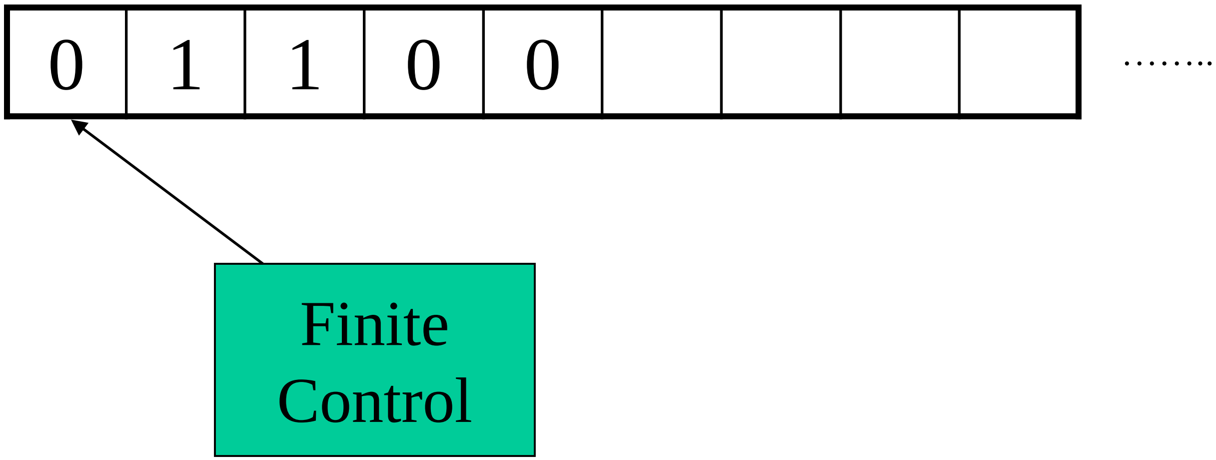
Finite Automata

Reading: Chapter 2

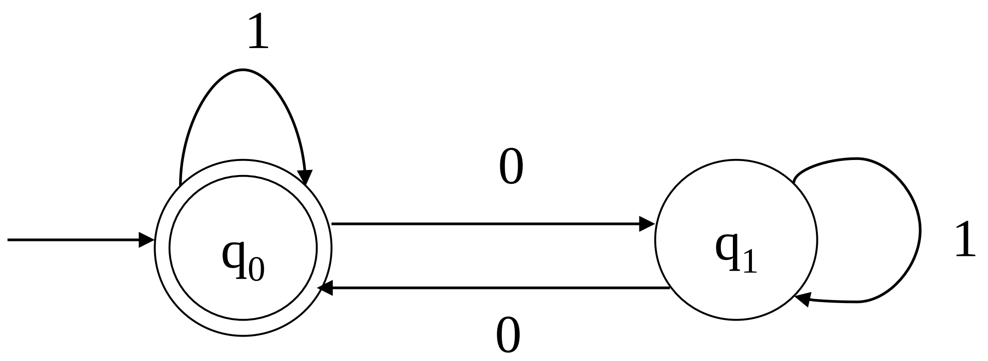
Deterministic Finite State Automata (DFA)



- One-way, infinite tape, broken into cells
- One-way, read-only tape head.
- Finite control, i.e., a program, containing the position of the read head, current symbol being scanned, and the current "state."
- A string is placed on the tape, read head is positioned at the left end, and the DFA will read the string one symbol at a time until all symbols have been read. The DFA will then either accept or reject.

- The finite control can be described by a transition diagram:

- Example #1:

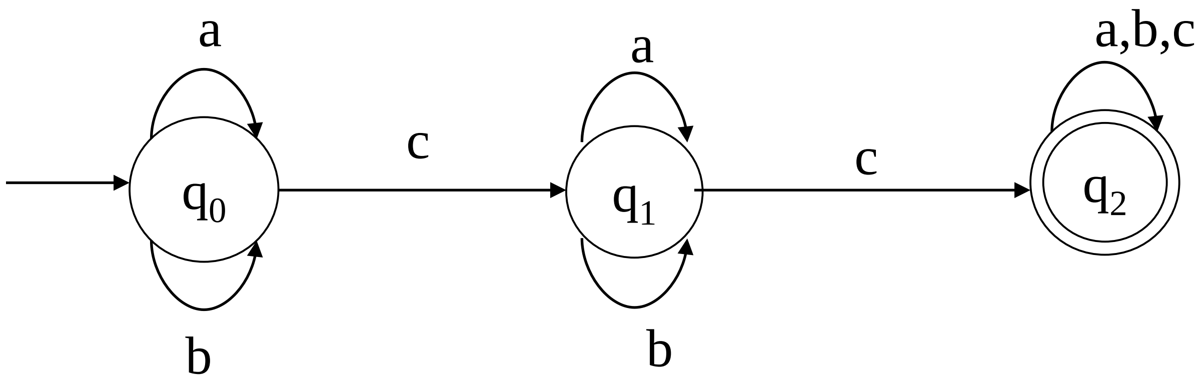


1 0 0 1 1

q_0 q_0 q_1 q_0 q_0 q_0

- One state is final/accepting, the other is rejecting.
- What strings does this DFA accept?

- Example #2:



a c c c b <u>accepted</u>

q_0 q_0 q_1 q_2 q_2 q_2

a a c <u>rejected</u>

q_0 q_0 q_0 q_1

- What strings does this DFA accept?
- Note that every state in a DFA has an implicit “assertion”

Formal Definition of a DFA

- A DFA is a five-tuple:

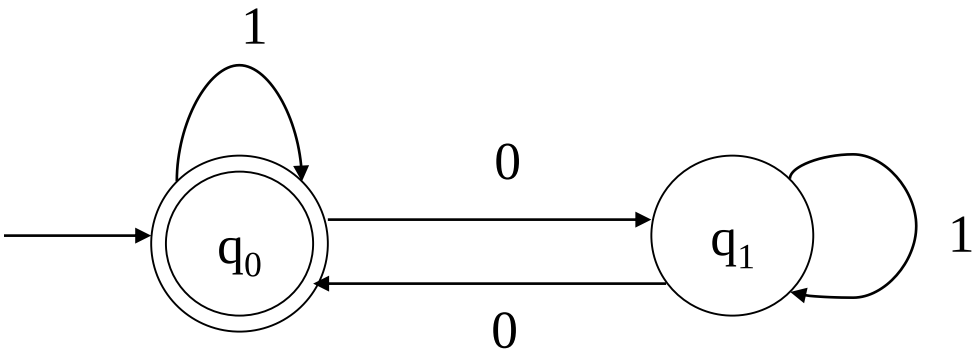
  $M = (Q, \Sigma, \delta, q_0, F)$

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 (zero or more) final/accepting states, which is a subset of Q
δ	A transition function, which is a <u>total</u> function from $Q \times \Sigma$ 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 another state q' in Q.

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

- For example #1:



$Q = \{q_0, q_1\}$

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

Start state is q_0

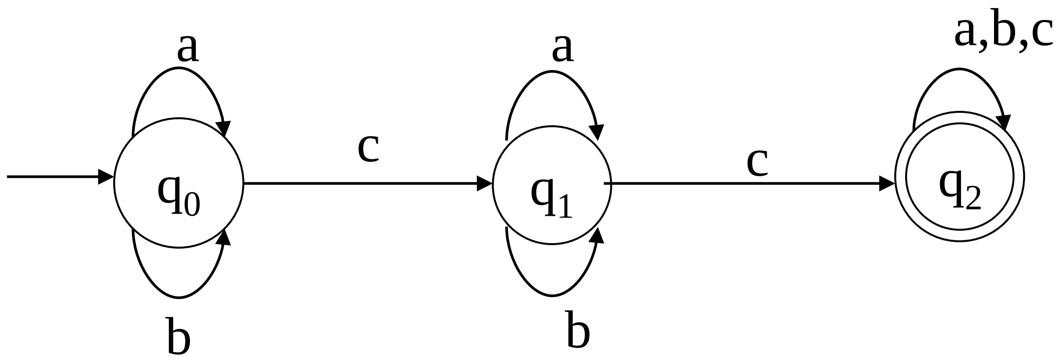
$F = \{q_0\}$

δ:

	0	1
q_0	q_1	q_0
q_1	q_0	q_1

- For example #2:

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



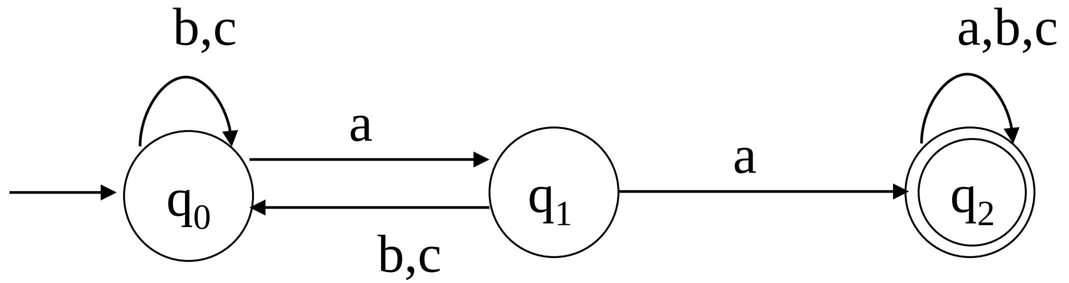
δ:

	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 <u>total</u> function, it is defined for every state q and symbol s.
- In other words, at each step M has exactly one option.
- It follows that for a given string, there is exactly one computation.

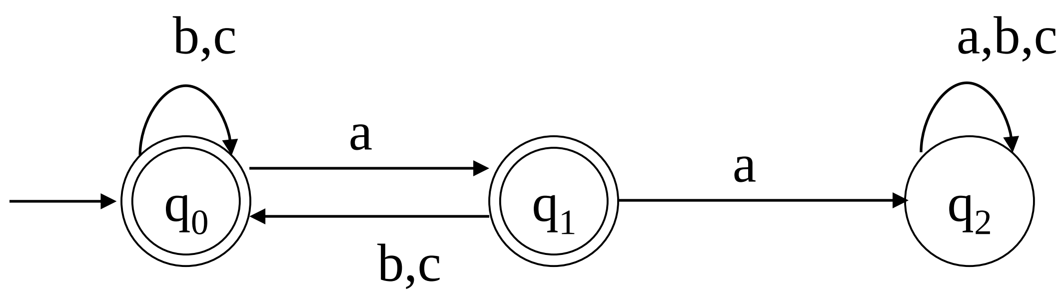
- Give a DFA that accepts the strings in the set:

  {x | x is a string of (zero or more) a's, b's and c's such that x contains the substring *aa*}



