FINITE AUTOMATA (FA)

ITEU133

AUTOMATA AND THEORY

OF COMPUTATION

COURSE SYLLABUS

- Finite Automata (FA)
  - Deterministic Finite Accepters (DFA)
  - Nondeterministic Finite Accepters (NFA)
  - Equivalence of Deterministic and Nondeterministic Finite Accepters
  - NFA with Epsilon Transition (NFA with ε)

FINITE AUTOMATON (FA)

- Informally, a state diagram that comprehensively captures all possible states and transitions that a machine can take while responding to a stream or sequence of input symbols
- Recognizer for “Regular Languages”

TYPES OF AUTOMATON

- Deterministic Finite Automata (DFA)
  – The machine can exist in only one state at any given time
- Non-deterministic Finite Automata (NFA)
  – The machine can exist in multiple states at the same time

THREE WAYS OF PRESENTING
FINITE AUTOMATA

- State Diagram / transition graph
- State Table
- Transition Function

TRANSITION GRAPHS

- A transition graph is a directed graph in that the vertices represent the internal states of the automaton, and the edges represent the transitions.
- The labels on the vertices are the names of the internal states, while the labels on the edges are the current values of the input symbols.

Deterministic Finite Automata (DFA) - Definition

- A DFA consists of:
  - Q → a finite set of states
  - ∑ → a finite set of input symbols (alphabet)
  - q_0 → a start state
  - F → set of final states
  - δ → a transition function that is a mapping between Q x ∑ ==> Q

- DFA is defined by the 5-tuple: {Q, ∑, q_0, F, δ}

DETERMINISTIC FINITE AUTOMATA (DFA)

- Design a DFA, M which accepts the language

  L(M) = {w ε (a, b)* :w does not contain three consecutive b's)}

  Let M = (Q, Σ, δ, q0 , F)

  where

- Q = {q0, q1, q2, q3} Σ = {a, b}
- q0 is the initial state F = {q0, q1, q2}
- δ is defined as follows:

What does a DFA do on reading an input string?

Input: a word w in Σ^*

Question: Is w acceptable by the DFA?

Steps:

- Start at the "start state" q_0
- For every input symbol in the sequence w do
  - Compute the next state from the current state, given the current input symbol in w and the transition function
- If after all symbols in w are consumed, the current state is one of the final states (F) then *accept* w;

DETERMINISTIC FINITE AUTOMATA (DFA)

Determine the DFA schematic for

$$M = (Q, \Sigma, \delta, q0\ , F\ )$$

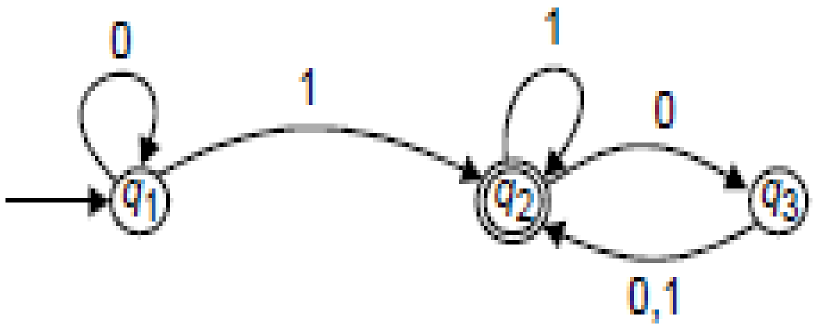
where $Q = \{q1,\ q2,\ q3\}$, $\Sigma = \{0,1\}$,

$q1$ is the start state, $F = \{q2\}$

Initial state q	Symbol σ	Final state $\delta(q,\sigma)$
q_1	0	q_1
q_1	1	q_2
q_2	0	q_3
q_2	1	q_2
q_3	0	q_2
q_3	1	q_2

Deterministic Finite Automata (DFA)

