Regular Languages: outline

- Regular grammars
- Automata
  - Finite State Automata (FSA)
- Regular expressions
- Minimization of DFSA
- Identification of non-regular languages
  - Pumping Lemma for Regular languages

Regular Grammars

- <u>Definition</u>: a PSG G = (N, T, P, S) is regular provided that:
  i. If there exists a λ-production, then it is of the form $S \rightarrow \lambda$ and S does not appear on the right hand side of any production.
  ii. All other productions are of the form:
    - $\alpha \rightarrow \beta$, $\alpha \in N$, $\beta \in T$, OR
    - $\alpha \rightarrow \beta\gamma$, $\alpha, \gamma \in N$, $\beta \in T$
- A language generated from a regular grammar is called a **Regular Language**.
- Ex1. G_1:

$$S \rightarrow aS \mid aB$$
$$B \rightarrow bB \mid b \quad L(G_1) = \{a^n b^m \mid n \geq 1, m \geq 1\}$$

Is G_1 a regular grammar? **Yes**, It is inline with the given definition

Regular Grammars: cont'd

- **Regular Grammar :** A grammar is regular if it has rules of form A -> a or A -> aB or A -> λ where λ is a special symbol called NULL.
  - Ex2. Write a grammar G_2 such that $L(G_2) = L(G_1) \cup \{\lambda\}$

    G_2:

    $S \rightarrow aA \mid aB \mid \lambda$

    $A \rightarrow aA \mid b$

    $B \rightarrow bB \mid b$

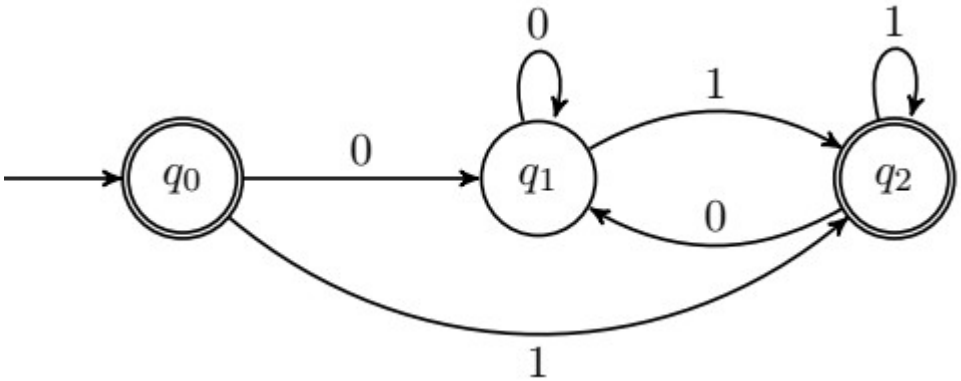
- Ex3: Let $G_3 = (\{S, A, B\}, \{0, 1\}, S, P)$, where P is geven by:
  - $P : S \rightarrow 1A;\ A \rightarrow 1A|0B|0;\ B \rightarrow 0B|0$

    Find the language generated by the Regular grammar G_3

- Given an NFSA M with alphabet X and states Q, with a single starting state $q_0 \in Q$ and no edges labelled λ (for example, a DFSA!), here is how to create a regular grammar $G = (V; T; S; P)$ such that $L(G) = L(M)$.
- • Let $T = X$ and $V = Q$.
- • Let $S = q_0$.
- • If there is an edge $(A; B)$ in M (where $A; B \in Q$) labelled $a \in X$, then make
- $A \rightarrow aB$ a production in P; if also B is an accepting state, also make $A \rightarrow a$ a production in P.
- • If $\lambda \in L(M)$, i.e if the starting state S of M is also accepting, then make $S \rightarrow \lambda$ a production in P.

… Cont'd

- **Example.** Find a regular grammar to generate $L(M)$, where M has state diagram as below:



- ***<u>Solution</u>***. Let $G = (V; T; S; P)$, where $V = \{q_0, q_1, q_2\}$, $T = \{0, 1\}$, $S = q_0$, and $P = \{q_0 \rightarrow 0q_1|1q_2|1|\lambda,\ q_1 \rightarrow 0q_1|1q_2|1,\ q_2 \rightarrow 0q_1|1q_2|1\}$:
- **Regular Languages :**
  - A language is regular if it can be expressed in terms of regular expression.

