Introduction to the Theory of Computation

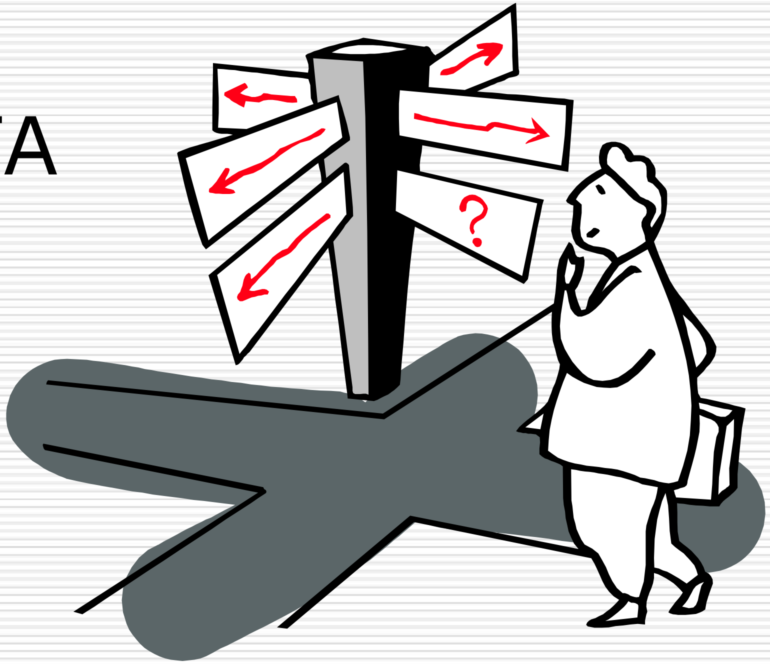
Part I: Automata and Languages

1. Regular Languages

- 1.1 Finite Automata
  - Deterministic FA
  - Non-Deterministic FA
  - ε-NFA
- 1.2 Regular Expressions
  - RE = FA
- 1.3 Properties of Regular Languages
  - Pumping lemma
  - Closure properties
  - Decision properties

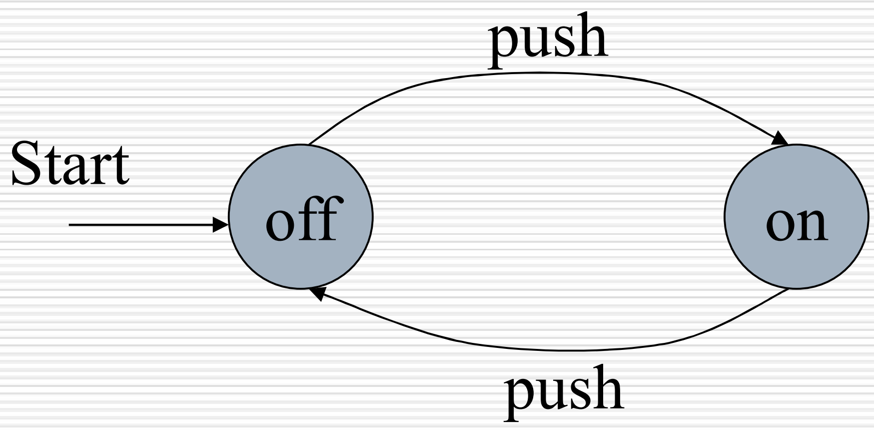
1.1 Finite Automata

- Deterministic FA
- Non-Deterministic FA
- NFA = DFA
- ε-NFA
- ε-NFA = NFA



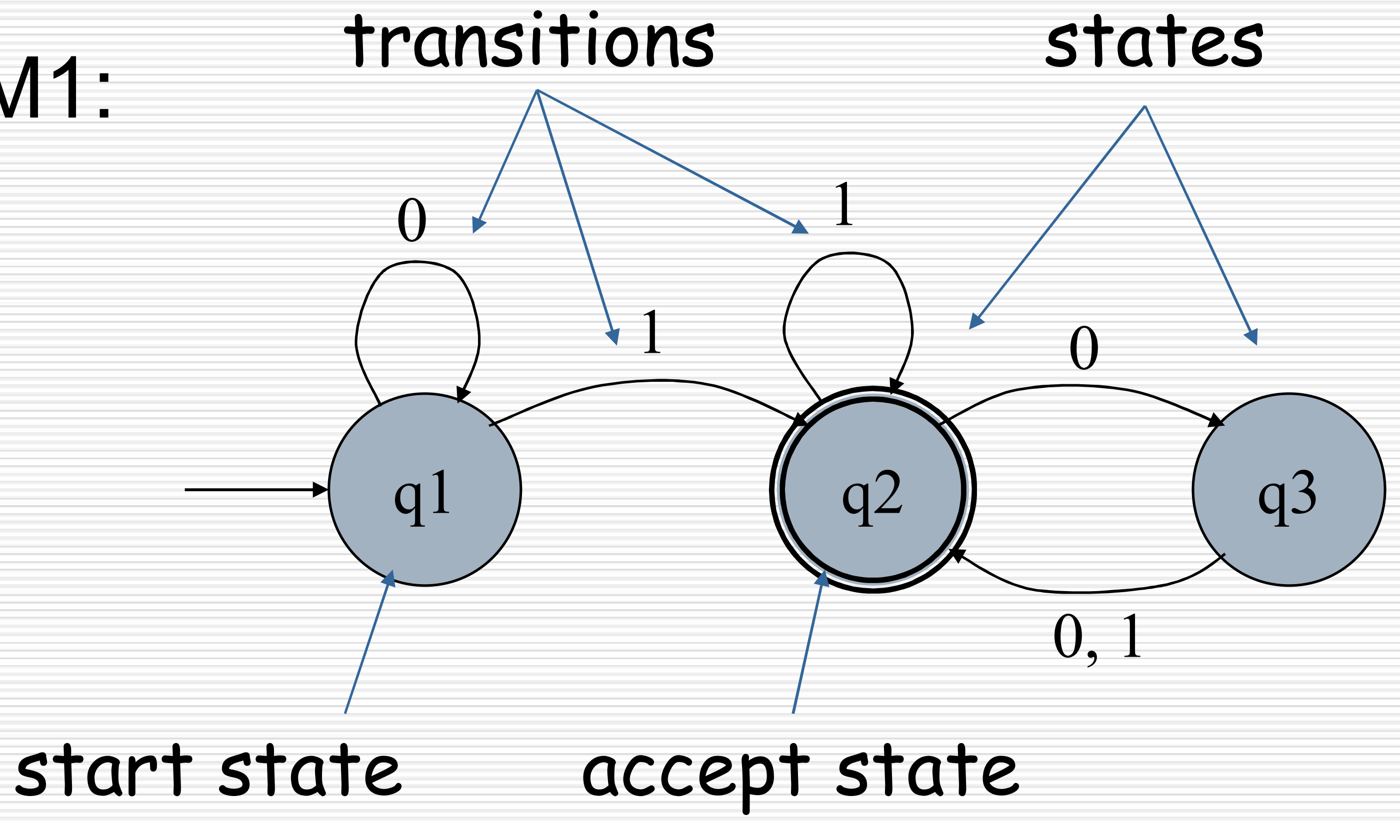
Finite Automata

- The simplest computational model, with very limited memory, but highly useful
- Example: a finite automaton modeling an on/off switch