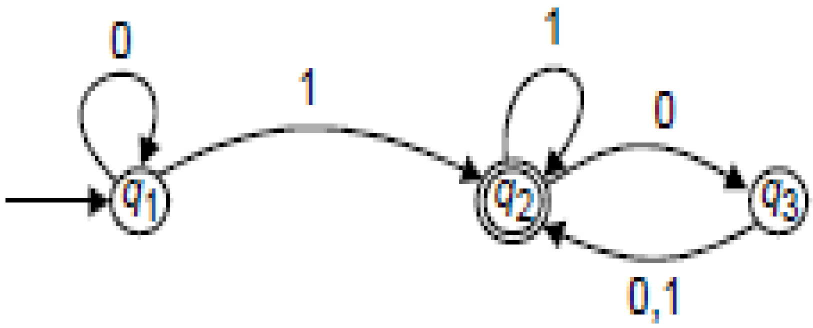
Initial state q	Symbol σ	Final state $\delta(q,\sigma)$
q_1	0	q_1
q_1	1	q_2
q_2	0	q_3
q_2	1	q_2
q_3	0	q_2
q_3	1	q_2



Deterministic Finite Automata (DFA)

Initial state q	Symbol σ	Final state $\delta(q,\sigma)$
q_1	0	q_1
q_1	1	q_2
q_2	0	q_3
q_2	1	q_2
q_3	0	q_2
q_3	1	q_2

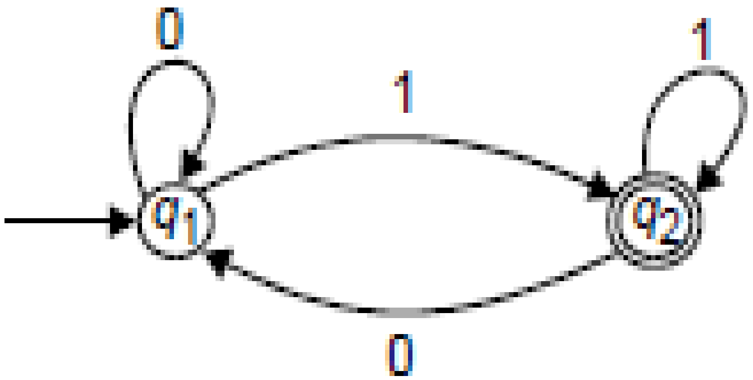
$L = \{w \mid w$ contains at least one 1 and an even number of 0s follow the last 1$\}$



DETERMINISTIC FINITE AUTOMATA (DFA)

- Sketch the DFA given
- $M = (\{q1, q2\}, \{0, 1\}, \delta, q1, \{q2\})$ and δ is given by
    - $\delta(q1, 0) = q1$
    - $\delta(q1, 1) = q2$
    - $\delta(q2, 0) = q1$
    - $\delta(q2, 1) = q1$

DETERMINISTIC FINITE AUTOMATA (DFA)



STATE DIAGRAM

L = 

DETERMINISTIC FINITE AUTOMATA (DFA)

- Design a DFA, the language recognized by the Automaton being $L = \{a^n b : n \geq 0\}$

EXAMPLE #1

Build a DFA for the following language:

L = {w | w is a binary string that contains 01 as a substring}

Steps for building a DFA to recognize L:

- $\sum = \{0,1\}$
- Decide on the states: Q
- Designate start state and final state(s)
- δ: Decide on the transitions:

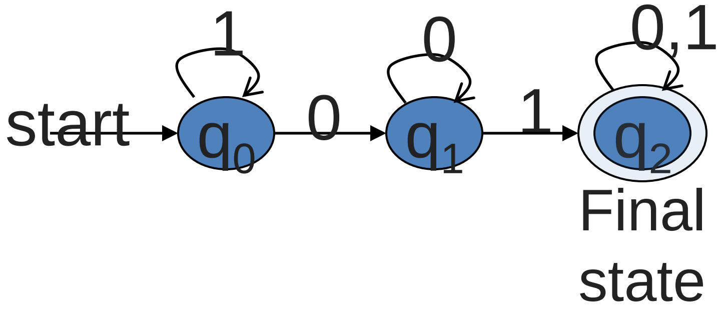
Final states == same as “accepting states”

Other states == same as “non-accepting states”

REGULAR EXPRESSION: (0+1)*01(0+1)*

DFA for strings containing 01

- What makes this DFA deterministic?



- What if the language allows empty strings?