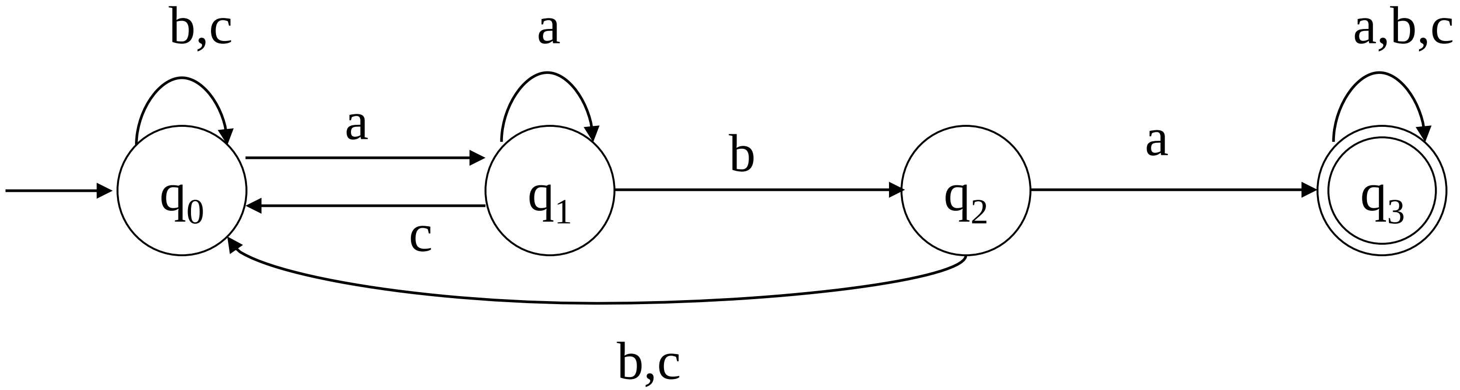
- Give a DFA M such that:

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*}



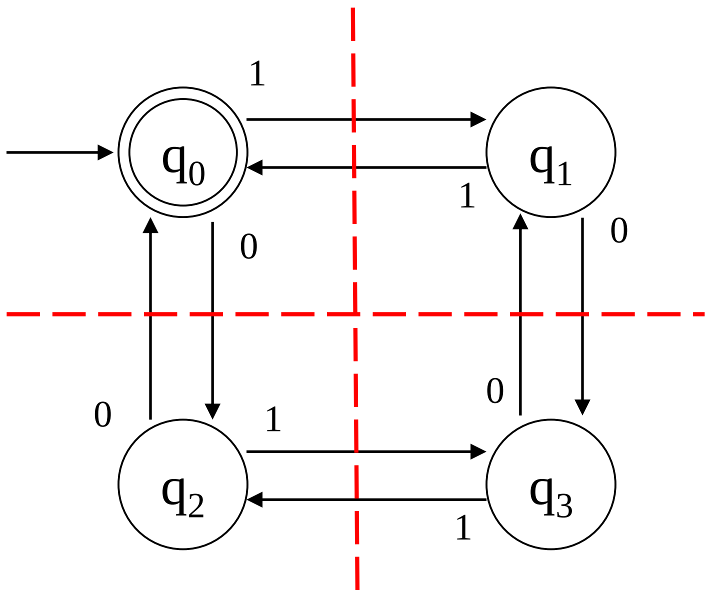
- Give a DFA M such that:

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



- Give a DFA M such that:

L(M) = {x | x is a string of 0's and 1's such that x contains an even number of 0's and an even number of 1's}



- Questions:
  - Which of the following are “valid” DFAs?
    - No final state
    - Unreachable state
    - Missing transition
    - Disconnected components
    - Single state
  - How difficult would it be to simulate a specific DFA?
  - How difficult would it be to automatically generate DFA simulators, i.e., Lex?

```
// variables; for the DFA that accepted all strings with the substring aba
n int = 4;
ch char;
cs int = 0;
FS int set = {0};
TM array[0..n-1,'a'..'c'] of int = { {1,0,0}, {1,2,0}, {3,0,0}, {3,3,3} };

// prompt for and process input string
print("Enter String:");
read(ch);
while (ch <> EOL) {
   cs = TM[cs,ch];
   read(ch)
}

//see if terminating state is a final state
if (cs is in F)
   print("accept");
Else
   print("reject");
```

Extension of δ to Strings

$\delta^{\wedge} : (Q \text{ x } \Sigma^{*}) \text{ --> } Q$

$\delta^{\wedge}(q,x)$ – The state entered after reading string x having started in state q.

Formally - given any string x in Σ^{*}, where $|x|>=0$:

1) $\delta^{\wedge}(q, \varepsilon) = q$, and if $|x|=0$, i.e., $x= \varepsilon$
2) For all w in Σ^{*} and a in Σ if $|x|>=1$, i.e., $x=wa$

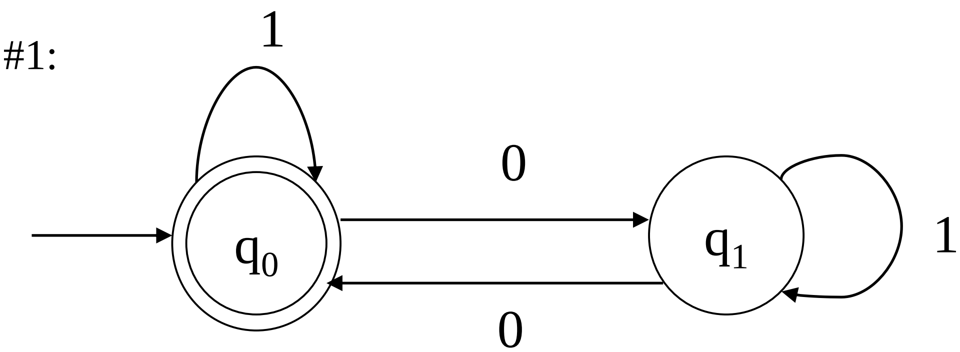
   $\delta^{\wedge}(q,wa) = \delta\ (\delta^{\wedge}(q,w), a)$

1) $\delta^\wedge(q, \varepsilon) = q$, and if $|x|=0$, i.e., $x= \varepsilon$

2) For all w in Σ^* and a in Σ if $|x|>=1$, i.e., $x=wa$

$\delta^\wedge(q,wa) = \delta(\delta^\wedge(q,w), a)$

- Recall Example #1:



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

$\delta^\wedge(q_0, 011)$	$= \delta(\delta^\wedge(q_0,01), 1)$	by rule #2
	$= \delta(\delta(\delta^\wedge(q_0,0), 1), 1)$	by rule #2
	$= \delta(\delta(\delta(\delta^\wedge(q_0, \varepsilon), 0), 1), 1)$	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 $\delta^\wedge(q_0, 011) = q_1$ is not a final state.

- Note that:

$$\begin{aligned} \delta^\wedge(q,a) &= \delta(\delta^\wedge(q, \varepsilon), a) && \text{by rule \#2} \\ &= \delta(q, a) && \text{by rule \#1} \end{aligned}$$

- More generally, it is obvious from the definition of $\delta^\wedge$ that:

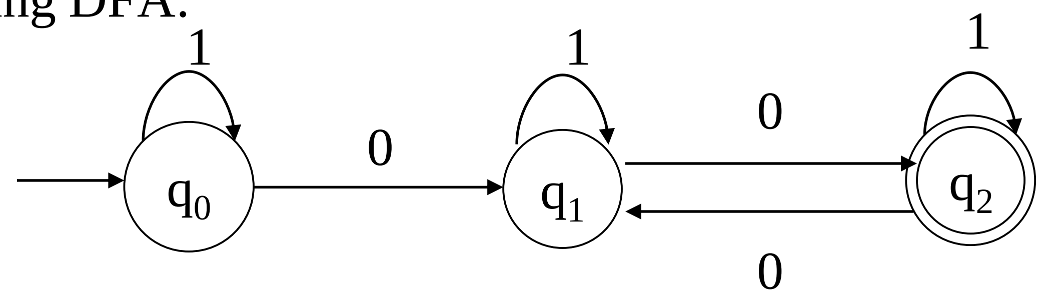
$$\delta^\wedge(q, a_1a_2\ldots a_n) = \delta(\delta(\ldots\delta(\delta(q, a_1), a_2)\ldots), a_n)$$

- Hence, we can (informally) use δ in place of $\delta^\wedge$:

$$\delta^\wedge(q, a_1a_2\ldots a_n) = \delta(q, a_1a_2\ldots a_n)$$

- In other words, $\delta^\wedge$ doesn't really add anything to δ.

- Consider the following DFA:

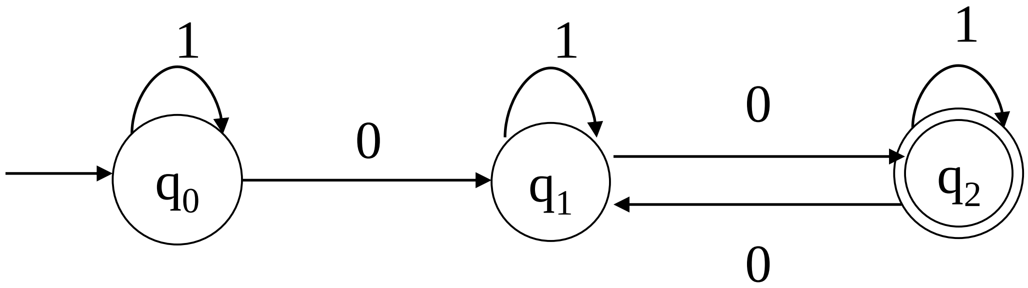


- What is $\delta(q_0, 011)$? Informally, it is the state entered by M after processing 011 having started in state q_0.
- Formally:

$\delta(q_0, 011)$	$= \delta(\delta(q_0,01), 1)$	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.

- Recall Example #2:



- 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

- Based on the above, can we conclude that 10 is 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!

Definitions for DFAs

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

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

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

- Other, equivalent, less formal definitions for L(M):

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

Definitions for DFAs

- 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 $\Sigma^* - L(M)$.

  - In the definition of a regular language, "=" means *exactly* equals.

  - If $L = L(M)$ then L is a subset of $L(M)$, and $L(M)$ is a subset of L.

  - 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

    $L_1 = \{x \mid x$ is a string of 0's and 1's containing an even number of 1's$\}$ and

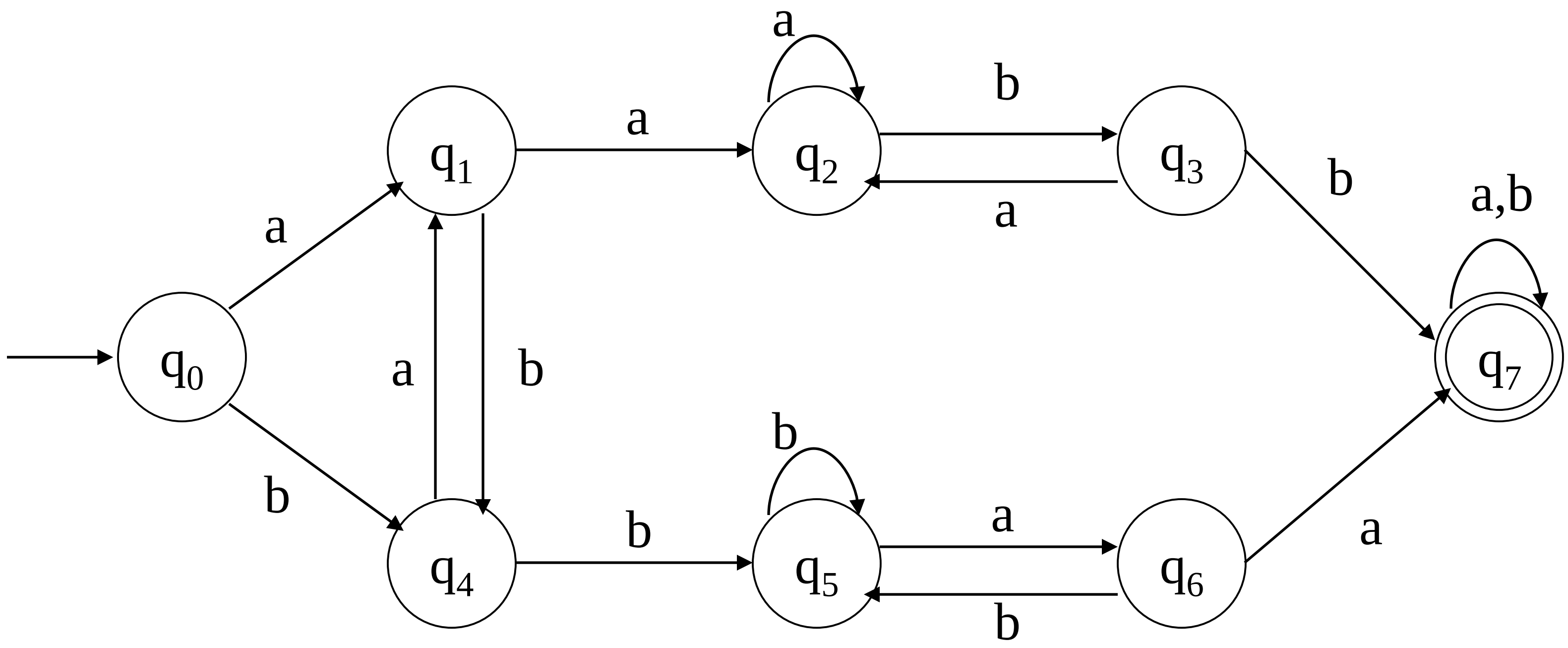
    $L_2 = \{x \mid x = 0^n1^n$ for some $n >= 0\}$

    then L_1 is regular but L_2 is not.