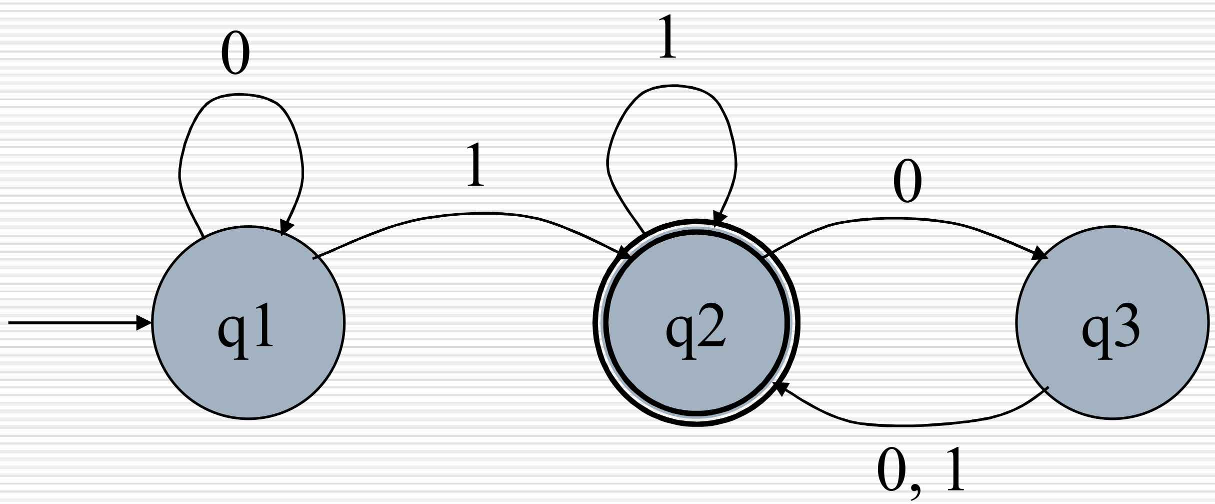


State Diagram of FA

M1:



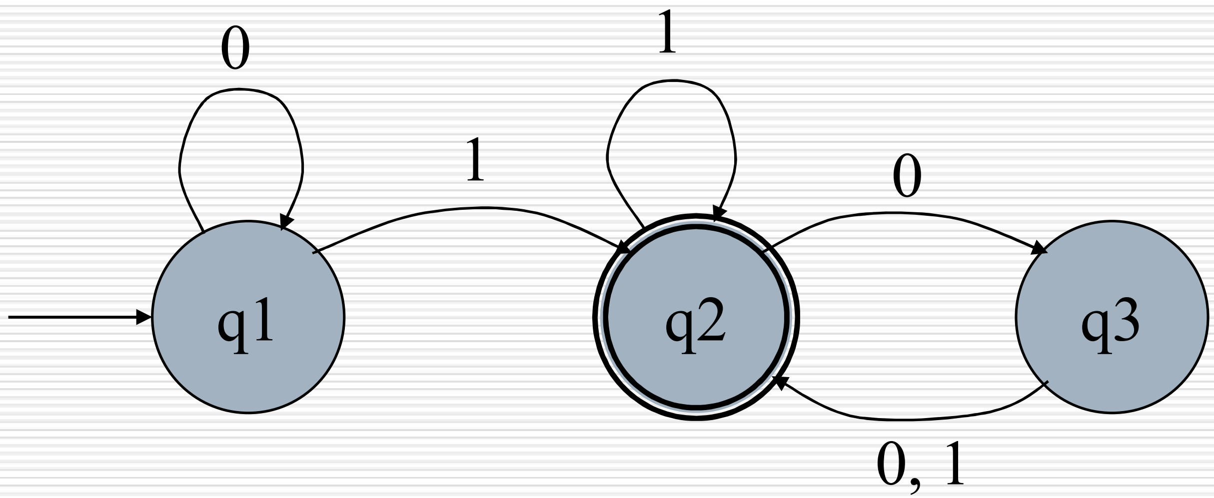
M1 Cont’d



on input “0110”, the machine goes:

q1→q1→q2→q2→q3 = “reject”

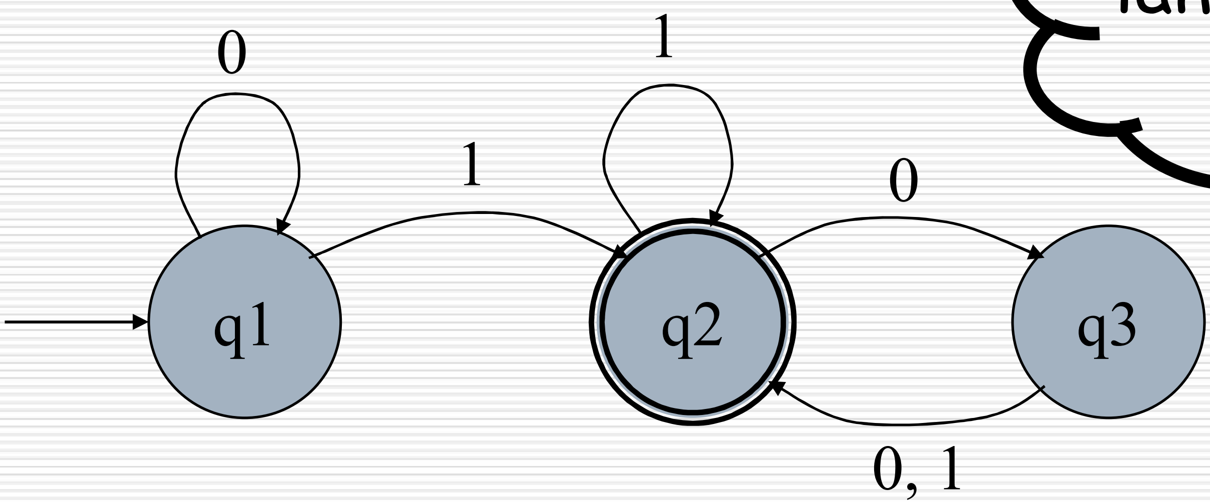
M1 Cont'd

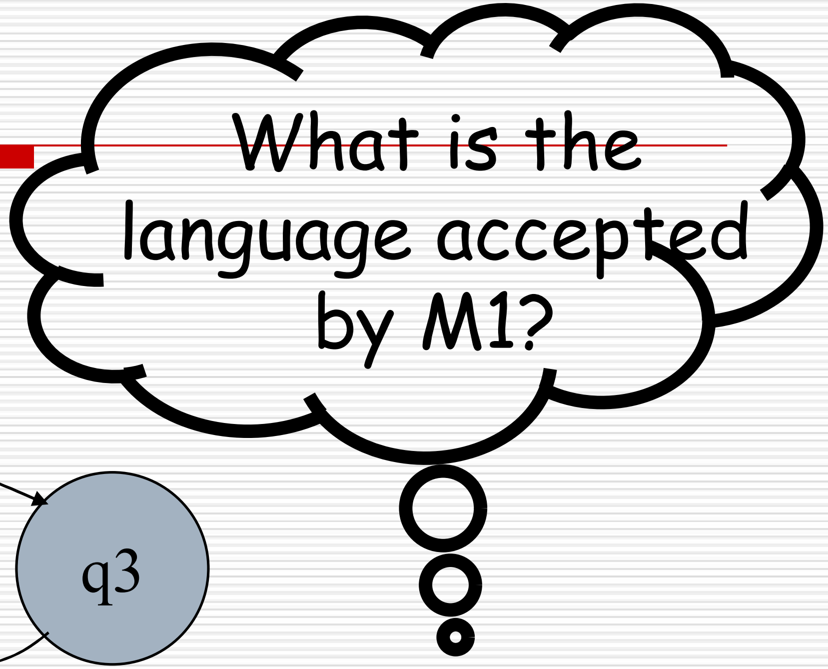


on input "101", the machine goes:

q1→q2→q3→q2 = "accept"

M1 Cont'd





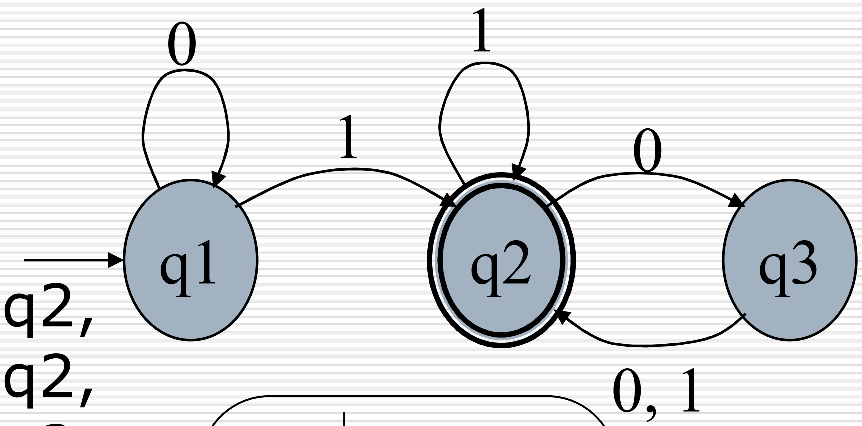
010: reject
11: accept
0110100: accept
010000010010: reject