- $Q = \{q_0, q_1, q_2\}$
- $\Sigma = \{0,1\}$
- start state = q_0
- $F = \{q_2\}$
- Transition table

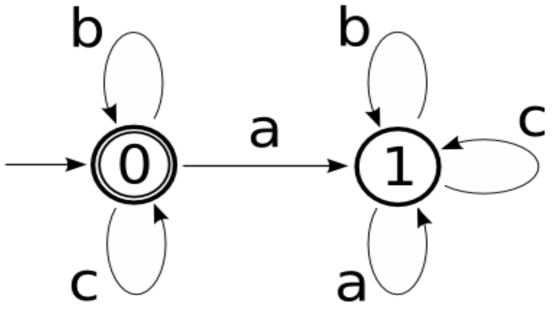
symbols

δ	0	1
q_0	q_1	q_0
q_1	q_1	q_2
$*q_2$	q_2	q_2

states

DEAD STATE

Are those non final state which transits in itself for all input symbol



Extension of transitions (δ) to Paths($\hat{\delta}$)

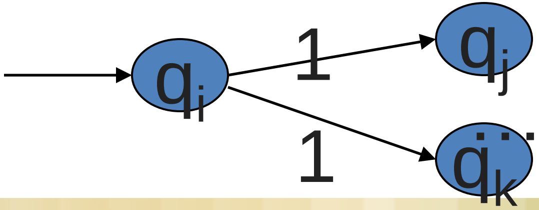
- $\hat{\delta}$ (q,w) = *destination state* from state *q* on input string *w*

- $\hat{\delta}(q,wa) = \delta(\hat{\delta}(q,w), a)$

  - Work out example #3 using the input sequence w=10010, a=1:

NON-DETERMINISTIC FINITE AUTOMATA (NFA)

- A Non-deterministic Finite Automaton (NFA)
  - is of course "non-deterministic"
    - Implying that the machine can exist in more than one state at the same time
    - Transitions could be non-deterministic



- Each transition function therefore maps to a <u>set</u> of states

NON-DETERMINISTIC FINITE AUTOMATA (NFA)

- A Non-deterministic Finite Automaton (NFA) consists of:
  - Q ==> a finite set of states
  - ∑ ==> a finite set of input symbols (alphabet)
  - q_0 ==> a start state
  - F ==> set of final states
  - δ ==> a transition function, which is a mapping between Q x ∑ ==> subset of Q
- An NFA is also defined by the 5-tuple:
  - {Q, ∑ , q_0,F, δ }

HOW TO USE AN NFA?

Input: a word w in Σ^*

Question: Is w acceptable by the NFA?

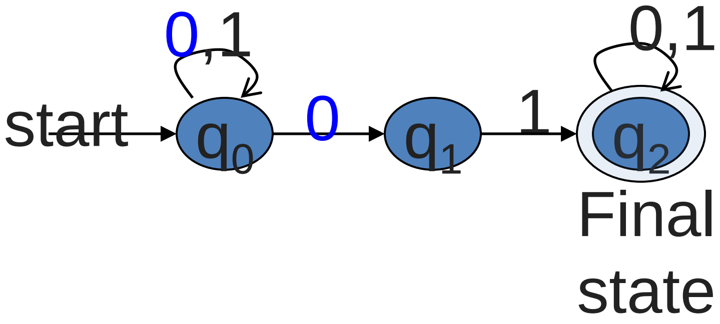
Steps:

- Start at the "start state" q_0
- For every input symbol in the sequence w do
  - Determine all possible next states from all current states, given the current input symbol in w and the transition function
- If after all symbols in w are consumed and if at least one of the current states is a final state then *accept w;*
- Otherwise, *reject w.*

Regular expression: (0+1)*01(0+1)*

NFA for strings containing 01

Why is this non-deterministic?



What will happen if at state q_1 an input of 0 is received?

