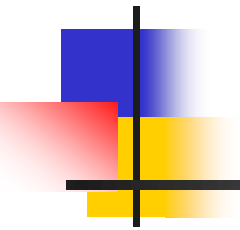
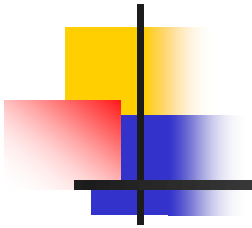


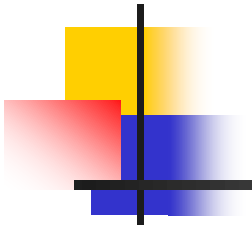
CS 2201: Finite Automata

Alphabets



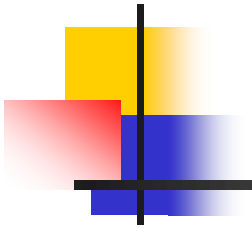
- An *alphabet* is any finite set of symbols
- Examples: ASCII, Unicode, {0,1} (*binary alphabet*), {a,b,c}.

Strings



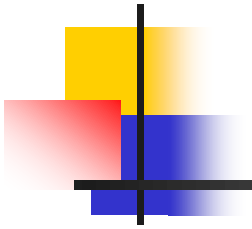
- Set of *strings* over an alphabet Σ is the set of lists, each element of which is a member of Σ
  - Strings shown with no commas, e.g., abc
- Σ* denotes this set of strings.
- ϵ stands for the *empty string* (string of length 0).

Example: Strings



- {0,1}* = {ϵ, 0, 1, 00, 01, 10, 11, 000, 001, . . . }
- Subtlety: 0 as a string, 0 as a symbol look the same.
  - Context determines the type.

Languages



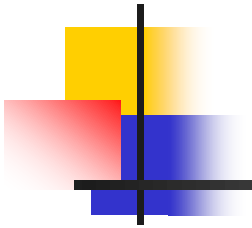
- A *language* is a subset of Σ^* for some alphabet Σ.
- Example: The set of strings of 0's and 1's with no two consecutive 1's.
- L = {ϵ, 0, 1, 00, 01, 10, 000, 001, 010, 100, 101, 0000, 0001, 0010, 0100, 0101, 1000, 1001, 1010, . . . }

Hmm… 1 of length 0, 2 of length 1, 3, of length 2, 5 of length 3, 8 of length 4. I wonder how many of length 5?

Finite Automata: Informal Explanation

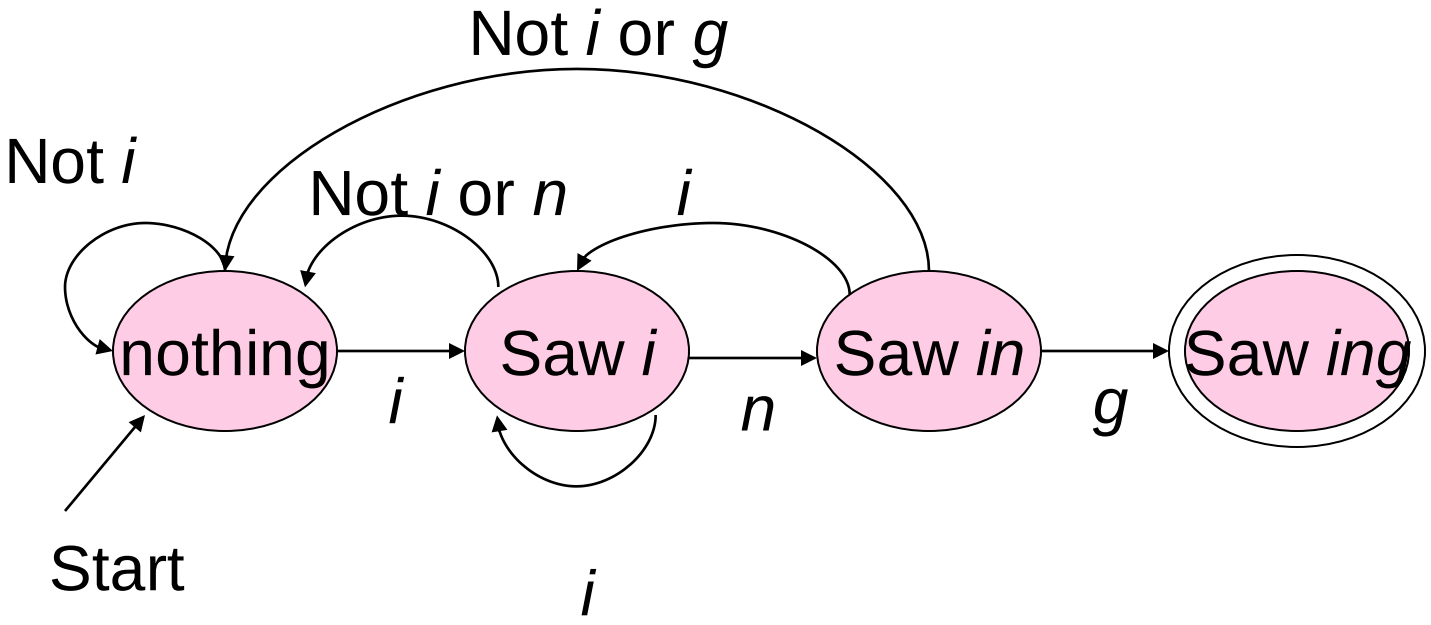
- Finite automata are finite collections of states with transition rules that take you from one state to another
- Original application: Sequential switching circuits, where the "state" was the settings of internal bits
- *Today*, several kinds of software can be modeled by FA