Formal Definition of FA

- A finite automaton is defined by a 5-tuple $(Q, \Sigma, \delta, q_0, F)$
  - Q: finite set of states
  - Σ: finite alphabet
  - δ: transition function, $\delta: Q \times \Sigma \rightarrow Q$, takes a state and input symbol as arguments, and returns a state
  - $q_0 \in Q$: start state
  - $F \subseteq Q$: set of accept states

M1's Formal Definition

- M1 = (Q, Σ, δ, q_0, F), where
  - Q = {q1, q2, q3}
  - Σ = {0, 1}
  - δ(q1,0)=q1, δ(q1,1)=q2, δ(q2,0)=q3, δ(q2,1)=q2, δ(q3,0)=q2, δ(q3,1)=q2
  - q1 is the start state
  - F = {q2}



	0	1
q1	q1	q2
q2	q3	q2
q3	q2	q2

Extension of δ to Strings

- Intuitively, an FA accepts a string $w = a_1a_2...a_n$ if there is a path in the state diagram that:
  1. Begins at the start state,
  2. Ends at an accept state, and
  3. Has sequence of labels $a_1, a_2, ..., a_n$.

- Formally, the transition function δ can be extended to $\delta^*(q, w)$, where w is any string of input symbols.
  - Basis: $\delta^*(q, \varepsilon) = q$
  - Induction: $\delta^*(q, wa) = \delta(\delta^*(q, w), a)$

Language of an FA

- An FA $M = (Q, \Sigma, \delta, q_0, F)$ accepts a string w if $\delta^*(q_0, w) \in F$.
- The language recognized by an FA $M= (Q, \Sigma, \delta, q_0, F)$ is

$$L(M) = \{w \mid \delta^*(q_0, w) \in F\}.$$

- A language is called a regular language if some finite automaton recognizes it.

Designing Finite Automata

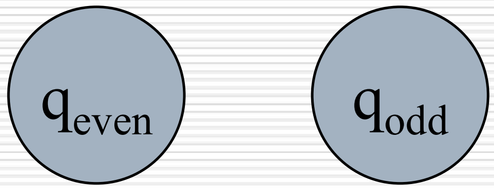
- Given some language, design a FA that recognizes it.
- Pretending to be a FA,
  - You see the input symbols one by one
  - You have finite memory, i.e. finite set of states, so remember only the crucial information (finite set of possibilities) about the string seen so far.

Example

- $\Sigma = \{0,1\}$, L = {w | w has odd number of 1s}, design a FA to recognize L.
  - What is the necessary information to remember? --- Is the number of 1s seen so far even or odd? Two possibilities.

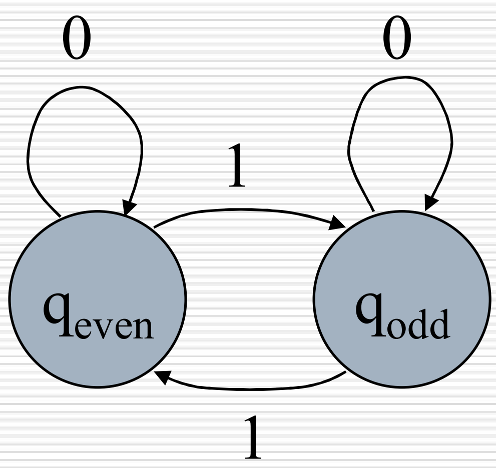
Example

- $\Sigma = \{0,1\}$, $L = \{w \mid w$ has odd number of 1s$\}$, design a FA to recognize L.
  - What is the necessary information to remember? --- Is the number of 1s seen so far even or odd? Two possibilities.
  - Assign a state to each possibility.



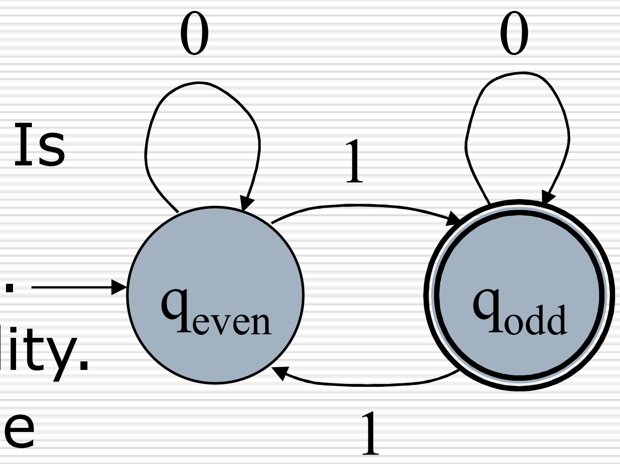
Example

- Σ ={0,1}, L = {w | w has odd number of 1s}, design a FA to recognize L.
  - What is the necessary information to remember? --- Is the number of 1s seen so far even or odd? Two possibilities.
  - Assign a state to each possibility.
  - Assign the transitions from one possibility to another upon reading a symbol.



Example

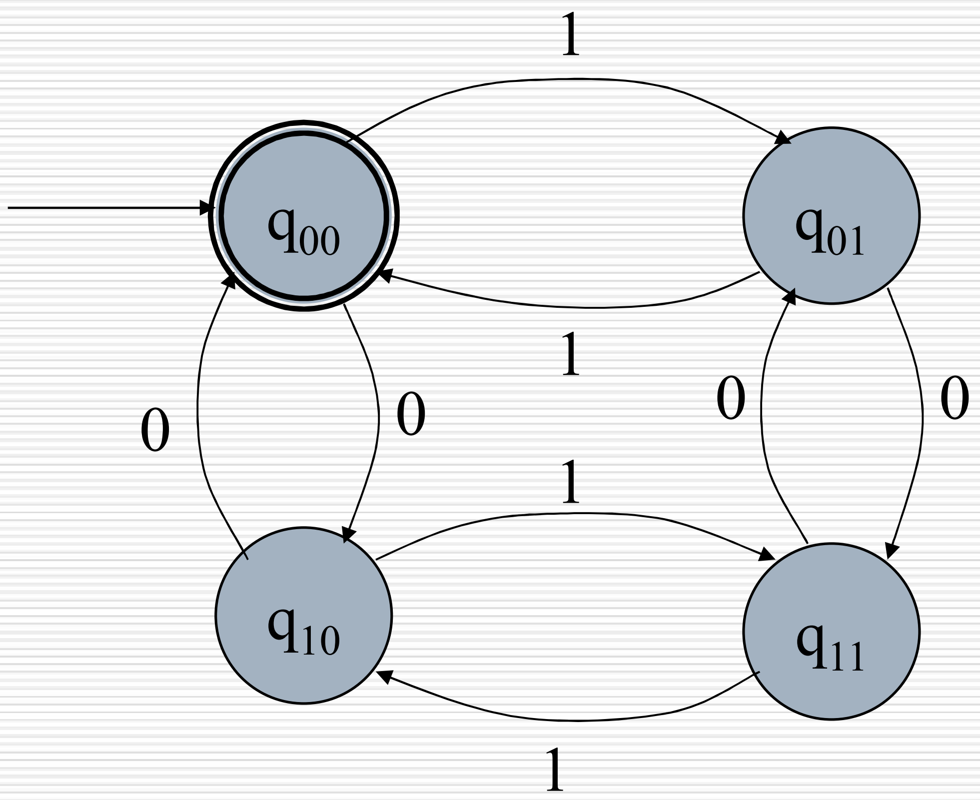
- $\Sigma = \{0,1\}$, L = {w | w has odd number of 1s}, design a FA to recognize L.
  - What is the necessary information to remember? --- Is the number of 1s seen so far even or odd? Two possibilities.
  - Assign a state to each possibility.
  - Assign the transitions from one possibility to another upon reading a symbol.
  - Set the start and accept states.



