Finite Automata

Reading: Chapter 2

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”

- 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

Deterministic Finite Automata - Definition

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

What does a DFA do on reading an input string?

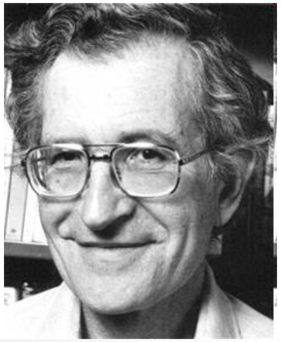
- Input: a word w in Σ^*
- Question: Is w acceptable by the DFA?
- Steps:
  - Start at the "start state" q_0
  - For every input symbol in the sequence w do
    - Compute the next state from the current state, given the current input symbol in w and the transition function
  - If after all symbols in w are consumed, the current state is one of the accepting states (F) then *accept w;*
  - Otherwise, *reject w.*

Regular Languages

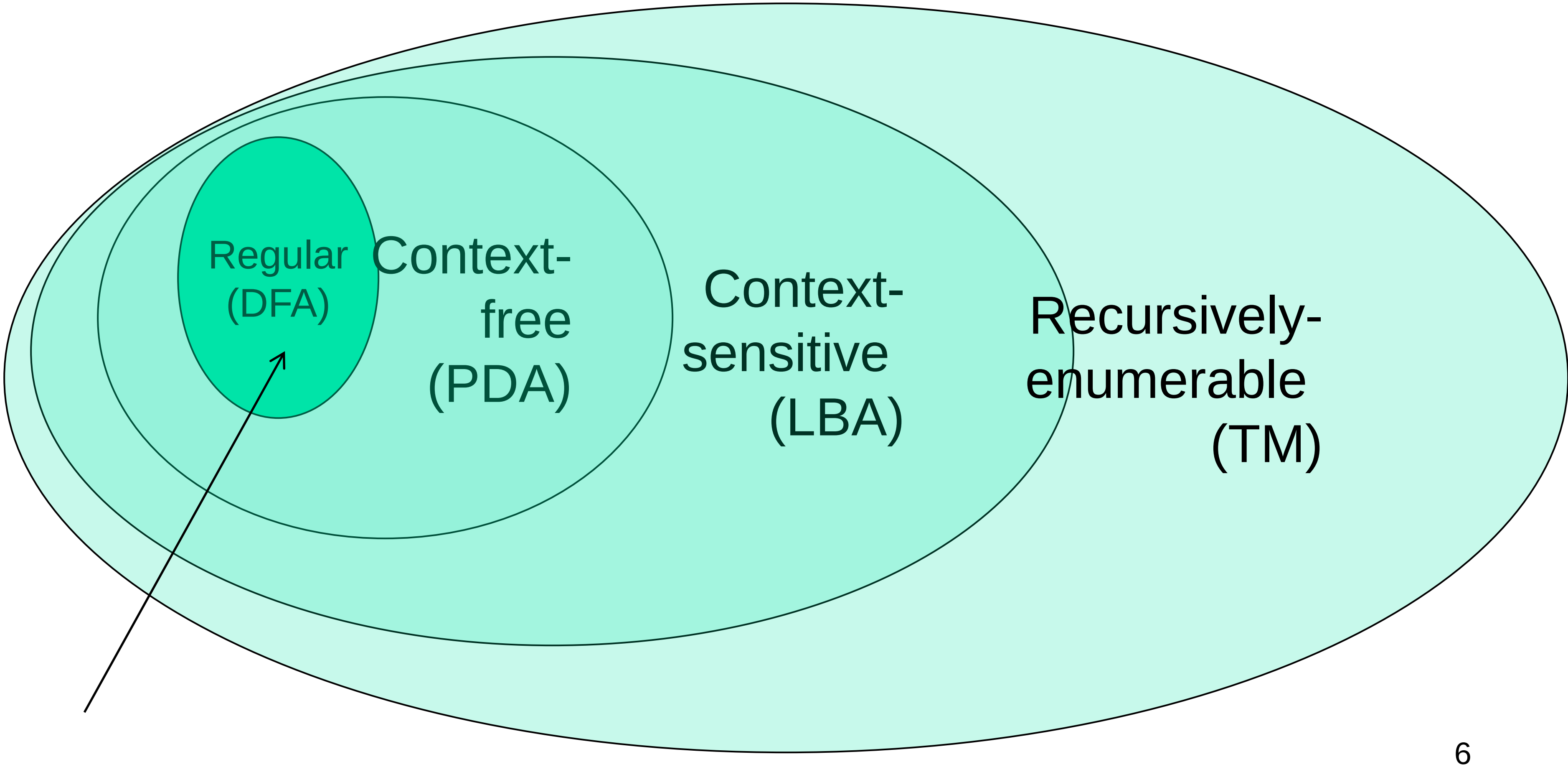
- Let L(A) be a language *recognized* by a DFA A.
  - Then L(A) is called a "*Regular Language*".