Representing FA

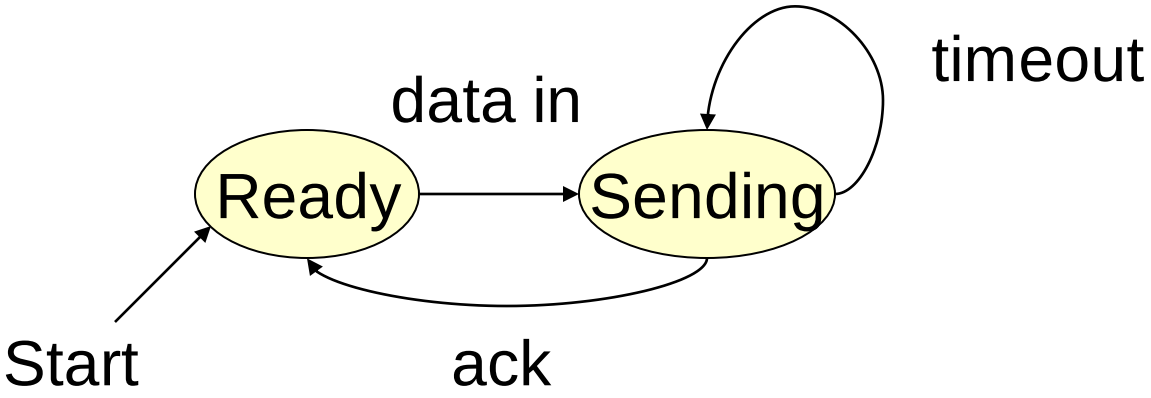


- Simplest representation is often a graph.
  - Nodes = states
  - Arcs indicate state transitions
  - Labels on arcs tell what causes the transition

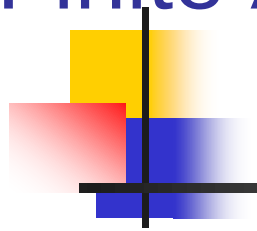
Example: Recognizing Strings Ending in "ing"



Example: Protocol for Sending Data

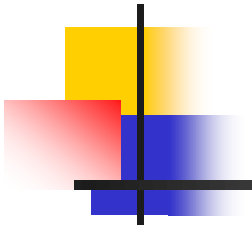


Finite Automaton (FA)



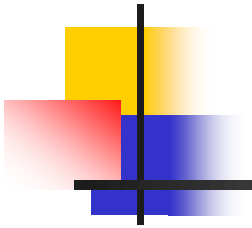
- Finite Automata (FA)
  - Recognizer for “Regular Languages”
- Deterministic Finite Automata (DFA)
  - Machine exists in only one state at any given time
- Non-deterministic Finite Automata (NFA)
  - Machine can exist in multiple states at the same time

Deterministic Finite Automata - Definition



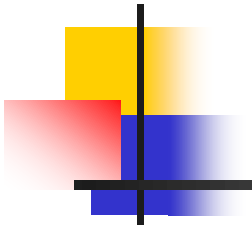
- A Deterministic Finite Automaton (DFA) consists of:
  - Q ==> a finite set of states
  - ∑ ==> a finite set of input symbols (alphabet)
  - q_0 ==> a start state
  - F ==> set of accepting states
  - δ ==> a transition function, which is a mapping between Q x ∑ ==> Q
- A DFA is defined by the 5-tuple: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

What does a DFA do on reading an input string?



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

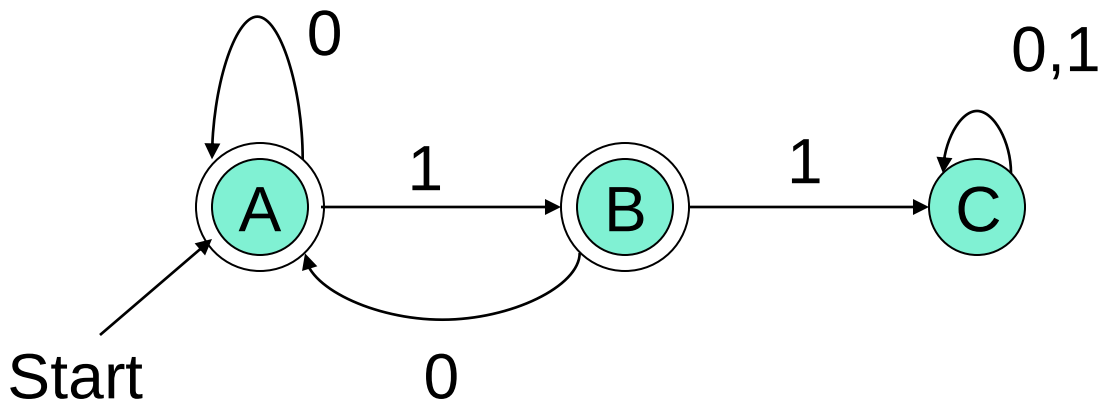
Language of a DFA



- Automata of all kinds define languages
- If A is an automaton, L(A) is its language
- For a DFA A, L(A) is the set of strings labeling paths from the start state to a final state
- Formally: L(A) = the set of strings w such that $\delta(q_0, w)$ is in F