Exercise

- $\Sigma = \{0,1\}$, L = {w | w has even number of 0s and even number of 1s}, design a FA to recognize L.
  - What to remember?
  - How many possibilities?

Exercise

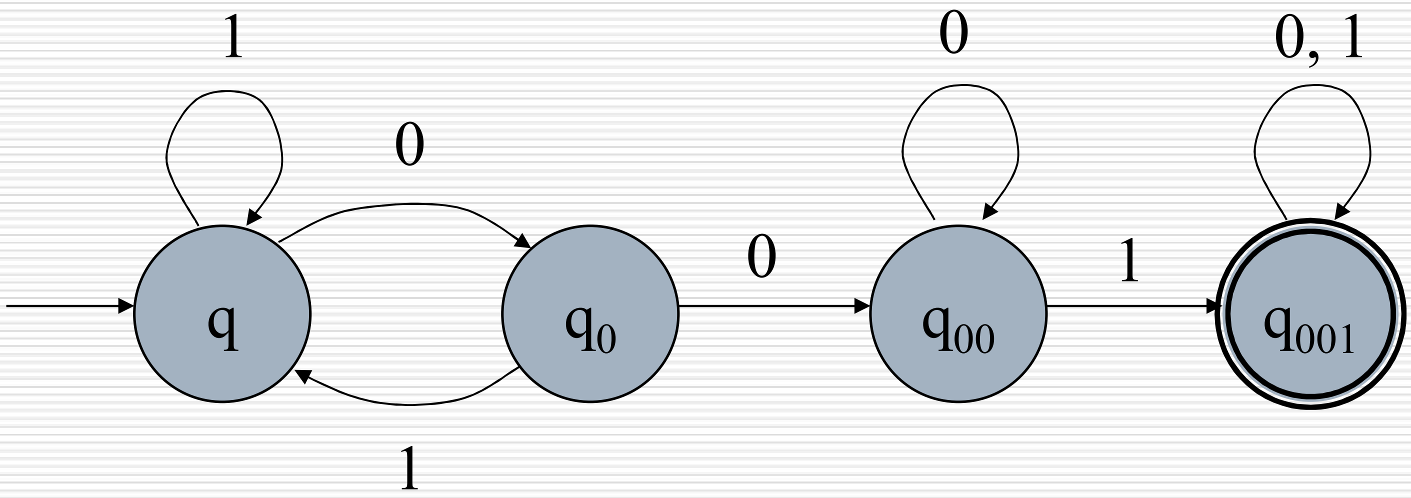


Example 1.9

- $\Sigma = \{0,1\}$, L = {w | w contains 001 as a substring}, design a FA to recognize L.
  - four possibilities:

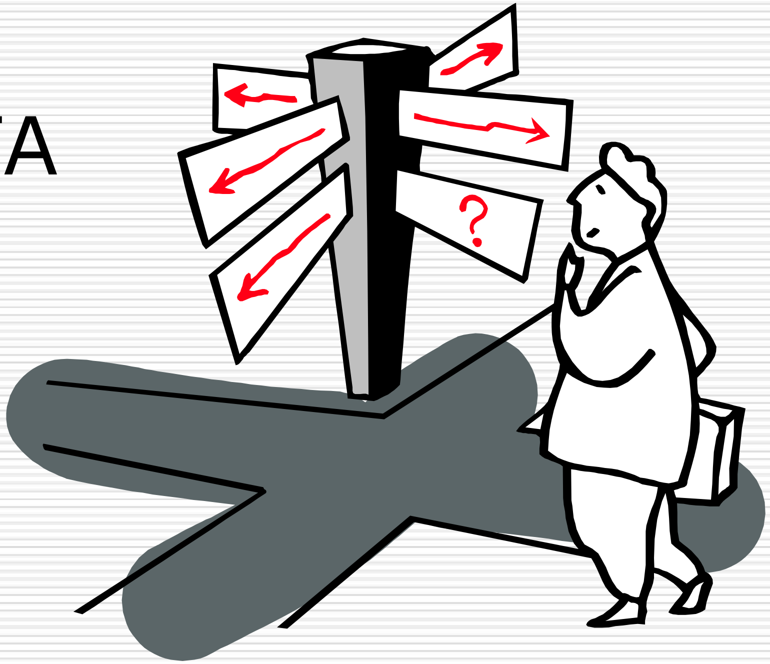
1. haven't just seen any symbols of 001 --- q
2. have just seen a 0 --- q_0
3. have just seen a 00 --- q_{00}
4. have seen the entire pattern 001 --- q_{001}

Example 1.9



1.1 Finite Automata

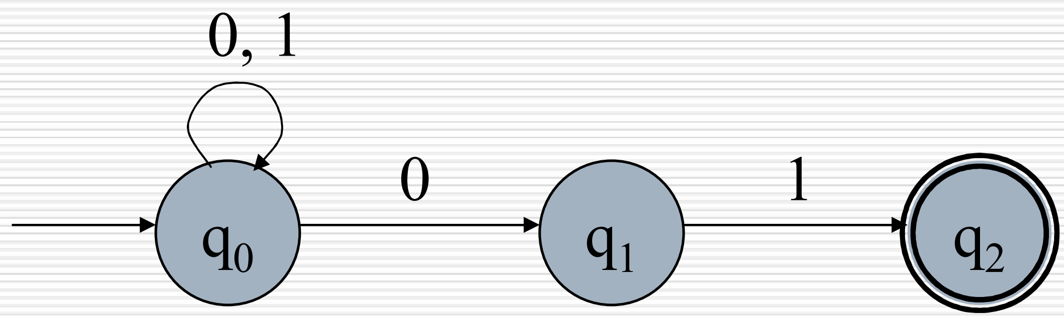
- Deterministic FA
- Non-Deterministic FA
- NFA = DFA
- ε-NFA
- ε-NFA = NFA



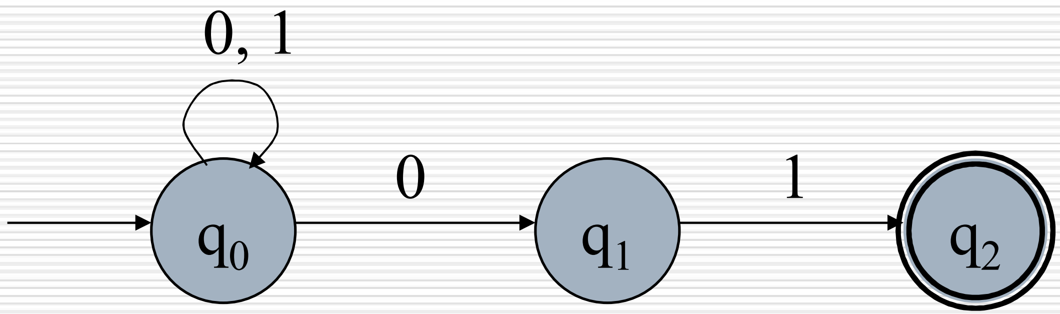
Nondeterministic Finite Automata

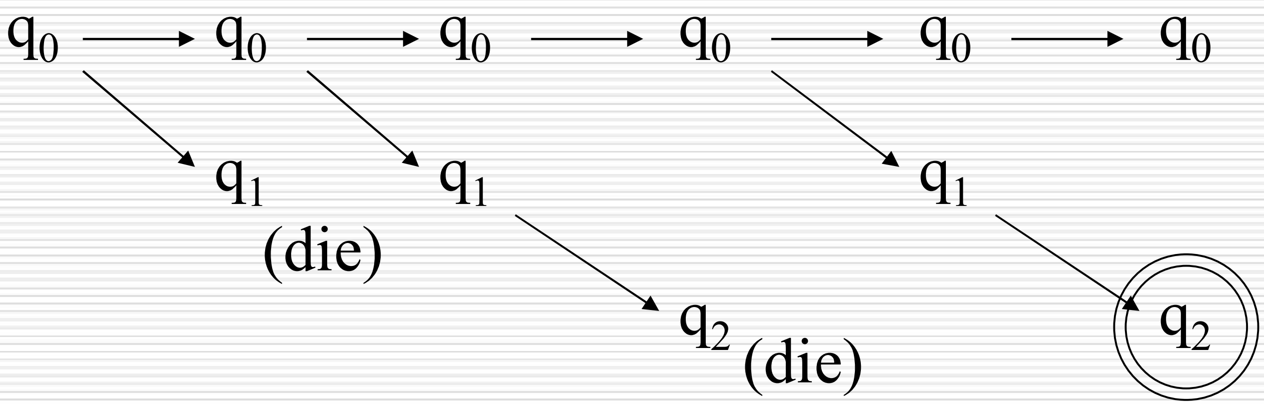
- Deterministic: At any point when the machine is in a state with an input symbol, there is a unique next state to go.
- Non-Deterministic: There can be more than one next state for each state-input pair.
- Example: an automaton that accepts all strings ending in 01.