- Questions:
  - How do we determine whether or not a given language is regular?

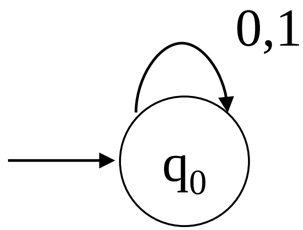
- Give a DFA M such that:

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

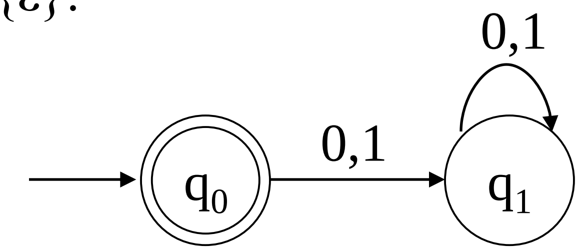


- Let $\Sigma = \{0, 1\}$. Give DFAs for $\{\}$, $\{\varepsilon\}$, Σ^*, and Σ^+.

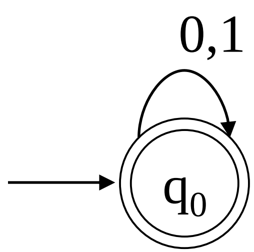
For $\{\}$:



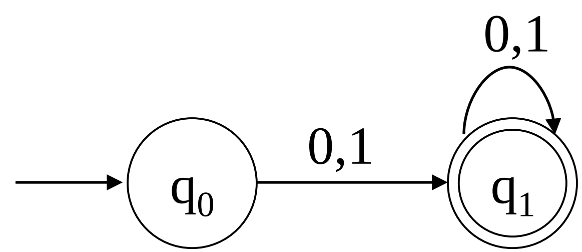
For $\{\varepsilon\}$:



For Σ^*:



For Σ^+:



Nondeterministic Finite State Automata (NFA)

- An NFA is a five-tuple:

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

Q	A finite set of states
Σ	A finite 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 $Q \times \Sigma$ to 2^Q

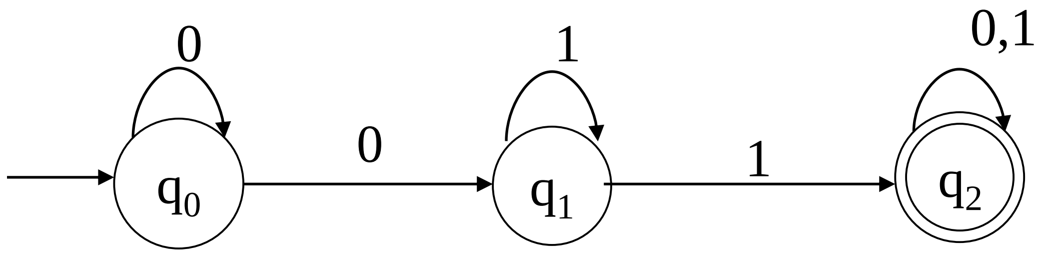
$\delta: (Q \times \Sigma) \rightarrow$ **2^Q** -2^Q is the power set of Q, the set of all subsets of Q

$\delta(q,s)$ -The set of all states p such that there is a transition labeled s from q to p

$\delta(q,s)$ is a function from $Q \times \Sigma$ to 2^Q (but not to Q)

- Example #1:

  $Q = \{q_0, q_1, q_2\}$
  $\Sigma = \{0, 1\}$
  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	$\{q_2\}$	$\{q_2\}$

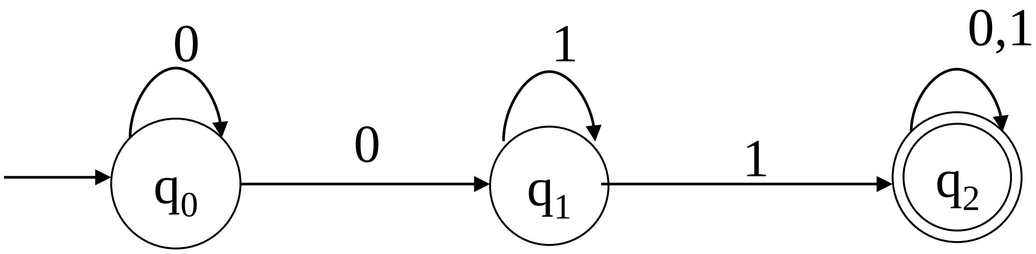
- How is a string such as 011 processed?

- For a given string there may be multiple paths; in fact, there are three types of paths.

- Example #1:

$Q = \{q_0, q_1, q_2\}$
$\Sigma = \{0, 1\}$
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	$\{q_2\}$	$\{q_2\}$

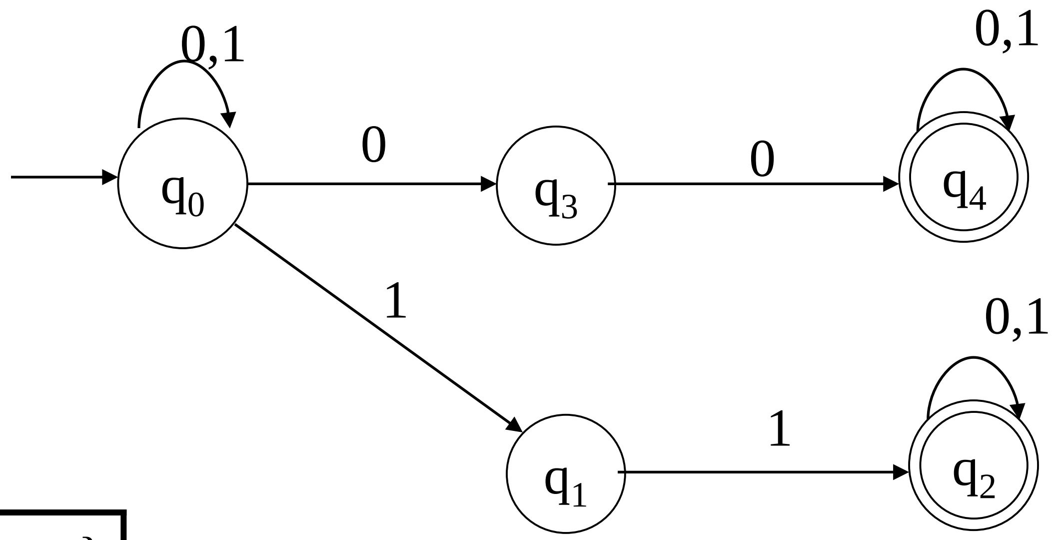


- A string is said to be accepted *if there exists* a path to some state in F that uses all the symbols in the string (try 011, 000, 01110, 1010, 0011).

- The language accepted by an NFA is the set of all accepted strings.
  – The above NFA accepts the set of all strings of 0's and 1's that start with one or more 0's, followed by one or more 1's, followed by any sequence of 0's and 1's.

- Example #2:

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



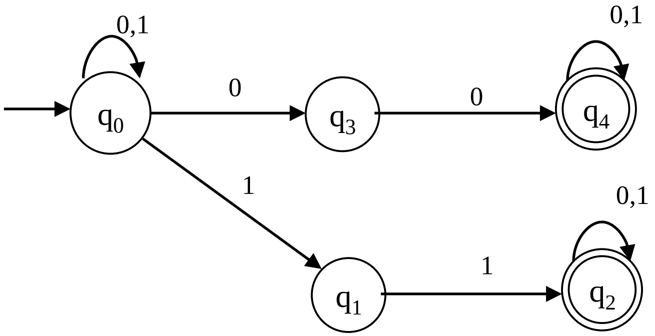
δ:

	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	$\{q_4\}$	$\{q_4\}$

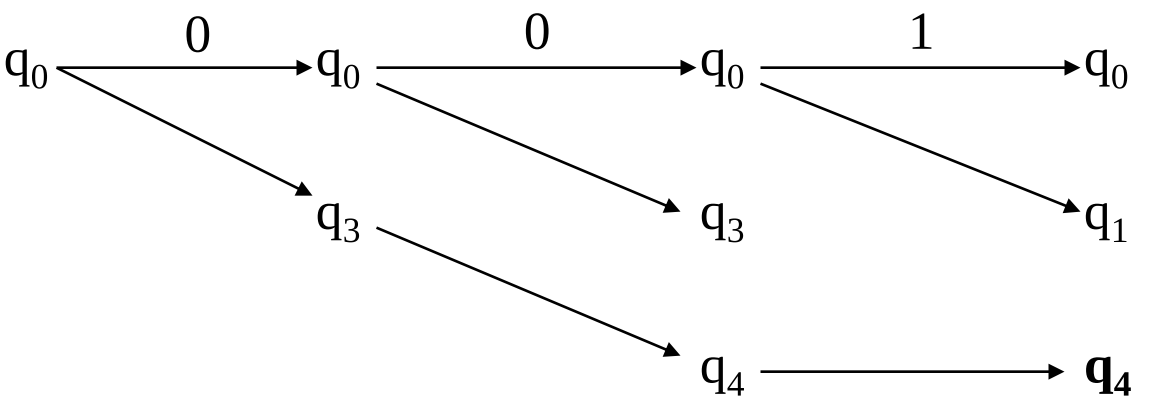
- Notes:
  - $\delta(q,s)$ may not be defined for some q and s (why?).
  - Could the previous two languages be accepted by a DFAs (exercise)?

- Question: How does an NFA find the correct/accepting path for a given string?
  - Doesn't really matter
  - NFAs are a non-intuitive computing model.
    - Designing NFAs is not a typical task.
    - Regardless, the notions of string and language acceptance are *well-defined*.
  - We are *primarily* interested in NFAs as language defining devices, i.e., do NFAs accept languages that DFAs do not?
  - Other questions are secondary, e.g., whether or not there is an algorithm for finding an accepting path through an NFA for a given string.

