to any state in *F*
  - The language accepted by an NFA is the set of all accepted strings

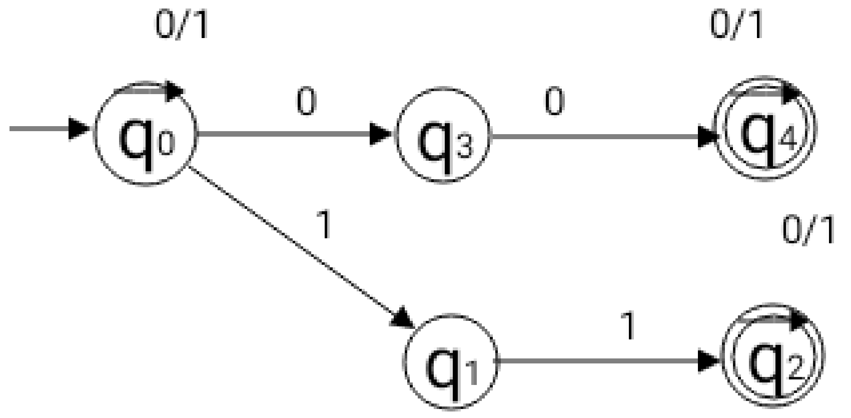
- Question: How does an NFA find the correct/accepting path for a given string?
  - NFAs are a non-intuitive computing model
  - You may use *backtracking* to find if there exists a path to a final state (following slide)
- Why NFA?
  - We are *primarily* interested in NFAs as language defining capability, i.e., do NFAs accept languages that DFAs do not?
  - Other secondary questions include practical ones such as whether or not NFA is easier to develop, or how does one implement NFA

- Determining if a given NFA (example #2) accepts a given string (001) can be done algorithmically:

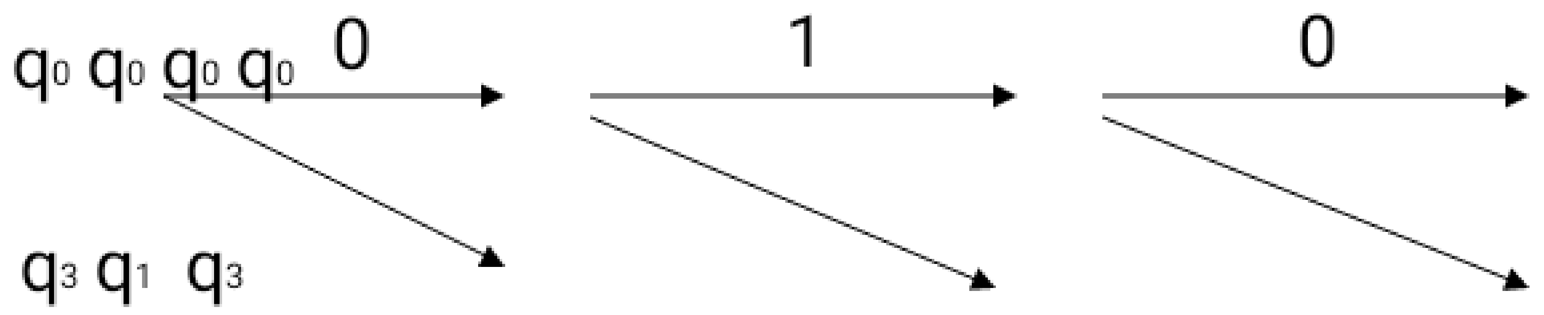


- Each level will have at most *n* states:

Complexity: O(|x|*n), for running over a string *x*

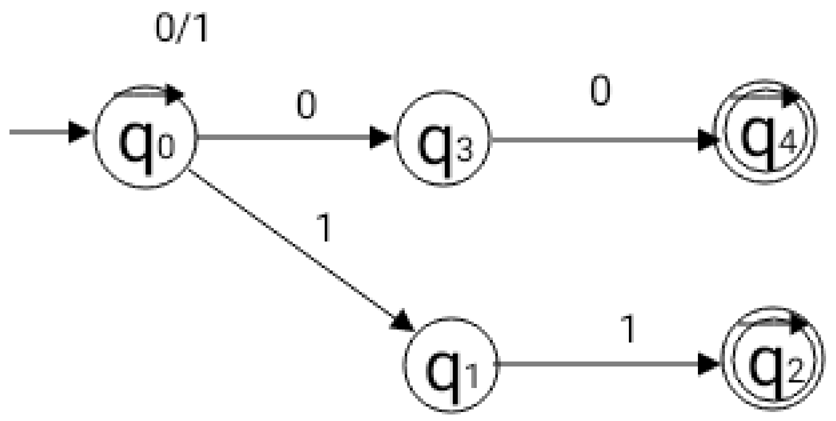


- Another example (010):



not accepted

- All paths have been explored, and none lead to an accepting state.

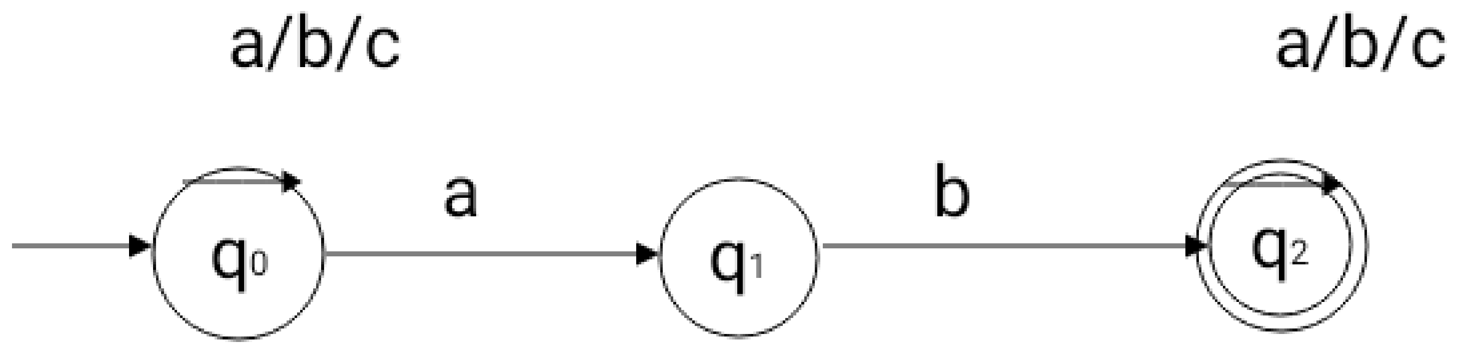


- Question: Why non-determinism is useful?
  - Non-determinism = Backtracking
  - Compressed information
  - Non-determinism hides backtracking
  - Programming languages, e.g., Prolog, hides backtracking => Easy to program at a higher level: *what we want to do, rather than how to do it*
  - Useful in algorithm complexity study

  - Is NDA more “powerful” than DFA, i.e., accepts type of languages that any DFA cannot?

- Let Σ = {a, b, c}. Give an NFA M that accepts:

L = {*x* | *x* is in Σ^* and *x* contains *ab*}


$$\begin{tikzcd}
\arrow[r] & q_0 \arrow[loop above, "a/b/c"] \arrow[r, "a"] & q_1 \arrow[r, "b"] & q_2 \arrow[loop above, "a/b/c"]
\end{tikzcd}$$

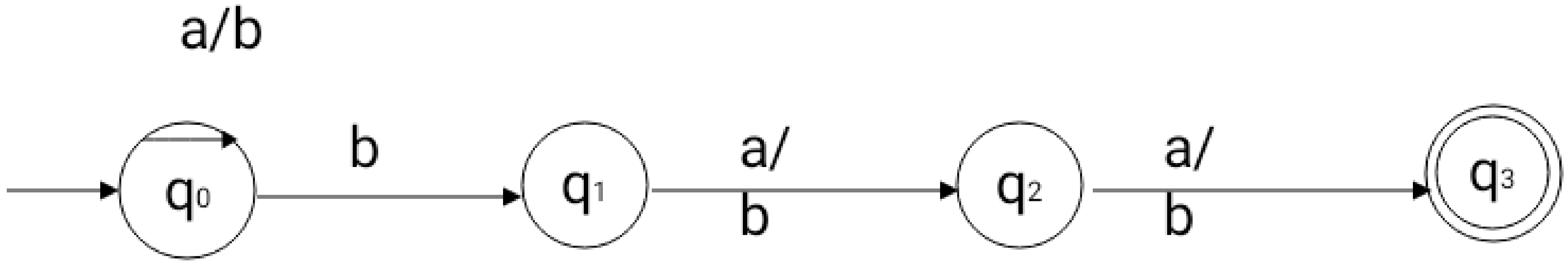
Is L a subset of L(M)? Or, does M accepts all string in L?

Is L(M) a subset of L? Or, does M rejects all strings not in L?

- Is an NFA necessary? Can you draw a DFA for this L?
- Designing NFAs is not as trivial as it seems: easy to create bug accepting string outside language

- Let Σ = {a, b}. Give an NFA M that accepts:

L = {*x* | *x* is in Σ^* and the third to the last symbol in *x* is *b*}

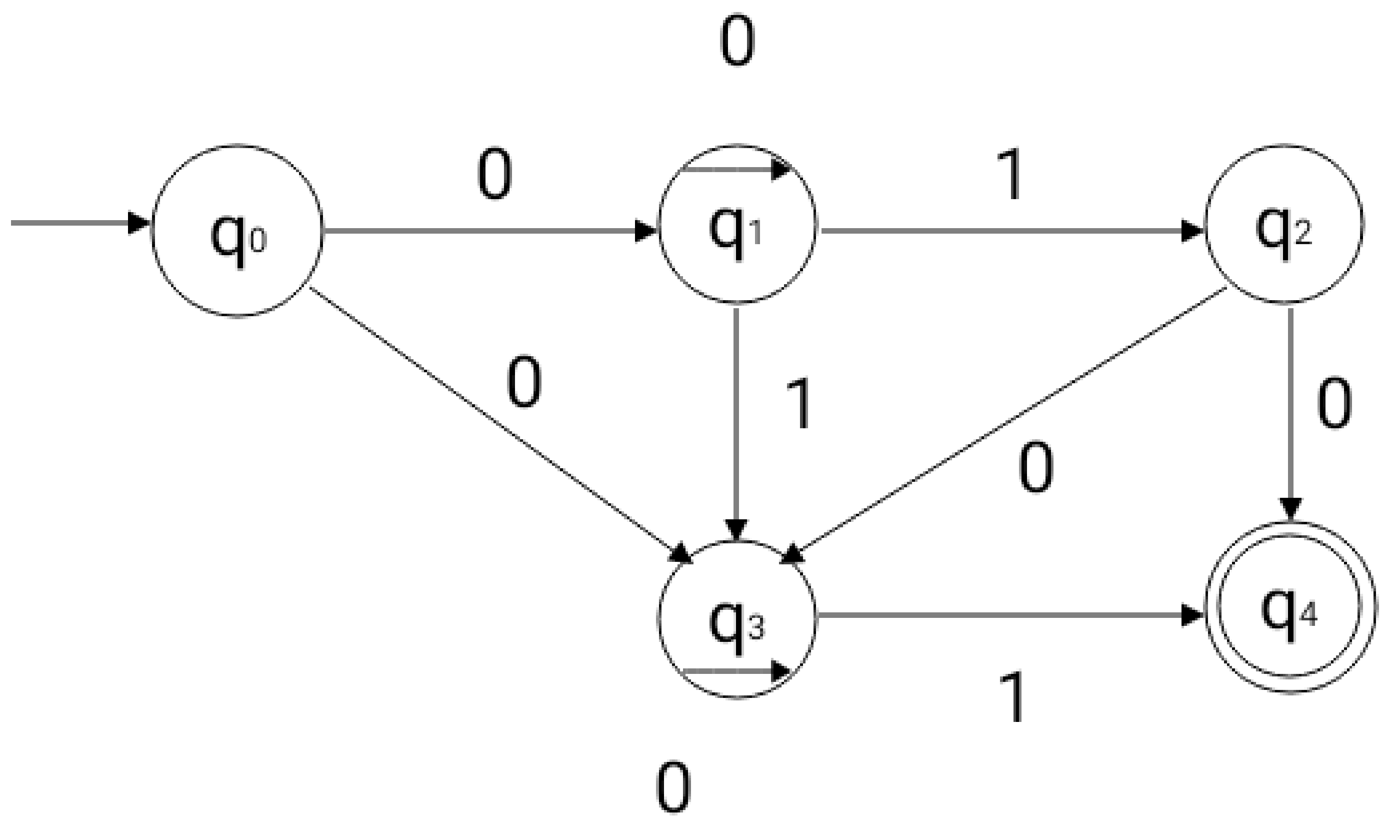


Is L a subset of L(M)?
Is L(M) a subset of L?

- Give an equivalent DFA as an exercise.

Extension of δ to Strings and Sets of States

- What we currently have: $\delta : (Q \times \Sigma) \rightarrow 2^Q$
- What we want (why?): $\delta : (2^Q \times \Sigma^*) \rightarrow 2^Q$
- We will do this in two steps, which will be slightly different from the book, and we will make use of the following NFA.