M2:



How an NFA computes?





Input: 0 0 1 0 1

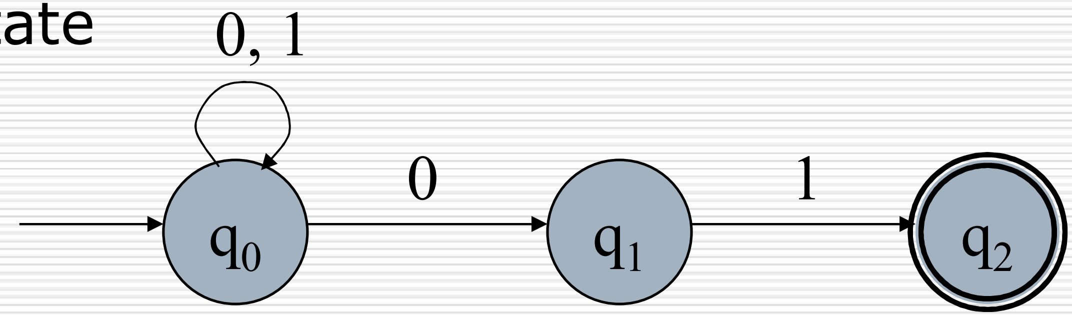
Formal Definition of NFA

- An NFA can be in several states at once, or viewed in another way, it can "guess" which state to go next.
- Formally, an NFA is a 5-tuple $N = (Q, \Sigma, \delta, q_0, F)$, where all is as DFA, but
  - $\delta: Q \times \Sigma \rightarrow \mathcal{P}(Q)$ is a transition function from $Q \times \Sigma$ to the power set of Q.

M2's Formal Definition

- M2 = $(Q, \Sigma, \delta, q_0, F)$, where
  - $Q = \{q_0, q_1, q_2\}$
  - $\Sigma = \{0, 1\}$
  - $\delta(q_0,0)=\{q_0,q_1\}$, $\delta(q_0,1)=\{q_0\}$, $\delta(q_1,1)=\{q_2\}$
  - q_0 is the start state
  - $F = \{q_2\}$

	0	1
q_0	$\{q_0,q_1\}$	$\{q_0\}$
q_1	$\varnothing$	$\{q_2\}$
q_2	$\varnothing$	$\varnothing$



Language of an NFA

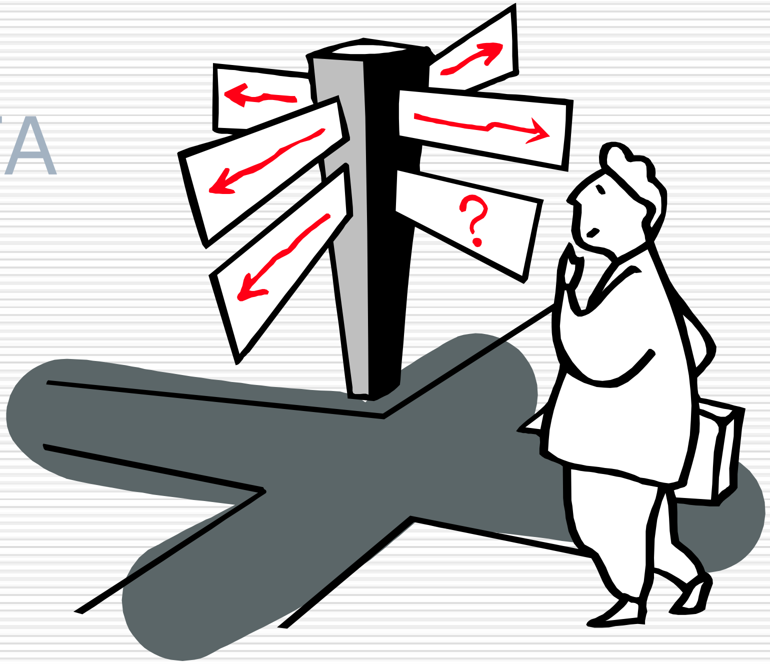
- Extension of δ to $\delta^*(q, w)$:
  - Basis: $\delta^*(q, \varepsilon) = \{q\}$
  - Induction: $\delta^*(q, wa) = \bigcup_{p \in \delta^*(q, w)} \delta(p, a)$
- Formally, the language recognized by $N = (Q, \Sigma, \delta, q_0, F)$ is

$$L(M) = \{w \mid \delta^*(q_0, w) \cap F \neq \emptyset\}.$$

(i.e. if *any* path from the start state to an accept state is labeled w.)

1.1 Finite Automata

- Deterministic FA
- Non-Deterministic FA
- NFA = DFA
- ε-NFA
- ε-NFA = NFA

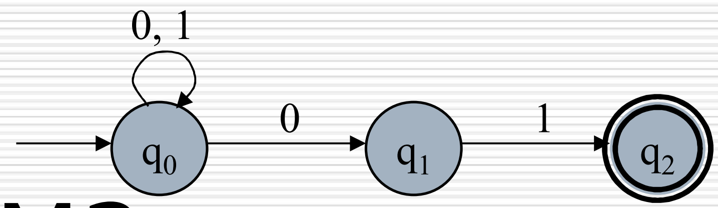


Equivalence of NFA and DFA

- NFA's are usually easier to "program" in.
- Surprisingly, for each NFA there is an equivalent (recognizes the same language) DFA.
- But the DFA can have exponentially many states.

Subset Construction

- Given an NFA $N=(Q_N, \Sigma, \delta_N, q_0, F_N)$, we will construct an DFA $D=(Q_D, \Sigma, \delta_D, \{q_0\}, F_D)$, such that $L(D) = L(N)$.
- Subset construction:
  - $Q_D = \{S \mid S \subseteq Q_N\}$, i.e. power set of Q_N
  - $\delta_D(S, a) = \cup_{p \in S} \delta_N(p, a)$
  - $F_D = \{S \mid S \cap F \neq \emptyset, S \in Q_D\}$

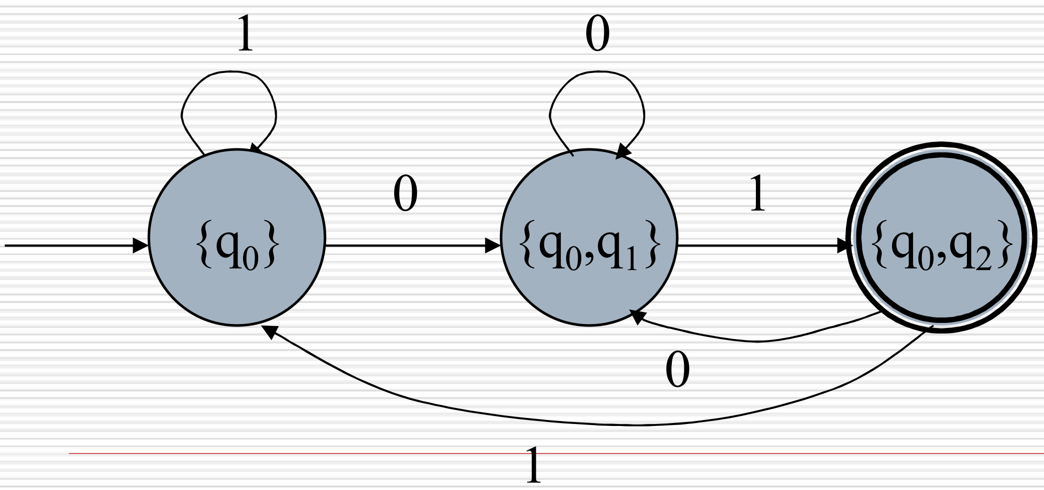


Construct DFA from M2

- Subset construction:

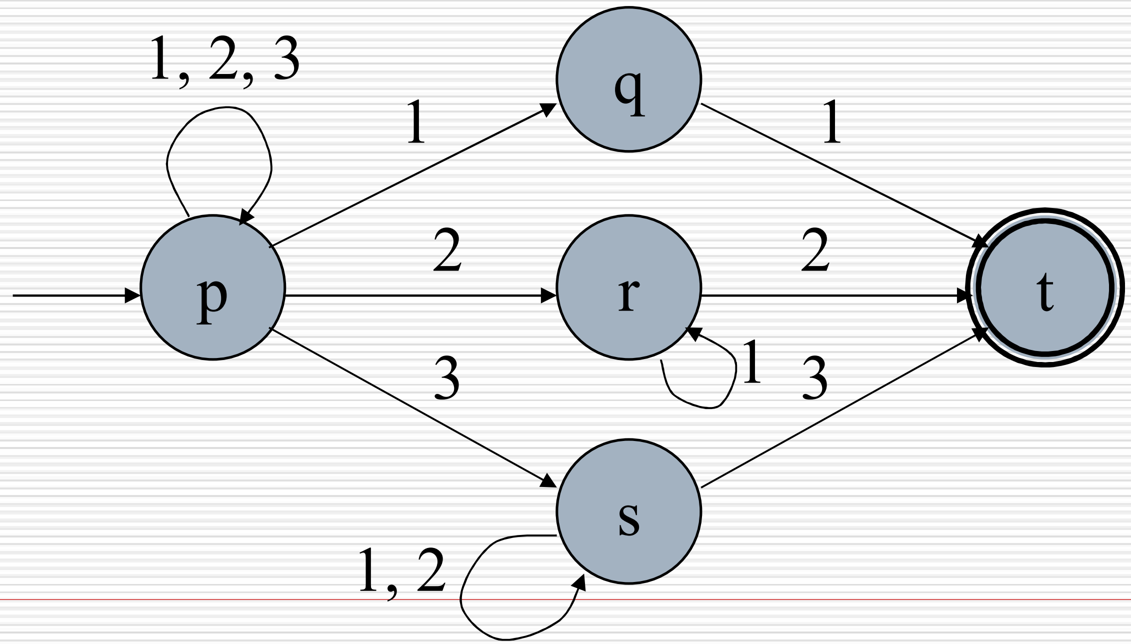
8 possible subsets,
3 accessible:

	0	1
~~$\varnothing$~~	~~$\varnothing$~~	~~$\varnothing$~~
$\{q_0\}$	$\{q_0,q_1\}$	$\{q_0\}$
~~$\{q_1\}$~~	~~$\varnothing$~~	~~$\{q_2\}$~~
~~$\{q_2\}$~~	~~$\varnothing$~~	~~$\varnothing$~~
$\{q_0,q_1\}$	$\{q_0,q_1\}$	$\{q_0,q_2\}$
$\{q_0,q_2\}$	$\{q_0,q_1\}$	$\{q_0\}$
~~$\{q_1,q_2\}$~~	~~$\varnothing$~~	~~$\{q_2\}$~~
~~$\{q_0,q_1,q_2\}$~~	~~$\{q_0,q_1\}$~~	~~$\{q_0,q_2\}$~~



Example NFA

- Design an NFA to accept strings over alphabet {1, 2, 3} such that the last symbol appears previously, without any intervening higher symbol, e.g., …11, …21112, …312123.



Equivalent DFA

- 32 possible subsets, 15 accessible:
- DFA is much larger than NFA

	1	2	3
p	pq	pr	ps
pq	pqt	pr	ps
pqt	pqt	pr	ps
pr	pqr	prt	ps
prt	pqr	prt	ps
ps	pqs	prs	pst
pst	pqs	prs	pst
prs	pqrs	prst	pst
prst	pqrs	prst	pst
pqs	pqst	prs	pst
pqst	pqst	prs	pst
pqr	pqrt	prt	ps
pqrt	pqrt	prt	ps
pqrs	pqrst	prst	pst
pqrst	pqrst	prst	pst

Proof: L(D)=L(N)

- Induction on |w| to show that

$$\delta^*_D(\{q_0\}, w) = \delta^*_N(q_0, w)$$

  - Basis: $w=\varepsilon$, the claim follows from the def.
  - Induction: $\delta^*_D(\{q_0\}, wa) = \delta_D(\delta^*_D(\{q_0\}, w), a)$

$$= \delta_D(\delta^*_N(q_0, w), a)$$

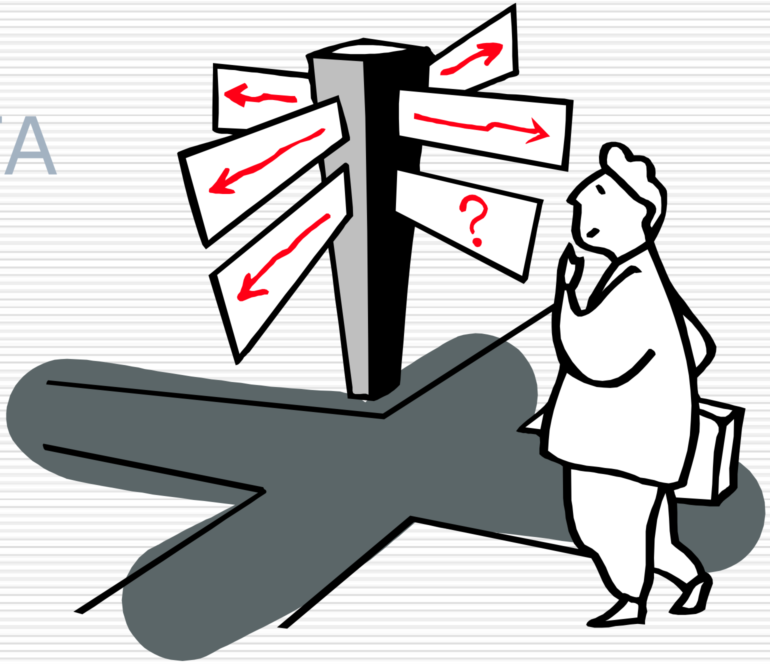
$$= \cup_{p\in\delta^*_N(q0,\ w)}\delta_N(p, a)$$

$$= \delta^*_N(q_0, wa)$$

- Then it follows that L(D) = L(N), why?