Properties of Regular Grammar

- <u>Theorem</u>: If L_1 and L_2 are two regular languages, then:
  a. $L_1 \cup L_2$ is also a regular language
  b. L_1L_2 is also a regular language
  c. L_1^* is also a regular language

a. **Theorem:** Given a regular grammar *G*, *L*(*G*) is a regular language. Conversely, given a regular language *L* over an alphabet *X*, there exists a regular grammar *G* such that *L*(*G*) = *L*.

b. Note that a non-regular grammar *G* can generate a regular language *L*(*G*): for example, if we use:
   a. $P = \{ S \rightarrow \lambda | AB, A \rightarrow 0A|0, B \rightarrow 1B|1 \};$
   b. G is not regular because of $S \rightarrow AB$
   c. *But the language generated is regular*

Cont'd…

- The language generated by G is:
  - $\{ 0^n 1^m \mid n, m \geq 1\} \cup \{ \lambda \} = 00^* 11^* + \lambda$
- We *cannot* say that $L(G)$ is a regular language if and only if G is a regular grammar. Still, there must be a regular grammar that generates it by the previous theorem.
- In fact one with the following choice of productions does it:
  - $P = \{ S \rightarrow 0A|\lambda, A \rightarrow 0A|1B|1, B \rightarrow 1B|1\}$
  - The above production makes the grammar regular
- **Complement :**
  - If L(G) is regular language, its complement L'(G) will also be regular. Complement of a language can be found by subtracting strings which are in L(G) from all possible strings. For example, L(G) = $\{a^n \mid n > 3\}$, L'(G) = $\{a^n \mid n <= 3\}$

Transition diagrams

- A regular grammar G=(N, T, P, S) can be represented by a transition diagram (a directed graph with labeled arcs as follows:
  - The nodes of the graph are to contain non-terminals
  - The arcs are labeled with terminals
  - One of the nodes is an **initial node** which is designated with a pointer
  - One (or more) of the nodes is designated as **final node** which is either a square or a double circle
  - If A→aB is in P, then the arc from A to B is labeled with a
  - If A→a is in P, then the arc from A to a final state is labeled with a

Transition diagrams: cont'd

- Example:

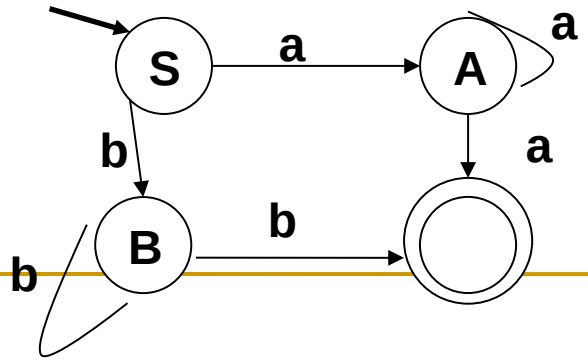
  Let G=(N, T, P, S) be a regular grammar with

  P: S→aA|bB

  A→aA|a

  B→bB|b

  Draw a transition diagram that represents G



Automata

- Abstract machines

  Characteristics
  - **Input**: input values (from an input alphabet ∑) are applied to the machine
  - **Output**: outputs of the machine
  - **States**: at any instant, the automation can be in one of the several states
  - **State relation**: the next state of the automation at any instant is determined by the present state and the present input
  - **Output relation**: output is related to either state only or to both the input and the state

Automata: cont'd

- Types of automata
  - Finite State Automata (FSA)
    - Deterministic FSA (DFSA)
    - Nondeterministic FSA (NFSA)
  - Push Down Automata (PDA)
    - Deterministic PDA (DPDA)
    - Nondeterministic PDA (NPDA)
  - Turing Machine (TM)

Deterministic FSA (DFSA)