Extension of δ to Strings and Sets of States

- Step #1:

Given $\delta: (Q \times \Sigma) \to 2^Q$ define $\delta^{\#}: (2^Q \times \Sigma) \to 2^Q$ as follows:

1) $\delta^{\#}(R, a) = \quad \delta(q, a)$ for all subsets R of Q, and symbols a in Σ

- Note that:

$\delta^{\#}(\{p\},a) = \quad \delta(q, a)$ by definition of $\delta^{\#}$, rule #1 above
$= \delta(p, a)$

- Hence, we can use δ for $\delta^{\#}$

$\delta(\{q_0, q_2\}, 0)$ These now make sense, but previously
$\delta(\{q_0, q_1, q_2\}, 0)$ they did not.

- Example:

$\delta(\{q_0, q_2\}, 0) = \delta(q_0, 0) \cup \delta(q_2, 0)$

$= \{q_1, q_3\} \cup \{q_3, q_4\}$

$= \{q_1, q_3, q_4\}$

$\delta(\{q_0, q_1, q_2\}, 1) = \delta(q_0, 1) \cup \delta(q_1, 1) \cup \delta(q_2, 1)$

$= \{\} \cup \{q_2, q_3\} \cup \{\}$

$= \{q_2, q_3\}$

- Step #2:

Given $\delta: (2^Q \times \Sigma) \rightarrow 2^Q$ define $\hat{\delta}: (2^Q \times \Sigma^*) \rightarrow 2^Q$ as follows:

$\hat{\delta}(R,w)$ – The set of states M could be in after processing string w, having started from any state in R.

Formally:

2) $\hat{\delta}(R, \varepsilon) = R$ for any subset R of Q
3) $\hat{\delta}(R,wa) = \delta(\hat{\delta}(R,w), a)$ for any w in Σ^*, a in Σ, and subset R of Q

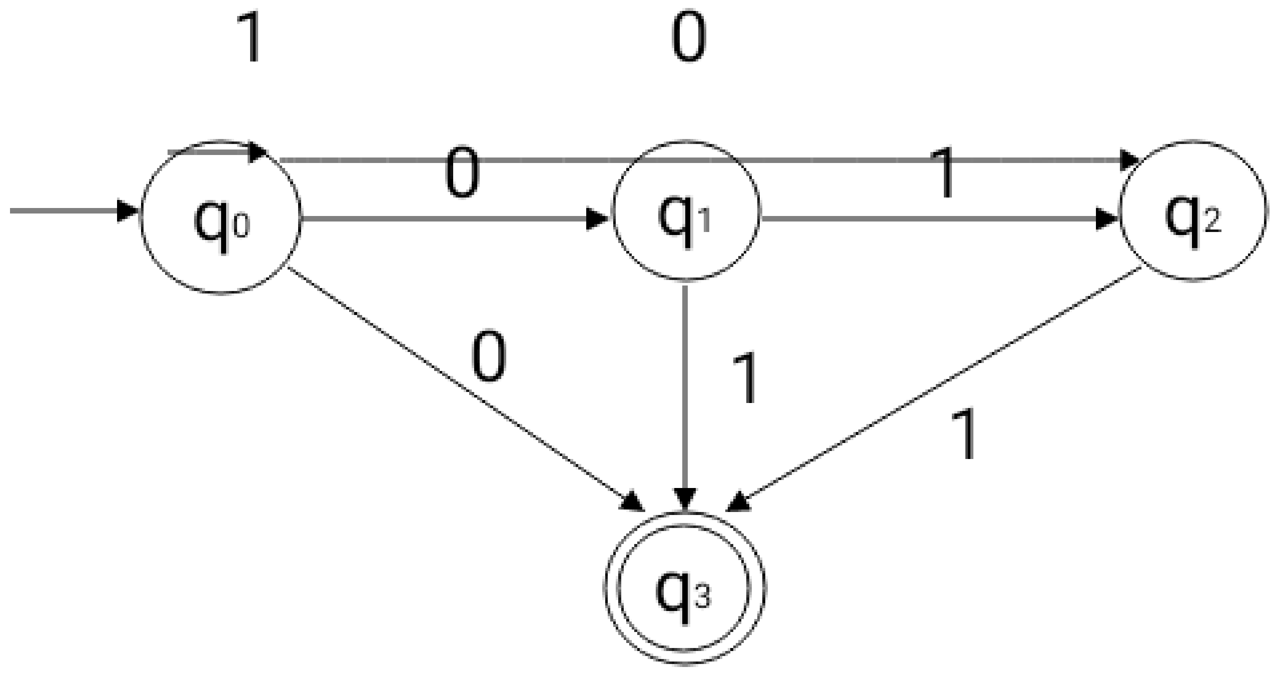
- Note that:

$\hat{\delta}(R, a) = \delta(\hat{\delta}(R, \varepsilon), a)$ by definition of $\hat{\delta}$, rule #3 above
$= \delta(R, a)$ by definition of $\hat{\delta}$, rule #2 above

- Hence, we can use δ for $\hat{\delta}$

$\delta(\{q_0, q_2\}, 0110)$ These now make sense, but previously
$\delta(\{q_0, q_1, q_2\}, 101101)$ they did not.

- Example:



What is $\delta(\{q_0\}, 10)$?

Informally: The set of states the NFA could be in after processing 10, having started in state q_0, i.e., $\{q_1, q_2, q_3\}$.

Formally: $\delta(\{q_0\}, 10) = \delta(\delta(\{q_0\}, 1), 0)$

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

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

Is 10 accepted? Yes!

- Example:

What is $\delta(\{q_0, q_1\}, 1)$?

$\delta(\{q_0, q_1\}, 1) = \delta(\{q_0\}, 1) \cup \delta(\{q_1\}, 1)$

$= \{q_0\} \cup \{q_2, q_3\}$

$= \{q_0, q_2, q_3\}$

What is $\delta(\{q_0, q_2\}, 10)$?

$\delta(\{q_0, q_2\}, 10) = \delta(\delta(\{q_0, q_2\}, 1), 0)$

$= \delta(\delta(\{q_0\}, 1)$ U $\delta(\{q_2\}, 1), 0)$

$= \delta(\{q_0\} \cup \{q_3\}, 0)$

$= \delta(\{q_0,q_3\}, 0)$

$= \delta(\{q_0\}, 0) \cup \delta(\{q_3\}, 0)$

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

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

- Example:

$\delta(\{q_0\}, 101) = \delta(\delta(\{q_0\}, 10), 1)$

$= \delta(\delta(\delta(\{q_0\}, 1), 0), 1)$

$= \delta(\delta(\{q_0\}, 0), 1)$

$= \delta(\{q_1, q_2, q_3\}, 1)$

$= \delta(\{q_1\}, 1) \cup \delta(\{q_2\}, 1) \cup \delta(\{q_3\}, 1)$

$= \{q_2, q_3\} \cup \{q_3\} \cup \{\}$

$= \{q_2, q_3\}$

Is 101 accepted? Yes! q_3 is a final state.

Definitions for NFAs

- Let M = (Q, Σ, δ,q_0,F) be an NFA and let w be in Σ^*. Then w is *accepted* by M iff δ({q_0}, w) contains at least one state in F.

- Let M = (Q, Σ, δ,q_0,F) be an NFA. Then the *language accepted* by M is the set:

L(M) = {w | w is in Σ^* and δ({q_0},w) contains at least one state in F}

- Another equivalent definition:

L(M) = {w | w is in Σ^* and w is accepted by M}

Equivalence of DFAs and NFAs

- Do DFAs and NFAs accept the same *class* of languages?
  - Is there a language L that is accepted by a DFA, but not by any NFA?
  - Is there a language L that is accepted by an NFA, but not by any DFA?

- Observation: Every DFA is an NFA, DFA is only restricted NFA.

- Therefore, if L is a regular language then there exists an NFA M such that L = L(M).

- It follows that NFAs accept all regular languages.

- But do NFAs accept more?

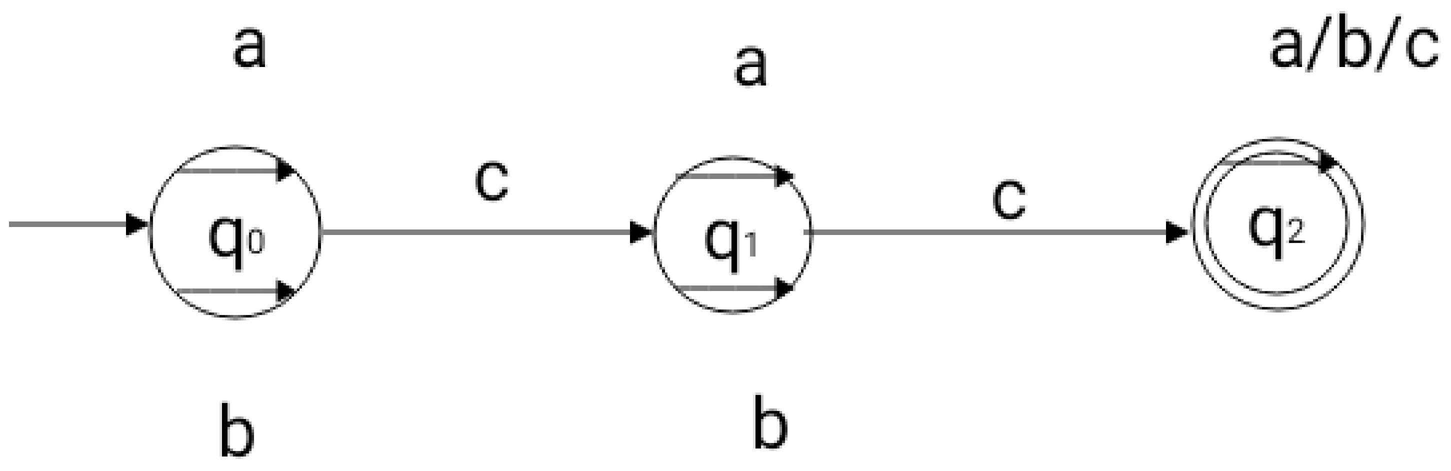
- Consider the following DFA: 2 or more c's

$Q = \{q_0, q_1, q_2\}$

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

Start state is q_0

$F = \{q_2\}$



δ:	a	b	c
q_0	q_0	q_0	q_1
q_1	q_1	q_1	q_2
q_2	q_2	q_2	q_2

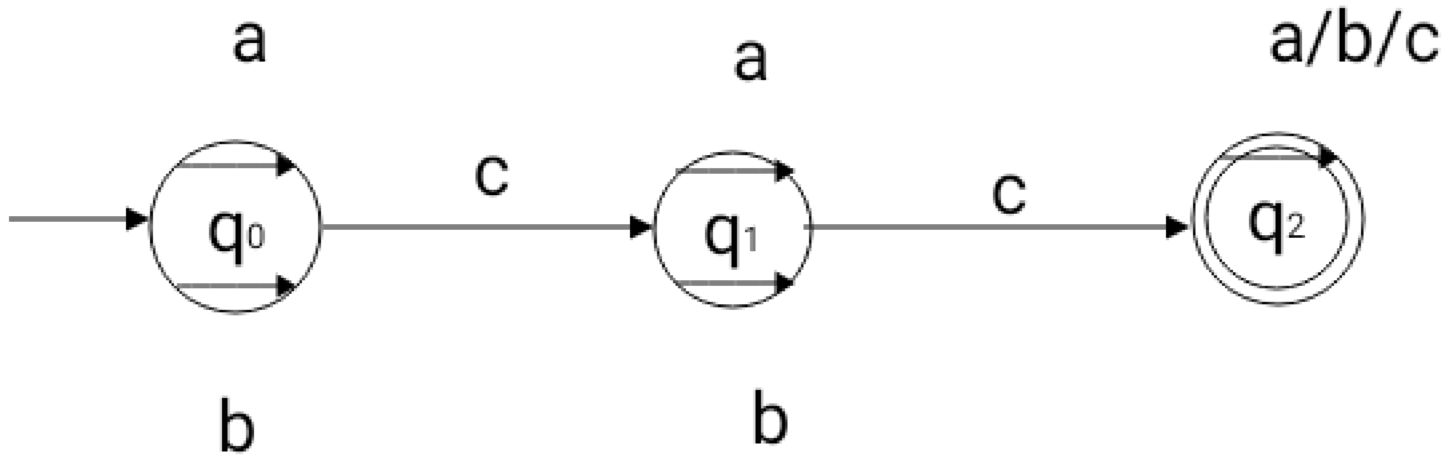
- An Equivalent NFA:

$Q = \{q_0, q_1, q_2\}$

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

Start state is q_0

$F = \{q_2\}$



δ:	a	b	c
q_0	$\{q_0\}$	$\{q_0\}$	$\{q_1\}$
q_1	$\{q_1\}$	$\{q_1\}$	$\{q_2\}$
q_2	$\{q_2\}$	$\{q_2\}$	$\{q_2\}$

- **Lemma 1:** Let M be an DFA. Then there exists a NFA M’ such that L(M) = L(M’).

- **Proof:** Every DFA is an NFA. Hence, if we let M’ = M, then it follows that L(M’) = L(M).

The above is just a formal statement of the observation from the previous slide.

- **Lemma 2:** Let M be an NFA. Then there exists a DFA M' such that L(M) = L(M').