- $Q = \{q_0, q_1, q_2\}$
- $\Sigma = \{0,1\}$
- start state = q_0
- $F = \{q_2\}$
- Transition table

symbols

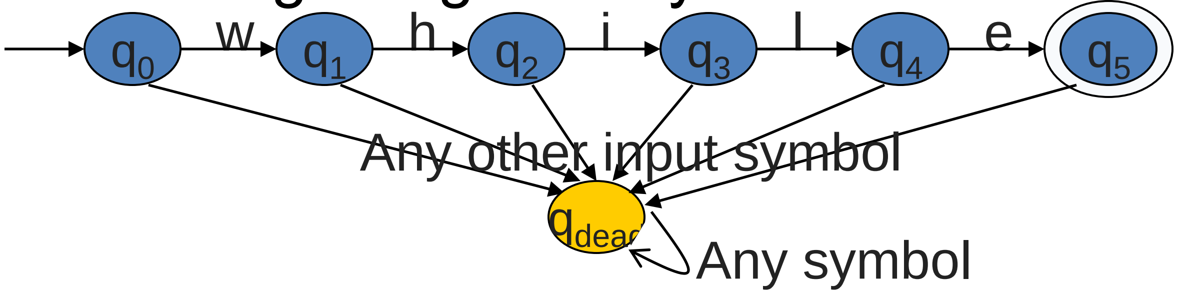
δ	0	1
→ q_0	$\{q_0,q_1\}$	$\{q_0\}$
q_1	Φ	$\{q_2\}$
$*q_2$	$\{q_2\}$	$\{q_2\}$

states

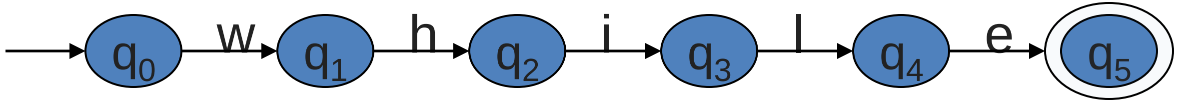
What is a “dead state”?

Note: Explicitly specifying dead states is just a matter of design convenience
(one that is generally followed in NFAs), and
this feature does not make a machine deterministic or non-deterministic.

- A DFA for recognizing the key word “*while*”



- An NFA for the same purpose:



Transitions into a dead state are implicit

NON-DETERMINISTIC FINITE AUTOMATA

1. $L=\{x \in \{a, b,c\}^* \mid x$ contains the pattern **abac** $\}$

2. Determine an NFA accepting all strings over {0,1} which end in 1 but does not contain the substring 00.

3. Design an NFA with no more than five states for the set $\{abab^n : n \geq 0\} \cup \{aba^n : n \geq 0\}$.

Language of an NFA

- An NFA accepts *w* if *there exists at least one* path from the start state to an accepting (or final) state that is labeled by *w*
- $L(N) = \{ w \mid \hat{\delta}(q_0, w) \cap F \neq \Phi \}$

ADVANTAGES & CAVEATS FOR NFA

- Great for modeling regular expressions
  - String processing - e.g., grep, lexical analyzer
- Could a non-deterministic state machine be implemented in practice?
  - A parallel computer could exist in multiple “states” at the same time
  - Probabilistic models could be viewed as extensions of non-deterministic state machines
    (e.g., toss of a coin, a roll of dice)

DIFFERENCES: DFA VS. NFA

But, DFAs and NFAs are equivalent in their power to capture languages !!

- **DFA**

1. All transitions are deterministic
   – Each transition leads to exactly one state
2. For each state, transition on all possible symbols (alphabet) should be defined
3. Accepts input if the last state is in F
4. Sometimes harder to construct because of the number of states
5. Practical implementation is feasible

- **NFA**

1. Some transitions could be non-deterministic
   – A transition could lead to a subset of states
2. Not all symbol transitions need to be defined explicitly (if undefined will go to a dead state – this is just a design convenience, not to be confused with "non-determinism")
3. Accepts input if *one of* the last states is in F
4. Generally easier than a DFA to construct
5. Practical implementation has to be deterministic (convert to DFA) or in the form of parallelism

EQUIVALENCE OF DFA & NFA

- **<u>Theorem</u>:**

Should be true for any L

A language L is accepted by a DFA *if and only if* it is accepted by an NFA.

- **<u>Proof</u>:**

1. If part:
   - Prove by showing every NFA can be converted to an equivalent DFA (in the next few slides…)
2. Only-if part is trivial:
   - Every DFA is a special case of an NFA where each state has exactly one transition for every input symbol. Therefore, if L is accepted by a DFA, it is accepted by a corresponding NFA.

