Chapter 1: Regular Languages

Finite Automata

- Models for computers with a limited amount of memory
- It reads one pass through the input
- Has no capability to write

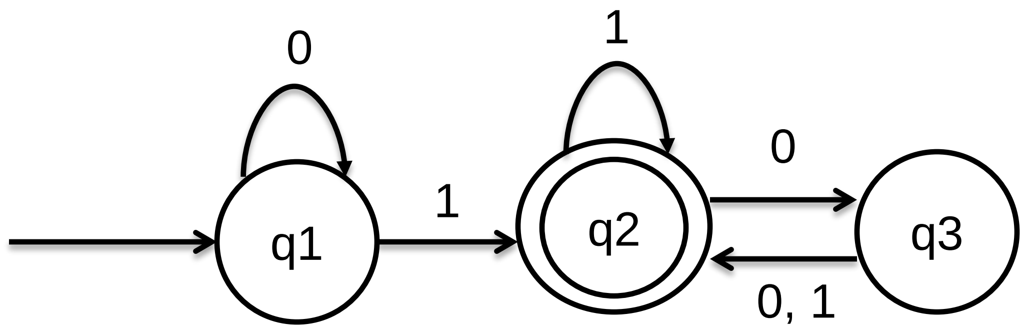
Deterministic Finite Automata (DFA)

A finite automaton is a 5-tuple (Q, Σ, δ, s, F), where

1. S is a finite set called the *states*
2. Σ is a finite set called the *alphabet*
3. δ:($Q \times \Sigma \rightarrow Q$) is the *transition function* between states

    i.e., (state, symbol) ---> next state
4. s is the *start state* (one special state)
5. $F \subseteq Q$ is the set of *accept states* (0 or more accept states)

State Diagram



q1: Start state q2: An accept state

The arrows going from one state to another are called transitions

How does a DFA work?

- An input string is placed on the tape (left-justified).
- Each cell contains one symbol
- The reading head is placed on the leftmost cell of the tape.
- DFA begins from the start state.
- On the symbol the head points to, DFA transit from one state to the next state (may be the same state)

How does a DFA work? (contd.)

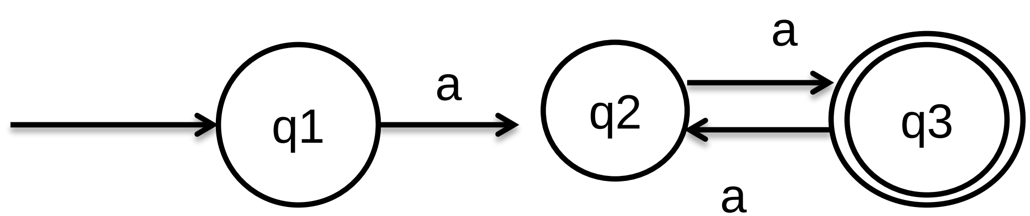
- DFA continue the transitions until the entire string is read.
  - In each step, DFA consults a transition table and changes state based on (s, σ) where
    - s - current state
    - σ - current symbol scanned by the head
- After reading the entire input string,
  - if DFA ends in an accept state, the input is accepted
  - if DFA ends in a non-accept state, the input is rejected.

Languages

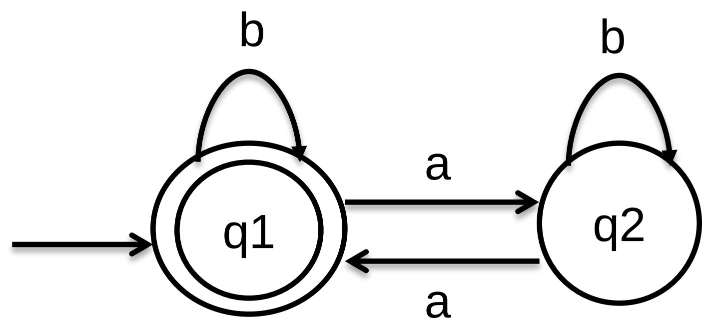
- A language L is a subset of Σ^*
  - i.e., language {0, 01, 11} is a subset of $\{0,1\}^*$
- The language accepted by a DFA D = L(D) is the set of all strings *w* such that *D* ends in an accept state on input *w.*
- A language is called a ***regular language*** if there exists a DFA that recognizes/accepts it.

$L = \{a^{2n} \mid n >= 1\}$

- L={aa, aaaa, aaaaaa, ……}



Example: L(M) = {w in {a, b}* | w contains even number of a's}



Regular Languages

- A Language is regular iff there is a finite automaton that accepts it.
- Examples: design DFAs for the following regular languages:
  - ϕ
  - $\{\varepsilon\}$
  - Σ^*
  - {w in {0,1}* | w starts with 1 and ends with 0}
  - {w in {0,1}* | the second symbol of w is 1}
  - {w in {0,1}* | w contains 1010 as a substring}

Closure properties of regular languages

- The class of regular languages are closed under the union, intersection, and complement operations

Example

- $\Sigma = \{a, b\}$
- $L_1 = \{ w \text{ in } \Sigma^* \mid w \text{ has even number of a's}\}$
- $L_2 = \{ w \text{ in } \Sigma^* \mid w \text{ has odd number of b's}\}$.
  - $L_1 \cup L_2 = ?$
  - $L_1 \cap L_2 = ?$
  - $\overline{L1}$

General construction of DFAs for the languages after union and intersection

- Let $M_1 = (Q_1, \Sigma, \delta_1, s_1, F_1)$ be the DFA for L_1 and $M_2 = (Q_2, \Sigma, \delta_2, s_2, F_2)$ be the DFA for L_2
  - $M = (Q, \Sigma, \delta, s, F)$ where:
    - $Q = Q_1 \times Q_2$
    - $s = (s_1, s_2)$
    - Σ is the same
    - $\delta((q_1, q_2), \sigma) = (\delta_1(q_1, \sigma), \delta_2(q_2, \sigma))$
    - for Union, $F = (Q_1 \times F_2) \cup (F_1 \times Q_2)$
    - for Intersection, $F = F_1 \times F_2$

DFAs for $L_1 \cup L_2$ and $L_1 \cap L_2$?

- Σ = {a, b}
- $\mathbf{L_1}$ = { w in Σ^* | w has even number of a's}
- $\mathbf{L_2}$ = { w in Σ^* | w has odd number of b's}.

Construction for Complement for DFAs

Given DFA M1 = (Q1, Σ, δ1, s1, F1)

L(M) = Complement of L(M1)

Swap the accept and non-accept states of M1 to create M that recognizes the complement language of L1:

$M = (Q, \Sigma, \delta, s, F)$
$Q = Q_1$
$s = s_1$
$F = Q - F_1$
$\delta = \delta_1$

Examples

Nondeterministic Finite Automaton (NFA)

- In a DFA, for a given state and the an input symbol, the next state is fixed
- In a NFA, several choices may exist for the next state at any point.
- NFA is a generalization of DFA. A NFA allows:
  - 0 or more next states for the same (state, symbol): guessing the next state,
  - Transitions can be labeled by the empty string ε : changing state without reading input,
  - No transition on an input symbol.

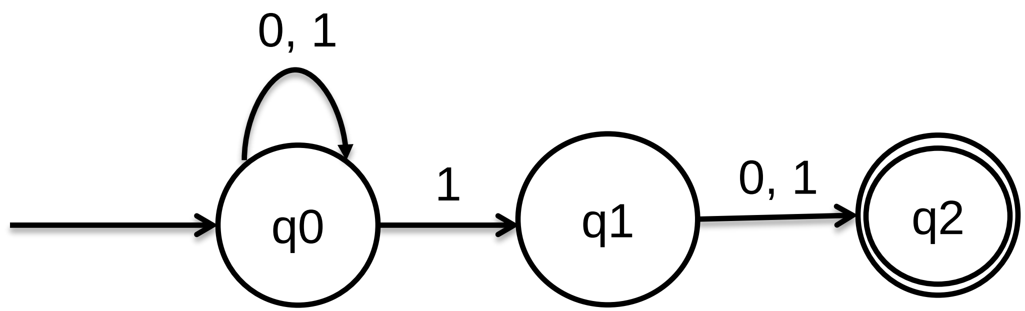
Formal definition of NFA

- $\Sigma_\varepsilon = \Sigma \cup \{\varepsilon\}$.
- NFA M = (Q, Σ, δ, s, F) where:
  - Q: finite set of states
  - Σ: finite input alphabet
  - δ: a subset of Q X Σ_ε X Q.
  - s: the start state
  - F $\subseteq$ Q – the set of accept states

How does an NFA work?