- **Proof:** (sketch)

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

Define a DFA $M' = (Q', \Sigma, \delta', q'_0, F')$ as:

$Q' = 2^Q$ Each state in M' corresponds to a
$= \{Q_0, Q_1, \ldots,\}$ subset of states from M

where $Q_u = [q_{i0}, q_{i1}, \ldots q_{ij}]$

$F' = \{Q_u \mid Q_u$ contains at least one state in $F\}$

$q'_0 = [q_0]$

$\delta'(Q_u, a) = Q_v$ iff $\delta(Q_u, a) = Q_v$

- Example: empty string or start and end with 0

$$\begin{tikzcd}
\arrow[r] & \boxed{q_0} \arrow[r, bend left, "0"] & q_1 \arrow[l, bend left, "0"] \arrow[loop above, "0/1"]
\end{tikzcd}$$

$Q = \{q_0, q_1\}$

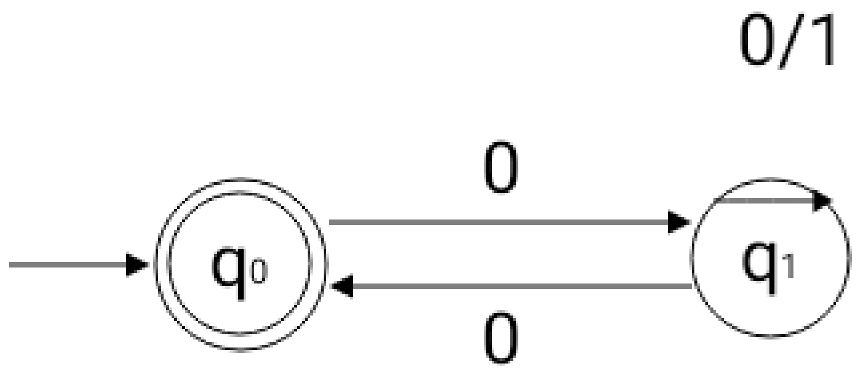
$\Sigma = \{0, 1\}$

Start state is q_0

$F = \{q_0\}$

δ:	0	1
q_0	$\{q_1\}$	$\{\}$
q_1	$\{q_0, q_1\}$	$\{q_1\}$

- Example of creating a DFA out of an NFA (as per the constructive proof):



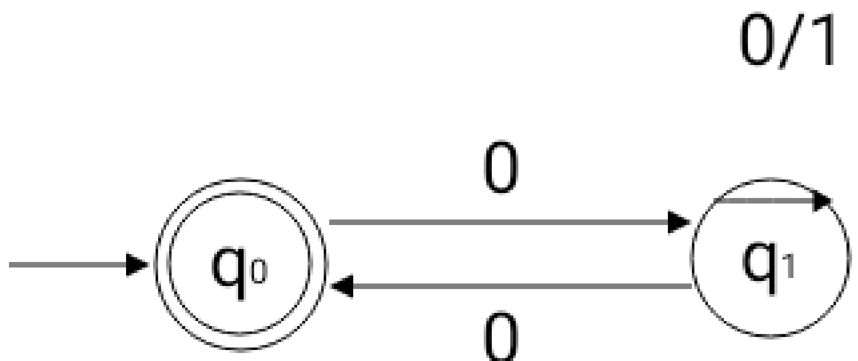
-->q_0

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

δ for DFA: 0 1 q_1

->q_0	$\{q_1\}$ write as $[q_1]$	{} write as [ ]
$[q_1]$		
[ ]		

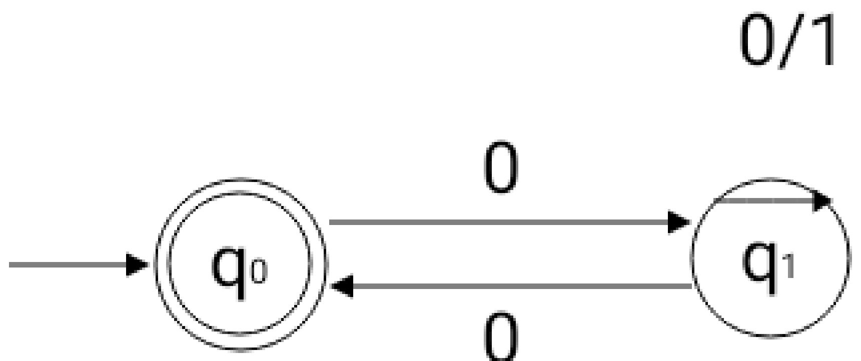
- Example of creating a DFA out of an NFA (as per the constructive proof):



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

δ:	0	1
->q_0	$\{q_1\}$ write as $[q_1]$	{}
$[q_1]$	$\{q_0,q_1\}$ write as $[q_{01}]$	$\{q_1\}$
[ ]		
$[q_{01}]$		

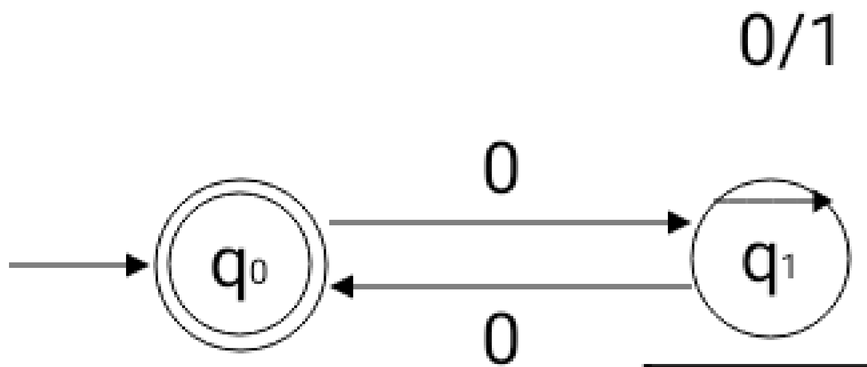
- Example of creating a DFA out of an NFA (as per the constructive proof):



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

δ:	0	1
->q_0	$\{q_1\}$ write as $[q_1]$	$\{\}$
$[q_1]$	$\{q_0,q_1\}$ write as $[q_{01}]$	$\{q_1\}$
[ ]	[ ]	[ ]
$[q_{01}]$		

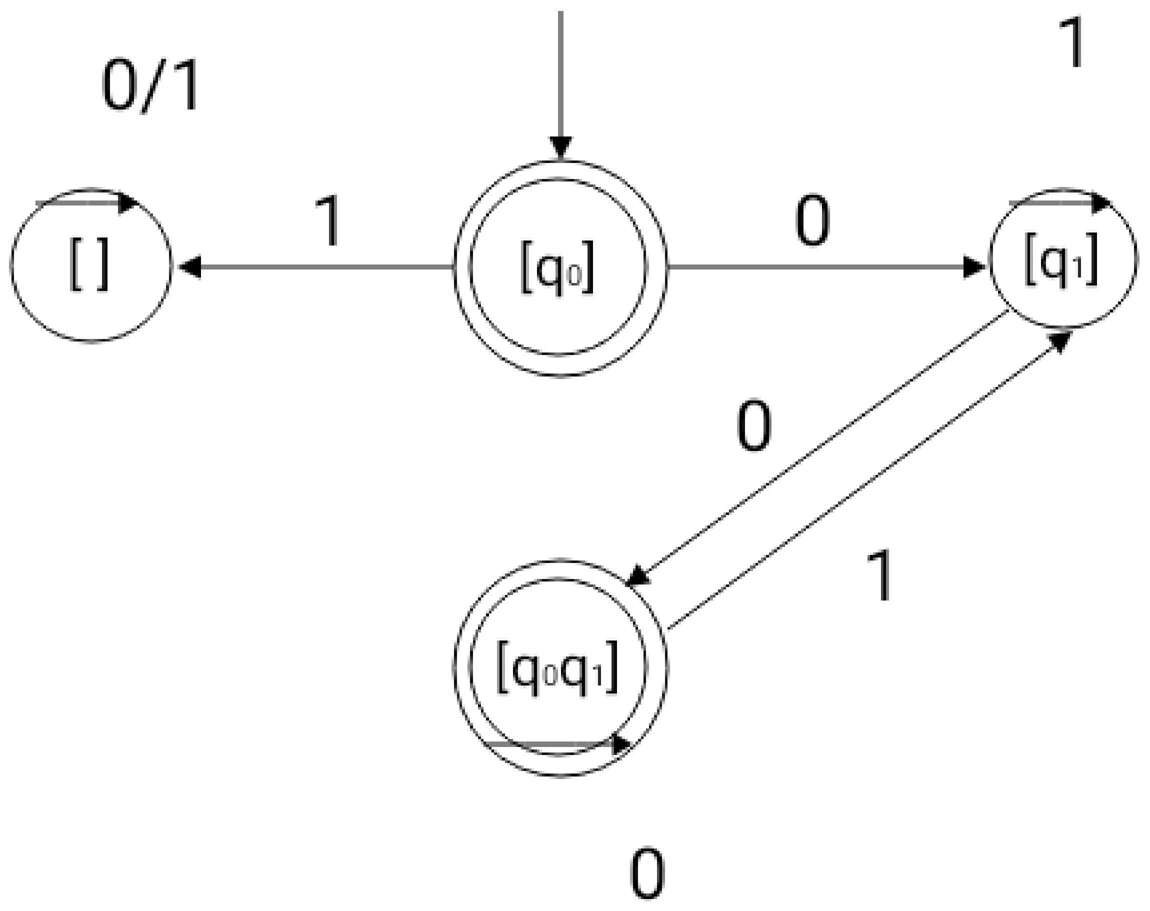
- Example of creating a DFA out of an NFA (as per the constructive proof):



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

δ:	0	1
->q_0	$\{q_1\}$ write as $[q_1]$	$\{\}$
$[q_1]$	$\{q_0,q_1\}$ write as $[q_{01}]$	$\{q_1\}$
[ ]	[ ]	[ ]
$[q_{01}]$	$[q_{01}]$	$[q_1]$

- Construct DFA M' as follows:



$\delta(\{q_0\}, 0) = \{q_1\} \Rightarrow \delta'([q_0], 0) = [q_1]$
$\delta(\{q_0\}, 1) = \{\} \Rightarrow \delta'([q_0], 1) = [\ ]$
$\delta(\{q_1\}, 0) = \{q_0, q_1\} \Rightarrow \delta'([q_1], 0) = [q_0q_1]$
$\delta(\{q_1\}, 1) = \{q_1\} \Rightarrow \delta'([q_1], 1) = [q_1]$
$\delta(\{q_0, q_1\}, 0) = \{q_0, q_1\} \Rightarrow \delta'([q_0q_1], 0) = [q_0q_1]$
$\delta(\{q_0, q_1\}, 1) = \{q_1\} \Rightarrow \delta'([q_0q_1], 1) = [q_1]$
$\delta(\{\}, 0) = \{\} \Rightarrow \delta'([\ ], 0) = [\ ]$
$\delta(\{\}, 1) = \{\} \Rightarrow \delta'([\ ], 1) = [\ ]$

- **Theorem:** Let L be a language. Then there exists an DFA M such that L = L(M) iff there exists an NFA M' such that L = L(M').

- **Proof:**

(if) Suppose there exists an NFA M' such that L = L(M'). Then by Lemma 2 there exists an DFA M such that L = L(M).

(only if) Suppose there exists an DFA M such that L = L(M). Then by Lemma 1 there exists an NFA M' such that L = L(M').

- **Corollary:** The NFAs define the regular languages.

- Note: Suppose R = {}

δ(R, 0) = δ(δ(R, ε), 0)
= δ(R, 0)
= δ(q, 0)
= {} Since R = {}

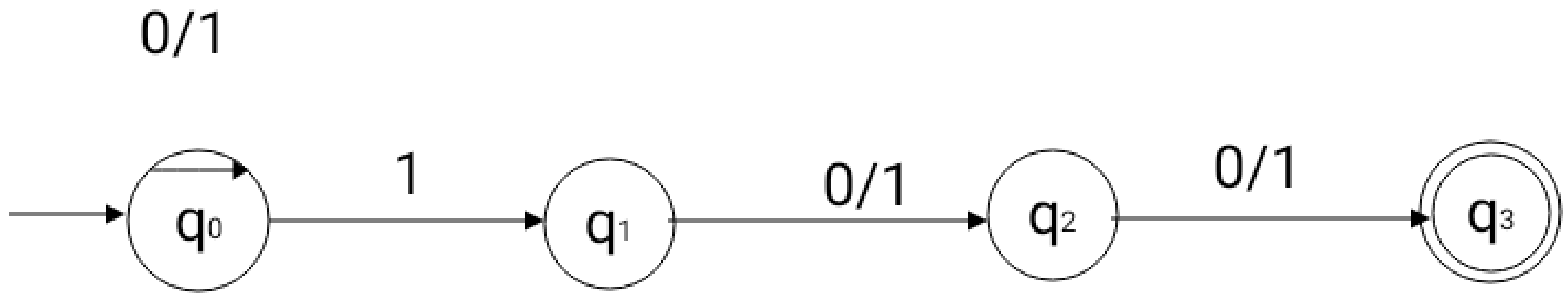
- Exercise - Convert the following NFA to a DFA:

Q = {q_0, q_1, q_2} δ: 0 1
Σ = {0, 1}
Start state is q_0 q_0
F = {q_0}
q_1

q_2

{q_0, q_1}	{ }
{q_1}	{q_2}
{q_2}	{q_2}

- Problem: Third symbol from last is 1



Now, can you convert this NFA to a DFA?