NFA to DFA by SUBSET CONSTRUCTION

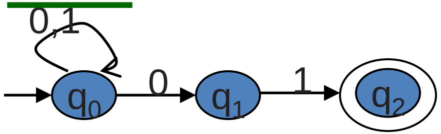
- Let $N = \{Q_N, \Sigma, \delta_N, q_0, F_N\}$
- Goal: Build $D = \{Q_D, \Sigma, \delta_D, \{q_0\}, F_D\}$ s.t. $L(D)=L(N)$
- Construction:
  1. Q_D = all subsets of Q_N (i.e., power set)
  2. F_D = set of subsets S of Q_N s.t. $S \cap F_N \neq \Phi$
  3. δ_D: for each subset S of Q_N and for each input symbol a in Σ:
     - $\delta_D(S,a) = \bigcup_{p \text{ in } s} \delta_N(p,a)$

NFA to DFA construction: Example

- *L = {w | w ends in 01}*

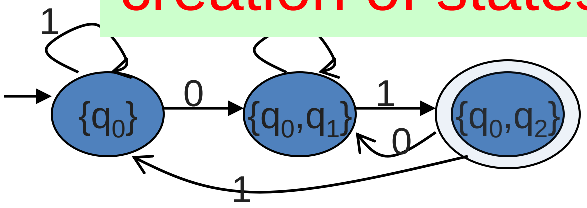
Idea: To avoid enumerating all of power set, do "lazy creation of states"

NFA:



δ_N	0	1
→q_0	$\{q_0,q_1\}$	$\{q_0\}$
q_1	Ø	$\{q_2\}$
*q_2	Ø	Ø

DFA:



δ_D	0	1
~~Ø~~	~~Ø~~	~~Ø~~
→$\{q_0\}$	$\{q_0,q_1\}$	$\{q_0\}$
~~$\{q_1\}$~~	~~Ø~~	~~$\{q_2\}$~~
~~*$\{q_2\}$~~	~~Ø~~	~~Ø~~
$\{q_0,q_1\}$	$\{q_0,q_1\}$	$\{q_0,q_2\}$
*$\{q_0,q_2\}$	$\{q_0,q_1\}$	$\{q_0\}$
~~*$\{q_1,q_2\}$~~	~~Ø~~	~~$\{q_2\}$~~
~~*$\{q_0,q_1,q_2\}$~~	~~$\{q_0,q_1\}$~~	~~$\{q_0,q_2\}$~~

⇒

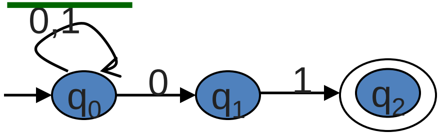
δ_D	0	1
→$\{q_0\}$	$\{q_0,q_1\}$	$\{q_0\}$
$\{q_0,q_1\}$	$\{q_0,q_1\}$	$\{q_0,q_2\}$
*$\{q_0,q_2\}$	$\{q_0,q_1\}$	$\{q_0\}$

0. Enumerate all possible subsets
1. Determine transitions
2. Retain only those states reachable from $\{q_0\}$

NFA to DFA: Repeating the example using *LAZY CREATION*

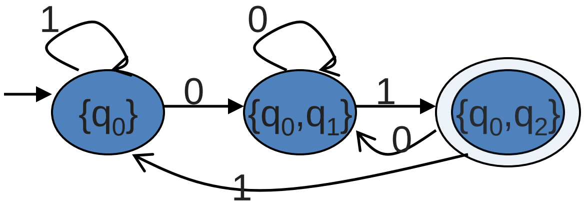
- *L = {w | w ends in 01}*

NFA:



δ_N	0	1
→ q_0	$\{q_0,q_1\}$	$\{q_0\}$
q_1	Ø	$\{q_2\}$
$*q_2$	Ø	Ø

DFA:



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

MAIN IDEA:
Introduce states as you go(on a need basis)