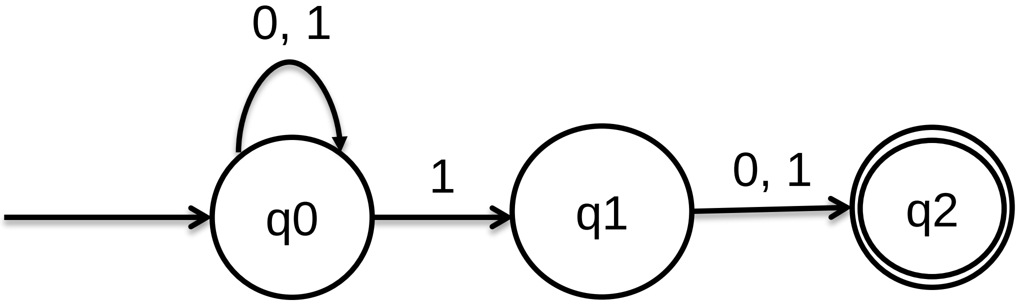
- String w is accepted by a NFA if there exists a sequence of guesses that lead to an accept state after reading the entire string w.
- Language accepted by a NFA is the set of all strings that are accepted by the NFA.

Example: {w in {0,1}* | the second to the last symbol of w is a 1}



NFA acceptance

- Define $\delta^*(q, w)$ as a set of states: $\{p \mid p \in \delta^*(q, w)$ if there is a directed path from *q* to *p* labeled with *w*.$\}$
  - $\delta^*(q_0, 1) = \{q_0, q_1\}$
  - $\delta^*(q_0, 11) = \{q_0, q_1, q_2\}$



NFA acceptance (cont'd)

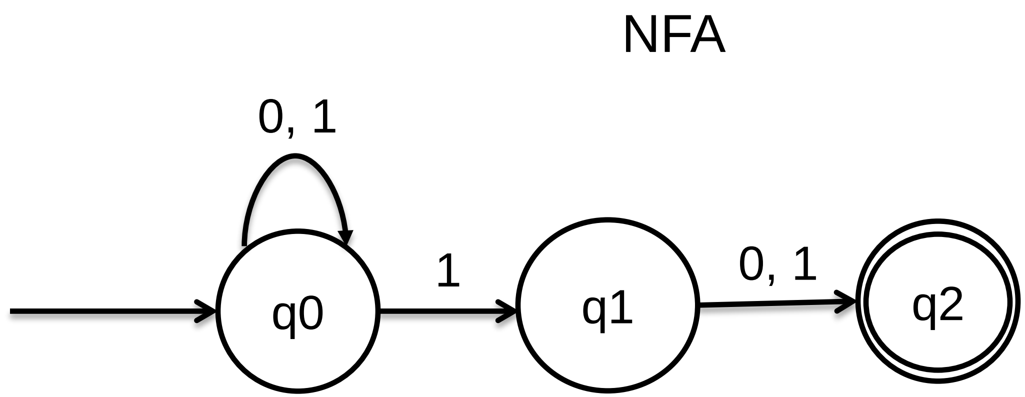
- w is accepted by NFA M iff $\delta^*(q_0, w) \cap F$ is not empty.

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

NFA vs. DFA

- Theorem: For every NFA M there is an equivalent DFA M'
  - NFA is not more powerful than DFA!
- Proof Idea:
  - DFA uses more states to get rid of the nondeterminism.

Example: Conversion from NFA to an equivalent DFA



δ	**0**	**1**
q0	{q0}	{q0,q1}
q1	{q2}	{q2}
q2	∅	∅

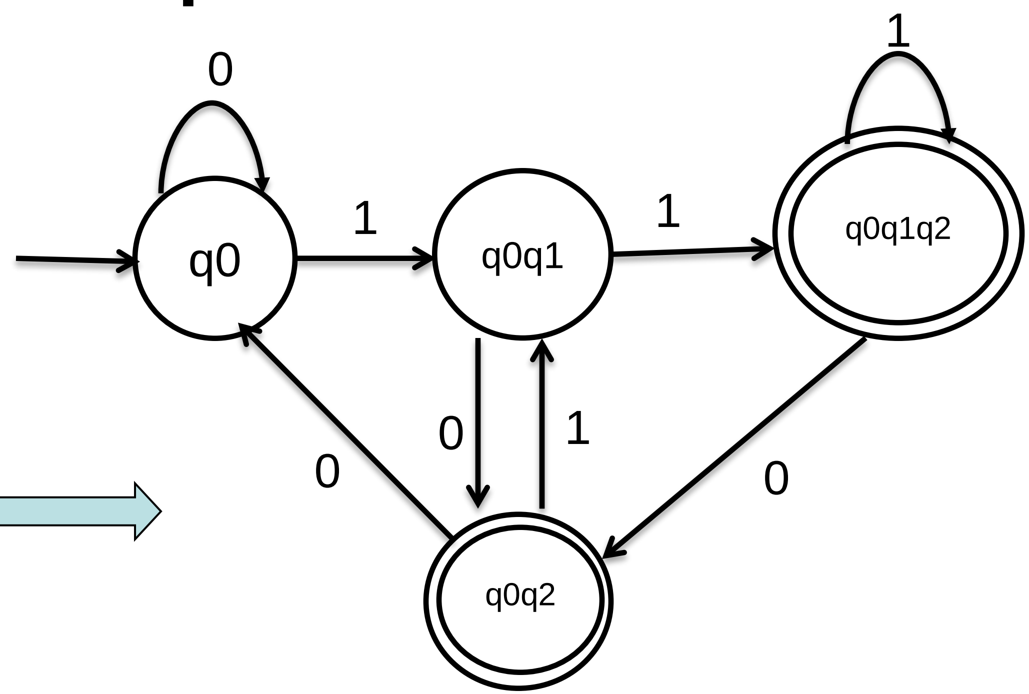
Traditional method: Conversion from NFA to an equivalent DFA

- For now, assume no transitions labeled by ε in the NFA (will get rid of this assumption later!)
- NFA $M = (Q, \Sigma, \Delta, s, F)$
- DFA $M' = (Q', \Sigma, \delta, s', F')$ where:
  - $Q' = 2^Q$
  - $s' = s$
  - $F' = \{q \mid q \cap F$ is not empty, i.e, q contains at least one accept state from the NFA M$\}$
  - $\delta(\{p_1, p_2, \ldots, p_m\}, \sigma) = \delta^*(p_1, \sigma) \cup \delta^*(p_2, \sigma) \cup \ldots \cup \delta^*(p_m, \sigma)$

1. Traditional method for the example

δ	0	1
q0	{q0}	{q0,q1}
q1	{q2}	{q2}
q2	∅	∅

δ	0	1
∅	∅	∅
q0	q0	q0q1
q1	q2	q2
q2	∅	∅
q0q1	q0q2	q0q1q2
q0q2	q0	q0q1
q1q2	q2	q2
q0q1q2	q0q2	q0q1q2



states Ø, q_1, q_2, and q_1q_2 can be deleted because they don't have incoming transitions: they cannot be reached from the start state q0.