NFAs with ε Moves

- An NFA-ε is a five-tuple:

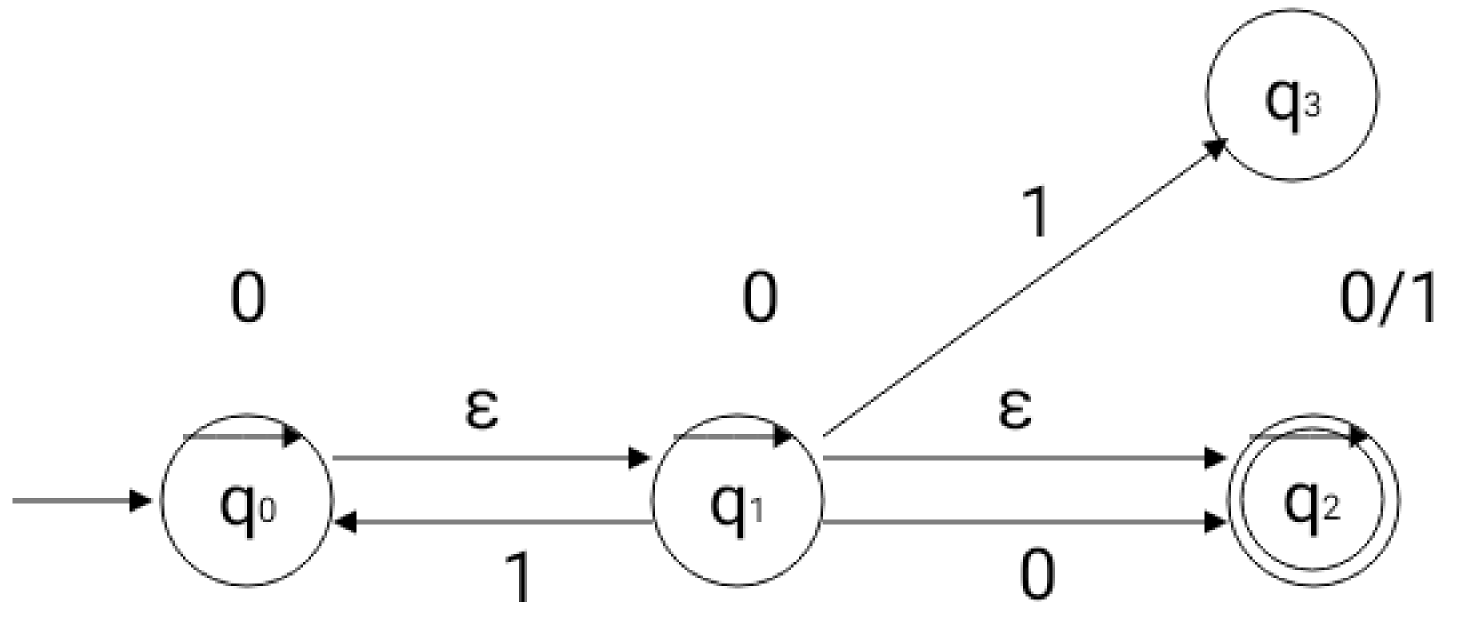
M = (Q, Σ, δ, q_0, F)

Q A <u>finite</u> set of states
Σ A <u>finite</u> input alphabet
q_0 The initial/starting state, q_0 is in Q
F A set of final/accepting states, which is a subset of Q
δ A transition function, which is a total function from Q x Σ U {ε} to 2^Q

δ: (Q x (**Σ U {ε}**)) –> 2^Q
δ(q,s) -The set of all states p such that there is a
transition labeled a from q to p, where a
is in Σ U {ε}

- Sometimes referred to as an NFA-ε other times, simply as an NFA.

- Example:



δ: 0 1 ε

	0	1	ε
q_0	$\{q_0\}$	$\{\}$	$\{q_1\}$
q_1	$\{q_1, q_2\}$	$\{q_0, q_3\}$	$\{q_2\}$
q_2	$\{q_2\}$	$\{q_2\}$	$\{\}$
q_3	$\{\}$	$\{\}$	$\{\}$

- A string $w = w_1w_2\ldots w_n$ is processed as $w = \varepsilon^* w_1 \varepsilon^* w_2 \varepsilon^* \ldots \varepsilon^* w_n \varepsilon^*$

- Example: all computations on 00:

$$\begin{array}{cccc} & 0 & \varepsilon & 0 \\ q_0 & q_0 & q_1 & q_2 \\ & & \vdots & \end{array}$$

Informal Definitions

- Let $M = (Q, \Sigma, \delta, q_0, F)$ be an NFA-ε.
- A String w in Σ^* is *accepted* by M iff there exists a path in M from q_0 to a state in F labeled by w and zero or more ε transitions.
- The language accepted by M is the set of all strings from Σ^* that are accepted by M.

ε-closure

- Define ε-closure(q) to denote the set of all states reachable from q by zero or more ε transitions.

- Examples: (for the previous NFA)

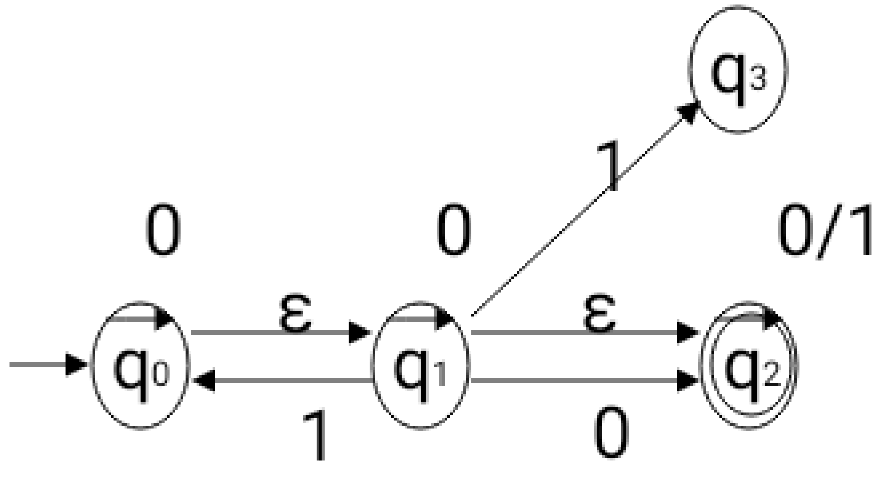
ε-closure(q_0) = {q_0, q_1, q_2} ε-closure(q_2) = {q_2}
ε-closure(q_1) = {q_1, q_2} ε-closure(q_3) = {q_3}

- ε-closure(q) can be extended to sets of states by defining:

ε-closure(P) =   ε-closure(q)

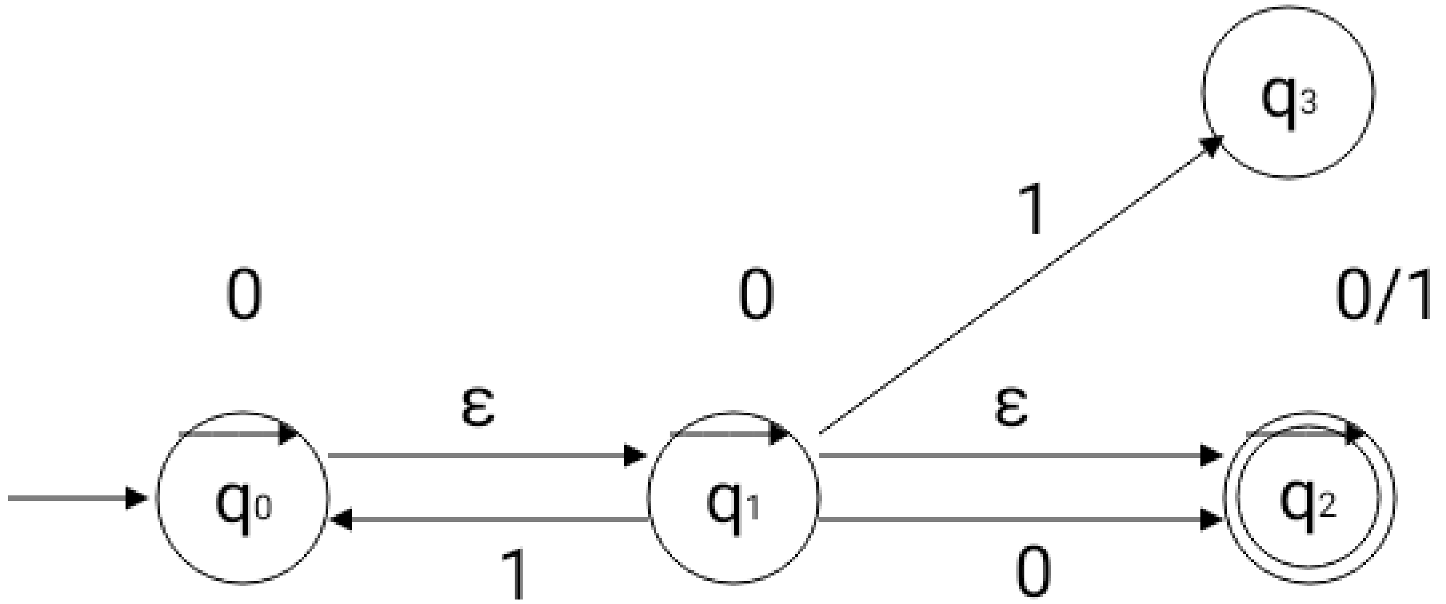
- Examples:

ε-closure({q_1, q_2}) = {q_1, q_2}
ε-closure({q_0, q_3}) = {q_0, q_1, q_2, q_3}



Extension of δ to Strings and Sets of States

- What we currently have: $\delta : (Q \times (\Sigma \cup \{\varepsilon\})) \rightarrow 2^Q$
- What we want (why?): $\delta : (2^Q \times \Sigma^*) \rightarrow 2^Q$
- As before, we will do this in two steps, which will be slightly different from the book, and we will make use of the following NFA.



- Step #1:

Given δ: $(Q \times (\Sigma \cup \{\varepsilon\})) \rightarrow 2^Q$ define $\delta^{\#}$: $(2^Q \times (\Sigma \cup \{\varepsilon\})) \rightarrow 2^Q$ as follows:

1) $\delta^{\#}(R, a) = \quad \delta(q, a)$ for all subsets R of Q, and symbols *a* in $\Sigma \cup \{\varepsilon\}$

- Note that:

$\delta^{\#}(\{p\},a) = \quad \delta(q, a)$ by definition of $\delta^{\#}$, rule #1 above
$= \delta(p, a)$

- Hence, we can use δ for $\delta^{\#}$

$\delta(\{q_0, q_2\}, 0)$ These now make sense, but previously
$\delta(\{q_0, q_1, q_2\}, 0)$ they did not.

- Examples:

What is $\delta(\{q_0, q_1, q_2\}, 1)$?

$\delta(\{q_0, q_1, q_2\}, 1) = \delta(q_0, 1) \cup \delta(q_1, 1) \cup \delta(q_2, 1)$

$= \{\ \} \cup \{q_0, q_3\} \cup \{q_2\}$

$= \{q_0, q_2, q_3\}$

What is $\delta(\{q_0, q_1\}, 0)$?

$\delta(\{q_0, q_1\}, 0) = \delta(q_0, 0) \cup \delta(q_1, 0)$

$= \{q_0\} \cup \{q_1, q_2\}$

$= \{q_0, q_1, q_2\}$

- Step #2:

Given $\delta: (2^Q \times (\Sigma \cup \{\varepsilon\})) \rightarrow 2^Q$ define $\delta^{\wedge}: (2^Q \times \Sigma^*) \rightarrow 2^Q$ as follows:

$\delta^{\wedge}(R,w)$ – The set of states M could be in after processing string w, having starting from any state in R.

Formally:

2) $\delta^{\wedge}(R, \varepsilon) = \varepsilon\text{-closure}(R)$ - for any subset R of Q
3) $\delta^{\wedge}(R,wa) = \varepsilon\text{-closure}(\delta(\delta^{\wedge}(R,w), a))$ - for any w in Σ^*, a in Σ, and subset R of Q

- Can we use δ for $\delta^{\wedge}$?

- Consider the following example:

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

$\hat{\delta}(\{q_0\}, 0) = \varepsilon\text{-closure}(\delta(\hat{\delta}(\{q_0\}, \varepsilon), 0))$ By rule #3

$= \varepsilon\text{-closure}(\delta(\varepsilon\text{-closure}(\{q_0\}), 0))$ By rule #2

$= \varepsilon\text{-closure}(\delta(\{q_0, q_1, q_2\}, 0))$ By ε-closure

$= \varepsilon\text{-closure}(\delta(q_0, 0) \cup \delta(q_1, 0) \cup \delta(q_2, 0))$ By rule #1

$= \varepsilon\text{-closure}(\{q_0\} \cup \{q_1, q_2\} \cup \{q_2\})$

$= \varepsilon\text{-closure}(\{q_0, q_1, q_2\})$

$= \varepsilon\text{-closure}(\{q_0\}) \cup \varepsilon\text{-closure}(\{q_1\}) \cup \varepsilon\text{-closure}(\{q_2\})$

$= \{q_0, q_1, q_2\} \cup \{q_1, q_2\} \cup \{q_2\}$

$= \{q_0, q_1, q_2\}$

- So what is the difference?