- Locate regular languages in the Chomsky Hierarchy

The Chomsky Hierachy



- A containment hierarchy of classes of formal languages



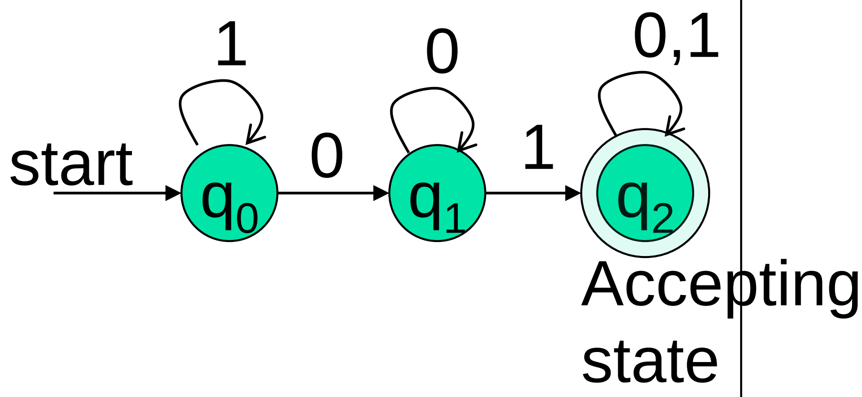
Example #1

- 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:
  - $\Sigma = \{0,1\}$
  - 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?



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

symbols

states

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

- What if the language allows empty strings?

Example #2

Clamping Logic:

- A clamping circuit waits for a ”1” input, and turns on forever. However, to avoid clamping on spurious noise, we’ll design a DFA that waits for *two consecutive 1s* in a row before clamping on.

- Build a DFA for the following language:
  L = { w | w is a bit string which contains the substring 11}
- State Design:
  - q_0 : start state (initially off), also means the most recent input was not a 1
  - q_1: has never seen 11 but the most recent input was a 1
  - q_2: has seen 11 at least once

Example #3

- Build a DFA for the following language:
  L = { w | w is a binary string that has even number of 1s and even number of 0s}
- ?

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:

    - $\hat{\delta}(q_0,wa) = ?$

Language of a DFA

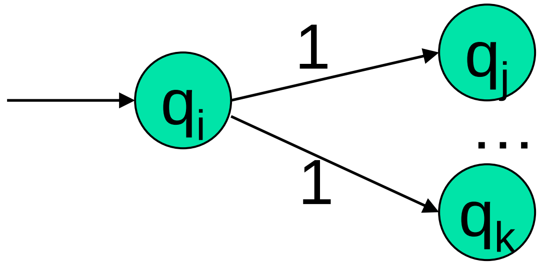
A DFA A accepts string w if there is a path from q_0 to an accepting (or final) state that is labeled by w

- *i.e.,* $L(A) = \{ w \mid \hat{\delta}(q_0,w) \in F \}$

- *I.e., L(A) = all strings that lead to an accepting state from* q_0

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 accepting 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 }