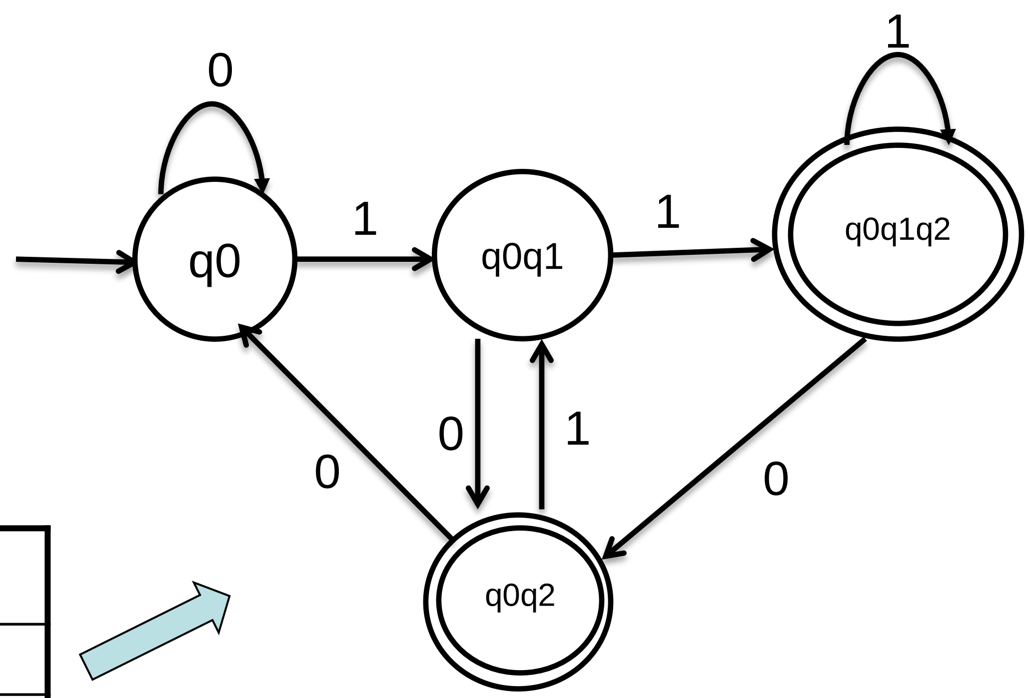
2. Subset Construction Method

- For every state in the NFA, determine all ***reachable states*** for every input symbol (first table).
- The set of reachable states constitute a ***single state*** in the converted DFA (Each state in the DFA corresponds to a subset of states in the NFA).
- Starting from the start state, find *reachable states* for each *new DFA state*, until no more new states can be found.

Example

δ	**0**	**1**
q0	{q0}	{q0,q1}
q1	{q2}	{q2}
q2	Ø	Ø

δ	**0**	**1**
q0	q0	q0q1
q0q1	q0q2	q0q1q2
q0q2	q0	q0q1
q0q1q2	q0q2	q0q1q2



How to handle ε transitions?

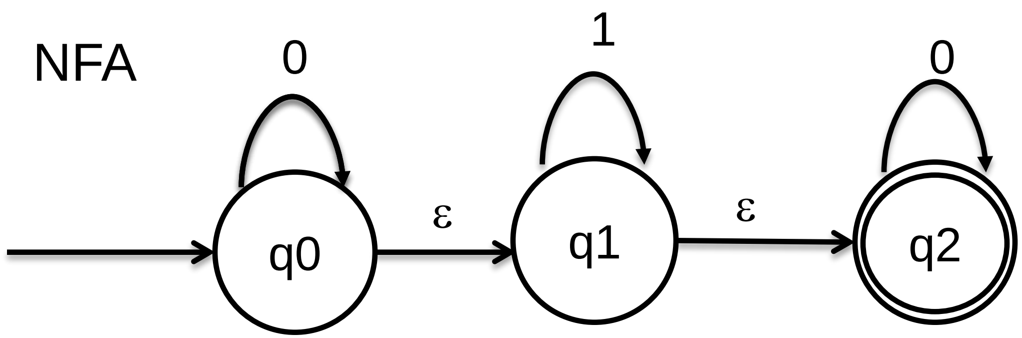
- Define ε-closure of state q as $\delta^*(q, \varepsilon)$.
  - notation: ε -closure(q)= $\delta^*(q, \varepsilon)$ (all the states including itself that can be reached from q via 0 or more ε's).
- Extend ε-closure to sets of states by:
  - ε-closure($\{s_1, \ldots, s_m\}$) = ε-closure(s_1) $\cup \ldots \cup$ ε-closure(s_m)
- For the equivalent DFA, the start state s' of the DFA is

  s' = ε-closure(s)

  and,

  $\delta(\{p_1, \ldots, p_m\}, \sigma) = \varepsilon\text{-closure}(\Delta^*(p_1, \sigma)) \cup \ldots \cup \varepsilon\text{-closure}(\Delta^*(p_m, \sigma))$
- Others are the same as the DFA construction from a NFA without ε transition.

Example: Convert a NFA with ε transitions to DFA



ε-closure(q0)={q0,q1,q2}
ε-closure(q1)={q1,q2}
ε-closure(q2)={q2}

δ	0	1
q0	{q0}	∅
q1	∅	{q1}
q2	{q2}	∅

after
ε-closure

δ	0	1
q0	{q0, q1,q2}	∅
q1	∅	{q1, q2}
q2	{q2}	∅

Using subset construction method

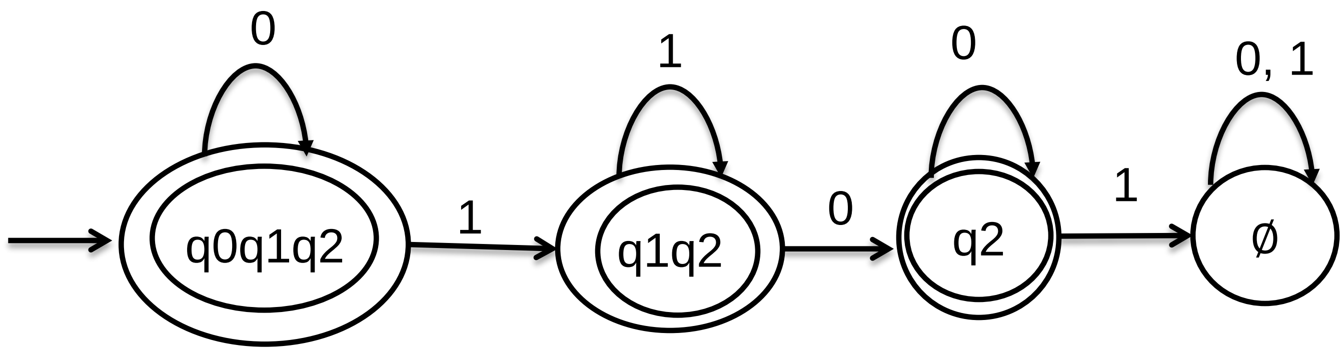
The start state of the NFA is q0, so the start state of the DFA is ε-closure(q0), which is **q0q1q2**. Other 3 tuples are constructed the same way as the conversion for NFAs without ε transitions.

δ	0	1
q0	{q0, q1,q2}	∅
q1	∅	{q1, q2}
q2	{q2}	∅

DFA

δ	0	1
q0q1q2	q0q1q2	q1q2
q1q2	q2	q1q2
q2	q2	∅
∅	∅	∅

DFA =



(The other four states do not have incoming transitions and thus cannot be reached from the start state. They are omitted here.)

Regular Operations

- Regular operations:
  - Union: $L1 \cup L2 = \{ x | x \in L1 \text{ or } x \in L2\}$
  - Concatenation: $L1 \bullet L2 = \{xy | x \in L1 \text{ and } y \in L2\}$
  - Star: $L^* = \{x_1x_2\ldots x_k | k \geq 0 \text{ and each } x_i \in L\}$
- Example: if $L1 = \{a^{2n+1} \mid n \geq 0\}$, $L2 = \{b^{2n} \mid n \geq 0\}$.
  - $L1 \bullet L2 = \{a^{2n+1}b^{2m} \mid n, m \geq 0\}$
  - $L1^* = \{a^n \mid n \geq 0\}$

Closure properties of regular languages

- Previously we discussed regular languages are closed under union, intersection, and complement.
- Regular languages are also closed under
  - Concatenation
  - Star