- Definition

  A DFSA is a 5-tuple $M=(Q, \sum, t, q_o, F)$ where:

  i. Q is a finite set of states

  ii. $\sum$ is an input alphabet

  iii. t is a, possibly partial, function

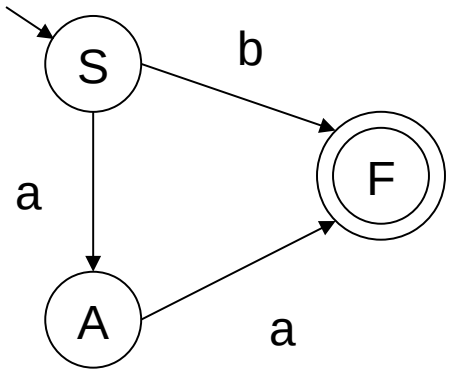
  $$t: Q X \sum \rightarrow Q$$

  iv. qo Є Q is the initial state

  v. $F \subseteq Q$ is a finite set of final states

DFSA: cont'd

- Example:



$M=(Q, \sum, t, q_o, F)$

$Q=\{S, A, F\}$

$\sum=\{a, b\}$

$Q_o=S$

$F=\{F\} \subseteq Q$

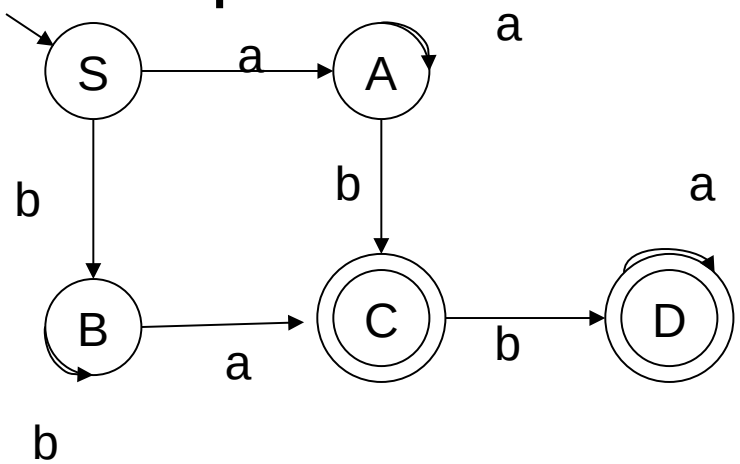
State	Input	Next state
S	a	A
S	b	F
A	a	F

DFSA: cont'd

- Acceptance of strings by DFSA
  - A string w in $\sum^*$ is accepted by a DFSA, M if
    - There exists a path which originates from some initial state, goes along the arrows, and terminates at some final state, and
    - The path value obtained by concatenation of all edge-labels of the path is equal to w.

DFSA: cont'd

- Example:



$Q = \{S, A, B, C, D\}$
$\sum = \{a, b\}$
$q_o = S$
$F = \{C, D\}$

State	Input	Next state
S	a	A
S	b	B
A	a	A
A	b	C
B	b	B
B	a	C
C	b	D
D	a	D

Check whether the following strings are accepted or not:

- **ab**
- **ba**
- **bbaba**
- **aa**
- **aaabbaaa**

DFSA: cont'd

- Extensions of t

t needs to be extended as:

$t: Q \times \Sigma^* \rightarrow Q$

let $x \in \Sigma^*$, then

i) $x = \lambda$

ii) $x = ay$, $a \in \Sigma$, $y \in \Sigma^*$

Then i) $t(q, \lambda) = q$

ii) $t(q, x) = t(q, ay)$

$= t(t(q, a), y)$

Eg. a) $t(S, ab)$

b) $t(S, bbba)$

c) $t(S, aaa)$

DFSA: cont'd

- Given DFSA, M, the set of strings accepted by M is given by L(M)

  $L(M) = \{x \varepsilon \Sigma^* \mid t(q_o,x) \varepsilon F\}$

Nondeterministic FSA (NFSA)