Example: String in a Language

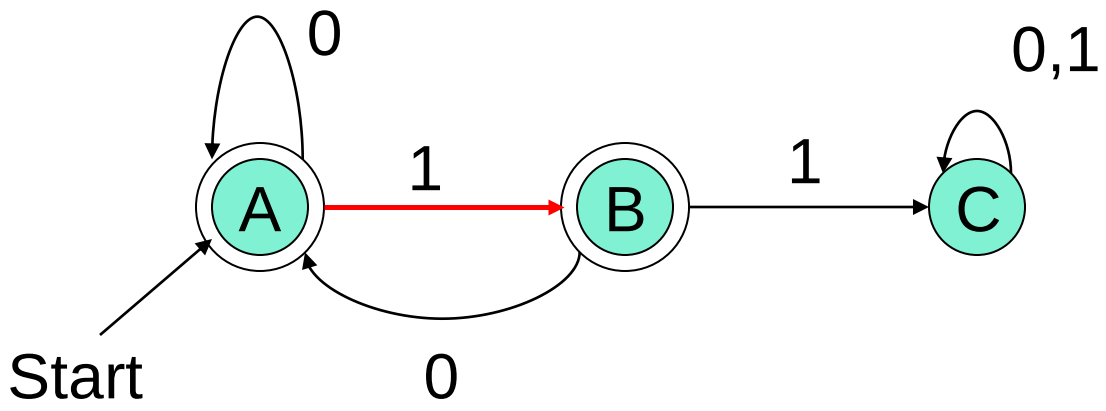
String 101 is in the language of the DFA below
Start at A



Example: String in a Language

String 101 is in the language of the DFA below.

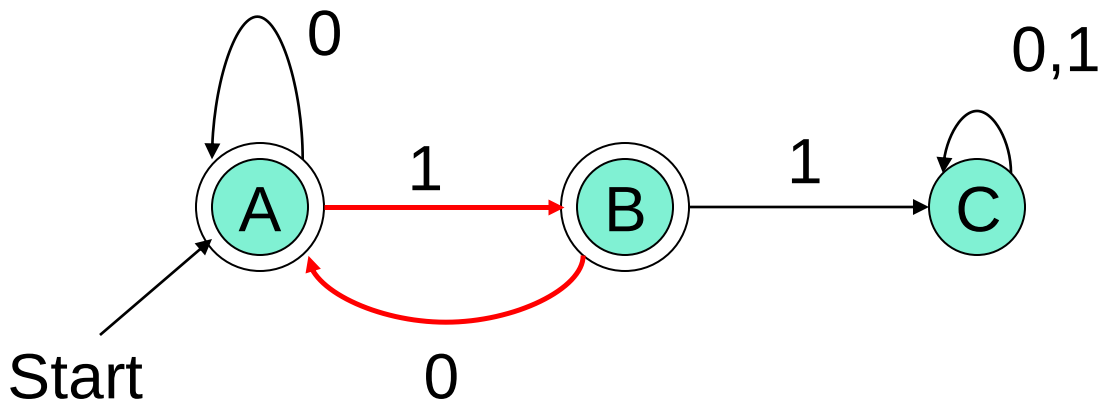
Follow arc labeled 1.



Example: String in a Language

String 101 is in the language of the DFA below.

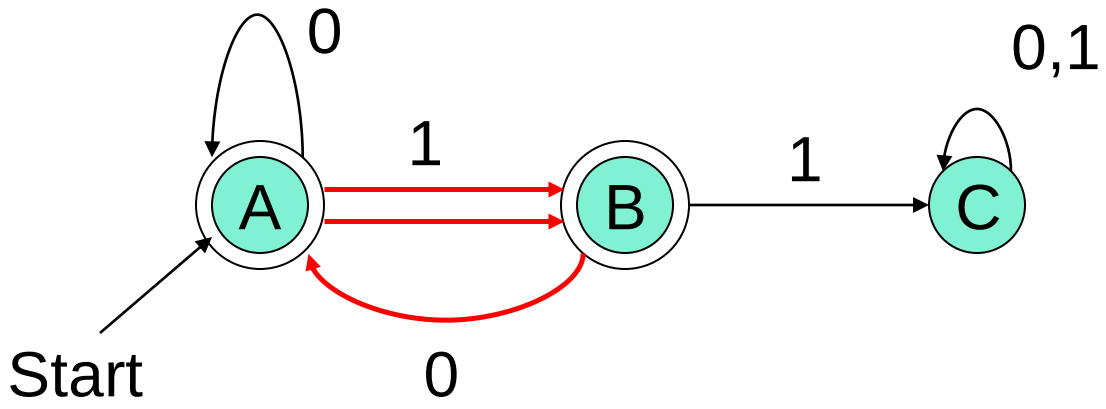
Then arc labeled 0 from current state B.



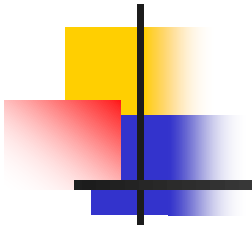
Example: String in a Language

String 101 is in the language of the DFA below

Finally arc labeled 1 from current state A. Result is an accepting state, so 101 is in the language



Example – Concluded



- The language of our example DFA is:

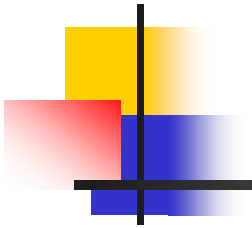
{w | w is in {0,1}* and w does not have two consecutive 1's}

Such that…

These conditions about w are true.