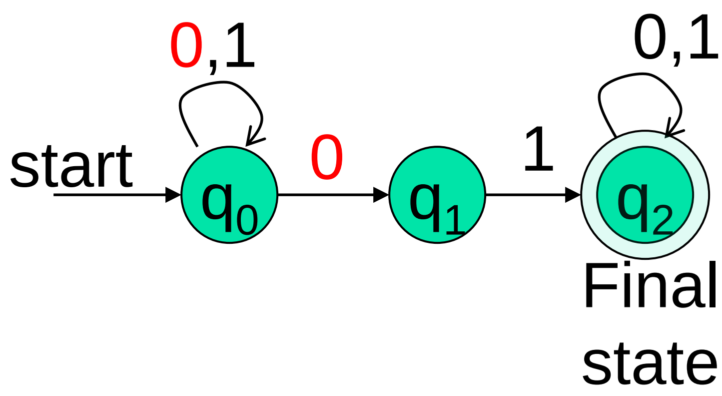
How to use an NFA?

- Input: a word w in $\sum^*$
- 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

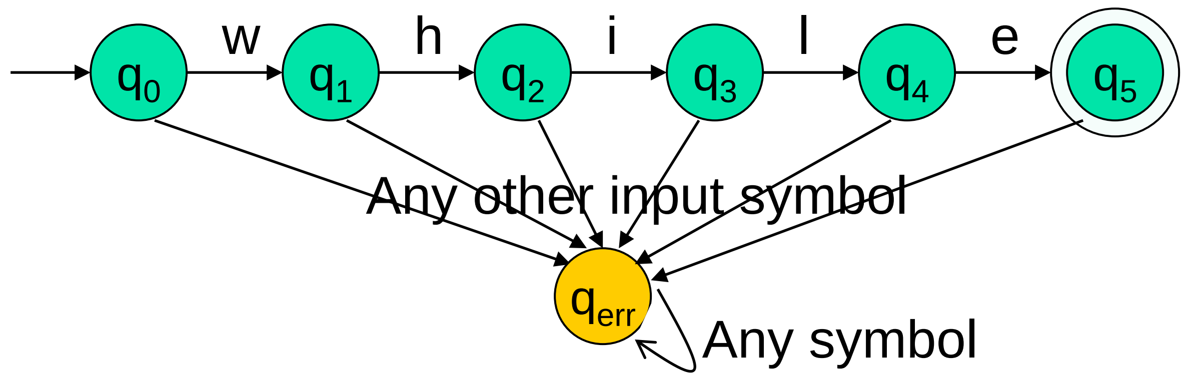
δ	**0**	**1**
→ **q_0**	$\{q_0, q_1\}$	$\{q_0\}$
q_1	Φ	$\{q_2\}$
***q_2**	$\{q_2\}$	$\{q_2\}$

states

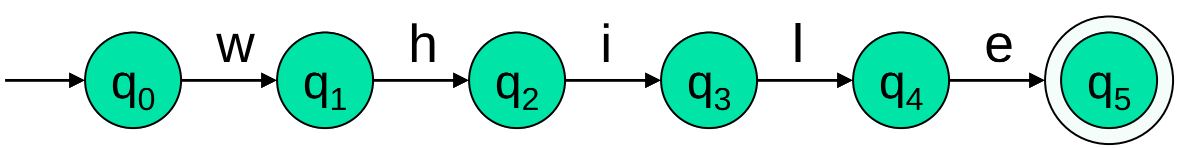
Note: Omitting to explicitly show error states is just a matter of design convenience
(one that is generally followed for NFAs), and
i.e., this feature should not be confused with the notion of non-determinism.

What is an “error state”?