- All that having been said…



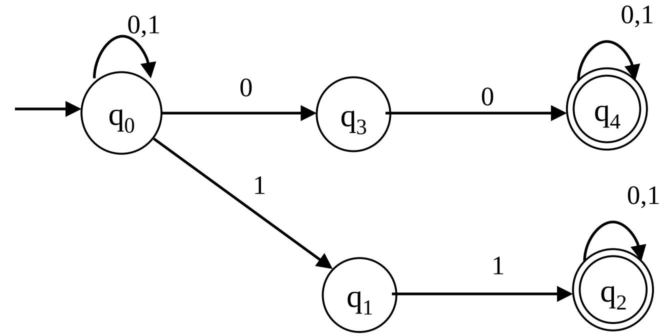
- Yes, determining if a given NFA (example #2) accepts a given string (001) can be done algorithmically:

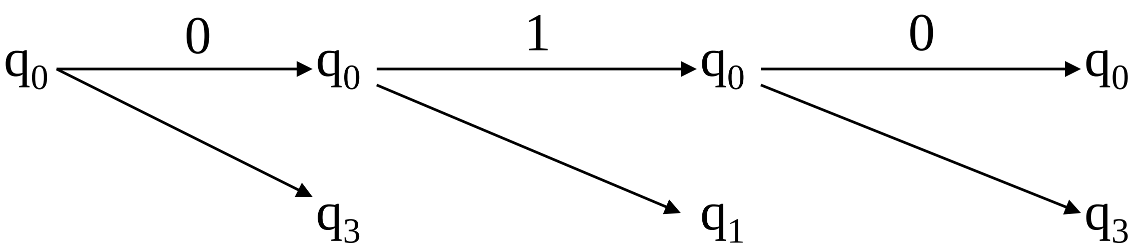


<u>accepted</u>

- Each level will have at most n states

- Another example (010):





<u>not accepted</u>

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

- How difficult would it be to simulate an NFA?

- Would the DFA simulation algorithm work?

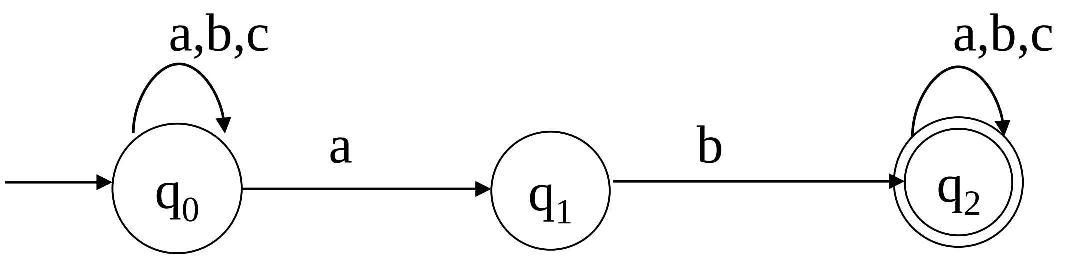
```
// variables; for NFA example #2
n int = 5;
ch char;
cs int set = {0};
new-cs int set;
FS int set = {2,4};
TM array[0..n-1,0..1] of int set = { {{0,3},{0,1}}, {{0},{}}, {{2},{2}}, {{4},{0}}, {{4},{4}} };

// prompt for and process input string
print("Enter String:");
read(ch);
while (ch <> EOL) {
   new-cs = {};
   for each s in cs {
      new-cs = new-cs U TM[s,ch];
   }
   cs = new-cs;
   read(ch)
}

//see if terminating state is a final state
if (any state in cs is in F)
   print("accept");
Else
   print("reject");
```

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

  $L = \{x \mid x \text{ is in } \Sigma^* \text{ and } x \text{ contains ab}\}$



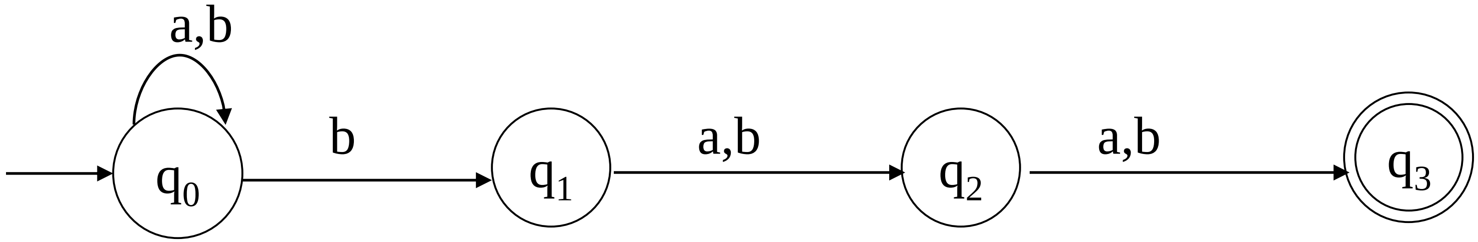
  Is L a subset of L(M)?

  Is L(M) a subset of L?

- Is an NFA necessary? Could a DFA accept L? Try and give an equivalent DFA as an exercise.

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

$$L = \{x \mid x \text{ is in } \Sigma^* \text{ and the third to the last symbol in x is b}\}$$



Is L a subset of L(M)?

Is L(M) a subset of L?

- What if q_3 had a transition to itself on *a* or *b*?

- Give an equivalent DFA as an exercise.

Extension of δ to Strings

What we currently have: $\delta : (Q \times \Sigma) \rightarrow 2^Q$

What we want (why?): $\delta^\wedge : (Q \times \Sigma^*) \rightarrow 2^Q$

$\delta^\wedge(q,x)$ – The *set* of states the NFA could be in after reading string x having started in state q.

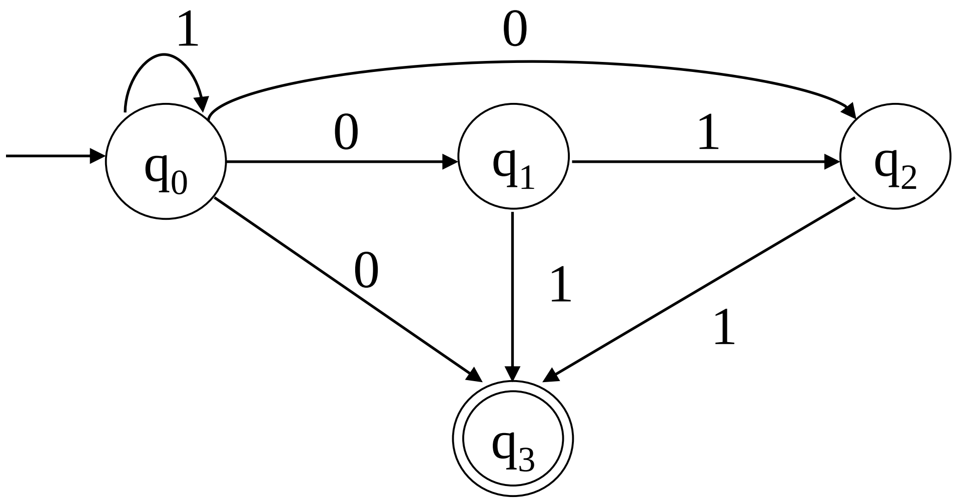
Formally - given any string x in Σ^*, where $|x|>=0$:

1) $\delta^\wedge(q, \varepsilon) = \{q\}$, and — if $|x|=0$
2) For all w in Σ^* and a in Σ, — if $|x|>=1$, i.e., x=wa

   if $\delta^\wedge(q, w) = \{p_1, p_2,\ldots, p_k\}$, and

   $\bigcup_{i=1}^{k} \delta(p_i, a) = \{r_1, r_2,\ldots, r_m\}$ then $\delta^\wedge(q,wa) = \{r_1, r_2,\ldots, r_m\}$

- Example:



What is $\delta^{\wedge}(q_0, 01)$?

Informally: The set of states the NFA could be in after processing 01, i.e., $\{q_2, q_3\}$

Formally: (bottom up)

a) $\delta^{\wedge}(q_0, \varepsilon)$	$= \{q_0\}$	def of $\delta^{\wedge}$, line #1
b) $\delta^{\wedge}(q_0, 0)$	$= \delta(q_0, 0) = \{q_1, q_2, q_3\}$	def of $\delta^{\wedge}$, line #2, δ, and a)
c) $\delta^{\wedge}(q_0, 01)$	$= \delta(q_1, 1) \cup \delta(q_2, 1) \cup \delta(q_3, 1)$	def of $\delta^{\wedge}$, line #2, δ, and b)
	$= \{q_2, q_3\} \cup \{q_3\} \cup \{\}$	def of δ
	$= \{q_2, q_3\}$	

Is 01 accepted? Yes! (see the book for a longer example)

- Note that just like with DFAs, $\delta^\wedge$ is a direct extension of δ, and doesn't really add anything.

- Consequently we can use δ in place of $\delta^\wedge$, i.e.,

$$\delta^\wedge(q, a_1a_2\ldots a_n) = \delta(q, a_1a_2\ldots a_n)$$

- $\delta(q, a_1a_2\ldots a_n)$ is the set of states the NFA could be in after processing $a_1a_2\ldots a_n$ having started in state q.

Definitions for NFAs

- Let $M = (Q, \Sigma, \delta, q_0, F)$ be an NFA and let w be in Σ^*. Then w is *accepted* by M iff $\delta(q_0, w)$ contains at least one state in F, i.e., $\delta(q_0, w) \cap F \neq \emptyset$.

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

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

- Other, equivalent, less formal definitions:

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

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

Equivalence of DFAs and NFAs

- Do DFAs and NFAs accept the same *class* of languages?

- Do they accept different classes 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?

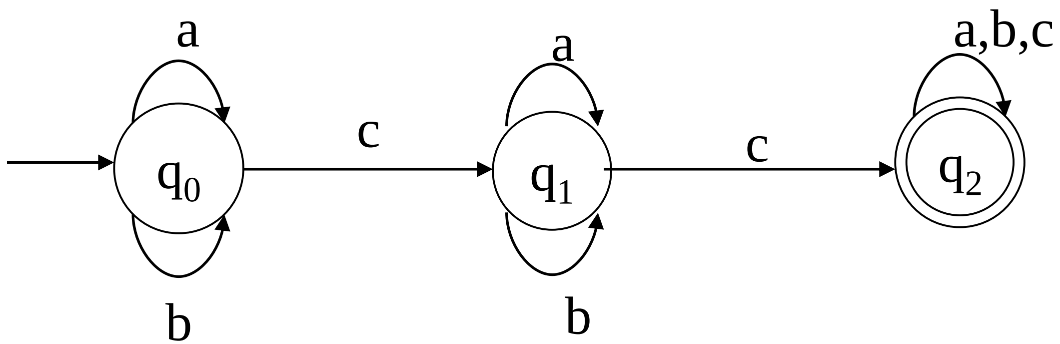
- Perhaps they accept overlapping classes of languages.

- In other words, is one of these two machine models more "powerful" than the other?

- Observation: Every DFA is an NFA.

- Consider the following DFA:

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

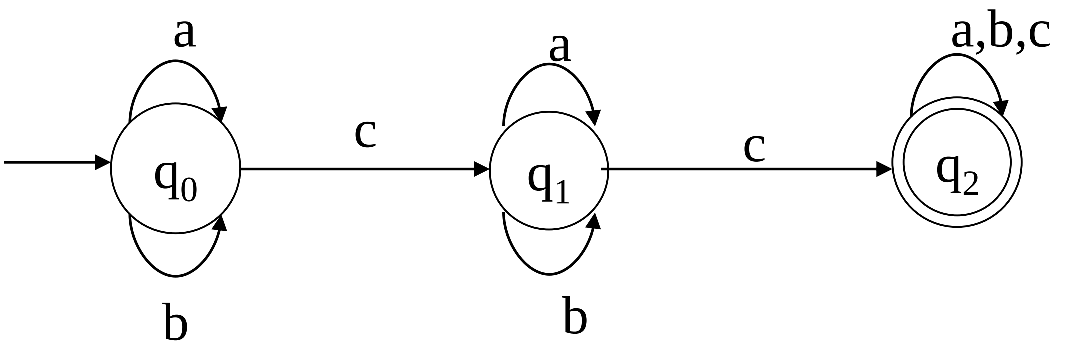


δ:

	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	$\{q_2\}$	$\{q_2\}$	$\{q_2\}$

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

- Thus, NFAs accept all regular languages, i.e., NFAs are at least as “powerful” as DFAs.