Read a *set former* as "The set of strings w…

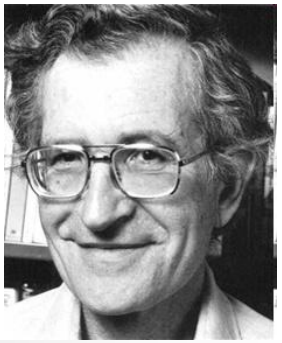
Language of a DFA



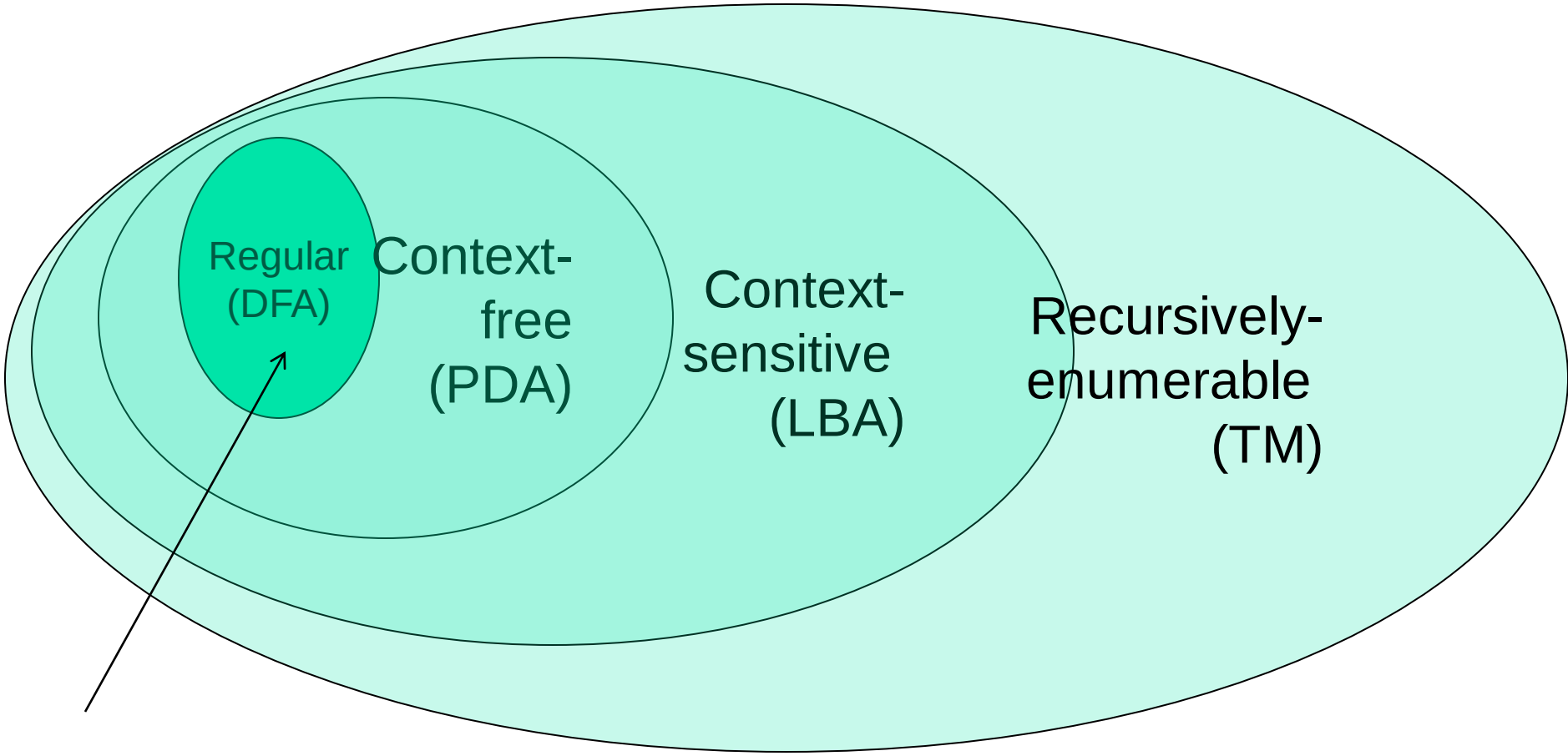
A DFA A accepts string *w* if there is a path from q_0 to an accepting (or final) state that is labeled by *w*

- *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

The Chomsky Hierarchy



- 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:
  - ∑ = {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"

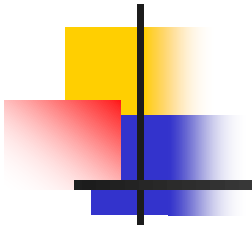
Example #2

- 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



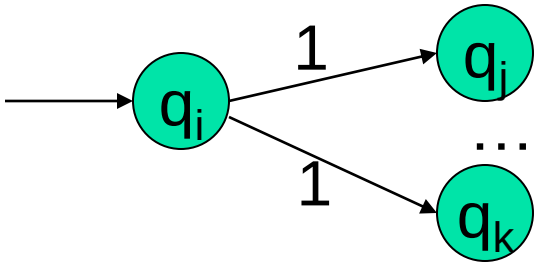
- Build a DFA for the following language:

  L = { w | w is a binary string that has even number of 1s and even number of 0s}

- ?