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



- An NFA for the same purpose:



Transitions into a dead state are implicit

Example #2

- Build an NFA for the following language:
  L = { w | w ends in 01}
- ?
- Other examples
  - Keyword recognizer (e.g., if, then, else, while, for, include, etc.)
  - Strings where the first symbol is present somewhere later on at least once

Extension of δ to NFA Paths

- Basis: $\hat{\delta}(q, \varepsilon) = \{q\}$

- Induction:
  - Let $\hat{\delta}(q_0, w) = \{p_1, p_2 \dots, p_k\}$
  - $\delta(p_i, a) = S_i$ for $i = 1, 2 \dots, k$
  - *Then,* $\hat{\delta}(q_0, wa) = S_1 \cup S_2 \cup \dots \cup S_k$

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*
- $L(N) = \{ w \mid \hat{\delta}(q_0,w) \cap F \neq \Phi \}$

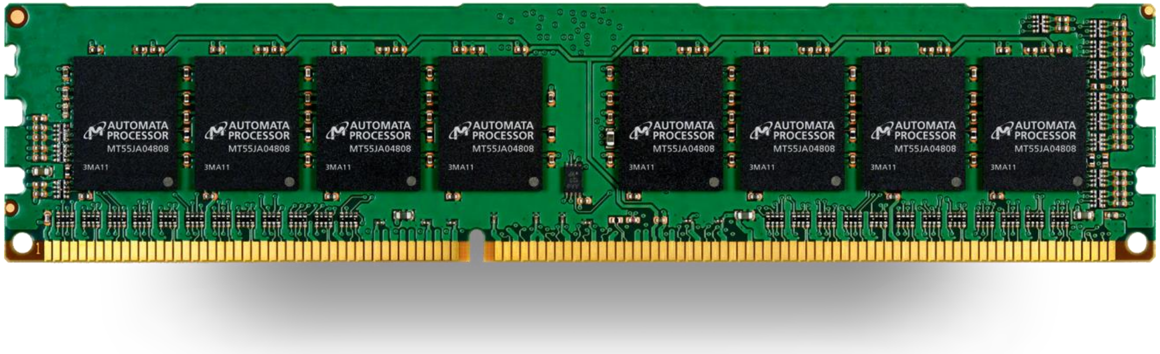
Advantages & Caveats for NFA

- Great for modeling regular expressions
  - String processing - e.g., grep, lexical analyzer

- Could a non-deterministic state machine be implemented in practice?
  - Probabilistic models could be viewed as extensions of non-deterministic state machines
    (e.g., toss of a coin, a roll of dice)
    - They are not the same though
  - A parallel computer could exist in multiple “states” at the same time

Technologies for NFAs

- Micron's Automata Processor (introduced in 2013)
- 2D array of MISD (multiple instruction single data) fabric w/ thousands to millions of processing elements.
- 1 input symbol = fed to all states (i.e., cores)
- Non-determinism using circuits
- http://www.micronautomata.com/



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

Differences: DFA vs. NFA

- **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 visited 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 an error 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 implementations limited but emerging (e.g., Micron automata processor)

Equivalence of DFA & NFA

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

Should be true for any L

- Proof:
  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. □

Proof for the if-part

- If-part: A language L is accepted by a DFA if it is accepted by an NFA
- rephrasing…
- Given any NFA N, we can construct a DFA D such that L(N)=L(D)

- How to convert an NFA into a DFA?
  - Observation: In an NFA, each transition maps to a *subset* of states
  - Idea: Represent:

    each “subset of NFA_states” → a single “DFA_state”

Subset construction

NFA to DFA by subset construction

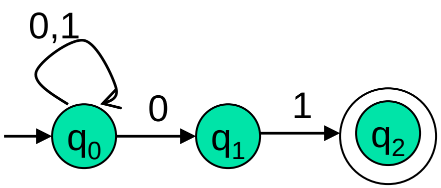
- Let $N = \{Q_N, \Sigma, \delta_N, q_0, F_N\}$
- Goal: Build $D=\{Q_D, \Sigma, \delta_D, \{q_0\}, F_D\}$ s.t. $L(D)=L(N)$
- Construction:
  1. Q_D = all subsets of Q_N (i.e., power set)
  2. F_D=set of subsets S of Q_N s.t. $S \cap F_N \neq \Phi$
  3. δ_D: for each subset S of Q_N and for each input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