EQUIVALENCE OF NFA AND DFA

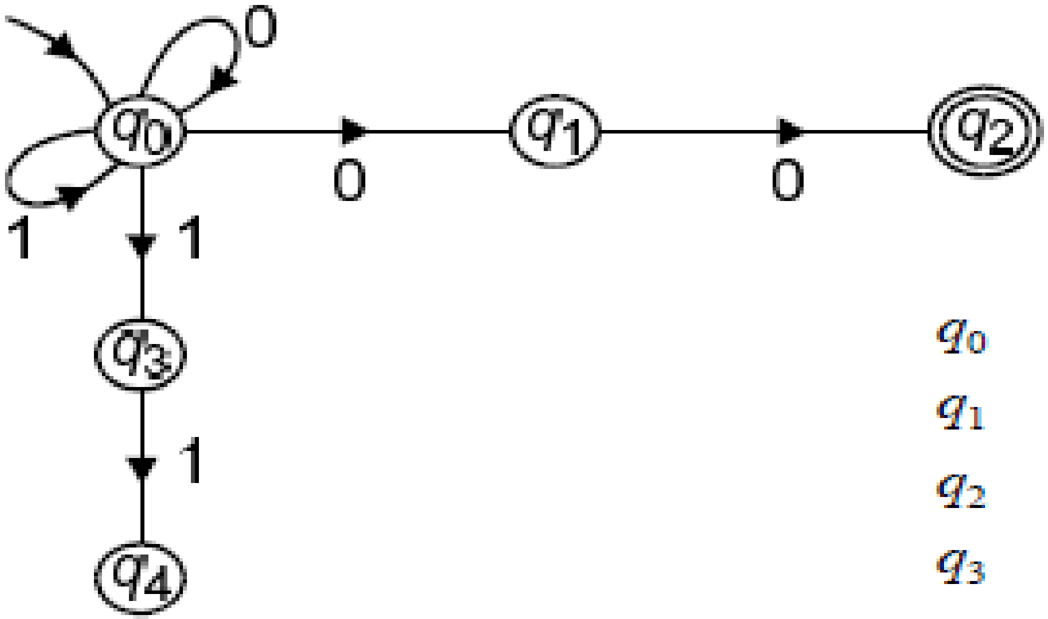
Determine a Deterministic Finite State Automaton from the given Nondeterministic FSA.

$M = (\{q0, q1\}, \{a, b\}, \delta, qo, \{q1\})$

with the state table diagram for δ given below

δ	a	b
q0	{q0, q1}	{q1}
q1	Φ	{q0, q1}

EQUIVALENCE OF NFA AND DFA



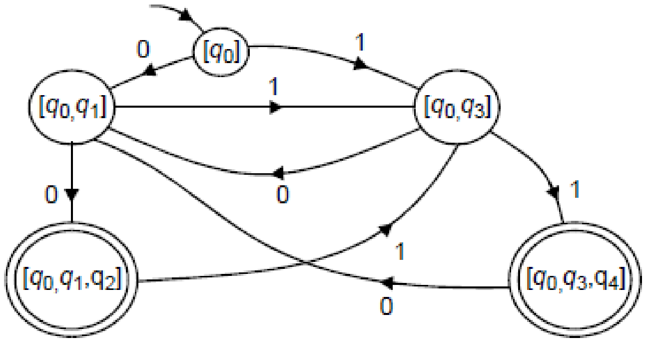
	0	1
q_0	$\{q_0, q_1\}$	$\{q_0, q_3\}$
q_1	$\{q_2\}$	$\varnothing$
q_2	$\varnothing$	$\varnothing$
q_3	$\varnothing$	$\{q_4\}$
q_4	$\varnothing$	$\varnothing$

EQUIVALENCE OF NFA AND DFA

	0	1
$\to [q_0]$	$[q_0, q_1]$	$[q_0, q_3]$
$[q_0, q_1]$	$[q_0, q_1, q_2]$	$[q_0, q_3]$
$[q_0, q_3]$	$[q_0, q_1]$	$[q_0, q_3, q_4]$
$[q_0, q_1, q_2]$	$[q_0, q_1, q_2]$	$[q_0, q_3]$
$[q_0, q_3, q_4]$	$[q_0, q_1]$	$[q_0, q_3, q_4]$

Any state containing q_2 or q_4 will be a final state.

The DFA is shown below.



APPLICATIONS

- Text indexing

  inverted indexing

  For each unique word in the database, store all locations that contain it using an NFA or a DFA

- Find pattern P in text T

  Example: Google querying

- Extensions of this idea:

  PATRICIA tree, suffix tree

A few subtle properties of DFAs and NFAs

- The machine never really terminates.
  - It is always waiting for the next input symbol or making transitions.
- The machine decides when to consume the next symbol from the input and when to ignore it.
  - (but the machine can never skip a symbol)
- => A transition can happen even *without* really consuming an input symbol (think of consuming ε as a free token)
- A single transition cannot consume more than one symbol.