- Stated formally:

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).

- So NFAs accept the regular languages, but do they accept more?

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:

$Q' = 2^Q$
$\quad = \{Q_0, Q_1, \ldots,\}$

Each state in M' corresponds to a 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 $F\}$

$q'_0 = \{q_0\}$

$$\delta'(Q_u, a) = \bigcup_{p \in Q_u} \delta(p, a)$$

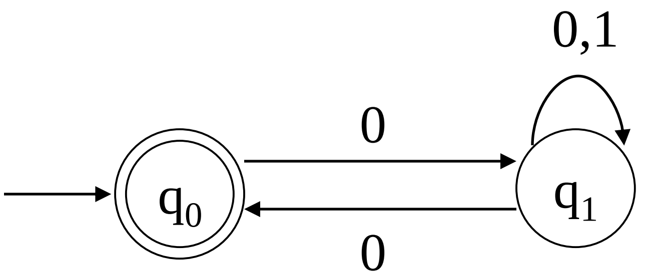
- Example:

$Q = \{q_0, q_1\}$

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

Start state is q_0

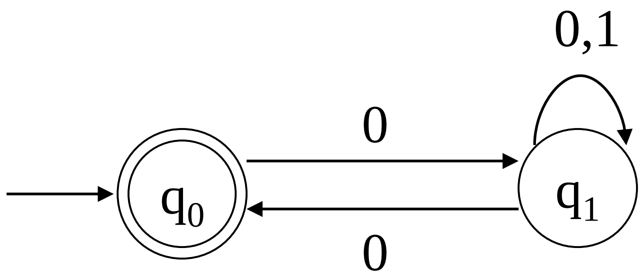
$F = \{q_0\}$

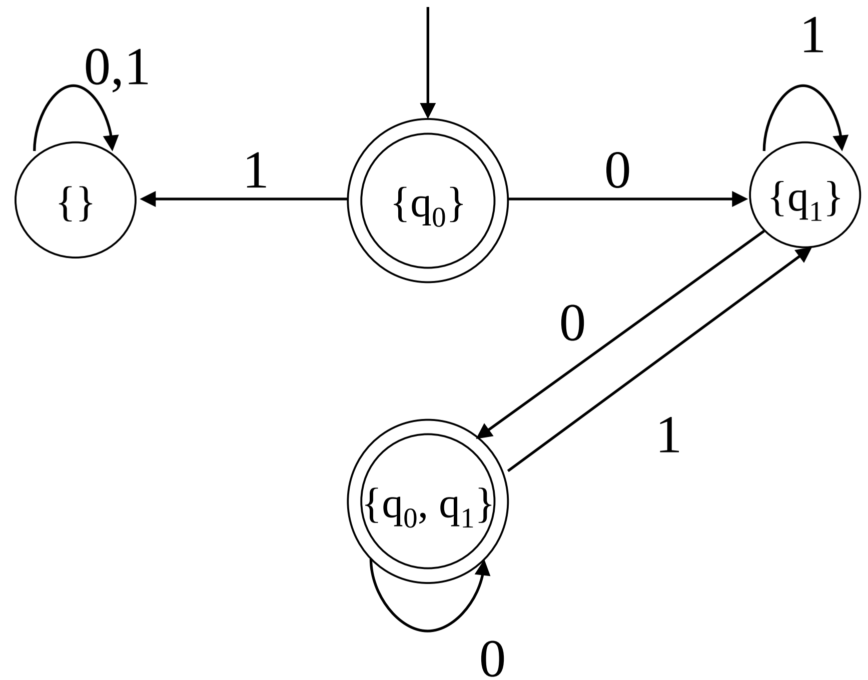


δ:

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

- Construct DFA M' as follows:





$\delta(q_0, 0) \cup \delta(q_1, 0) = \{q_0, q_1\}$	=>	$\delta'(\{q_0, q_1\}, 0) = \{q_0, q_1\}$
$\delta(q_0, 1) \cup \delta(q_1, 1) = \{q_1\}$	=>	$\delta'(\{q_0, q_1\}, 1) = \{q_1\}$
$\delta(q_0, 0) = \{q_1\}$	=>	$\delta'(\{q_0\}, 0) = \{q_1\}$
$\delta(q_0, 1) = \{\}$	=>	$\delta'(\{q_0\}, 1) = \{\}$
$\delta(q_1, 0) = \{q_0, q_1\}$	=>	$\delta'(\{q_1\}, 0) = \{q_0, q_1\}$
$\delta(q_1, 1) = \{q_1\}$	=>	$\delta'(\{q_1\}, 1) = \{q_1\}$

- Lastly, for the state corresponding to the empty set…

- Suppose R = {}

$$\begin{aligned} \delta'(R, 0) &= \bigcup_{q \in R} \delta(q, 0) && \text{From the construction} \\ &= \{\} && \text{Since } R = \{\} \end{aligned}$$

- So why does this construction work?
- Consider a simulation of the original NFA on the string 0010
- Consider a simulation of the resulting DFA on the same string
- So does this complete the proof?
- No! Technically, we need to prove that a string x is accepted by the DFA if and only if it is accepted by the NFA.
  - Performed by induction on the length of x
  - See the book

- Note the *constructive* nature of the proof, i.e., it shows how to construct the DFA (not all proofs are constructive).

- In fact, the construction could be programmed…

- The construction is not particularly efficient, however, i.e., the resulting DFA is not guaranteed to be minimum.
  - Some states in the DFA may not be reachable.
  - The book uses “lazy evaluation” to eliminate unreachable states.

- As an exercise, try the construction on some of the NFAs from class.

- Exercise - Convert the following NFA to a DFA:

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

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

- Exercise - Convert the following NFA to a DFA:

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

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

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.

NFAs with ε Moves

- An NFA-ε is a five-tuple:

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

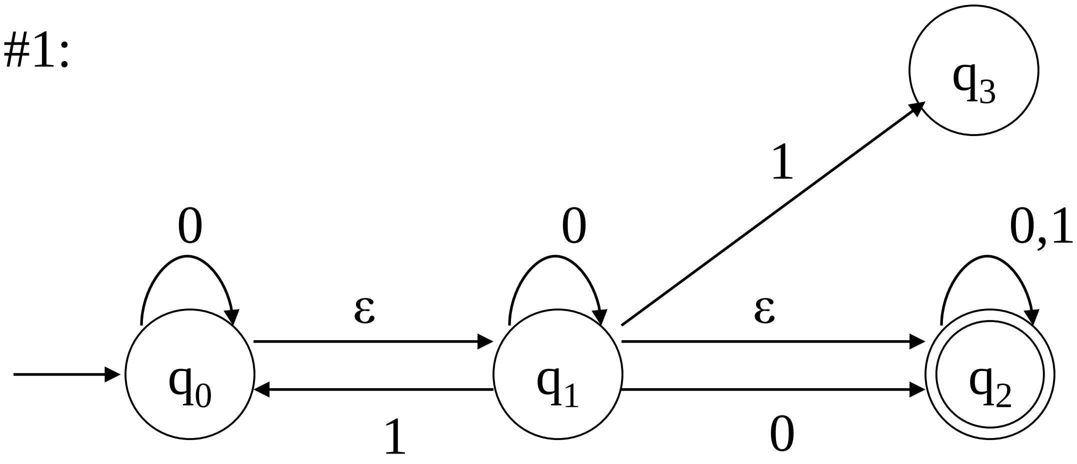
Q	A <u>finite</u> set of states
Σ	A <u>finite</u> input alphabet
q_0	The initial/starting state, q_0 is in Q
F	A set of final/accepting states, which is a subset of Q
δ	A transition function, which is a total function from Q x $(\Sigma \cup \{\varepsilon\})$ to 2^Q

  $\delta: (Q \times (\Sigma \cup \{\varepsilon\})) \rightarrow 2^Q$

  $\delta(q,s)$ -The set of all states p such that there is a transition labeled a from q to p, where a is in $\Sigma \cup \{\varepsilon\}$

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

- Example #1:



δ:

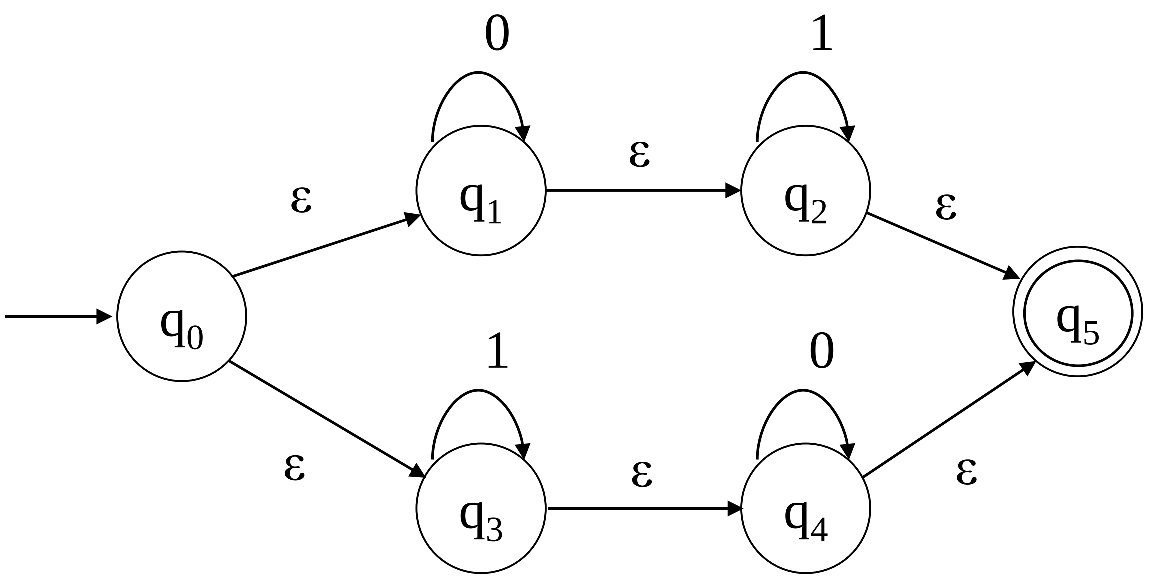
	0	1	ε
q_0	$\{q_0\}$	$\{\ \}$	$\{q_1\}$
q_1	$\{q_1, q_2\}$	$\{q_0, q_3\}$	$\{q_2\}$
q_2	$\{q_2\}$	$\{q_2\}$	$\{\ \}$
q_3	$\{\ \}$	$\{\ \}$	$\{\ \}$

- 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:

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

:

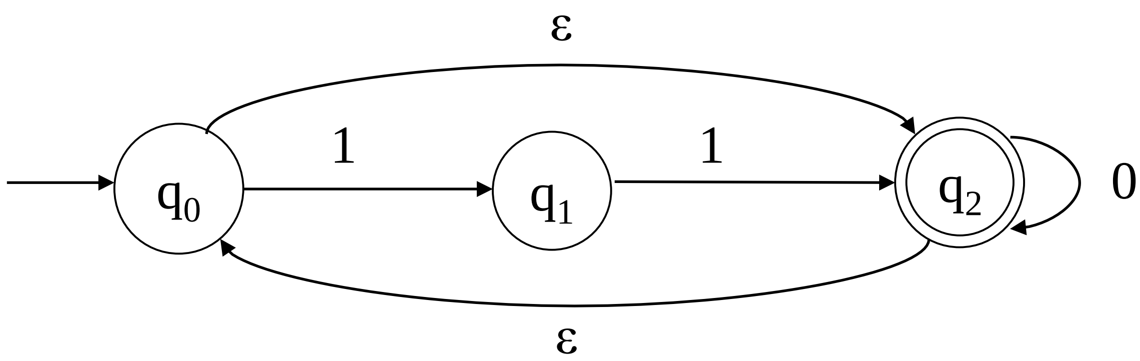
- Example #2:



- What language does the above NFA-ε accept?