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

NFA to DFA construction: Example

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

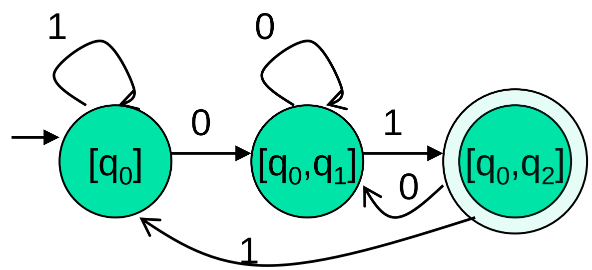
NFA:



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

DFA:

δ_D
~~Ø~~
→$[q_0]$
~~$[q_1]$~~
~~$^*[q_2]$~~
$[q_0,q_1]$
$^*[q_0,q_2]$
~~$^*[q_1,q_2]$~~
~~$^*[q_0,q_1,q_2]$~~



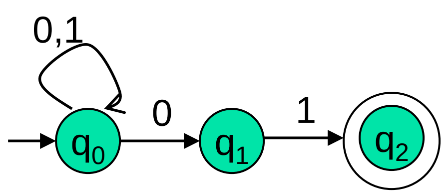
δ_D	0	1
→$[q_0]$	$[q_0,q_1]$	$[q_0]$
$[q_0,q_1]$	$[q_0,q_1]$	$[q_0,q_2]$
$^*[q_0,q_2]$	$[q_0,q_1]$	$[q_0]$

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*

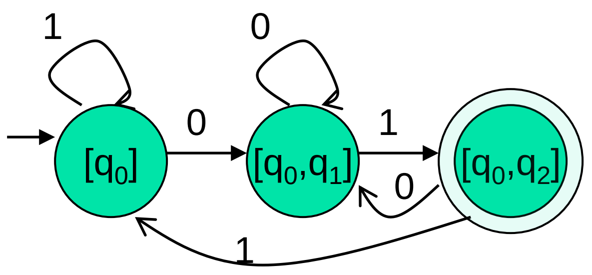
- *L = {w | w ends in 01}*

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)

Correctness of subset construction

<u>Theorem:</u> If D is the DFA constructed from NFA N by subset construction, then L(D)=L(N)

- <u>Proof:</u>
  - Show that $\hat{\delta}_D(\{q_0\},w) \equiv \hat{\delta}_N(q_0,w\}$, for all w
  - Using induction on w's length:
    - Let w = xa
    - $\hat{\delta}_D(\{q_0\},xa) \equiv \delta_D(\hat{\delta}_N(q_0,x\}, a) \equiv \hat{\delta}_N(q_0,w\}$

A bad case where #states(DFA)>>#states(NFA)

- L = {w | w is a binary string s.t., the k^{th} symbol from its end is a 1}
  - NFA has k+1 states
  - But an equivalent DFA needs to have at least 2^k states

<u>(Pigeon hole principle)</u>

- *m* holes and >*m* pigeons
  - => at least one hole has to contain two or more pigeons

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 <u>consume</u> the next symbol from the input and when to <u>ignore</u> it.
  - (but the machine can never <u>skip</u> a symbol)
- => A transition can happen even *without* really consuming an input symbol (think of consuming ε as a free token) – if this happens, then it becomes an ε-NFA (see next few slides).
- A single transition *cannot* consume more than one (non-ε) symbol.