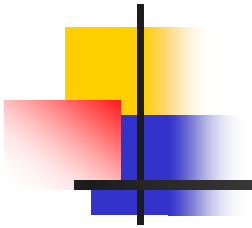
Non-deterministic Finite Automata (NFA)

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



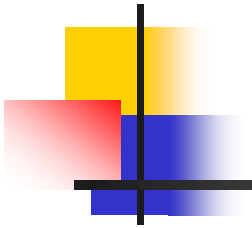
• Each transition function therefore maps to a set of states

Non-deterministic Finite Automata (NFA)



- A Non-deterministic Finite Automaton (NFA) consists of:
  - Q ==> a finite set of states
  - $\sum$ ==> 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 $\sum$ ==> subset of Q
- An NFA is also defined by the 5-tuple:
  - $\{Q, \sum, q_0, F, \delta\}$

How to use an NFA?

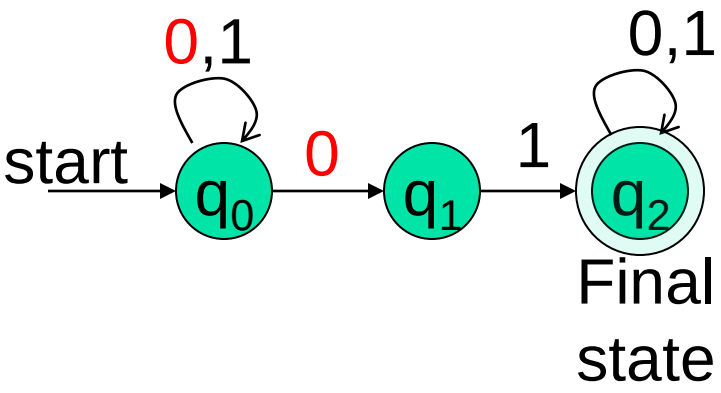


- Input: a word w in $\sum^*$
- Question: Is w acceptable by the NFA?
- Steps:
  - Start at the "start state" q_0
  - For every input symbol in the sequence w do
    - Determine all possible next states from all current states, given the current input symbol in w and the transition function
  - If after 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 = $\{q_0,q_1,q_2\}$
- Σ = {0,1}
- start state = q_0
- F = $\{q_2\}$
- Transition table

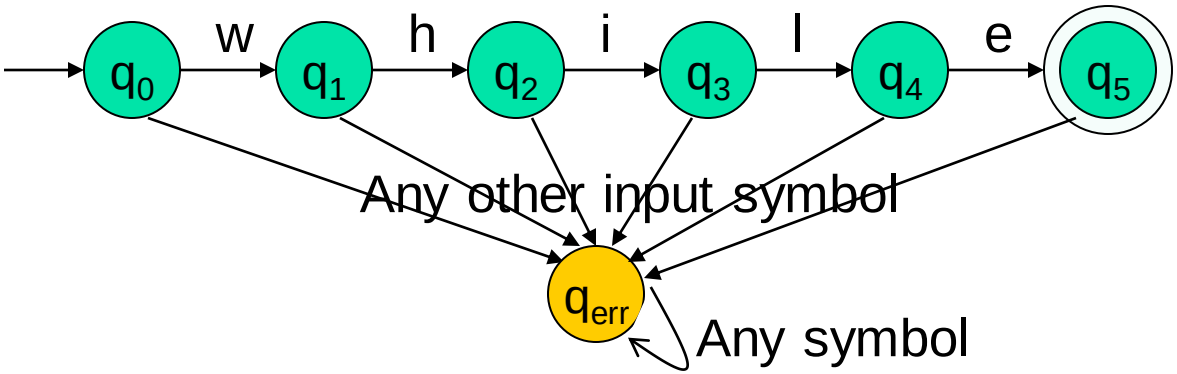
	δ	0	1
states →	q_0	$\{q_0,q_1\}$	$\{q_0\}$
	q_1	Φ?	$\{q_2\}$
	$*q_2$	$\{q_2\}$	$\{q_2\}$

symbols

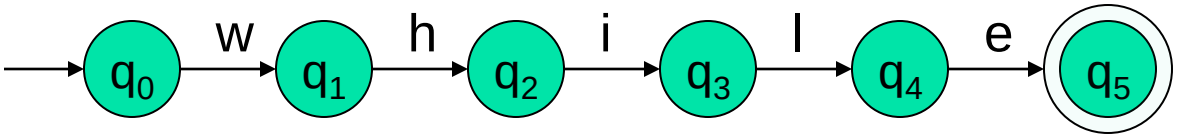
Note: Omitting to explicitly show error states is just a matter of design convenience
(one that is generally followed for NFAs), and
i.e., this feature should not be confused with the notion of non-determinism.

What is an "error state"?