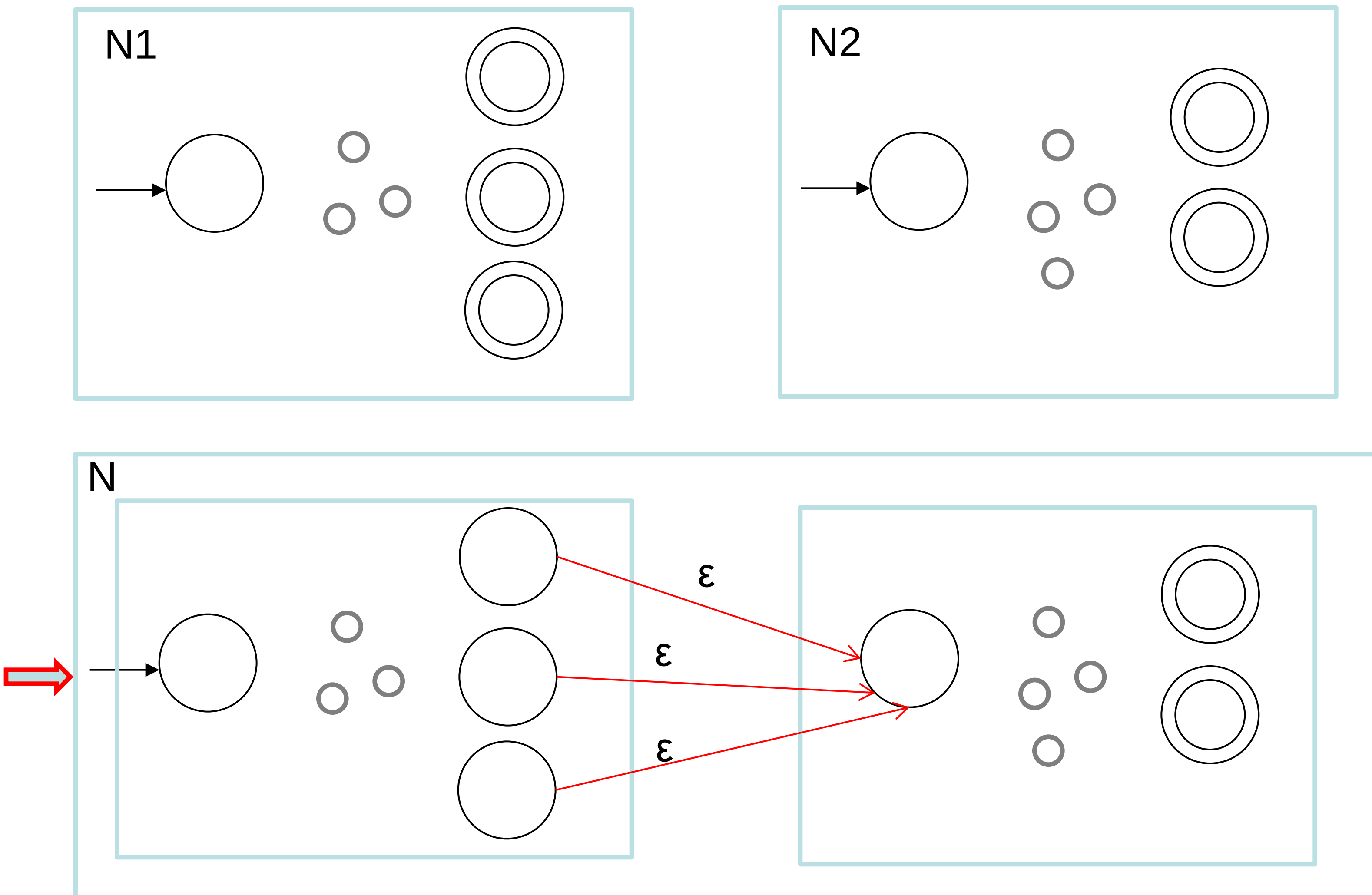
Construction for L1•L2

L1= (Q1, Σ, δ1, s1, F1)

L2= (Q2, Σ, δ2, s2, F2)

L= (Q, Σ, **δ**, s, F)
$Q = Q_1 \cup Q_2$
$s = s_1$
$F = F_2$
$\boldsymbol{\delta} = \boldsymbol{\delta}_1 \cup \boldsymbol{\delta}_2 \cup F_1 \times \{\varepsilon\} \times \{s_2\}$

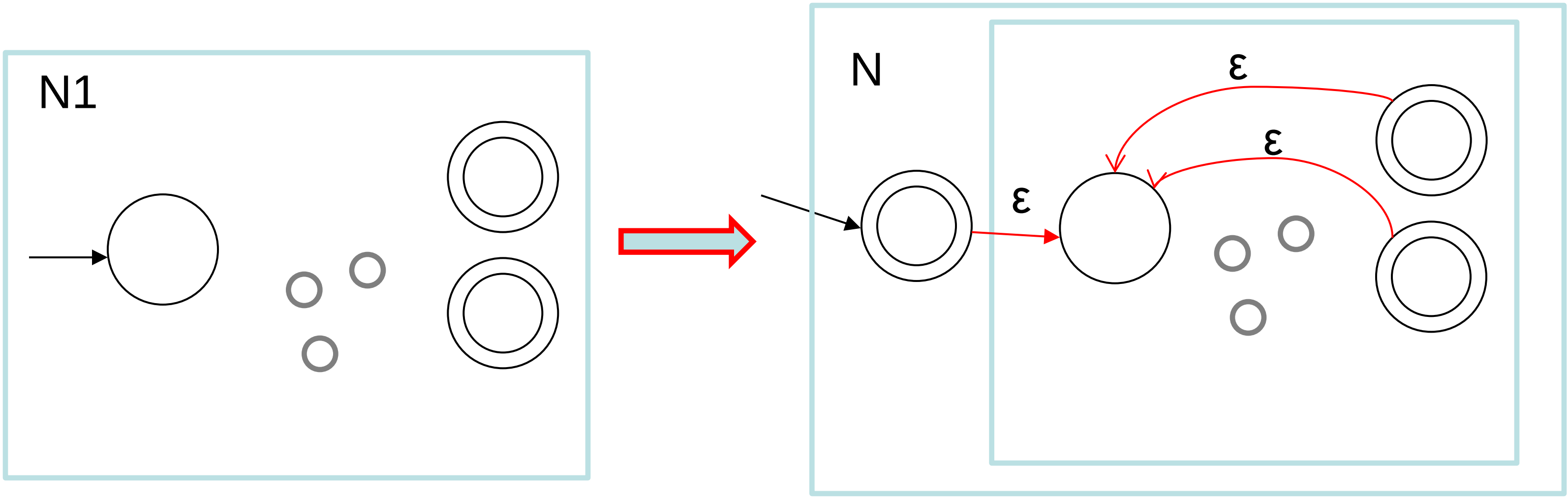
Construction for L1•L2 (cont'd)



Construction for Star

L1= (Q1, Σ, δ1, s1, F1)

L=L1*= (Q, Σ, δ, s, F)
$Q = Q_1 \cup \{s\}$
s is the new start state
$F = F1 \cup \{s\}$
$\delta = \delta_1 \cup F_1 \times \{\varepsilon\} \times \{s1\}$