1.1 Finite Automata

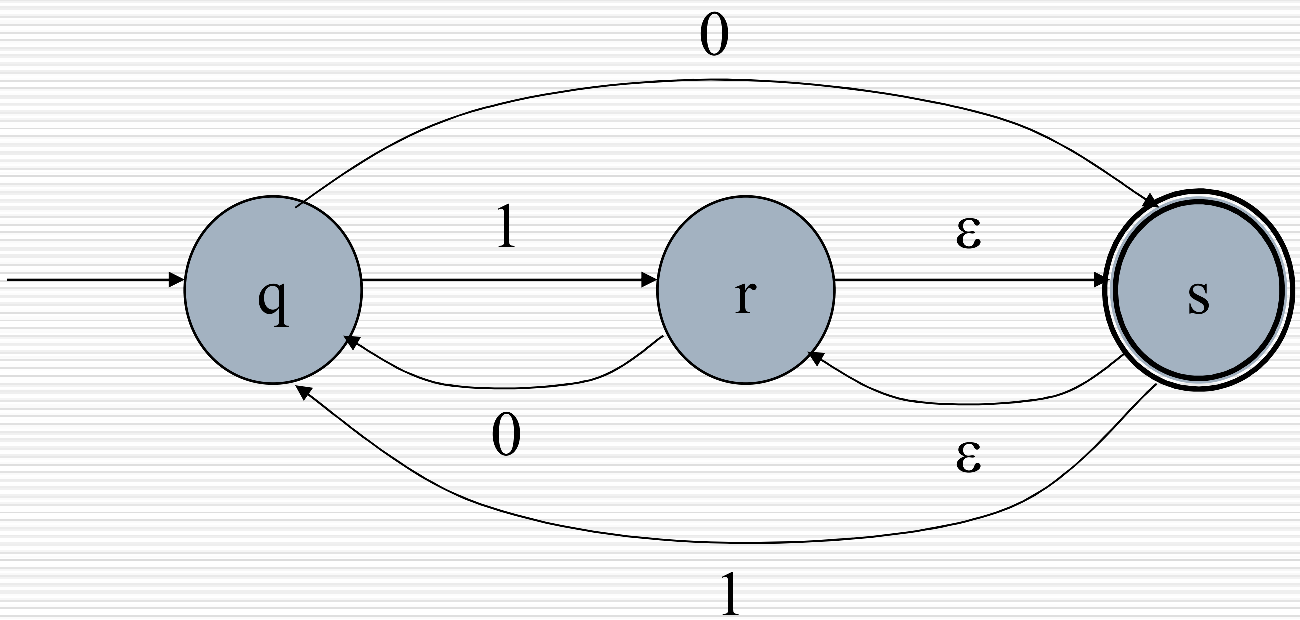
- Deterministic FA
- Non-Deterministic FA
- NFA = DFA
- ε-NFA
- ε-NFA = NFA



Finite Automata with ε-Transitions

- Allow ε to be a label on arcs.
- Formally the transition function δ of an ε-NFA is from Q x $\Sigma \cup \{\varepsilon\}$ to $\mathcal{P}(Q)$.
- Nothing else changes: acceptance of *w* is still the existence of a path from the start state to an accept state with label *w*. But ε can appear on arcs, and means the empty string (i.e., no visible contribution to w).

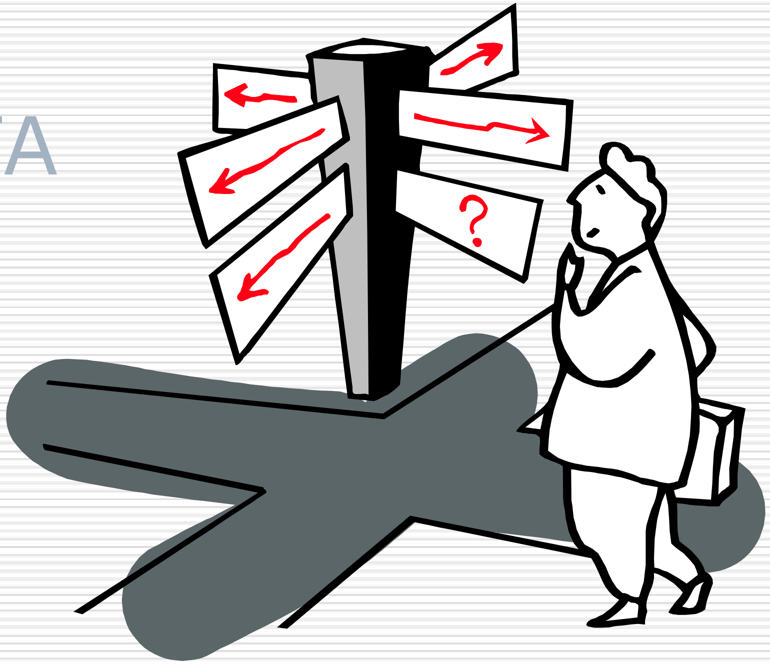
Example of an ε-NFA



"001" is accepted by the path
q→s→r→q→r→s with label 0ε01ε = 001

1.1 Finite Automata

- Deterministic FA
- Non-Deterministic FA
- NFA = DFA
- ε-NFA
- ε-NFA = NFA



Elimination of ε-Transitions

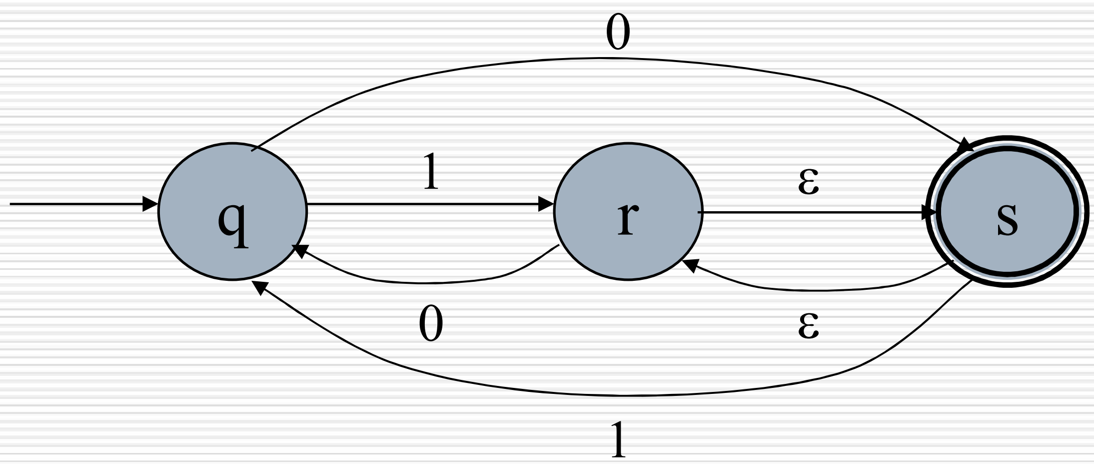
- ε-transitions are a convenience, but do not increase the power of FA's. To eliminate ε-transitions:
  1. Compute the transitive closure of the ε-arcs only.
  2. If a state p can reach state q by ε-arcs, and there is a transition from q to r on input a (not ε), then add a transition from p to r on input a.
  3. Make state p an accept state if p can reach some accept state q by ε-arcs.
  4. Remove all ε-transitions.

ε-CLOSURE

- ε-CLOSURE(q): all states reachable from q by a sequence $\varepsilon\varepsilon\ldots\varepsilon$
  - $q \in \varepsilon$-CLOSURE(q);
  - $p \in \varepsilon$-CLOSURE(q) and $r \in \delta(p, \varepsilon) \rightarrow r \in \varepsilon$-CLOSURE(q)