- Definition

  An NFSA is a 5-tuple $M=(Q, \sum, t, q_o, F)$ where:

  i. Q is a finite set of states

  ii. $\sum$ is an input alphabet

  iii. t is a, total, function

  $$t: Q X \sum \rightarrow 2^Q$$

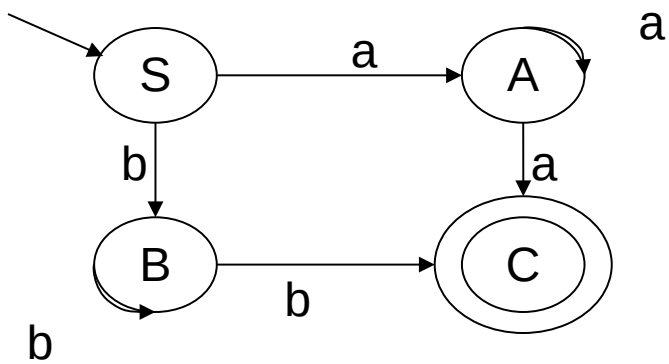
  iv. $q_o \in Q$ is the initial state

  v. $F \subseteq Q$ is a finite set of final states

NFSA: cont'd

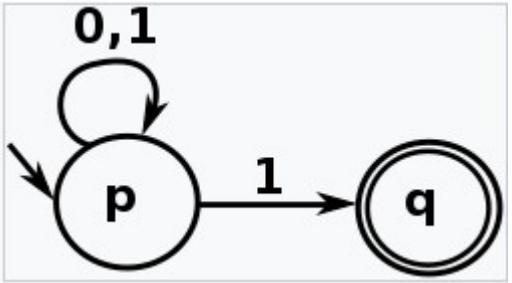
The state diagram is NFA as the inputs a to A and b to B are producing multiple next states

Example



State	Input	Next state(s)
S	a	{A}
S	b	{B}
A	a	{A, C}
A	b	{}
B	a	{}
B	b	{B, C}
C	a	{}
C	b	{}

The strings aa, bbb are both accepted by the NFSA.



The automaton *M* is not deterministic since in state *p* reading a 1 can lead to *p* or to *q*.

NFSA: cont'd

- Extensions of t:

1. t: $Q \times \Sigma \rightarrow 2^Q$ to t: $2^Q \times \Sigma \rightarrow 2^Q$ by defining

   $t(Q', a) = \cup t(q, a), q \in Q', Q' \subseteq Q, a \in \Sigma$

1. t: $2^Q \times \Sigma \rightarrow 2^Q$ to t: $2^Q \times \Sigma^* \rightarrow 2^Q$

   let $x \in \Sigma^*$, then

   i) $x = \lambda \Rightarrow t(Q',x) = Q'$

   ii) $x = ay, a \in \Sigma, y \in \Sigma^*$

   $\Rightarrow t(Q', x) = t(Q', ay) = t(t(Q', a), y)$

<u>Example</u>: Evaluate the following using the NFSA in the previous slide

a) t(S, aaa)

b) t(S, aaba)

NFSA: cont'd

- Let M be an NFSA,

  L(M) = set of strings accepted by M

  $= \{x \in \Sigma^* \mid t(q_o, x) \cap F \neq \emptyset\}$

Equivalence of DFSA and NFSA

- <u>Theorem</u>: Let L be a language. L is accepted by a DFSA iff L is accepted by NFSA.

  <u>proof</u>:

  (=>)

  Any DFSA can have a total function by introducing a dummy state such that all undefined transitions are defined to that state

  Let $L \subseteq \Sigma^*$ such that L=L(M) for some DFSA, M.

  Construct NFSA, M' such that L(M)=L(M')

  Given $M=(Q, \Sigma, t, q_o, F)$

  $M' = (Q, \Sigma, t', q_o, F)$ where $t'(q, a) = \{t(q, a)\}$

  L(M) = L(M')

Equivalence: cont'd

(<=)

Given NFSA, M, we want to construct M', DFSA, such that L(M) = L(M')

Let M = (Q, ∑, t, q_o, F), to construct M':

1. Collect all sets such that t(q_o, a) = Q', for all a∈∑
2. Collect all sets such that t(Q', a) = Q'', for all a∈∑
3. Repeat the process until no new subsets of Q are constructed

Thus, M' = (Q*, ∑, t*, q_o', F') where

Q* = all subsets of Q reachable from q_o

q_o' = {q_o}

F' = Q' C Q such that Q' n F ≠ Ø

t' is an extension of t such that t: 2^Q X Σ → 2^Q

L(M) = L(M')

<u>Illustration</u>:

Equivalence: cont'd

- <u>Theorem</u>: The following statements are equivalent:
  i. L is accepted by NFSA
  ii. L is accepted by DFSA
  iii. L is generated by a regular grammar

  Proof: (i) => (ii) => (iii) => (i)

  (i) => (ii) proved in the previous theorem.