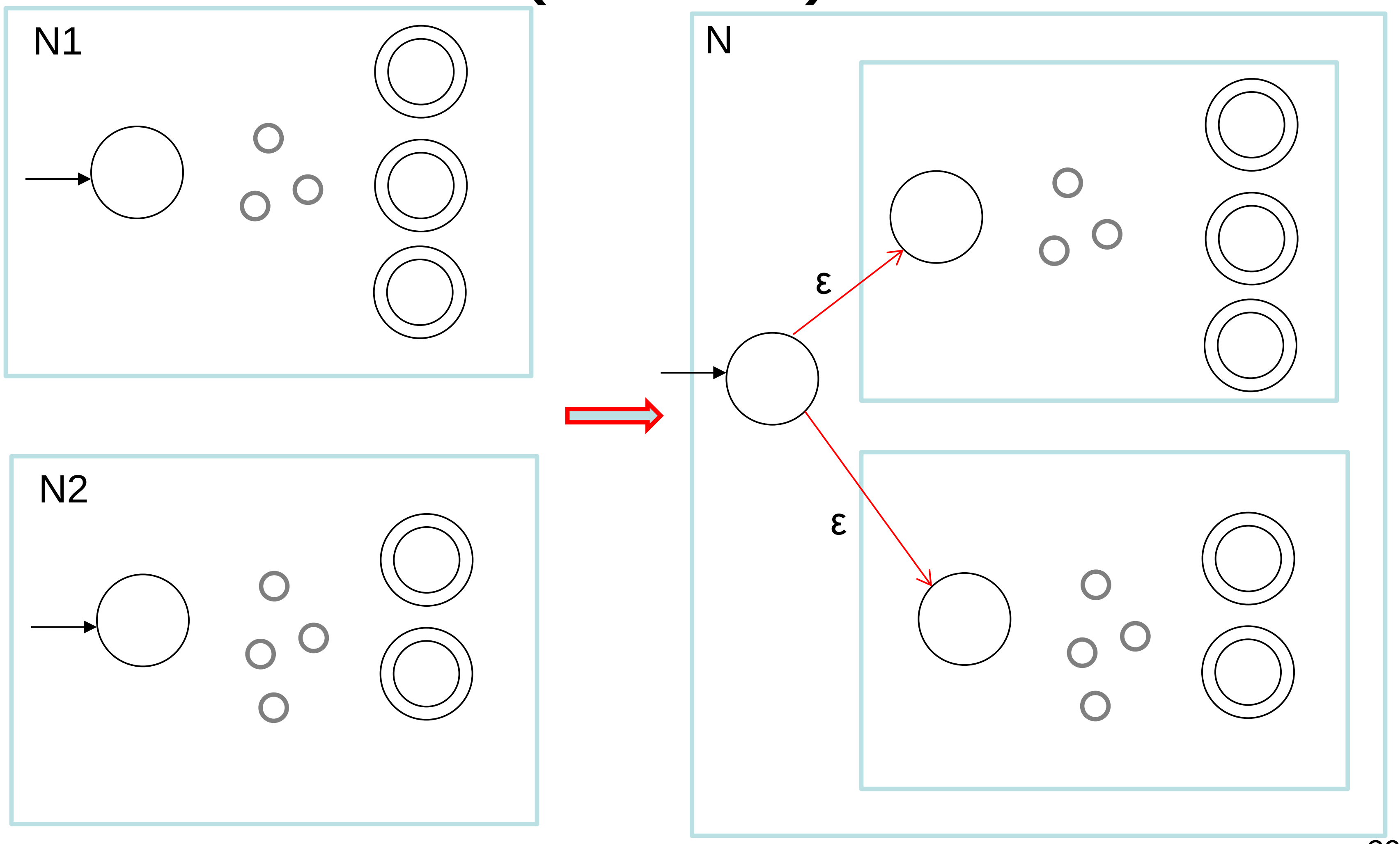
Construction the NFA for $L1 \cup L2$

$L1 = (Q1, \Sigma, \delta1, s1, F1)$

$L2 = (Q2, \Sigma, \delta2, s2, F2)$

$L = (Q, \Sigma, \delta, s, F)$
$Q = \{s\} \cup Q_1 \cup Q_2$
s is the new start state
$F = F_1 \cup F_2$
$\delta = \delta_1 \cup \delta_2 \cup s \times \{\varepsilon\} \times \{s_1, s_2\}$

Construction for L1 ∪ L2 (cont'd)



Regular expressions

- It is another way to view regular languages.
- Definition of <u>Regular expressions</u>:
  - a for some a in the alphabet Σ
  - ε
  - ϕ
  - ($R1 \cup R2$), where R1 and R2 are regular expressions
  - ($R1 \bullet R2$), where R1 and R2 are regular expressions
  - ($R1^*$), where R1 is a regular expressions

Examples of regular expressions

- Note: We drop parentheses and dots when not required, i.e.,
  - $(a \cup b)$ is written as $a \cup b$
  - $a \bullet b$ is written as ab
- Let $\Sigma = \{a, b\}$, the following are regular expressions:
  - ϕ, a, b
  - ϕ^*, a^*, b^*, ab, $a \cup b$
  - $(a \cup b)^*$, a^*b^*, $(ab)^*$
  - $(a \cup b)^*ab$

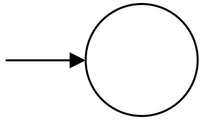
Some exercises on regular expressions

- What is the language of <u>$((a \cup b)^*a(a \cup b)^*)$</u>?
  - Answer: L={w in {a, b}* | w contains at least one a}
- Write regular expressions for:
  - {w in {a, b}* | the length of w (the number of symbols, |w|) is even}.
  - {w in {a, b}* | w does not have ab as a substring}.
  - {w in {a, b}* | no b in w can come before any a in w}.

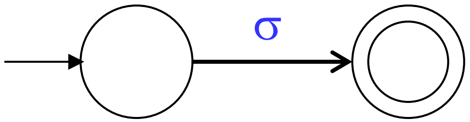
Answer: 1. $(a \cup b)^{2n}$, $n \geq 0$; 2. b^*a^*; 3. a^*b^*

Regular expressions vs. FA's

- a) For every regular expression there is an equivalent NFA
- b) For every DFA there is an equivalent regular expression.
- Proof of (a):
  - For ϕ, the NFA is:

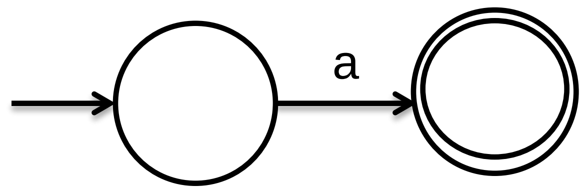


  - For σ, the NFA is:

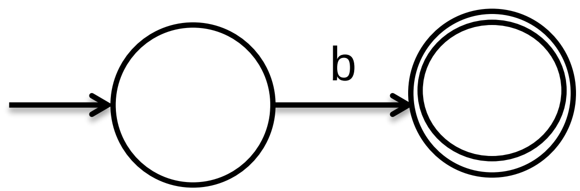


Example: convert regular expression (a U b)*b to NFA

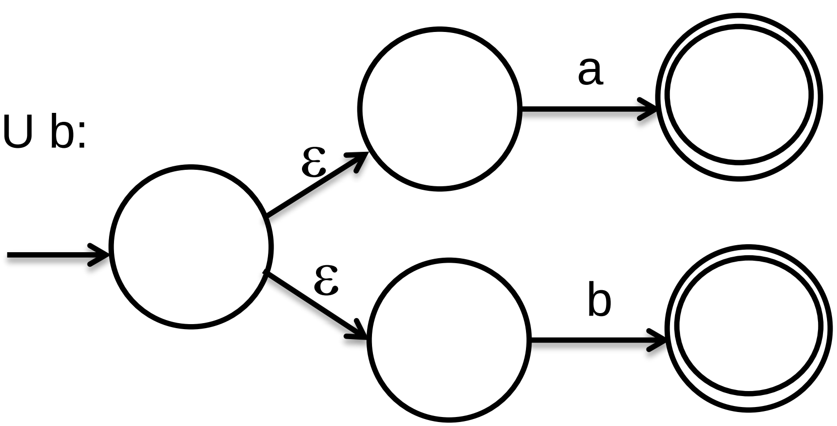
a:



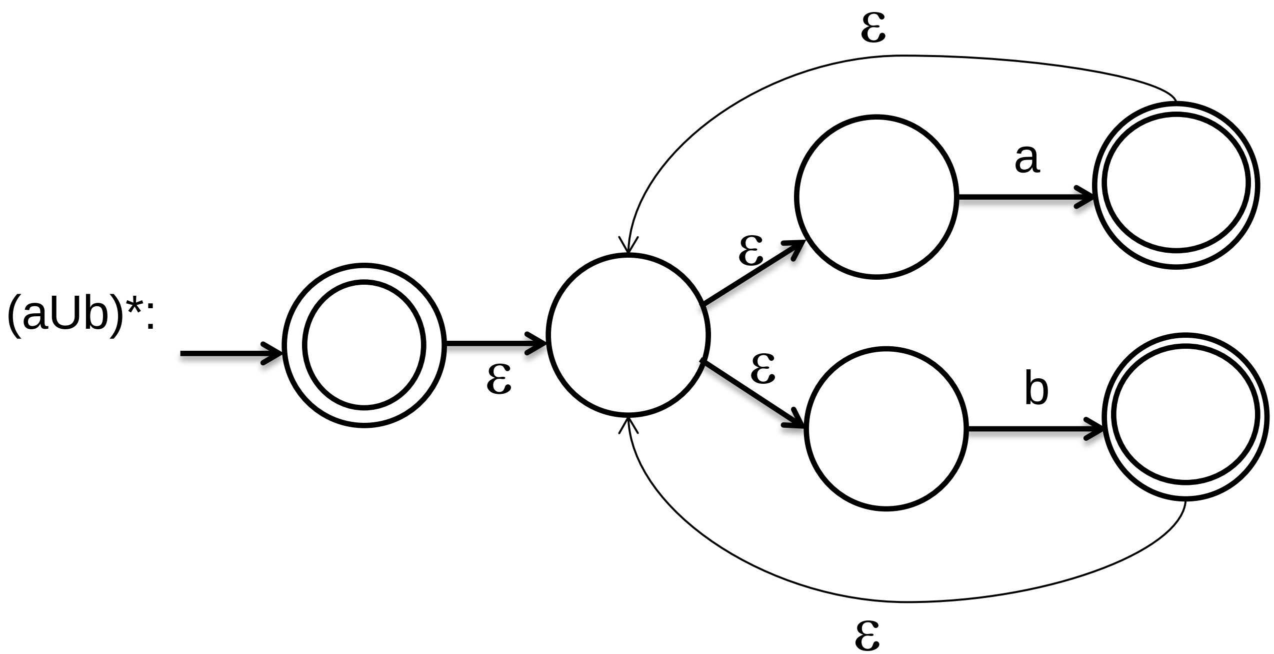
b:



a U b:

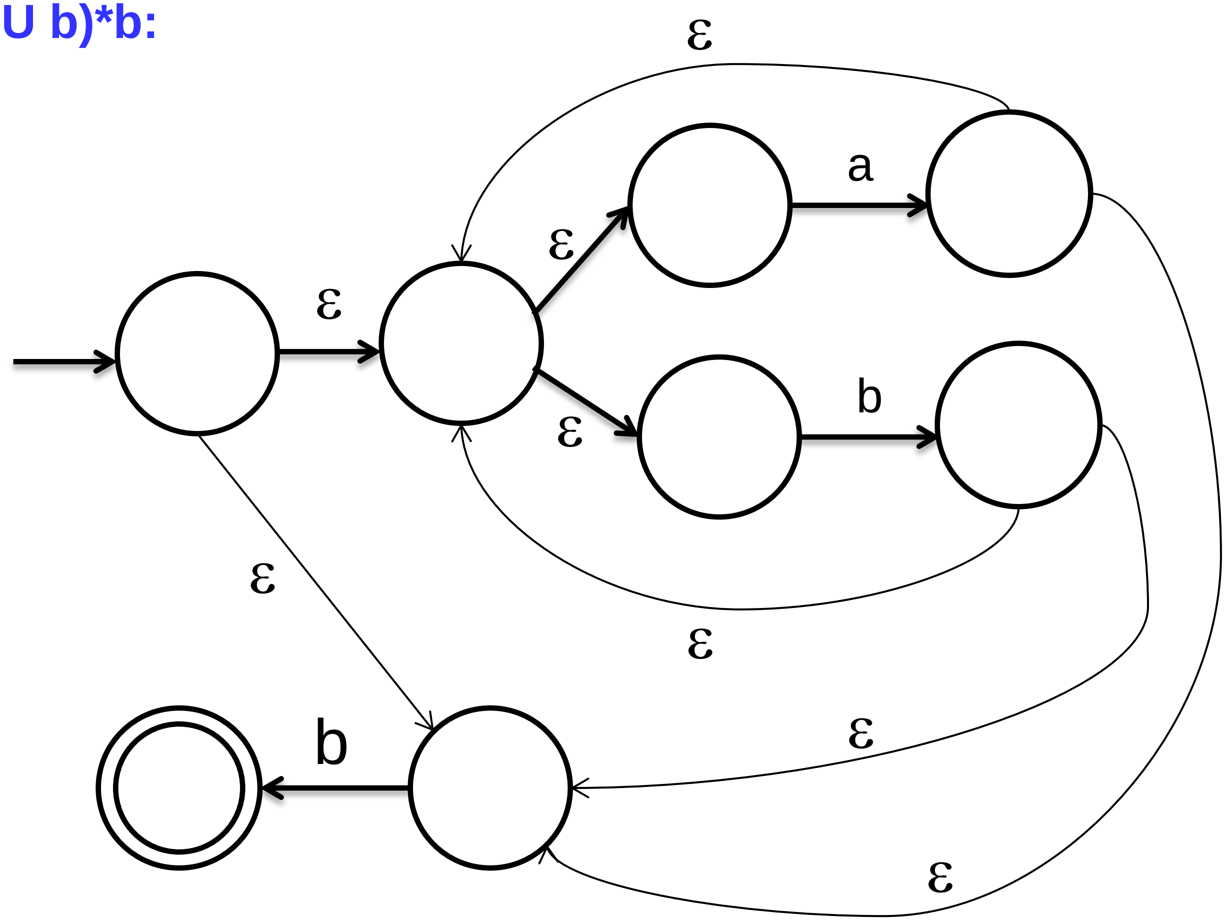


Example (cont'd): NFA for (a U b)*b



Example (contd.) : NFA for (a U b)*b

(a U b)*b:



Exercise

- Convert the Regular expression ab U a* to a NFA

Convert a DFA to a regular expression

- Steps:
  - DFA $\rightarrow$ GNFA $\rightarrow$ regular expression.
- GNFA (Generalized NFA)
  - In GNFA, the labels on the transitions can be regular expressions.
- Need special GNFA that satisfies:

  (1) The start state has no incoming transitions;

  (2) Only one accept state;

  (3) The accept state has no outgoing transitions.

Convert a DFA to a regular expression (cont'd)

- Steps:
  1. Convert the DFA to a special GNFA;
  2. Eliminate one state at a time, except the start state and the accept state, until only the start state and the accept state are left;
  3. Output the label on the single transition from the start state to the accept state.

Eliminating state $q_{\{rip\}}$

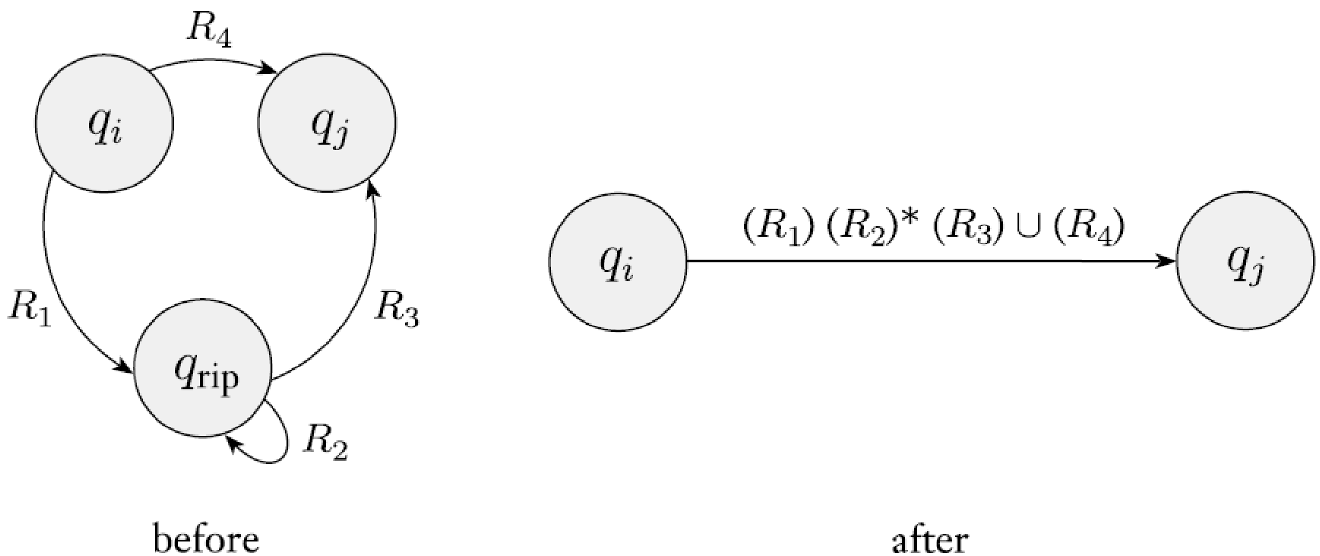


FIGURE 1.63
Constructing an equivalent GNFA with one fewer state

This figure is taken from the book *Introduction to Theory of Computation, Michael Sipser,* page 72.