- A DFA for recognizing the key word "*while*"

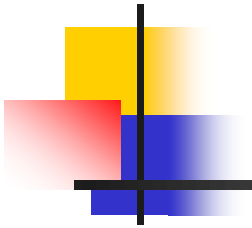


- An NFA for the same purpose:



Transitions into a dead state are implicit

Example #2

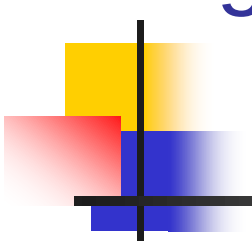


- Build an NFA for the following language:

  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

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

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

Differences: DFA vs. NFA

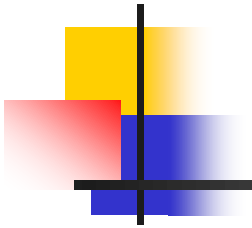
- **DFA**

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)

NFA to DFA by subset construction



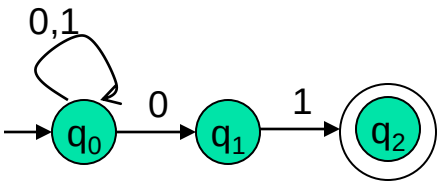
- Let $N = \{Q_N, \Sigma, \delta_N, q_0, F_N\}$
- Goal: Build $D=\{Q_D, \Sigma, \delta_D, \{q_0\}, F_D\}$ s.t. $L(D)=L(N)$
- Construction:
  1. Q_D= all subsets of Q_N (i.e., power set)
  2. F_D=set of subsets S of Q_N s.t. $S \cap F_N \neq \Phi$
  3. δ_D: for each subset S of Q_N and for each input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

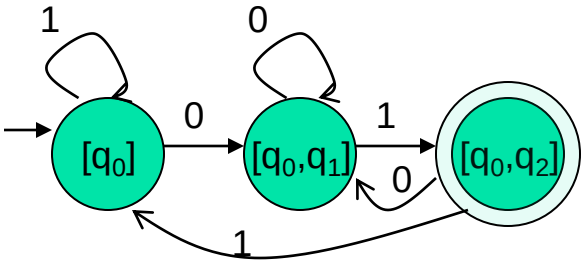
- *L = {w | w ends in 01}*

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

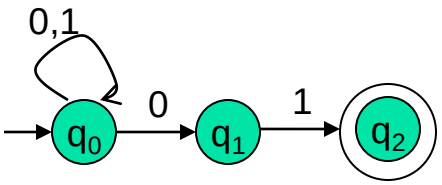
~~δ_D~~
~~Ø~~
→ $[q_0]$
~~$[q_1]$~~
~~*$[q_2]$~~
$[q_0,q_1]$
*$[q_0,q_2]$
~~*$[q_1,q_2]$~~
~~*$[q_0,q_1,q_2]$~~

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*

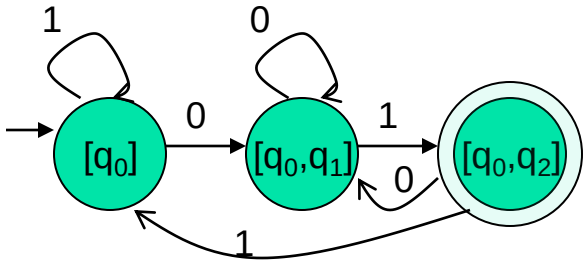
- *L = {w | w ends in 01}*

NFA:



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

DFA:

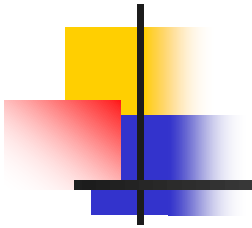


δ_D	0	1
→ $[q_0]$	$[q_0,q_1]$	$[q_0]$

Main Idea:

Introduce states as you go (on a need basis)

FA with ε-Transitions



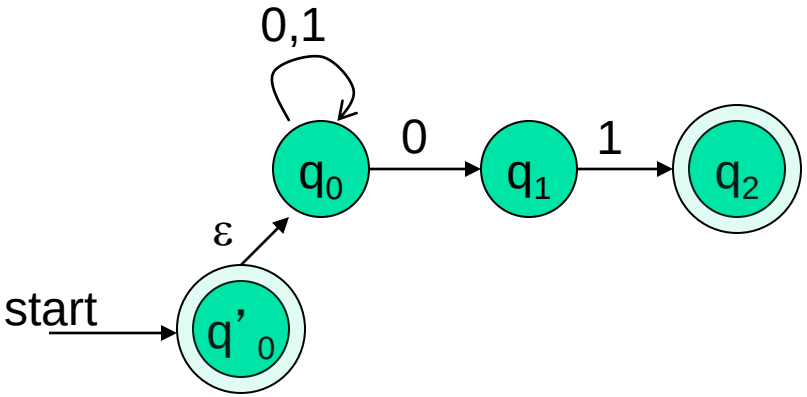
- We can allow <u>explicit</u> ε-transitions in finite automata
  - i.e., a transition from one state to another state without consuming any additional input symbol
  - Explicit ε-transitions between different states introduce non-determinism
  - Makes it easier sometimes to construct NFAs

<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	
→ *q'_0	Ø	Ø	{q'_0,q_0}	ECLOSE(q'_0)
q_0	{q_0,q_1}	{q_0}	{q_0}	ECLOSE(q_0)
q_1	Ø	{q_2}	{q_1}	ECLOSE(q_1)
*q_2	Ø	Ø	{q_2}	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.