FA with ε-Transitions

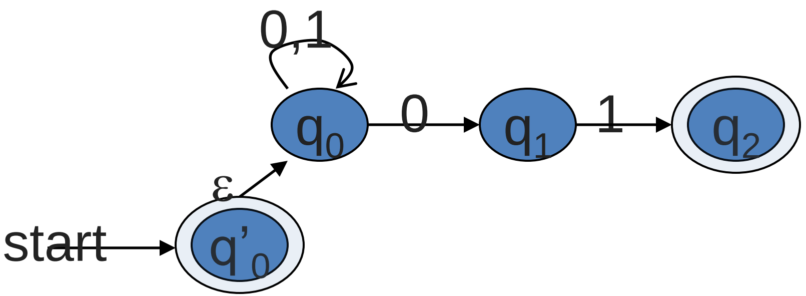
- Allow <u>explicit</u> ε-transitions in finite automata
  - i.e., a transition from one state to another state without consuming any additional input symbol
  - Makes it easier sometimes to construct NFAs

<u>Definition:</u> ε -NFAs are those NFAs with at least one explicit ε-transition defined.

- ε -NFAs have one more column in their transition table

Example of an ε-NFA

L = {w | w is empty, <u>or</u> if non-empty will end in 01}



δ_E	0	1	ε
$*q'_0$	Ø	Ø	$\{q'_0, q_0\}$
q_0	$\{q_0, q_1\}$	$\{q_0\}$	$\{q_0\}$
q_1	Ø	$\{q_2\}$	$\{q_1\}$
$*q_2$	Ø	Ø	$\{q_2\}$

ECLOSE(q'_0)

ECLOSE(q_0)

ECLOSE(q_1)

ECLOSE(q_2)

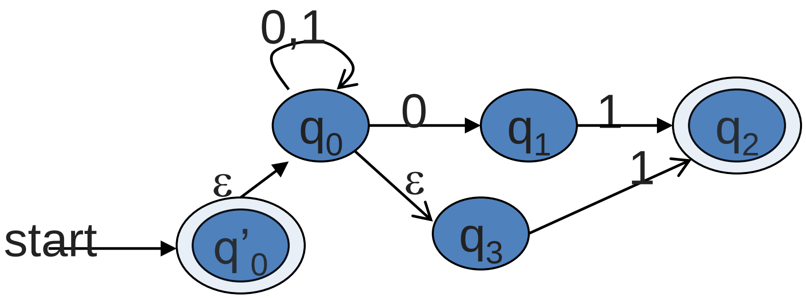
- ε-closure of a state q, ***ECLOSE(q)***, is the set of all states (including itself) that can be *reached* from q by repeatedly making an arbitrary number of ε-transitions.

To simulate any transition:

Step 1) Go to all immediate destination states.

Step 2) From there go to all their ε-closure states as well.

Example of another ε-NFA



δ_E	0	1	ε
→$*q'_0$	Ø	Ø	$\{q'_0,q_0,q_3\}$
q_0	$\{q_0,q_1\}$	$\{q_0\}$	$\{q_0,q_3\}$
q_1	Ø	$\{q_2\}$	$\{q_1\}$
$*q_2$	Ø	Ø	$\{q_2\}$
q_3	Ø	$\{q_2\}$	$\{q_3\}$

Simulate for w=101:

Equivalency of DFA, NFA, ε-NFA

- Theorem: A language L is accepted by some ε-NFA if and only if L is accepted by some DFA

- Implication:
  - DFA ≡ NFA ≡ ε-NFA
  - (all accept Regular Languages)