  (ii) => (iii)

  Let L be a language accepted by DFSA, M. Construct a regular grammar G such that L(M) = L(G)

  Let M = (Q, ∑, t, q_o, F)

  Construct of G = (N, T, P, S) as follows:

  1. N = Q, T = ∑, S = q_o
  2. P:

     qi → aqj, if t(qi, a) = qj

     qi → a, if t(qi, a) Є F

  L(M) = L(G)

Equivalence: cont'd

(iii) => (i)

Let L be a language generated by regular grammar G.

We want to construct NFSA M such that L = L(M) = L(G)

Let G = (N, T, P, S)

Construct M = (Q, ∑, t, q_o, F) as follows:

1. Q = N, ∑ = T, q_o= S, F = {x∈N| x → a ∈ P, a ∈ T}

   Q = N U {F_f}, F_f not in N

2. t is given by:

   if q_i → aq_j, then t({q_i}, a) n {q_j} ≠Ø

   q_i → a, then t({q_i}, a) n F ≠Ø

   L(M) = L(G)

<u>Illustration</u>:

Equivalence: cont'd

- <u>Theorem</u>: If L is a regular language, then so is L bar (L').

  <u>proof</u>:

  L is regular → there exists a DFSA M = (Q, ∑, t, q_o, F) such that L = L(M)

  x∈L → t(q_o, x) ∈ F

  x ∈L' → x is not in L → t(q_o, x) is not in F

  Hence, M' = (Q, ∑, t, q_o, Q\F) such that L' = L(M')

Equivalence: cont'd

- <u>Theorem</u>: If L_1 and L_2 are regular languages, then so is L_1 n L_2.

  <u>proof</u>:

  L_1 is regular → L_1' is regular

  L_2 is regular → L_2' is regular

  → L_1' U L_2' is regular

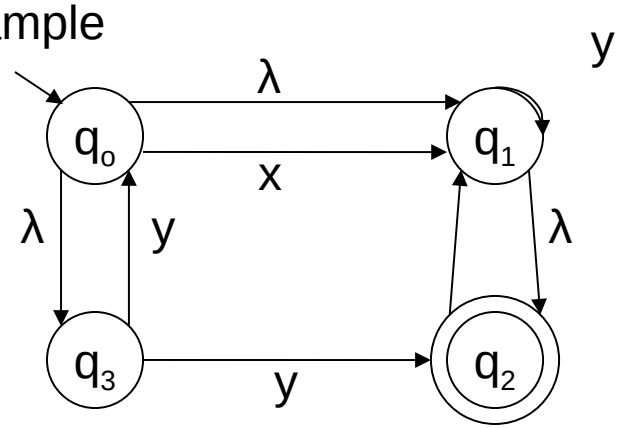
  → (L_1' U L_2')' is regular

  → L_1 n L_2 is regular

FSA with λ moves

- Definition: Let $M = (Q, \sum, t, q_o, F)$ be a FSA, M is said to be with λ-moves if $t: Q X (\sum \cup \{\lambda\}) \rightarrow Q$