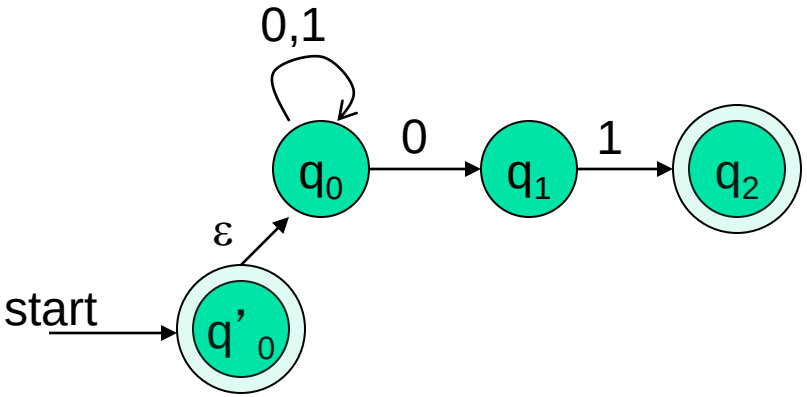
To simulate any transition:

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}

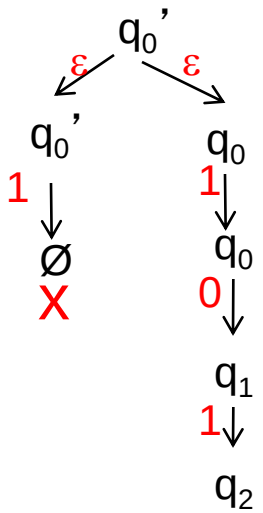


δ_E	0	1	ε
→ *q'_0	Ø	Ø	{q'_0,q_0}
q_0	{q_0,q_1}	{q_0}	{q_0}
q_1	Ø	{q_2}	{q_1}
*q_2	Ø	Ø	{q_2}

ECLOSE(q'_0)

ECLOSE(q_0)

Simulate for w=101:

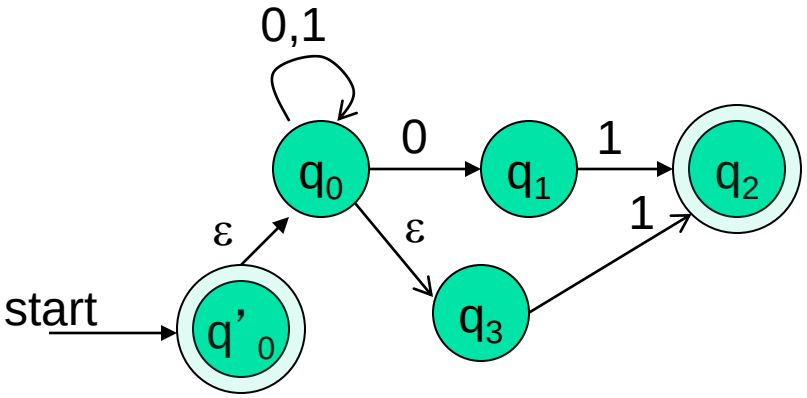


To simulate any transition:

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

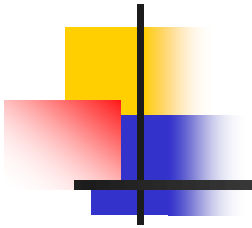


Simulate for w=101:

?

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	$\{q_2\}$	$\{q_3\}$

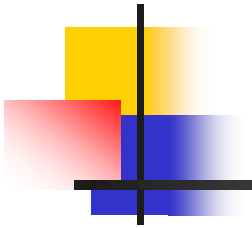
Equivalency of DFA, NFA, ε-NFA



- 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

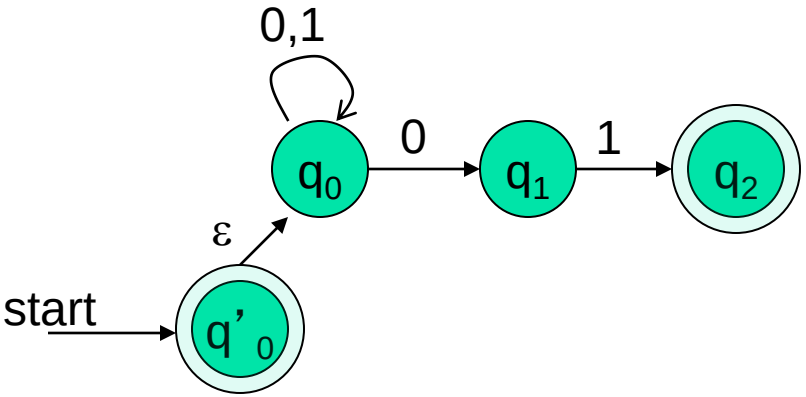
Goal: To build DFA $D=\{Q_D, \Sigma, \delta_D, \{q_D\}, F_D\}$ s.t. L(D)=L(E)

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 $R = \bigcup_{r \text{ in } R} \delta_E(p,a)$ // go to destination states
   - $\delta_D(S,a) = \bigcup_{p \text{ in } s} ECLOSE(p)$ // 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}