Eliminating ε-transitions

Let $E = \{Q_E,\Sigma,\delta_E,q_0,F_E\}$ be an ε-NFA

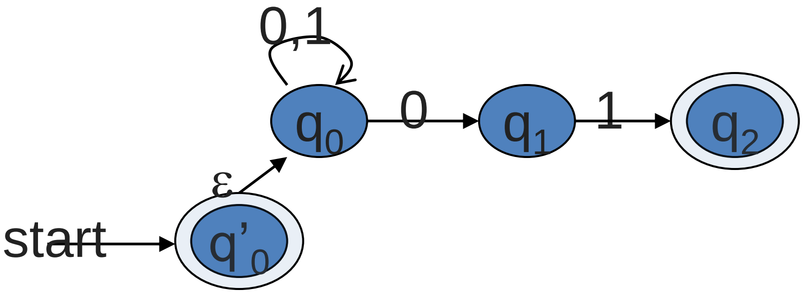
Goal: To build DFA $D=\{Q_D,\Sigma,\delta_D,\{q_D\},F_D\}$ s.t. $L(D)=L(E)$

Construction:

1. Q_D= all reachable subsets of Q_E factoring in ε-closures
2. q_D = ECLOSE(q_0)
3. F_D=subsets S in Q_D s.t. $S\cap F_E\neq\Phi$
4. δ_D: for each subset S of Q_E and for each input symbol $a\in\Sigma$:
   - Let $R= \bigcup_{p \text{ in } s} \delta_E(p,a)$ // go to destination states
   - $\delta_D(S,a) = \bigcup_{r \text{ in } R}$ ECLOSE(r) // from there, take a union of all their ε-closures

Example: ε-NFA ➔ DFA

L = {w | w is empty, or if non-empty will end in 01}

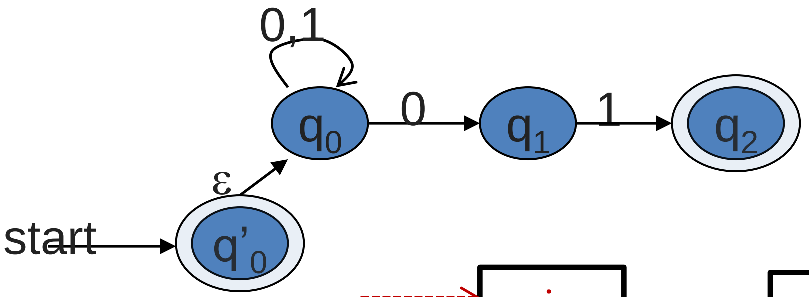


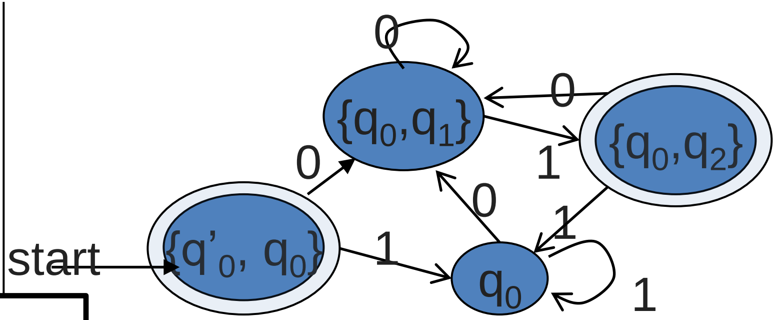
δ_E	0	1	ε
→ $*q'_0$	Ø	Ø	$\{q'_0, q_0\}$
q_0	$\{q_0, q_1\}$	$\{q_0\}$	$\{q_0\}$
q_1	Ø	$\{q_2\}$	$\{q_1\}$
$*q_2$	Ø	Ø	$\{q_2\}$

δ_D	0	1
→ $*\{q'_0, q_0\}$		
…		

Example: ε-NFA ➔ DFA

L = {w | w is empty, or if non-empty will end in 01}





union

δ_E	0	1	ε
→ $*q'_0$	Ø	Ø	$\{q'_0,q_0\}$
q_0	$\{q_0,q_1\}$	$\{q_0\}$	$\{q_0\}$
q_1	Ø	$\{q_2\}$	$\{q_1\}$
$*q_2$	Ø	Ø	$\{q_2\}$

union

δ_D	0	1
→ $*\{q'_0,q_0\}$	$\{q_0,q_1\}$	$\{q_0\}$
$\{q_0,q_1\}$	$\{q_0,q_1\}$	$\{q_0,q_2\}$
$\{q_0\}$	$\{q_0,q_1\}$	$\{q_0\}$
$*\{q_0,q_2\}$	$\{q_0,q_1\}$	$\{q_0\}$