Example: convert DFA to regular expression

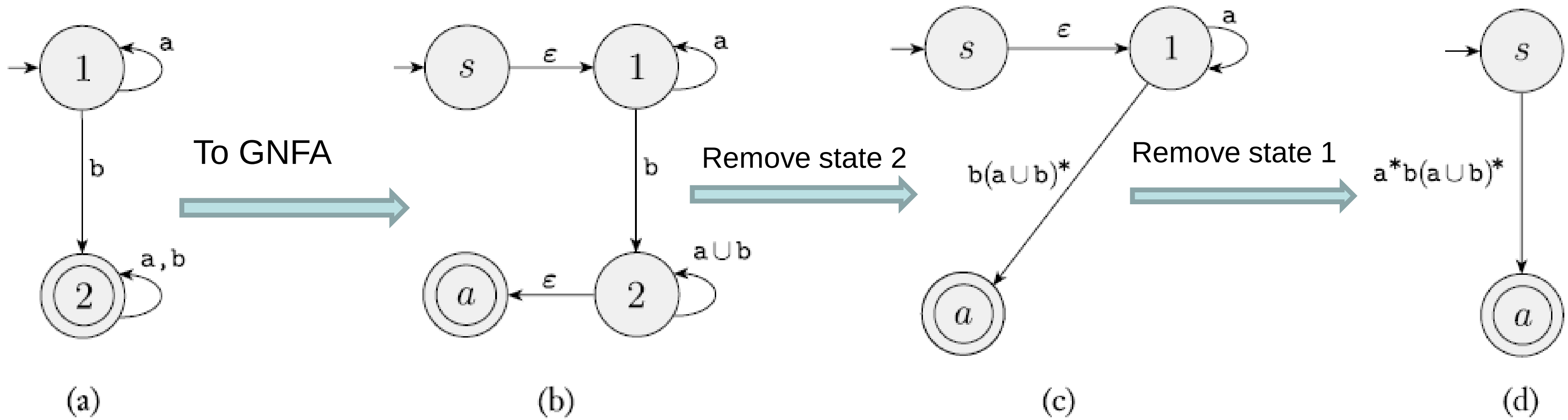


FIGURE 1.67
Converting a two-state DFA to an equivalent regular expression

This figure (a), (b), (c), (d) are taken from Figure 1.67 on the book *Introduction to Theory of Computation, Michael Sipser,* page 75.

Pumping Lemma

- Not all languages are regular
- Pumping lemma is used to show that some languages are not regular.

Statement of Pumping Lemma

If A is a regular language, then there is a number p (the pumping length) where, if s is any string in A of length at least p, then s may be divided into three pieces, $s = xyz$, satisfying the following conditions:

1) $|y| > 0$,
2) $|xy| \leq p$, *and*
3) for each $i \geq 0$, $xy^iz \in$ A.

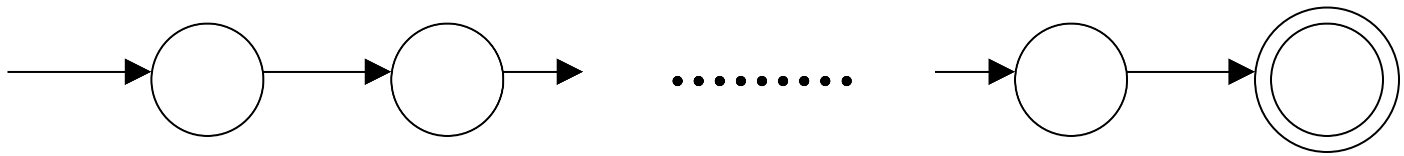
Recall that $|s|$ represents the length of string s, which is the number of symbols in s.

Describing the pumping lemma

For a DFA with *m* states,

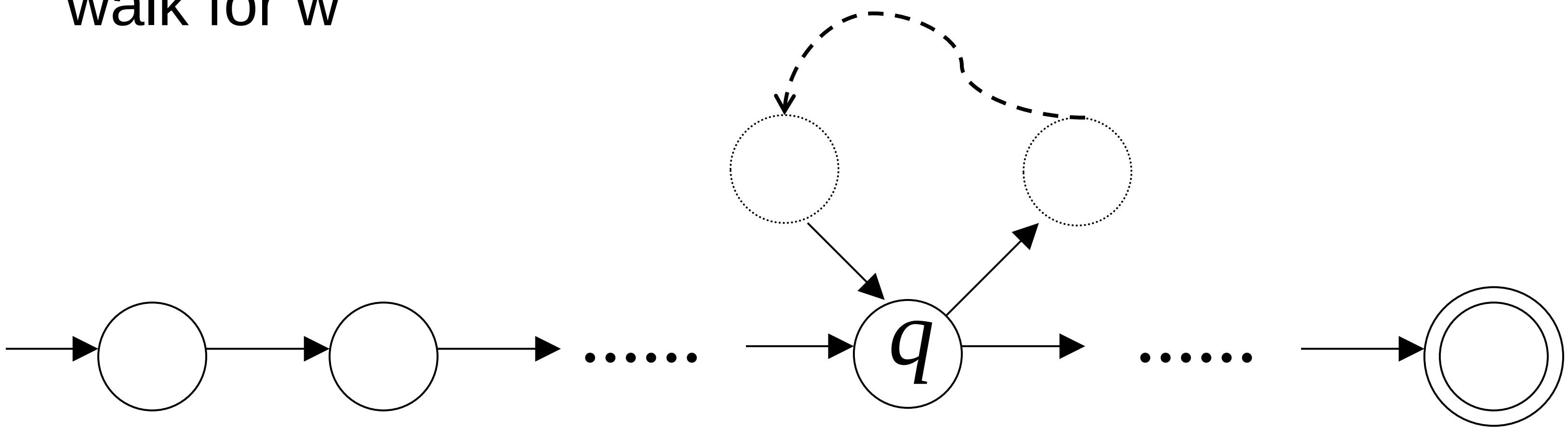
Take string w , $w \in L$

Since $w \in L$, there is a walk from the start state to a final state labeled with w



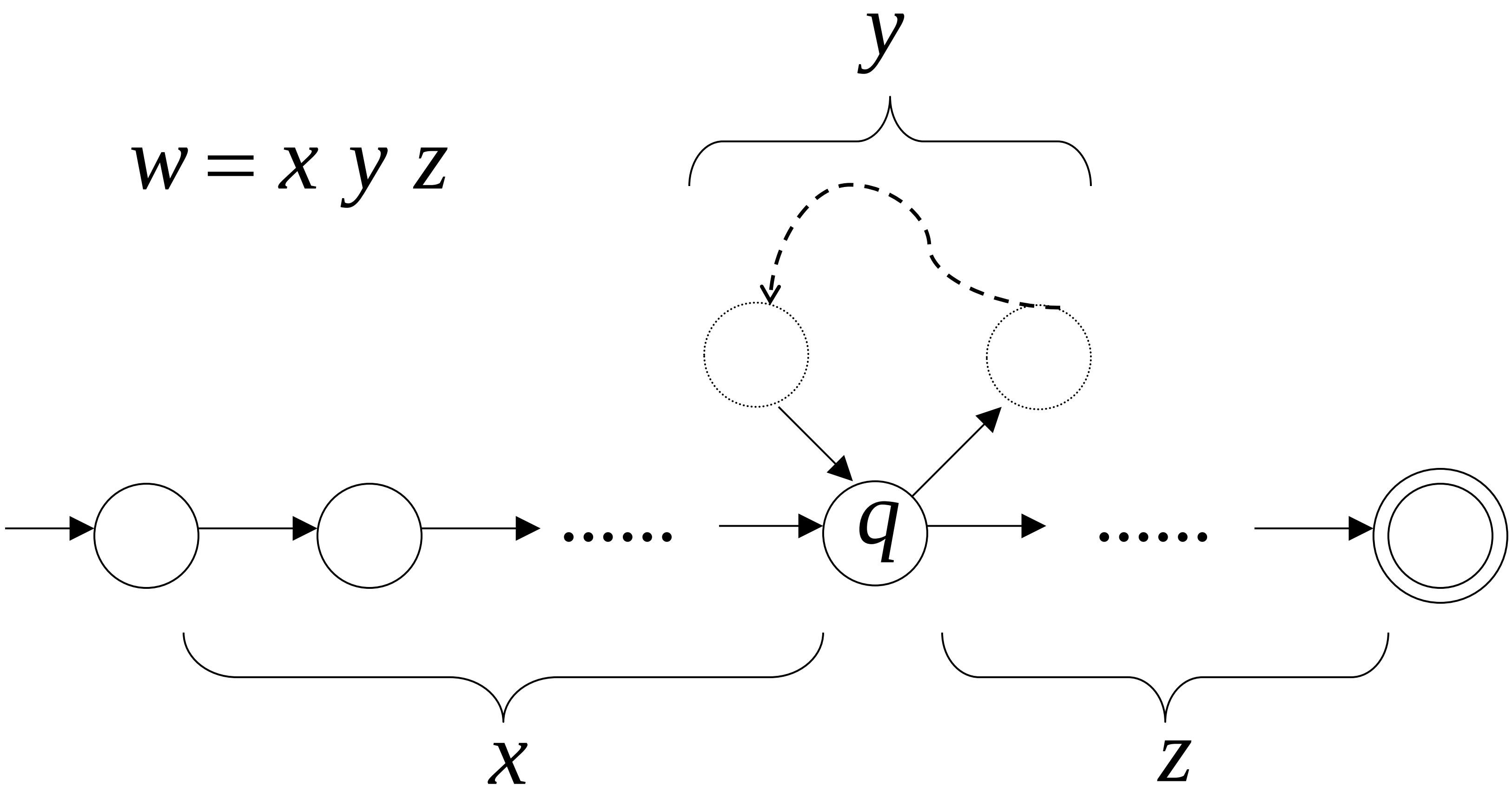
Describing the pumping lemma (cont'd.)