$\delta(q_0, 0)$ - Processes 0 as a single symbol, without ε transitions.
$\hat{\delta}(q_0, 0)$ - Processes 0 using as many ε transitions as are possible.

- Example:

$\hat{\delta}(\{q_0\}, 01) = \varepsilon\text{-closure}(\delta(\hat{\delta}(\{q_0\}, 0), 1))$ By rule #3

$= \varepsilon\text{-closure}(\delta(\{q_0, q_1, q_2\}), 1)$ Previous slide

$= \varepsilon\text{-closure}(\delta(q_0, 1) \cup \delta(q_1, 1) \cup \delta(q_2, 1))$ By rule #1

$= \varepsilon\text{-closure}(\{\ \} \cup \{q_0, q_3\} \cup \{q_2\})$

$= \varepsilon\text{-closure}(\{q_0, q_2, q_3\})$

$= \varepsilon\text{-closure}(\{q_0\}) \cup \varepsilon\text{-closure}(\{q_2\}) \cup \varepsilon\text{-closure}(\{q_3\})$

$= \{q_0, q_1, q_2\} \cup \{q_2\} \cup \{q_3\}$

$= \{q_0, q_1, q_2, q_3\}$

Definitions for NFA-ε Machines

- Let $M = (Q, \Sigma, \delta, q_0, F)$ be an NFA-ε and let w be in Σ^*. Then w is *accepted* by M iff $\hat{\delta}(\{q_0\}, w)$ contains at least one state in F.

- Let $M = (Q, \Sigma, \delta, q_0, F)$ be an NFA-ε. Then the *language accepted* by M is the set:

$L(M) = \{w \mid w \text{ is in } \Sigma^* \text{ and } \hat{\delta}(\{q_0\},w) \text{ contains at least one state in } F\}$

- Another equivalent definition:

$L(M) = \{w \mid w \text{ is in } \Sigma^* \text{ and } w \text{ is accepted by } M\}$

Equivalence of NFAs and NFA-εs

- Do NFAs and NFA-ε machines accept the same *class* of languages?
  - Is there a language L that is accepted by a NFA, but not by any NFA-ε?
  - Is there a language L that is accepted by an NFA-ε, but not by any DFA?
- Observation: Every NFA is an NFA-ε.
- Therefore, if L is a regular language then there exists an NFA-ε M such that L = L(M).
- It follows that NFA-ε machines accept all regular languages.
- But do NFA-ε machines accept more?

- **Lemma 1:** Let M be an NFA. Then there exists a NFA-ε M' such that L(M) = L(M').

- **Proof:** Every NFA is an NFA-ε. Hence, if we let M' = M, then it follows that L(M') = L(M).

The above is just a formal statement of the observation from the previous slide.

- **Lemma 2:** Let M be an NFA-ε. Then there exists a NFA M' such that L(M) = L(M').

- **Proof:** (sketch)

Let M = $(Q, \Sigma, \delta, q_0, F)$ be an NFA-ε.

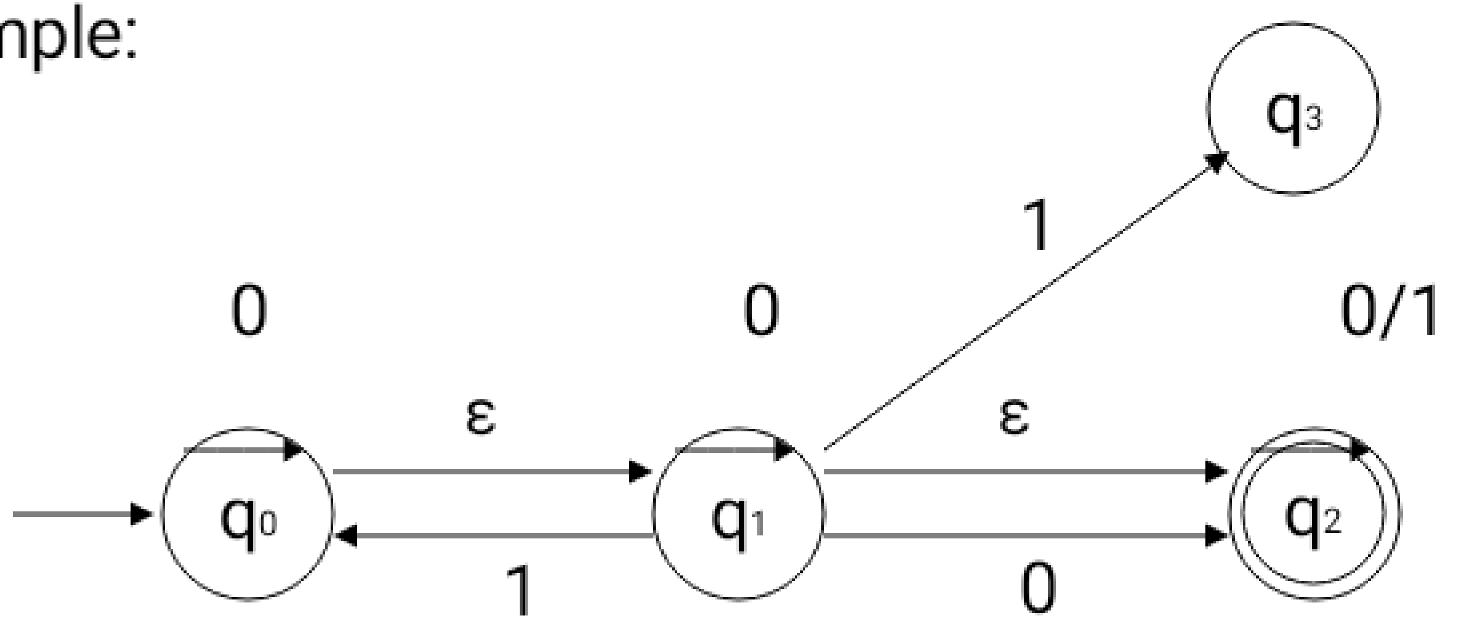
Define an NFA M' = $(Q, \Sigma, \delta', q_0, F')$ as:

$F' = F \cup \{q\}$ if ε-closure(q) contains at least one state from F
$F' = F$ otherwise

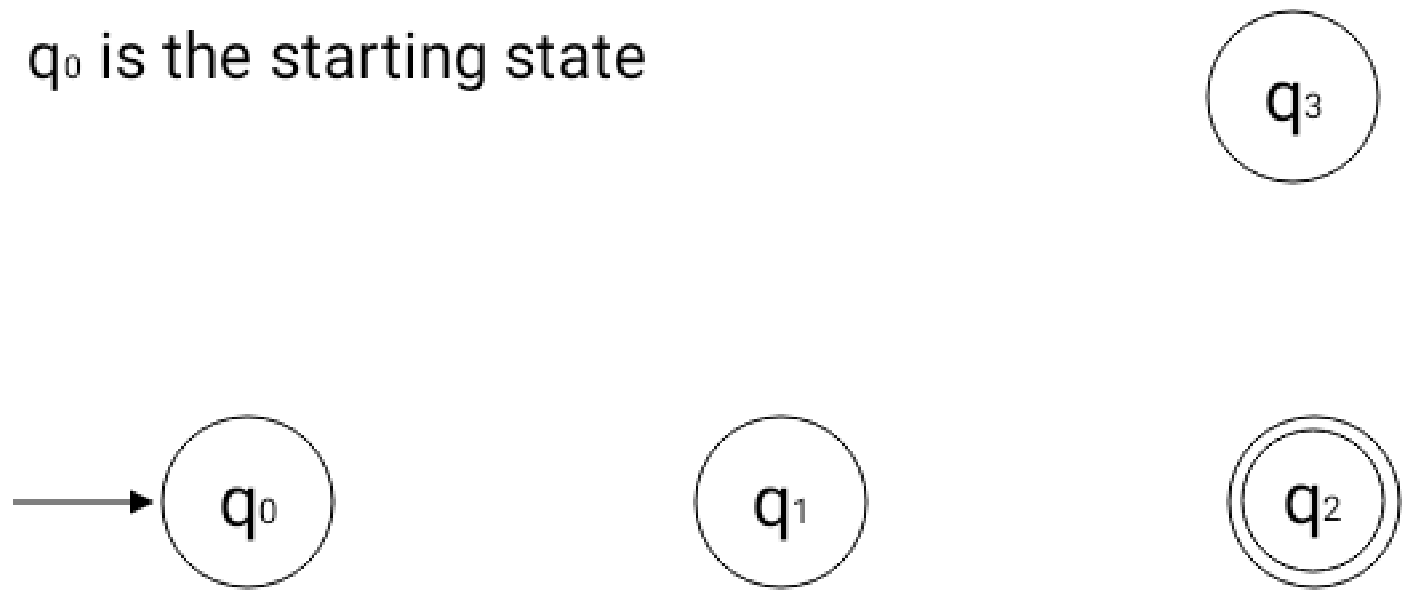
$\delta'(q, a) = \hat{\delta}(q, a)$ - for all q in Q and a in Σ

- Notes:
  – $\delta': (Q \times \Sigma) \rightarrow 2^Q$ is a function
  – M' has the same state set, the same alphabet, and the same start state as M
  – M' has no ε transitions

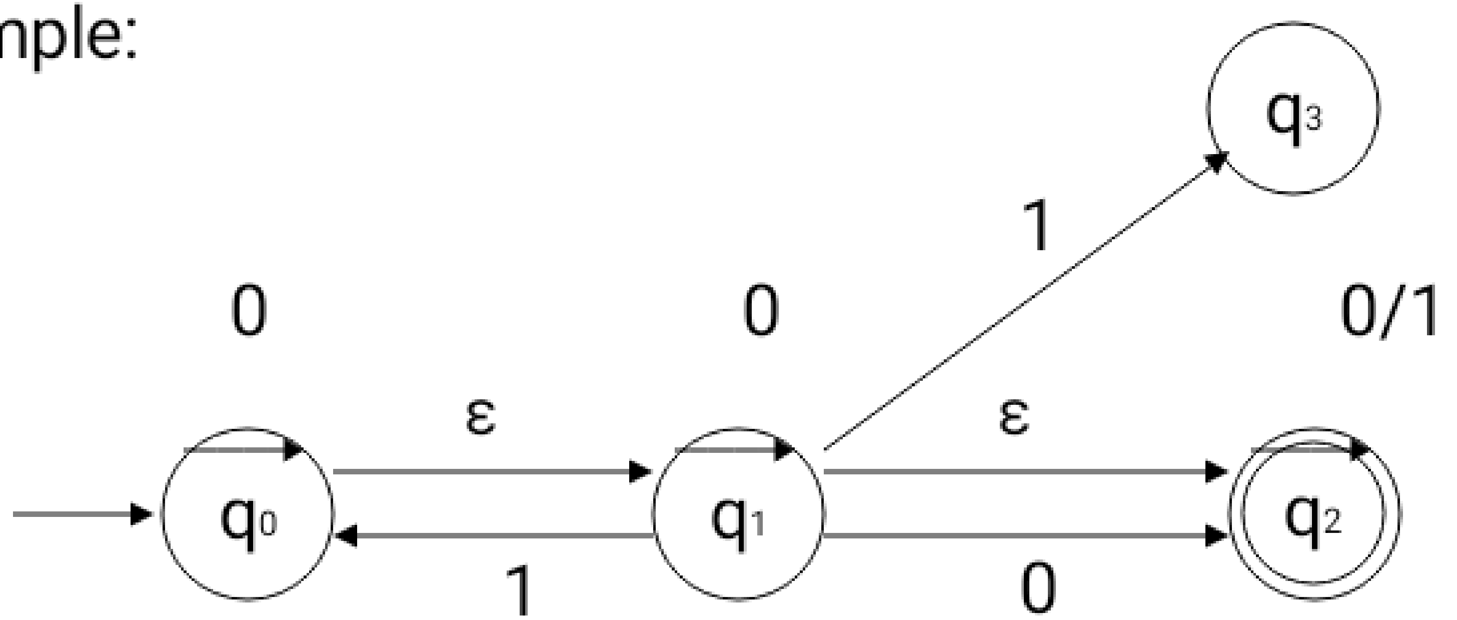
- Example:



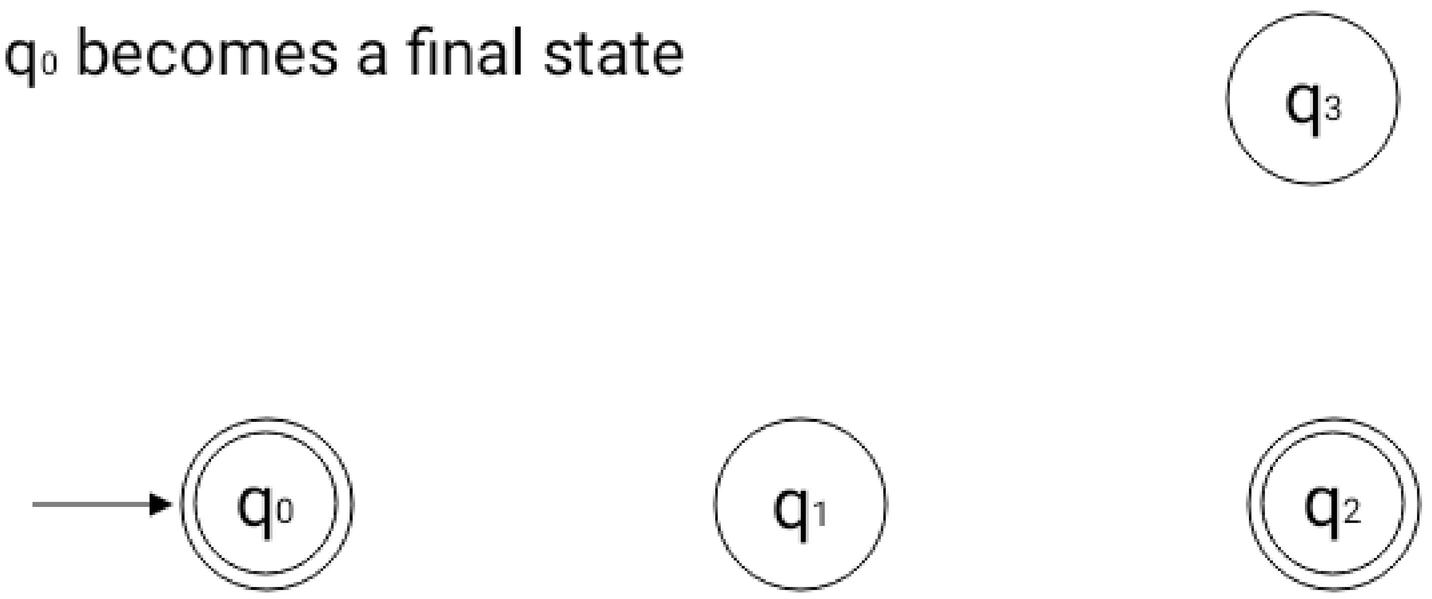
- Step #1:
  - Same state set as M
  - q_0 is the starting state



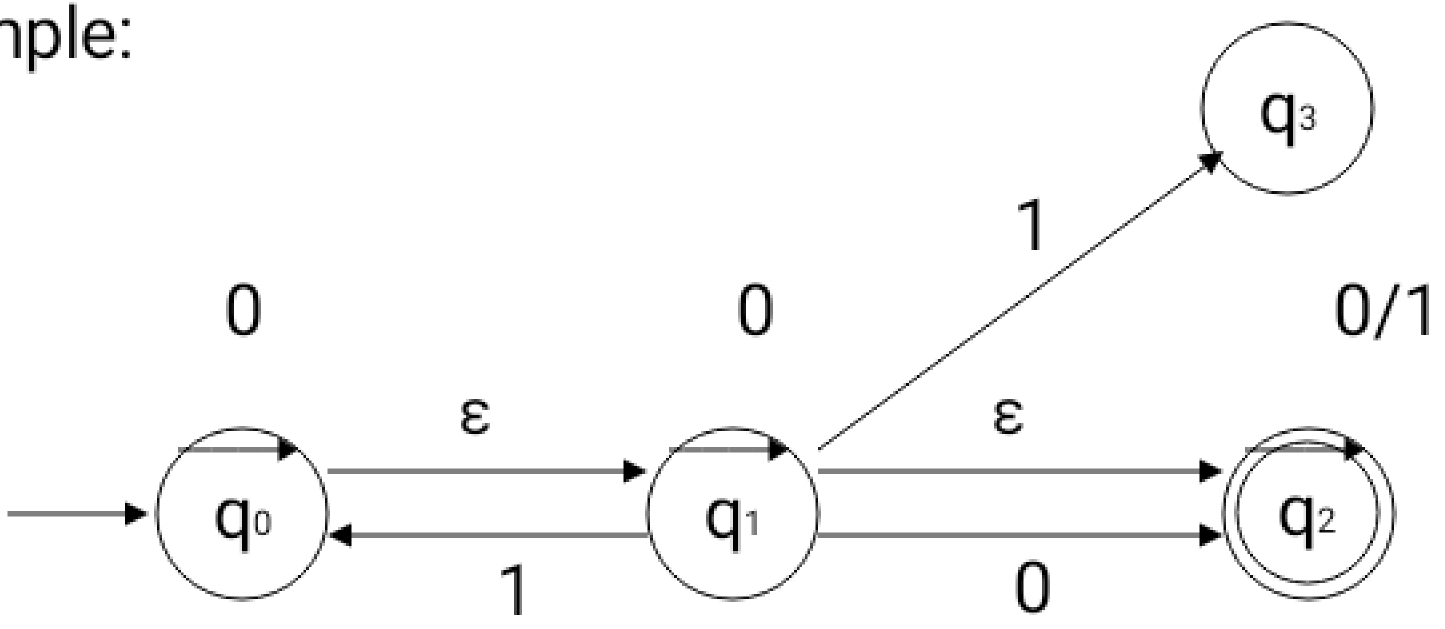
- Example:



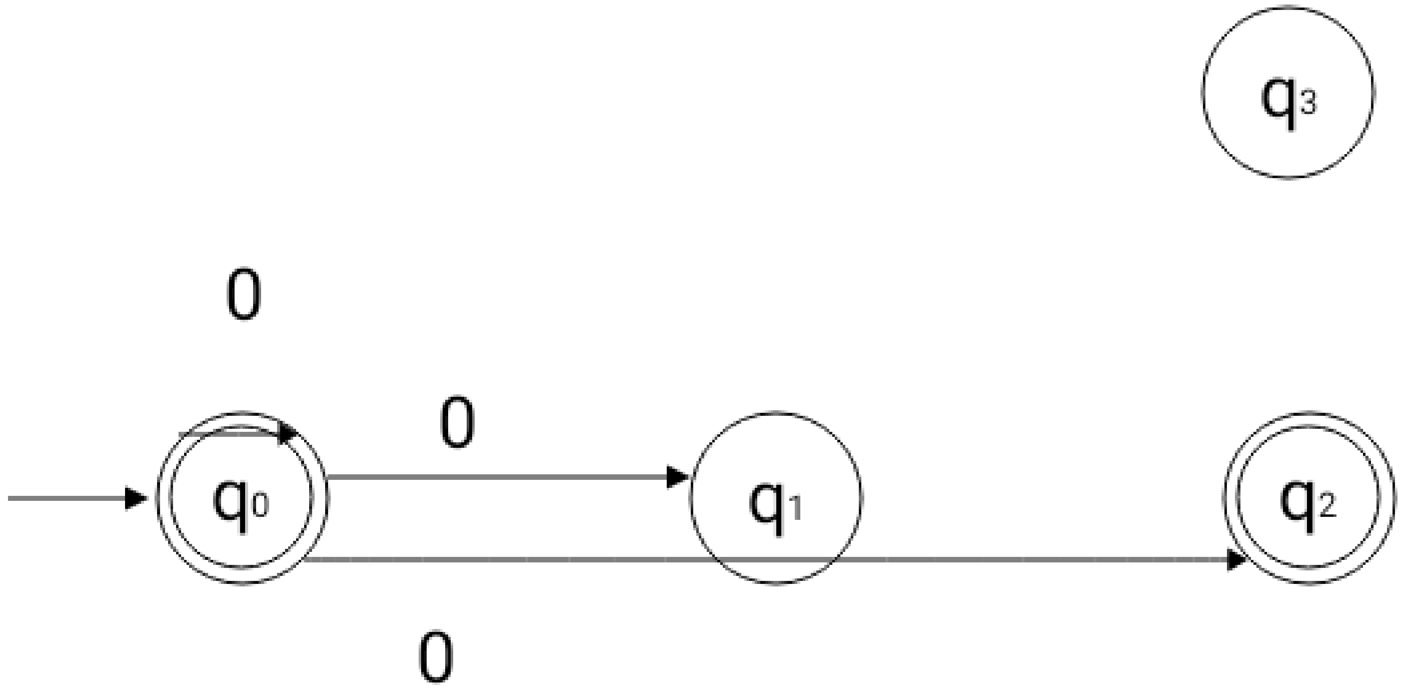
- Step #2:
  - q₀ becomes a final state



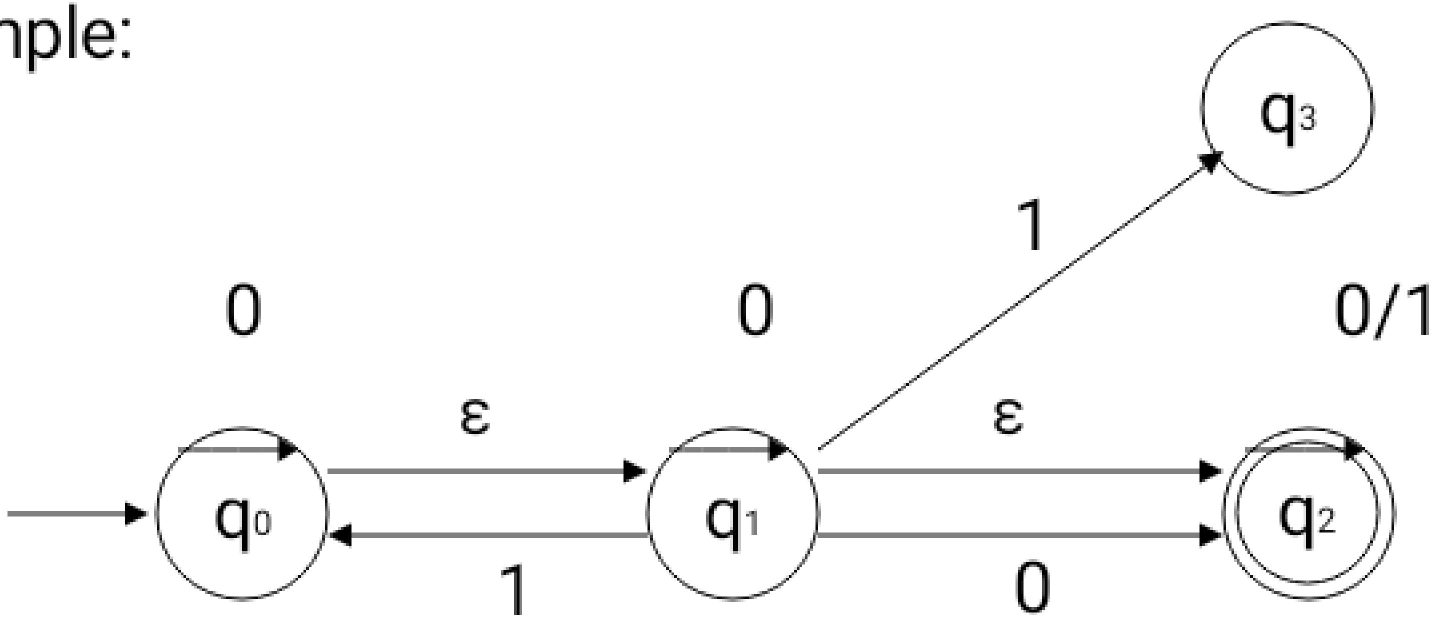
- Example:



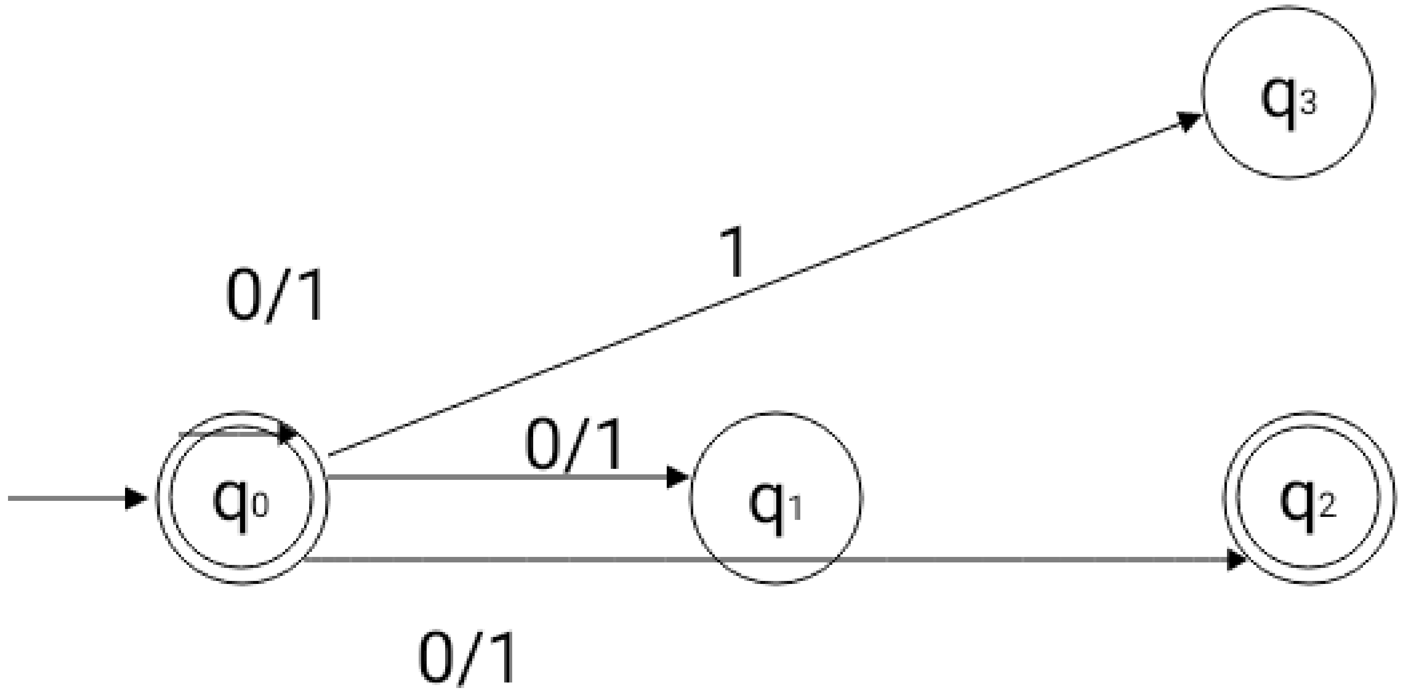
- Step #3:



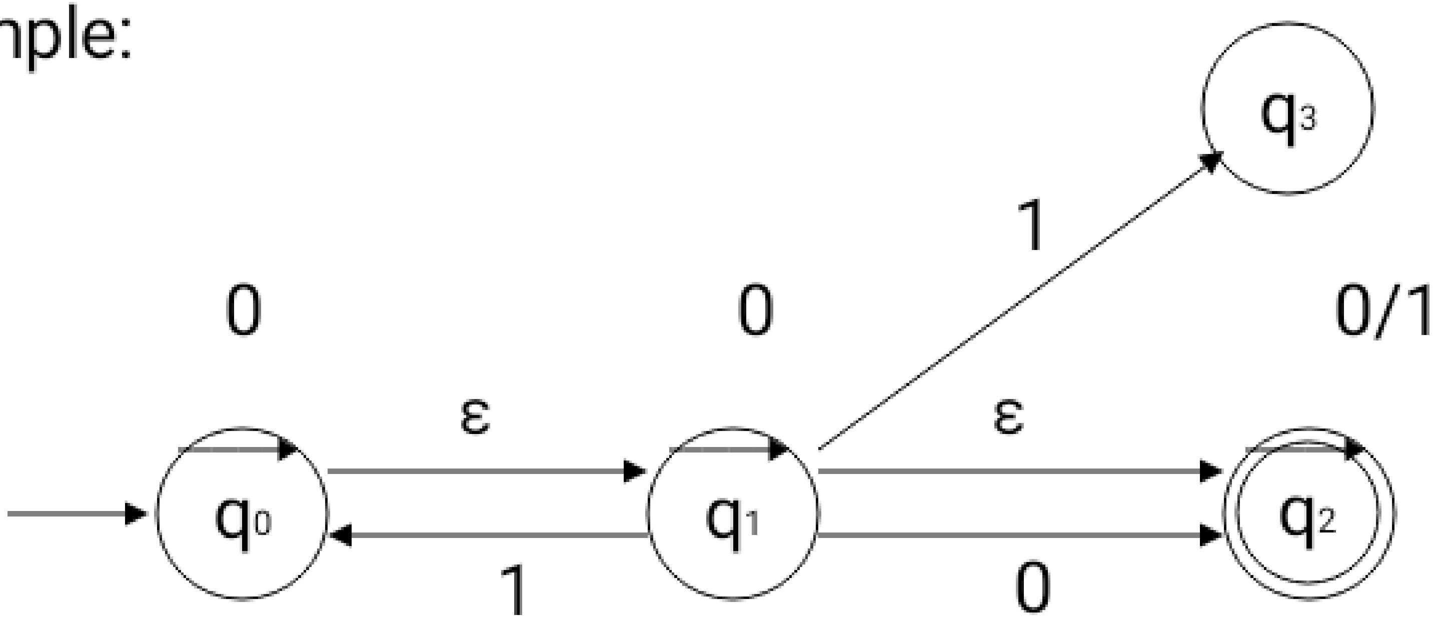
- Example:



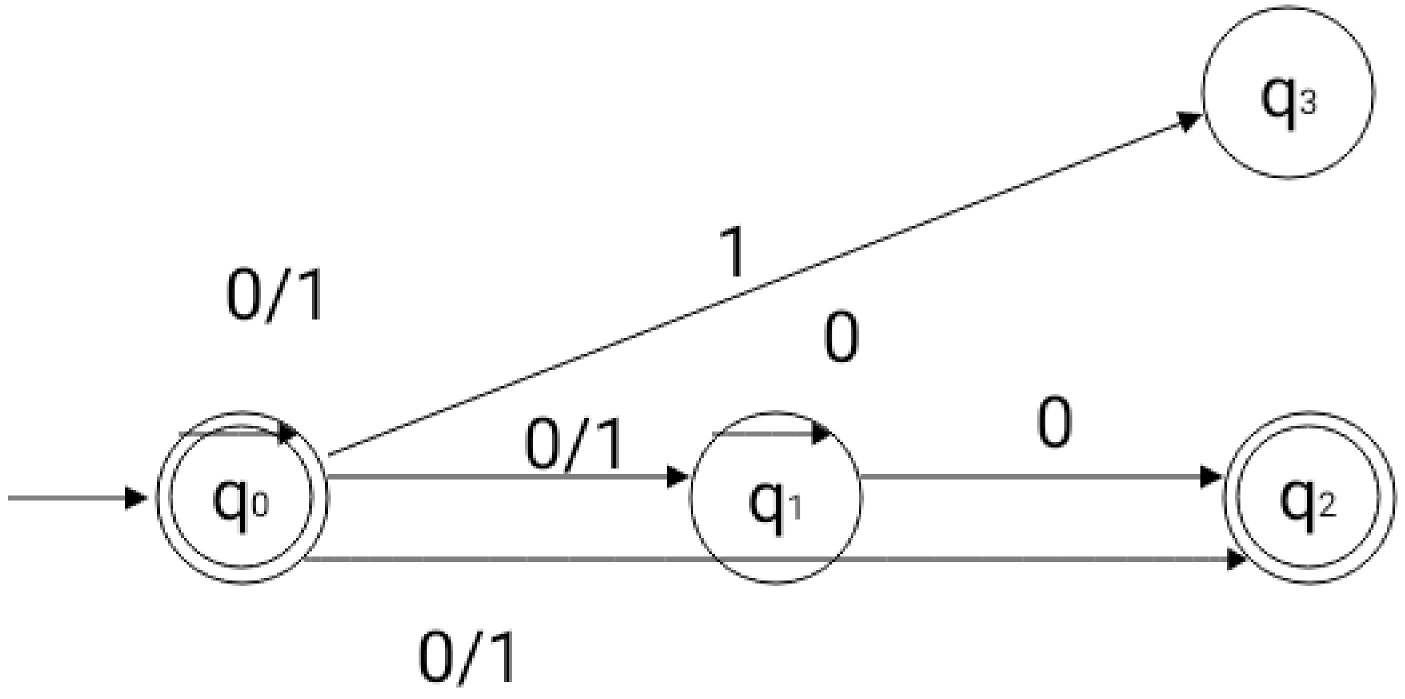
- Step #4:



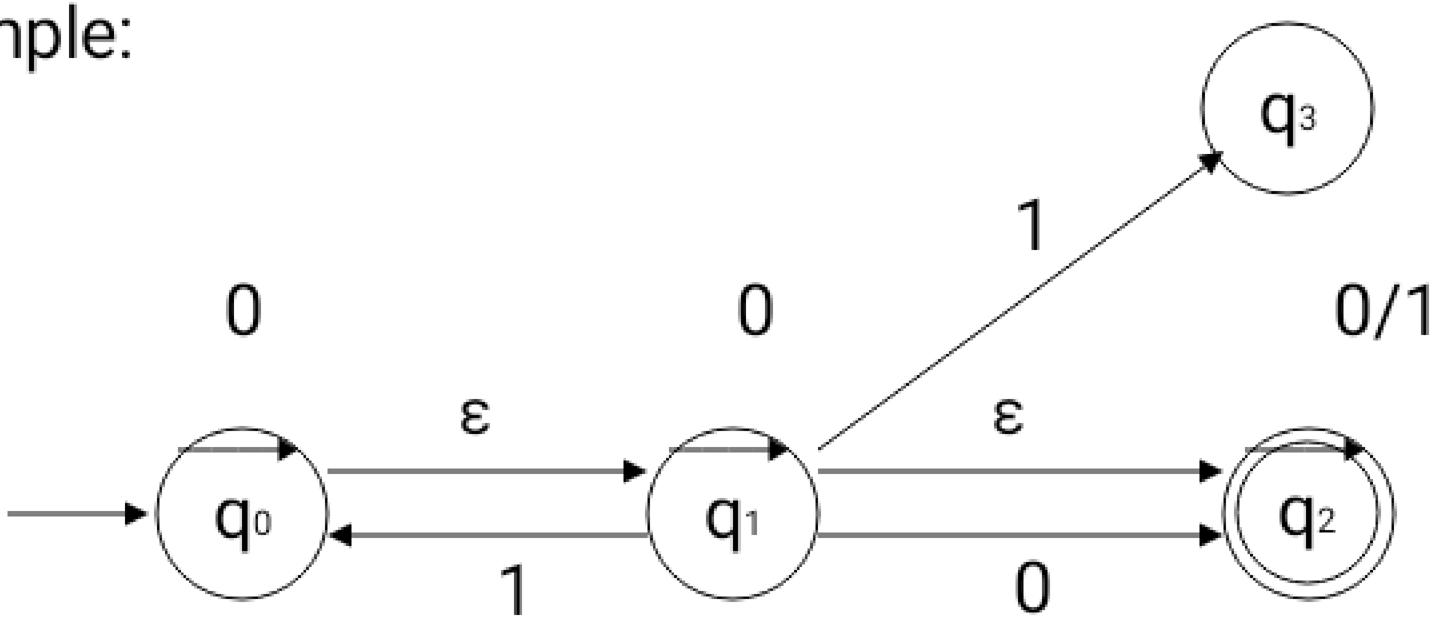
- Example:



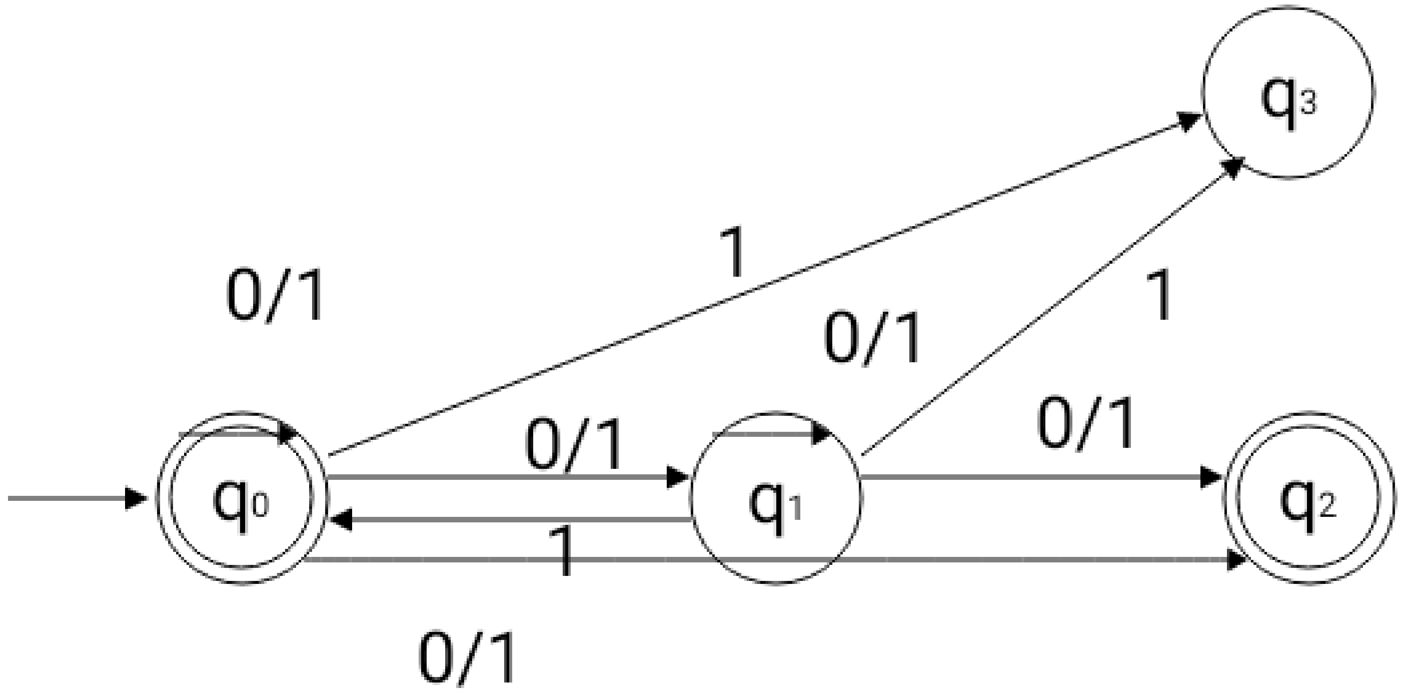
- Step #5:



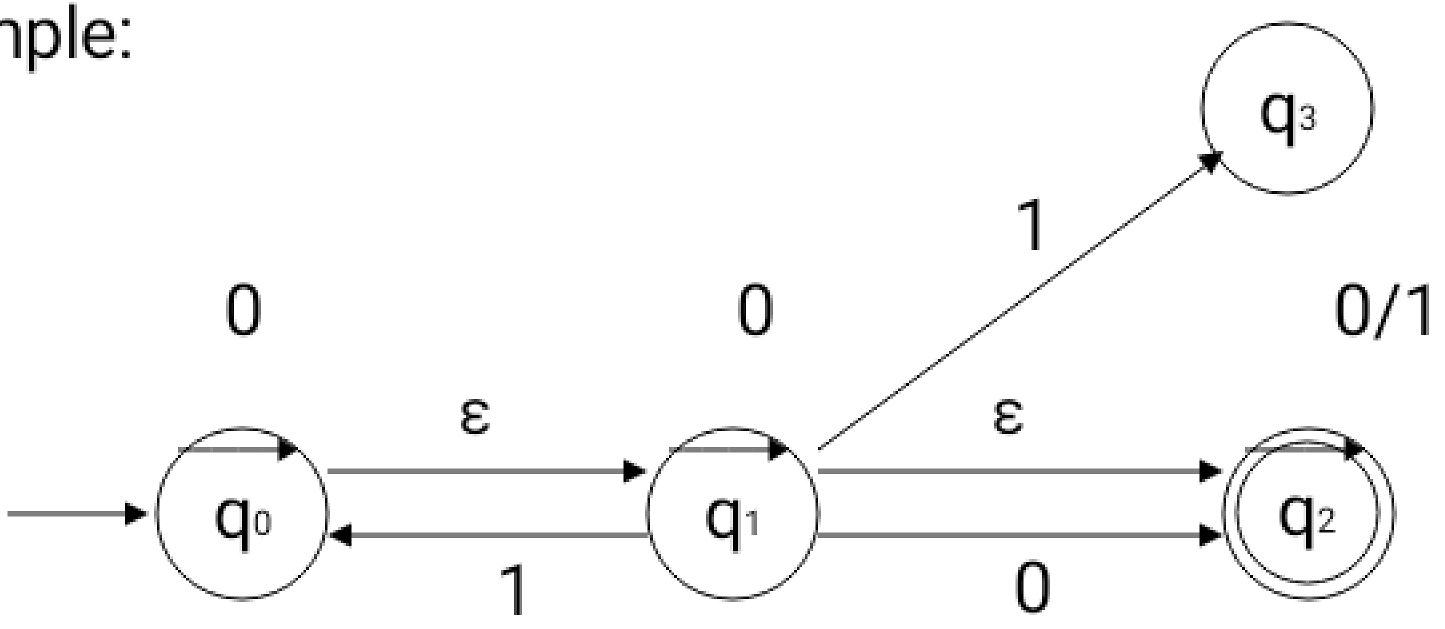
- Example:



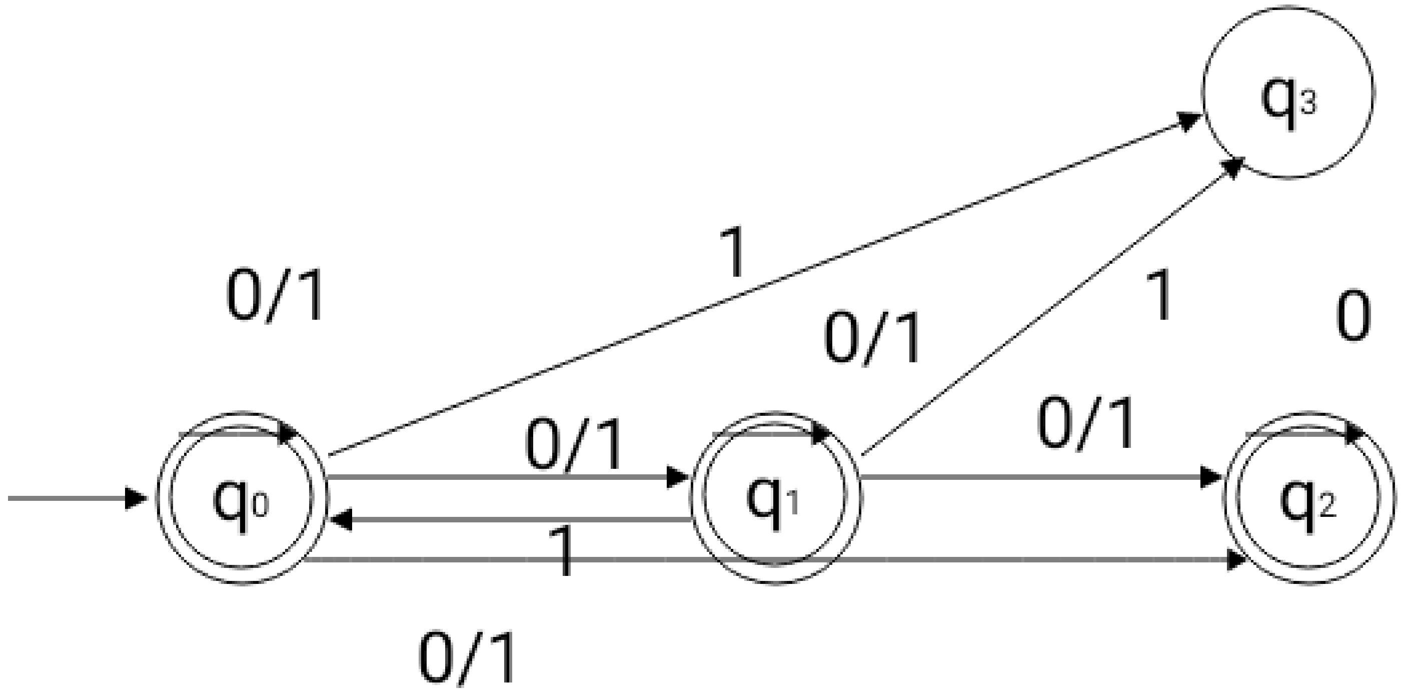
- Step #6:



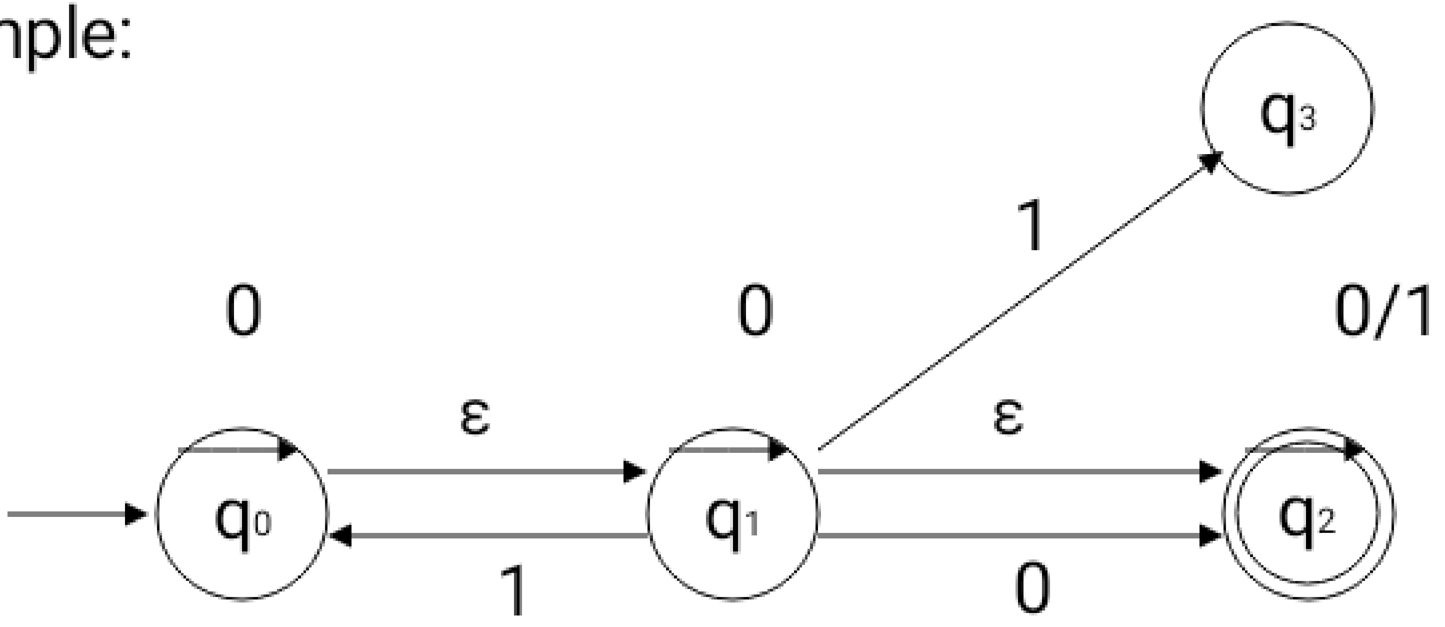
- Example:



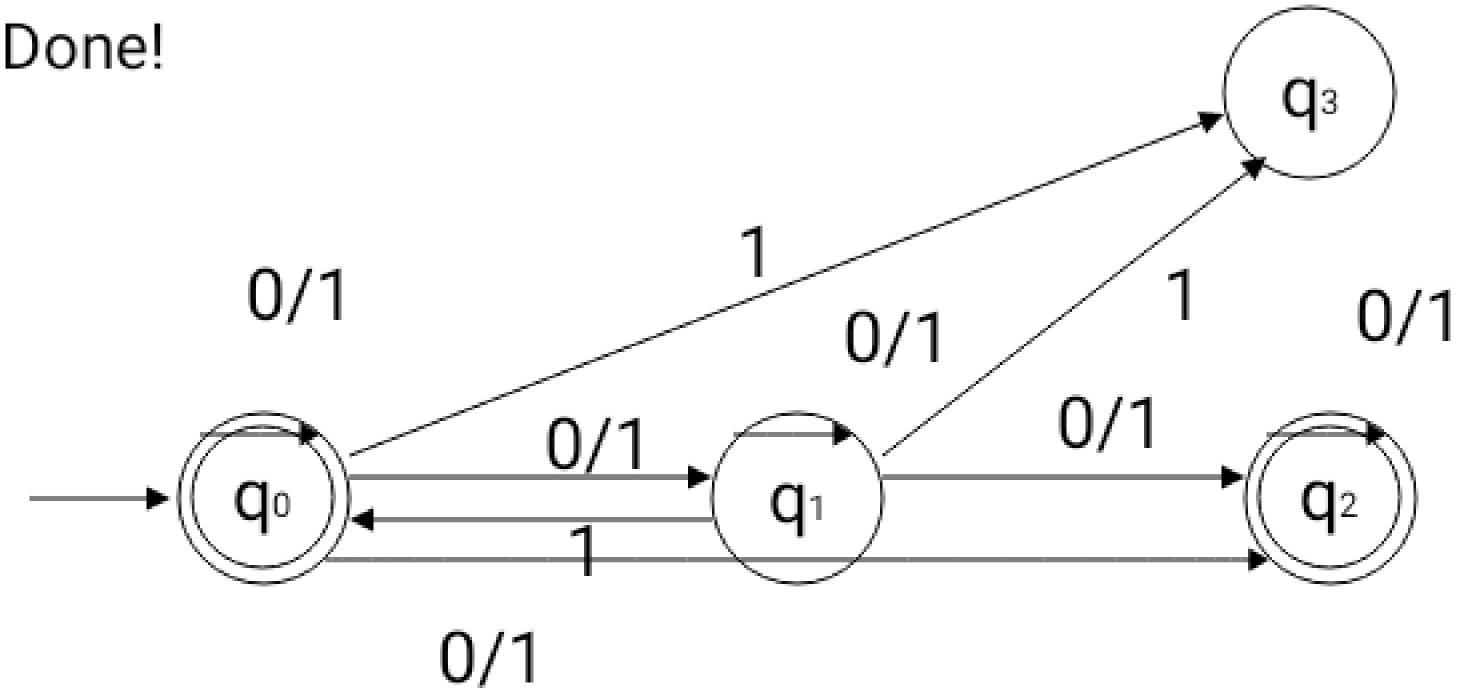
- Step #7:



- Example:



- Step #8: [use table of e-closure]
  - Done!



- **Theorem:** Let L be a language. Then there exists an NFA M such that L= L(M) iff there exists an NFA-ε M' such that L = L(M').

- **Proof:**

(if) Suppose there exists an NFA-ε M' such that L = L(M'). Then by Lemma 2 there exists an NFA M such that L = L(M).

(only if) Suppose there exists an NFA M such that L = L(M). Then by Lemma 1 there exists an NFA-ε M' such that L = L(M').

- **Corollary:** The NFA-ε machines define the regular languages.