FA with ε-Transitions

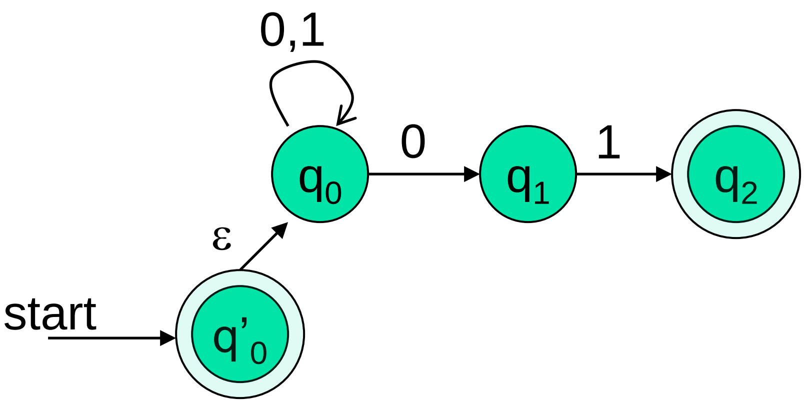
- We can allow <u>explicit</u> ε-transitions in finite automata
  - i.e., a transition from one state to another state without consuming any additional input symbol
  - Explicit ε-transitions between different states introduce non-determinism.
  - Makes it easier sometimes to construct NFAs

<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, or if non-empty will end in 01}



δ_E	0	1	ε
$\rightarrow$ *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.

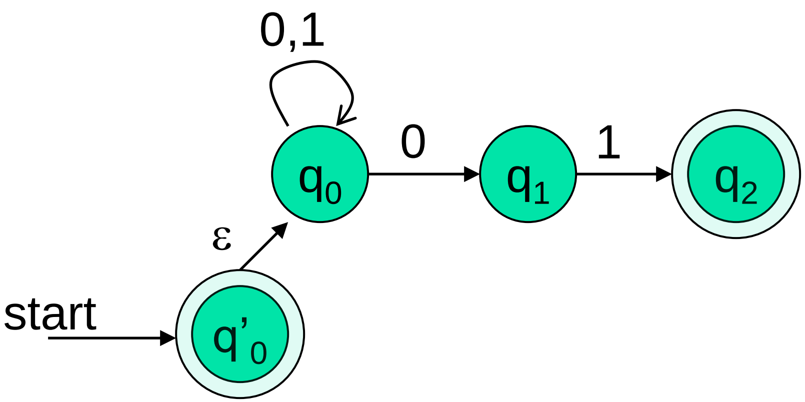
To simulate any transition:

Step 1) Go to all immediate destination states.

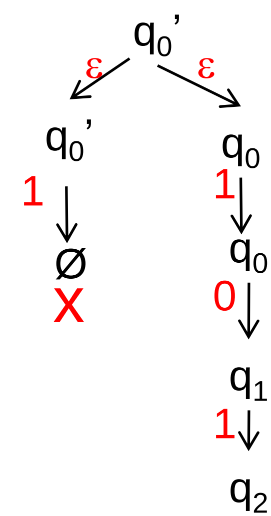
Step 2) From there go to all their ε-closure states as well.

Example of an ε-NFA

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



Simulate for w=101:



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)

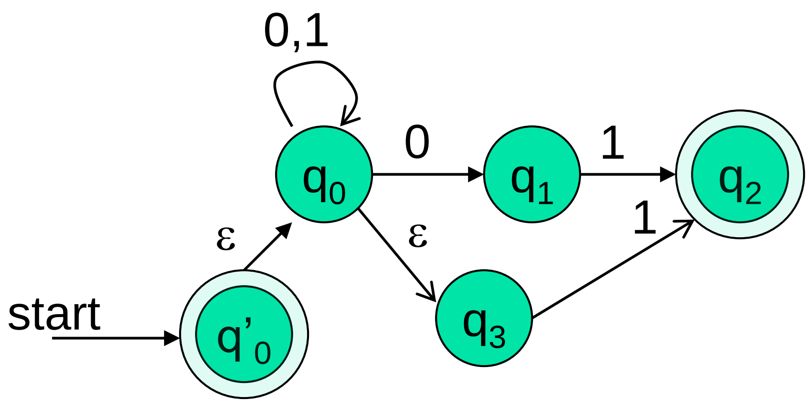
ECLOSE(q_0)