If the length of W is greater than the number of states m, then there must be a state, say *q* that is repeated in the walk for w



Describing the pumping lemma (cont'd.)

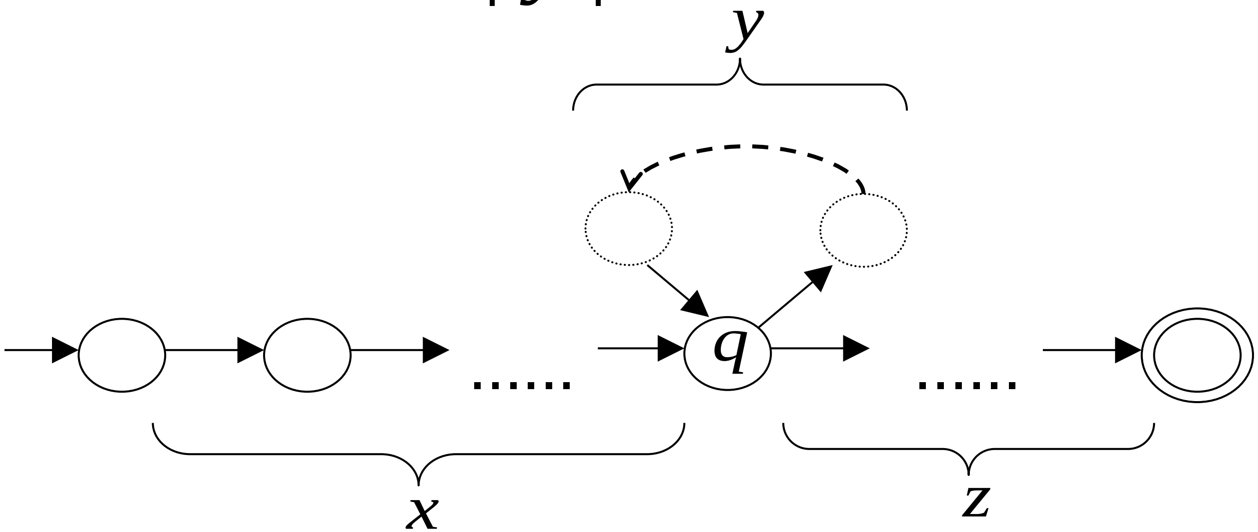
$w = x\ y\ z$



Describing the pumping lemma (cont'd)

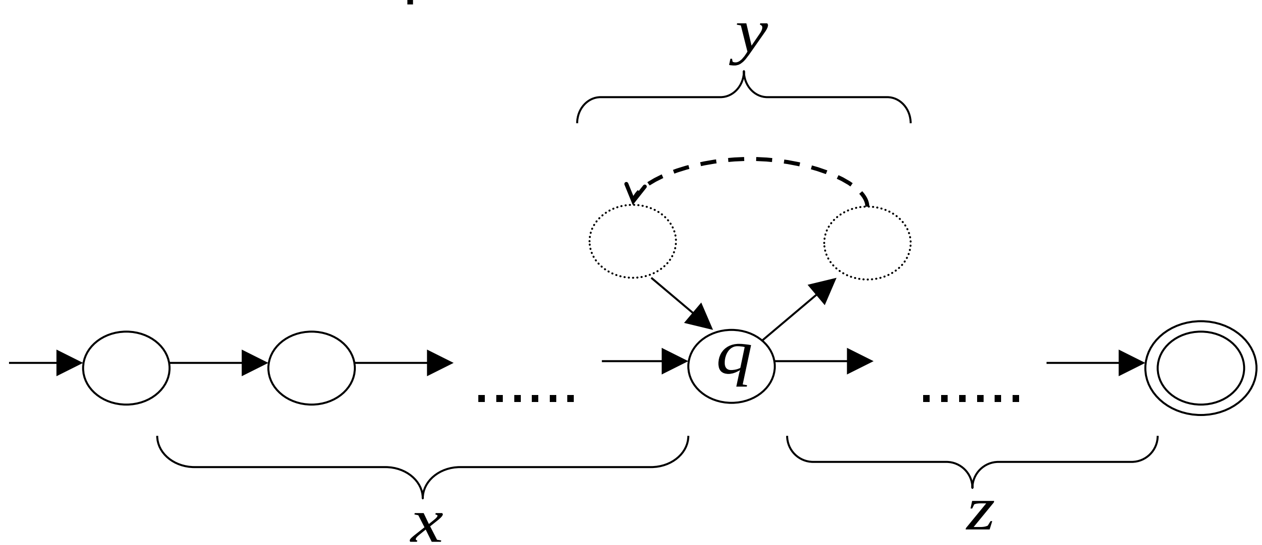
Observations : length $|x\ y| \leq m$ number of states

length $|y| \geq 1$



Describing the pumping lemma (cont'd.)

In General: The string $x\, y^i\, z \qquad i = 0, 1, 2, \ldots$
is accepted



Some Applications of Pumping Lemma

The following languages are not regular.

1. $\{a^n b^n \mid n \geq 0\}$.
2. $\{ww \mid w \text{ in } \{a, b\}^*\}$.
3. $\{w = w^R \mid w \text{ in } \{a, b\}^*\}$ (language of palindromes).
4. $\{1^{n^2} \mid n \geq 0\}$.

Prove $L = \{a^n b^n \mid n \geq 0\}$ is not regular

Proof:

Since L is infinite, the pumping lemma applies to L.

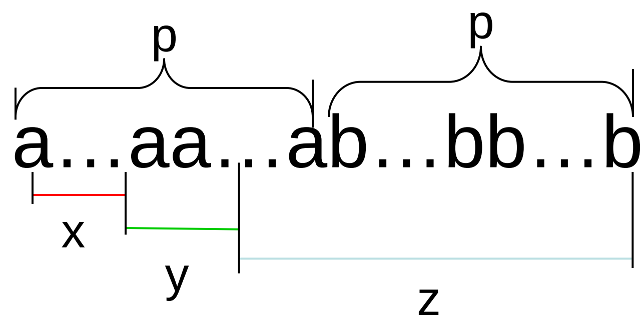
- Assume L is regular.
- Let p be the pumping length
- Let $w = a^p b^p$, $w \in L$, and $|w| \geq p$

Prove $L = \{a^n b^n \mid n \geq 0\}$ is not regular (cont'd)

According to pumping lemma,

$$a^p b^p = xyz$$

and since $|xy| \leq p$



$$x = a^k,\ y = a^m,\ z = a^{p-k-m} b^p$$

$$|y| = m > 0,\ 0 < |xy| = k+m \leq p$$

Prove L = $\{a^n b^n \mid n \geq 0\}$ is not regular (cont'd)

$$xy^2z = xyyz = a^k a^m a^m a^{p-k-m} b^p$$
$$= a^{p+m} b^p$$

But $a^{p+m}b^p \notin L$ since $m > 0$, which contradict pumping lemma (3). Therefore, the assumption that L is a regular language is not true.

Important points of Using Pumping Lemma

- Cannot use a specific number for p
  - Choosing p=3 or any number is not right
- String w must belong to L and |w| is at least the pumping length.
  - Choosing $w = a^2b^2$ is wrong since we do not know the exact value of the pumping length p.
- Must consider all possibilities for what the substrings x, y and z can be, such that w = xyz and $|xy| \leq p$.
- The pumping lemma is used to show that a language is not regular; it cannot be used to show that a language is regular.

Practice

- Design a DFA A such that L(A)={w in {a,b}* | w contains aab as a substring}

Practice 2

- Given: L1 = {all strings that have two consecutive a’s}
- L2 = {all strings that have two consecutive b’s}
- Question: find the automaton A such that L(A) = L1 U L2