- What does an equivalent DFA look like?

- Example #3:



- What language does the above NFA-ε accept?

- What does an equivalent DFA look like?

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.

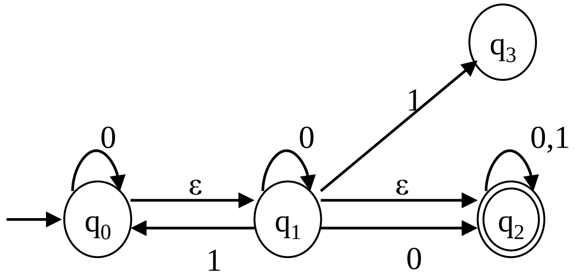
- Formalizing these concepts is a bit more tricky than it was for DFAs and NFAs…

ε-closure

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

- Examples: (for example #1)

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



ε-closure

- Formal (recursive) Definition:

  For any state q

  1) $q \in \varepsilon\text{-closure}(q)$

  2) if $p \in \varepsilon\text{-closure}(q)$ and $r \in \delta(p, \varepsilon)$ then $r \in \varepsilon\text{-closure}(q)$

Extension of δ to Strings

What we currently have: $\delta : (Q \times (\Sigma \cup \{\varepsilon\})) \rightarrow 2^Q$

What we want (why?): $\delta^\wedge : (Q \times \Sigma^*) \rightarrow 2^Q$

$\delta^\wedge(q,w)$ – The set of states the NFA-ε could be in after reading string w having started in state q.

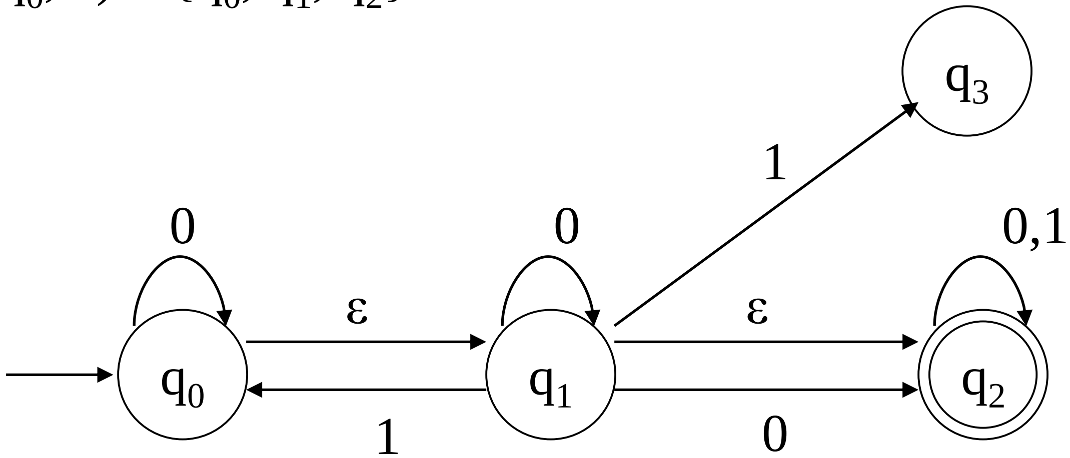
Formally - given any string x in Σ^*, where $|x|>=0$:

1) $\delta^\wedge(q, \varepsilon) = \varepsilon\text{-closure}(q)$, and if $|x| = 0$
2) For all w in Σ^* and a in Σ, if $|x| >= 1$, i.e., x=wa
   if $\delta^\wedge(q, w) = \{p_1, p_2,\ldots, p_k\}$, and $\bigcup_{i=1}^{k}\delta(p_i, a) = \{r_1, r_2,\ldots, r_m\}$
   then $\delta^\wedge(q,wa) = \bigcup_{i=1}^{m} \varepsilon\text{-closure}(r_i)$

- Note the difference between:

$\delta(q_0, 0) = q_0$

$\delta^{\wedge}(q_0, 0) = \{q_0, q_1, q_2\}$



So, unlike with DFAs and NFAs, we can't substitute δ for $\delta^{\wedge}$.

- See the book for a sample derivation.

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 $\delta^\wedge(q_0, w)$ contains at least one state in F, i.e., $\delta^\wedge(q_0, w) \cap F \neq \emptyset$.

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

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

- Other equivalent, less formal, definitions:

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

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

Equivalence of NFAs and NFA-εs

- Do NFAs and NFA-ε machines accept the same *class* of languages?

- Do they accept different classes 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 NFA?

- Perhaps they accept overlapping classes of languages.

- In other words, is one of these two machine models more "powerful" than the other?

- 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.

- Stated formally:

  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).

- But do NFA-ε machines accept more?

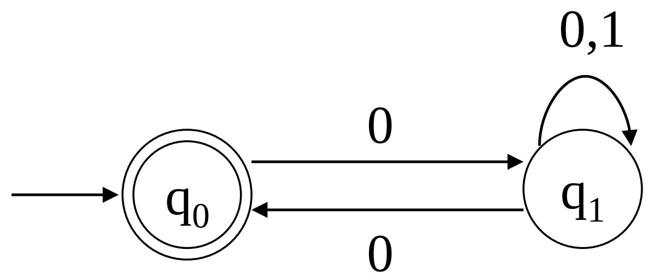
- Example: (NFA)

$Q = \{q_0, q_1\}$

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

Start state is q_0

$F = \{q_0\}$



δ:

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

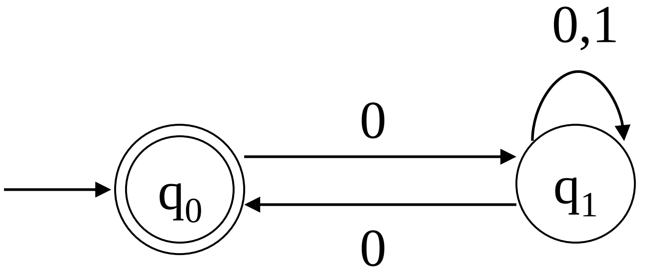
- Example: NFA-ε

$Q = \{q_0, q_1\}$

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

Start state is q_0

$F = \{q_0\}$



δ:

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

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.

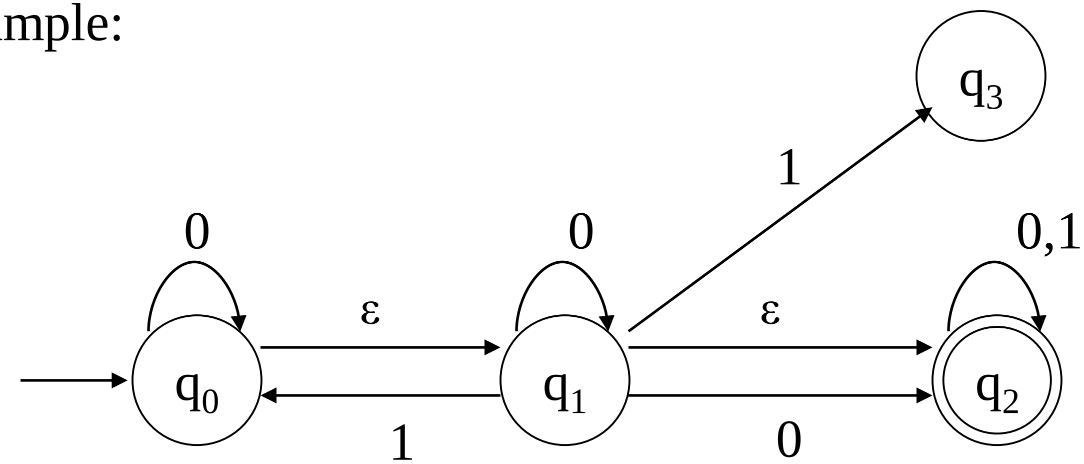
Define an NFA $M' = (Q, \Sigma, \delta', q_0, F')$ as:

$F' = F \cup \{q_0\}$ if ε-closure(q_0) contains at least one state from F
$F' = F$ otherwise

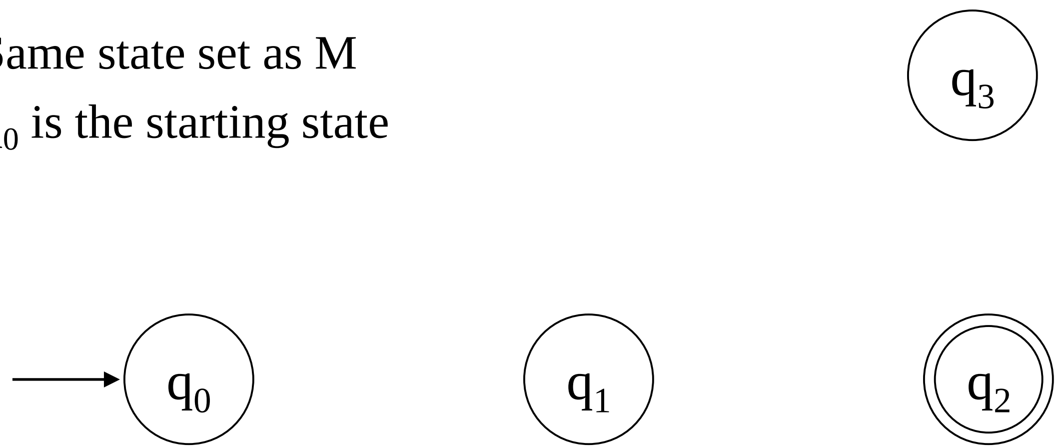
$\delta'(q, a) = \delta^{\wedge}(q, a)$ - for all q in Q and a in Σ

- 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

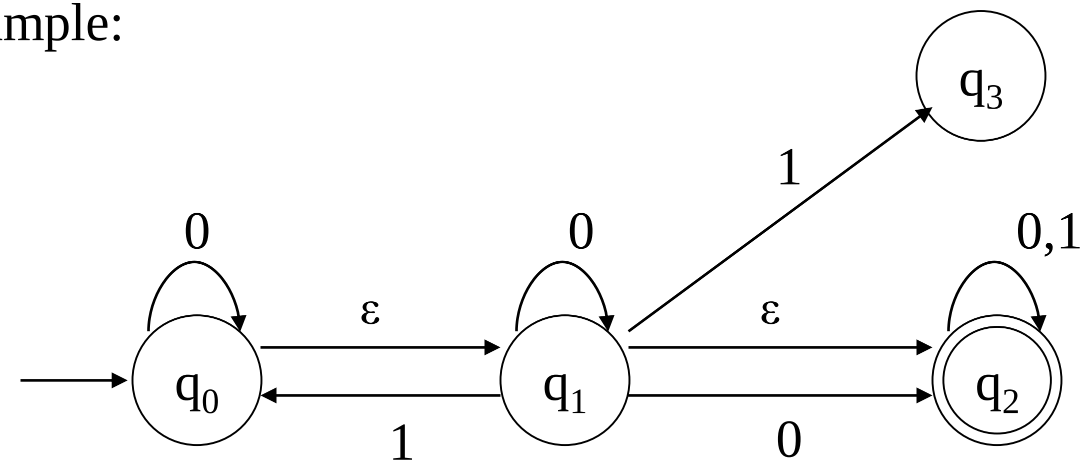
- Example:



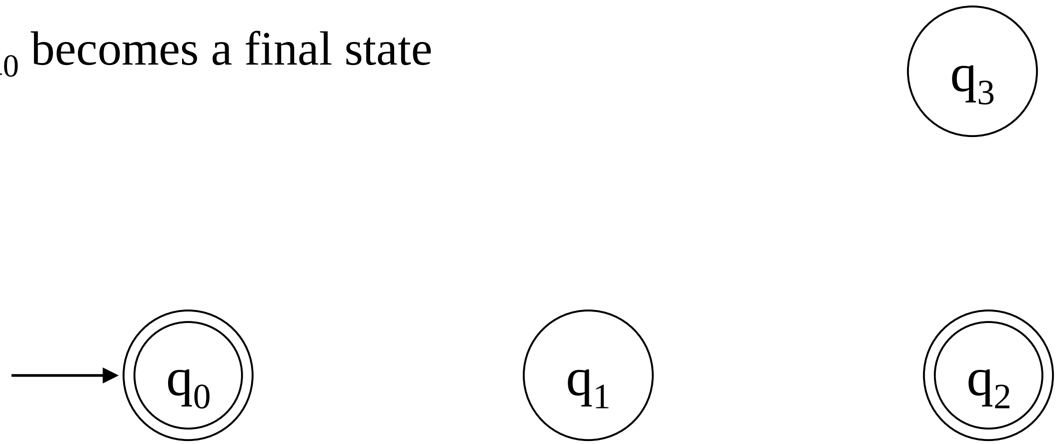
- Step #1:
  - Same state set as M
  - q_0 is the starting state



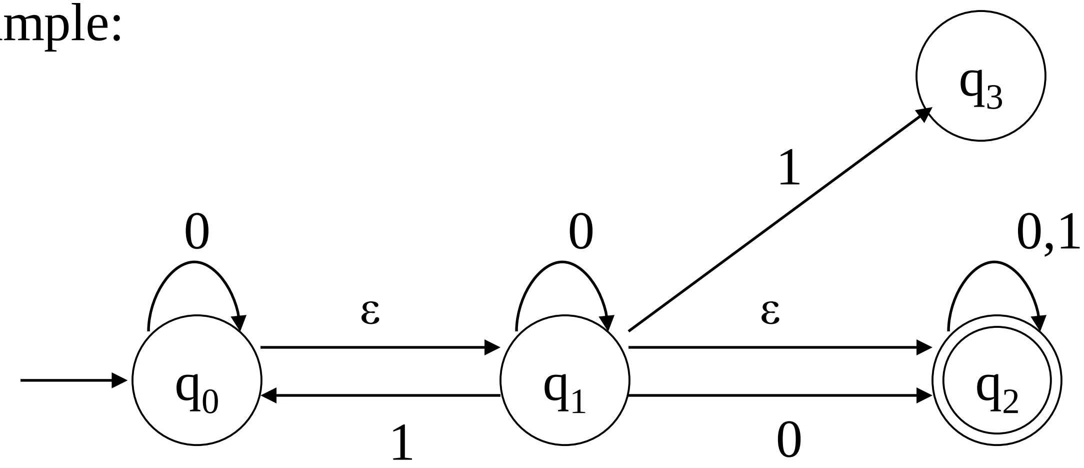
- Example:



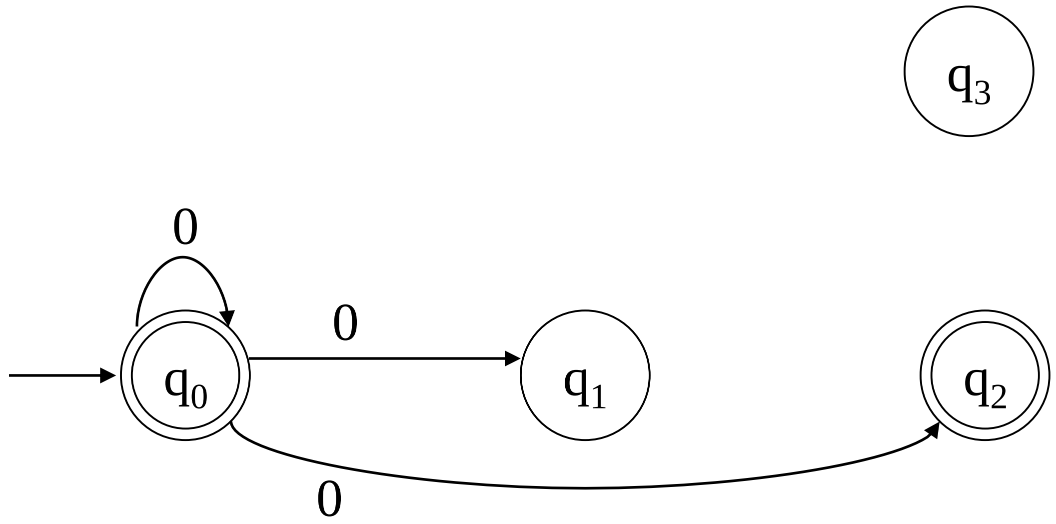
- Step #2:
  - q_0 becomes a final state



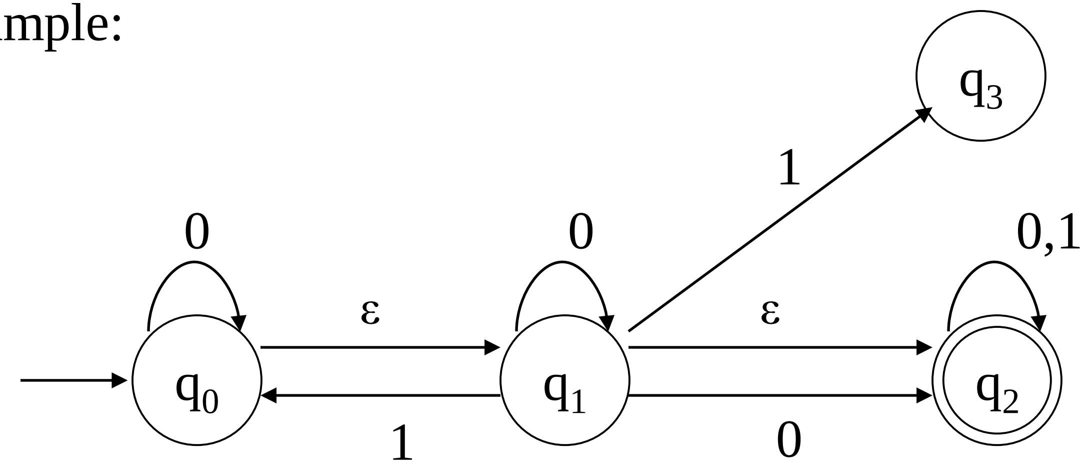
- Example:



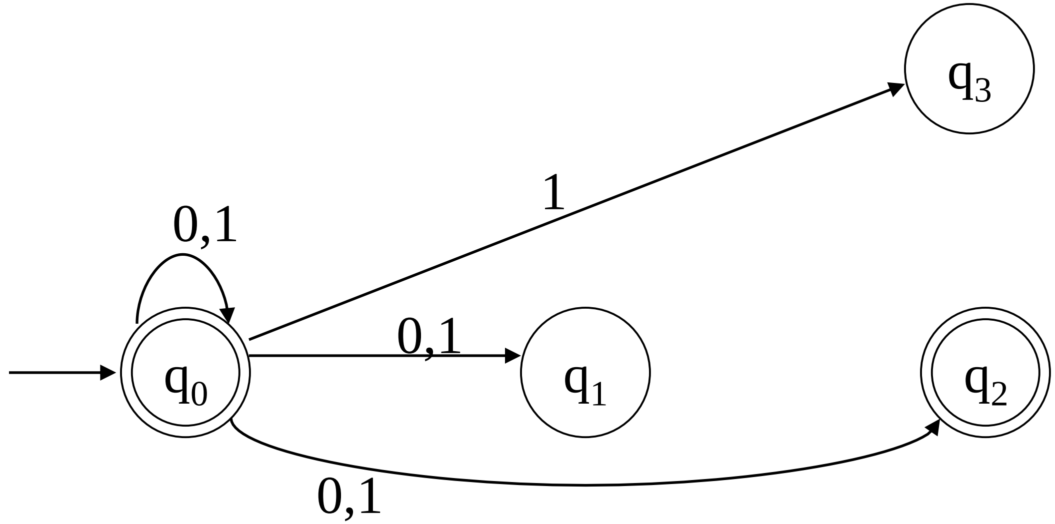
- Step #3:



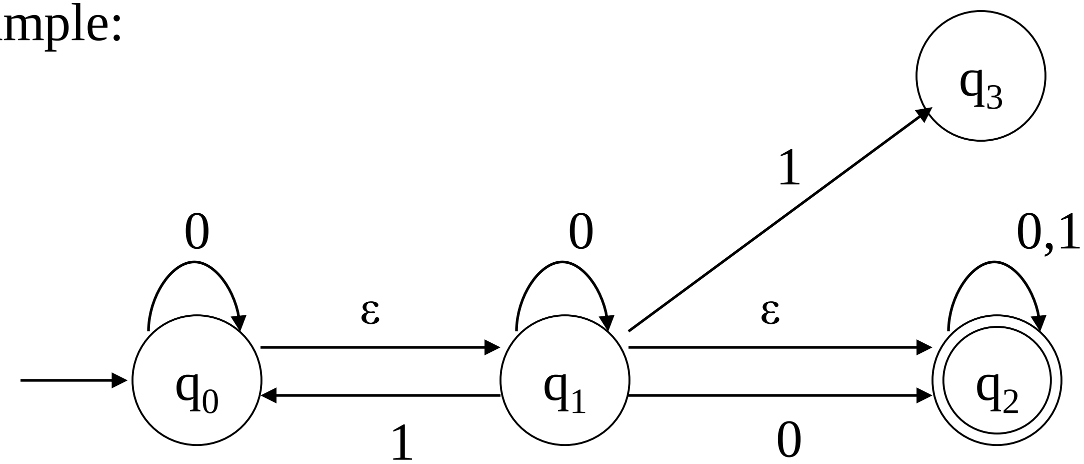
- Example:



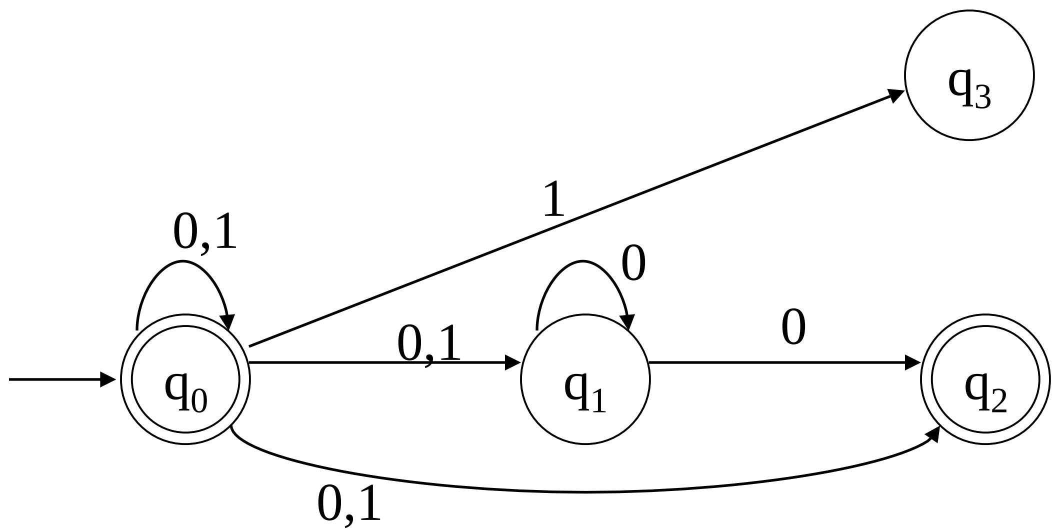
- Step #4:



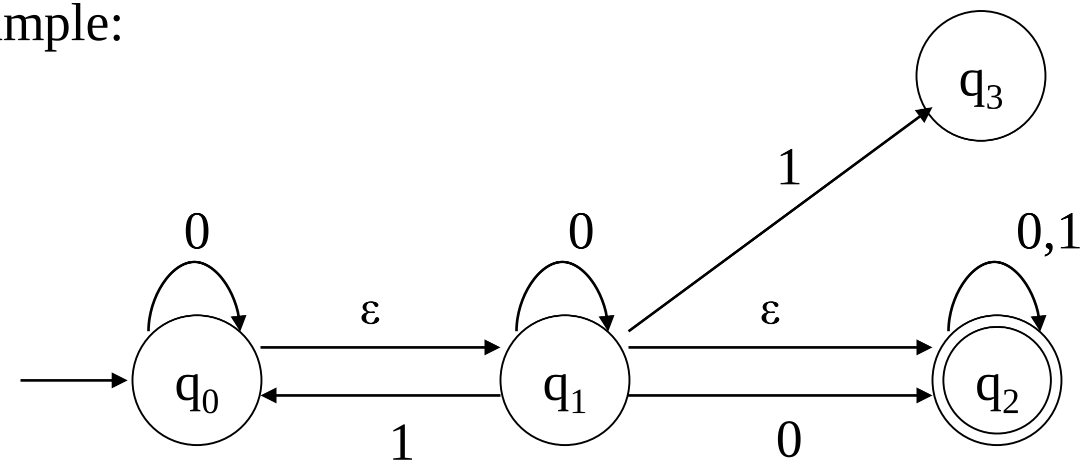
- Example:



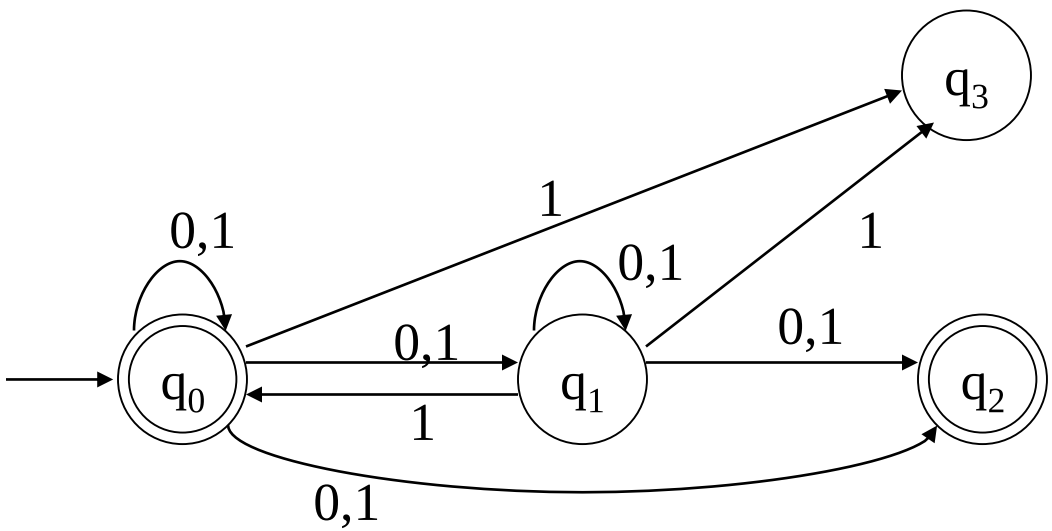
- Step #5:



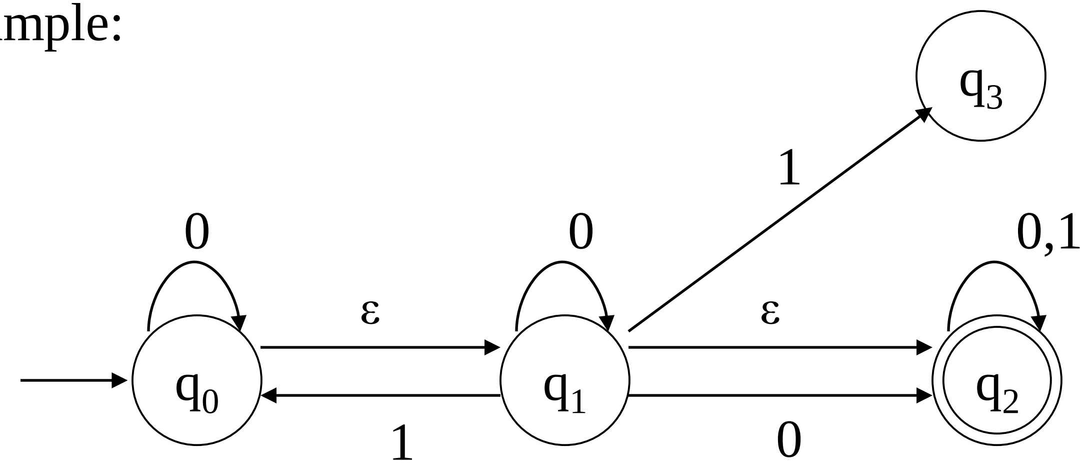
- Example:



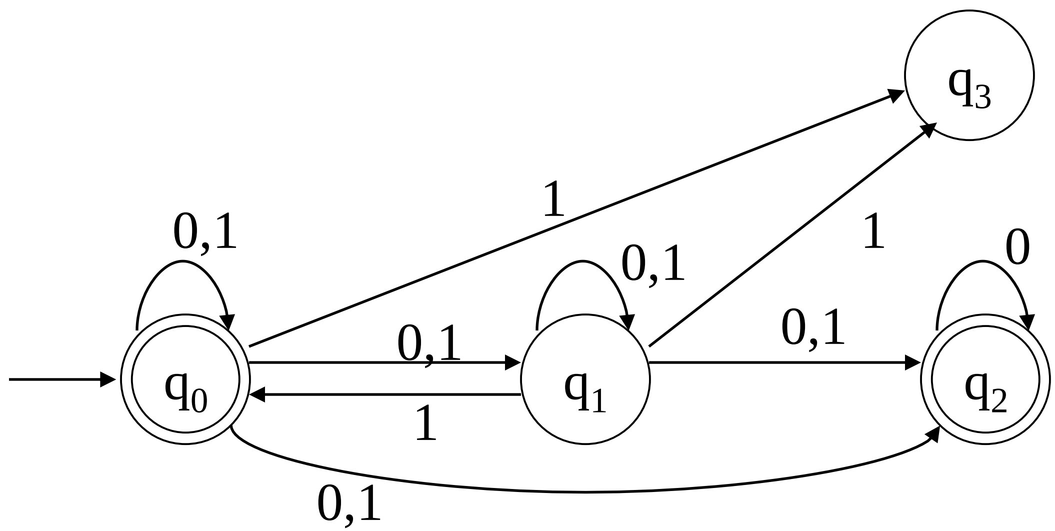
- Step #6:



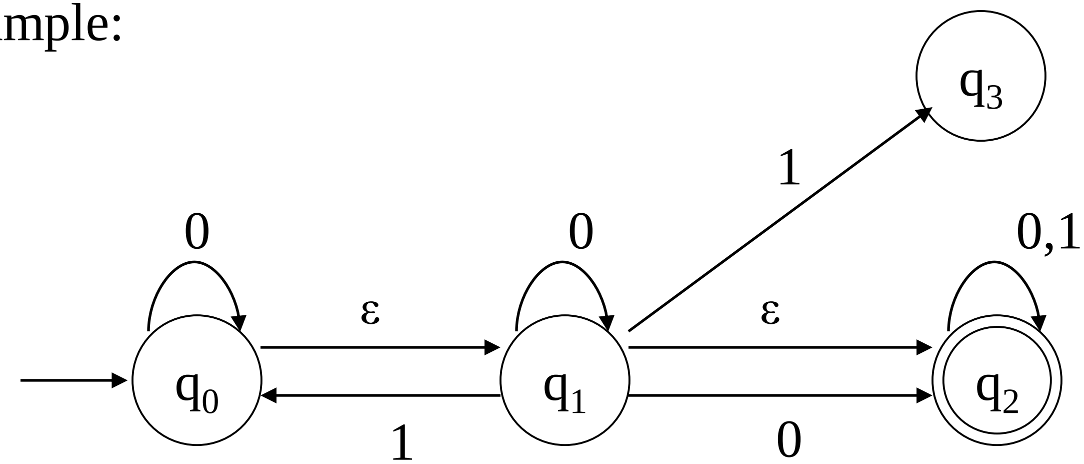
- Example:



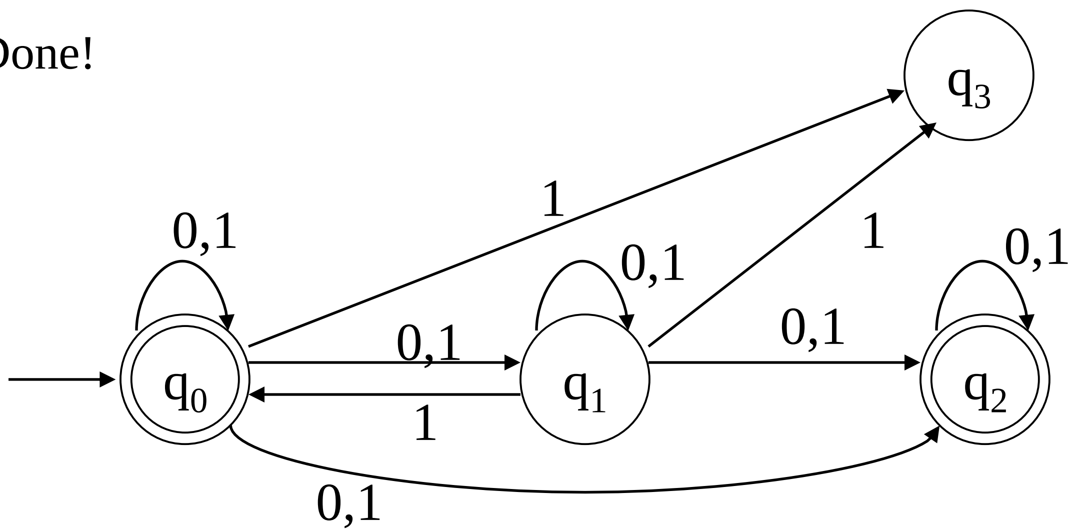
- Step #7:



- Example:



- Step #8:
  - 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.

- Finally, once again note the *constructive* nature of the proof, i.e., it shows how to construct the NFA.

- In fact, the construction could also be programmed…

- Give a DFA M such that:

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