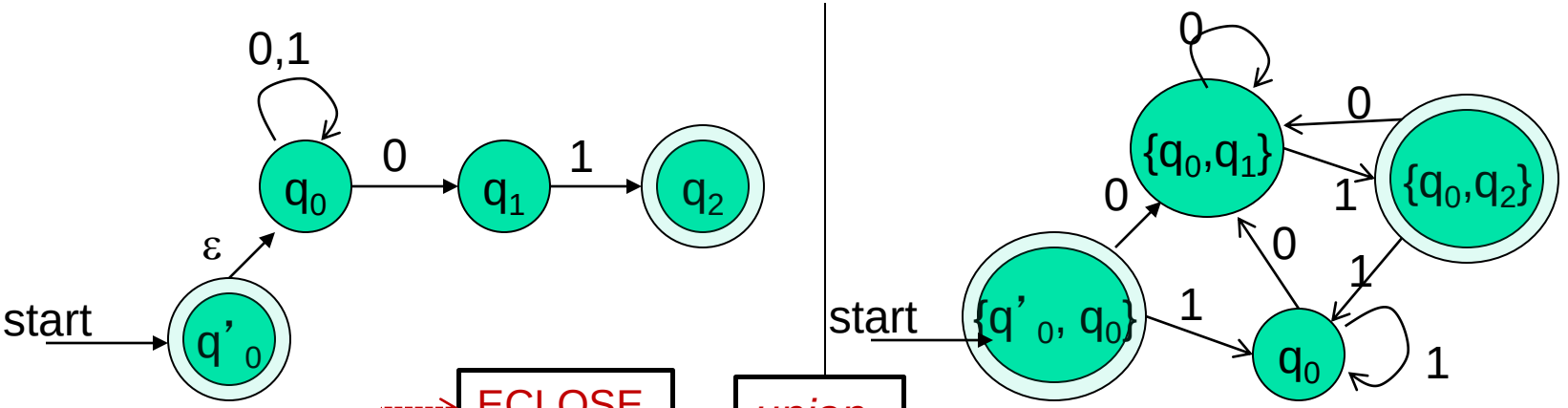


δ_E	0	1	ε
→ *q'_0	Ø	Ø	{q'_0,q_0}
q_0	{q_0,q_1}	{q_0}	{q_0}
q_1	Ø	{q_2}	{q_1}
*q_2	Ø	Ø	{q_2}

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}



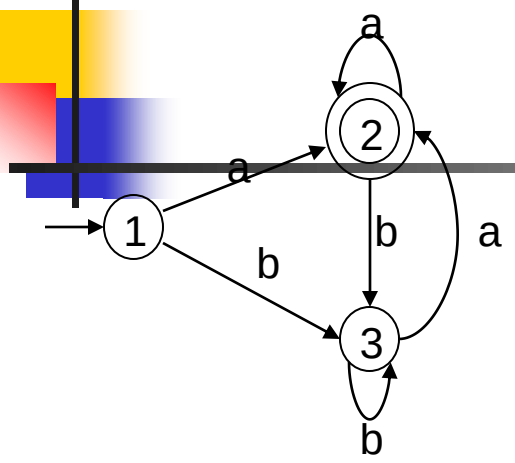
δ_E	0	1	ε
→ *q'_0	Ø	Ø	$\{q'_0,q_0\}$
q_0	$\{q_0,q_1\}$	$\{q_0\}$	$\{q_0\}$
q_1	Ø	$\{q_2\}$	$\{q_1\}$
*q_2	Ø	Ø	$\{q_2\}$

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

Minimizing Number of States of a DFA

- partition the set of states into two groups:
  - G_1 : set of accepting states
  - G_2 : set of non-accepting states
- For each new group G
  - partition G into subgroups such that states s_1 and s_2 are in the same group iff for all input symbols a, states s_1 and s_2 have transitions to states in the same group
- Start state: the group containing the start state of the original DFA
- Accepting states: the groups containing the accepting states of the original DFA

Minimizing DFA - Example



$G_1 = \{2\}$

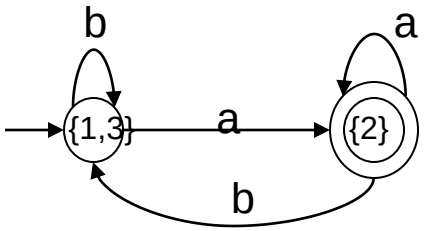
$G_2 = \{1,3\}$

G_2 cannot be partitioned because

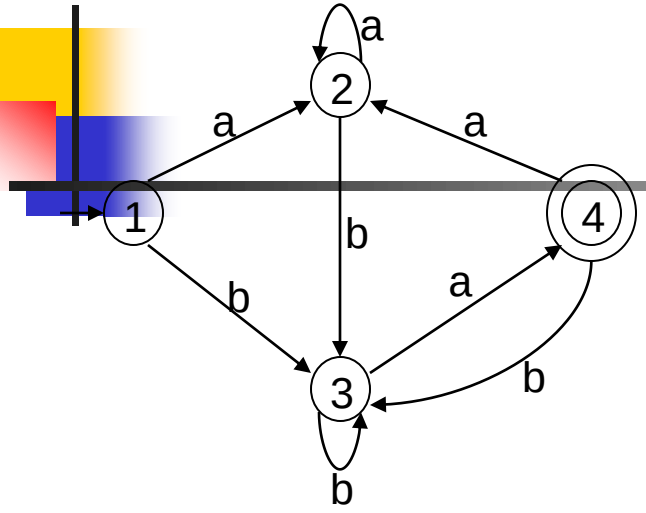
move(1,a)=2 move(1,b)=3

move(3,a)=2 move(3,b)=3

So, the minimized DFA (with minimum states)



Minimizing DFA – Another Example



Groups: {1,2,3} {4}

{1,2} {3}

no more partitioning

a	b
1->2	1->3
2->2	2->3
3->4	3->3

So, the minimized DFA