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. $L(D)=L(E)$

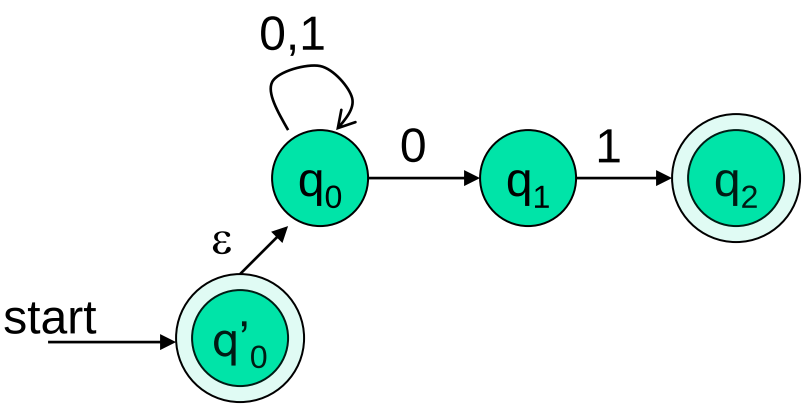
Construction:

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
   - $\delta_D(S,a) = \bigcup_{r \text{ in } R} \text{ECLOSE}(r)$ // from there, take a union of all their ε-closures

Reading: Section 2.5.5 in book

Example: ε-NFA → DFA

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

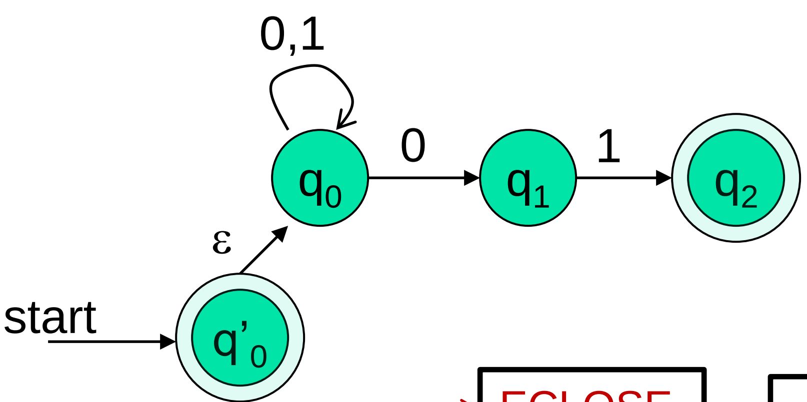


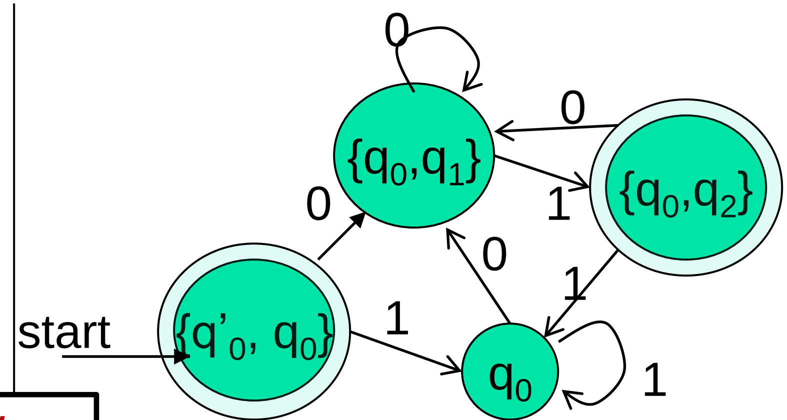
δ_E	0	1	ε
→*q'_0	Ø	Ø	{q'_0,q_0}
q_0	{q_0,q_1}	{q_0}	{q_0}
q_1	Ø	{q_2}	{q_1}
*q_2	Ø	Ø	{q_2}

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}





ECLOSE

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$_2$}

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$_0$}

Summary

- DFA
  - Definition
  - Transition diagrams & tables
- Regular language
- NFA
  - Definition
  - Transition diagrams & tables
- DFA vs. NFA
- NFA to DFA conversion using subset construction
- Equivalency of DFA & NFA
- Removal of redundant states and including dead states
- ε-transitions in NFA
- Pigeon hole principles
- Text searching applications