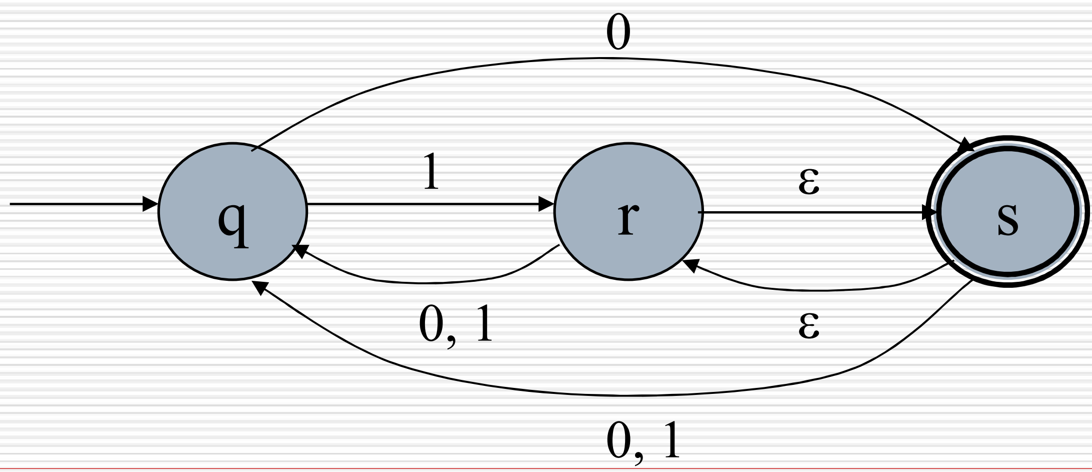
Example

1. ε-CLOSURE(q)={q}, ε-CLOSURE(r)={r,s}, ε-CLOSURE(s)={r,s}



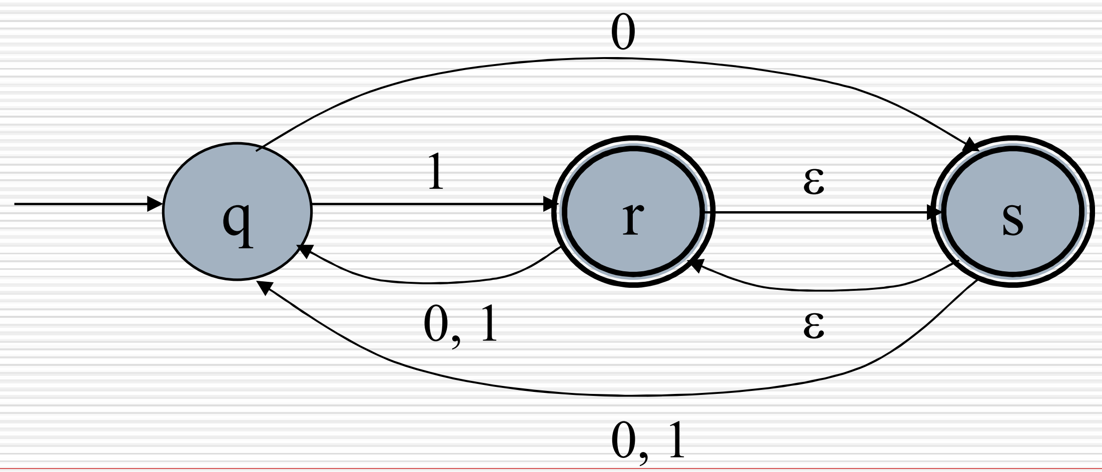
Example

1. ε-CLOSURE(q)={q}, ε-CLOSURE(r)={r,s}, ε-CLOSURE(s)={r,s}
2. Add δ(s, 0)={q}, δ(r, 1)={q}



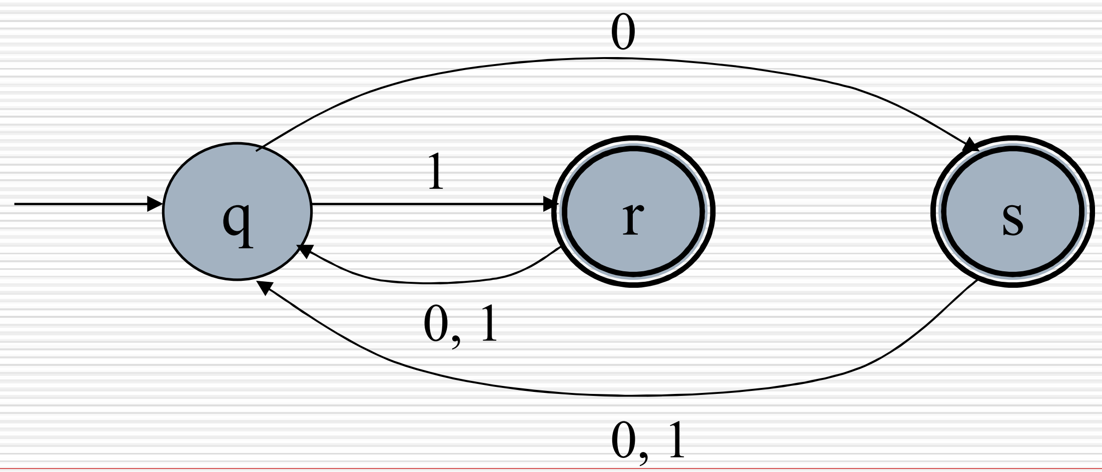
Example

1. ε-CLOSURE(q)={q}, ε-CLOSURE(r)={r,s}, ε-CLOSURE(s)={r,s}
2. Add δ(s, 0)={q}, δ(r, 1)={q}
3. Add r into F as an accept state

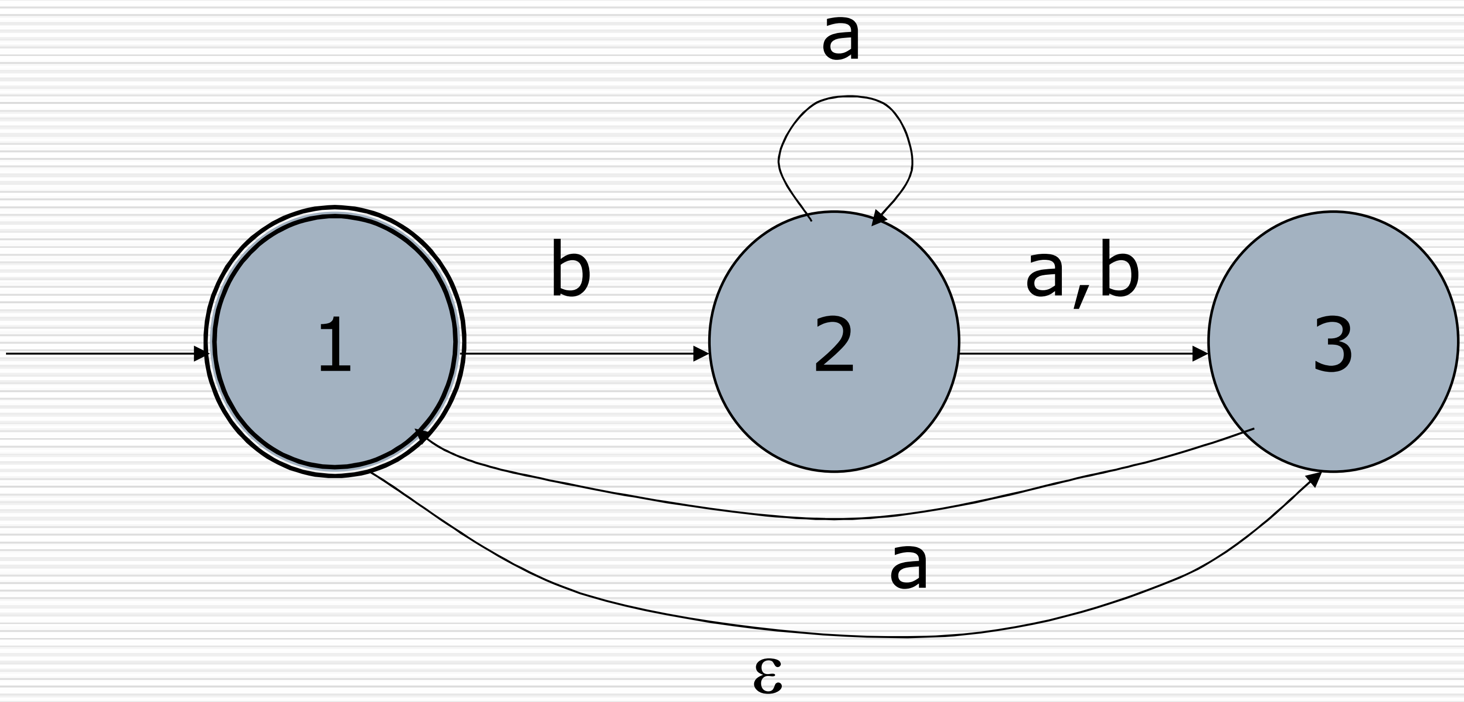


Example

1. ε-CLOSURE(q)={q}, ε-CLOSURE(r)={r,s}, ε-CLOSURE(s)={r,s}
2. Add δ(s, 0)={q}, δ(r, 1)={q}
3. Add r into F as an accept state
4. Remove δ(s, ε)={r}, δ(r, ε)={s}



Example 1.21



Summary of Finite Automata

- DFA, NFA, and ε-NFA are all equivalent.

ε-NFA → NFA → DFA

ε-elimination subset construction