Example

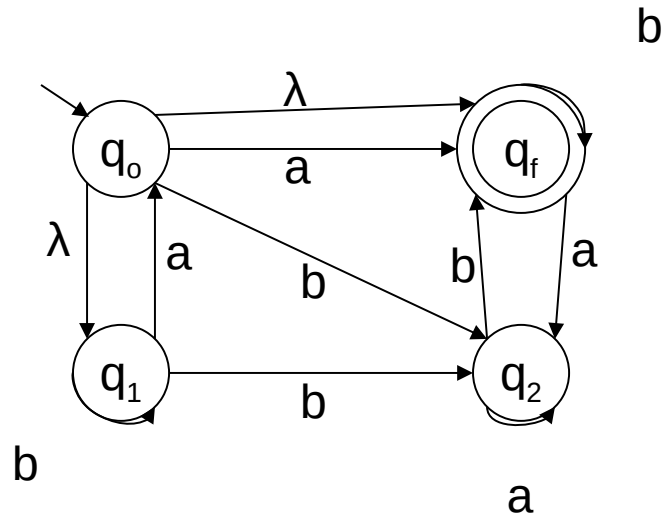


R(q) = the set of λ-reachable states
For ex. $R(q_o) = \{q_o, q_1, q_2, q_3\}$

If $t(q_i, \lambda) = q_j$, then q_j is λ-reachable from q_i

For $Q' \subseteq Q$, $R(Q') = \cup R(q')$, $q' \in Q'$

FSA with λ moves: cont'd



$Q = \{q_0, q_1, q_2, q_f\}$

$\Sigma = \{a, b, \lambda\}$

Let q, q' Є Q. If q' Є t(q, λ), then q' is λ-reachable denoted by $q \overset{*}{\underset{\lambda}{\rightarrow}} q'$

Let R(q) be the set of states that are reachable, $R(q) = \{q' \mid q \rightarrow q'\}$

If $Q' \subseteq Q$ then $R(Q') = \cup R(q')$, q Є Q'

FSA with λ moves: cont'd

- Example: consider the previous NFSA

  R(qo) = ?

  R(q1) = ? R(q2) = ? R(q3) = ? R(qf) = ?

  R({qo, q1, q2, qf}) = ?

String acceptance by FSA with λ moves

- Extend t to t' such that

  $t': Q \times (\sum \cup \{\lambda\}) \rightarrow 2^Q$ defined as

  $t'(q, \lambda) = R(q)$

  $t'(q, a) = \cup R(t(q', a)), q' \in R(q), a \in \sum \cup \{\lambda\}$

  <u>Ex</u>. consider the previous NFSA

  $t'(q_o, \lambda) = ?$

  $t'(q_o, a) = ?$

  $t'(q_o, b) = ?$

- Extend t' such that $t': 2^Q \times (\sum \cup \{\lambda\}) \rightarrow 2^Q$ defined as

  $t'(Q', \lambda) = R(Q'), Q' \subseteq Q$

  $t'(Q', a) = \cup t'(q', a), q' \in Q', a \in \sum$

String acceptance: cont'd

- Extend t' such that t': $2^Q \times \Sigma^* \rightarrow 2^Q$ defined as

  $t'(Q', ax) = t'(t'(Q', a), x), a \in \Sigma, x \in \Sigma^*$

  Thus, $L(M) = \{x \in \Sigma^* | t'(qo, x) \cap F \neq \emptyset\}$

  = set of strings accepted by NFSA with λ-moves

Cont'd

- <u>Theorem</u>: If L is accepted by NFSA with λ-moves, then L is a regular language.

  <u>proof</u>: It suffices to construct an equivalent NFSA without λ-moves

  Let $M = (Q, \sum, t, q_o, F)$ be NFSA with λ-moves

  To construct $M' = (Q, \sum, t', q_o, F')$, define

  $t'(q, a) = t(q, a)$, for all $a \in \sum$ and t is the 3rd extension

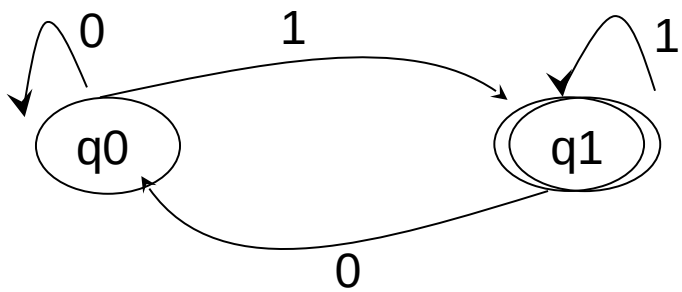
  $F' = F$ if $R(q_o) \cap F = \emptyset$

  $= F \cup \{q_o\}$ otherwise

  $L(M) = L(M')$

Exercises

- Given the formal description of finite state automaton M_2 below, draw a corresponding state diagram for M_2. $M_2 = (Q, \Sigma, P, q_0, F)$, where Q={q0, q1}, Σ = {0, 1}, P is defined as: $q_0 \rightarrow 0q_0|1q_1$, $q_1 \rightarrow 1q_1|0q_0$
- q_0 is the start state and q_1 is the final state, F= $\{q_1\}$

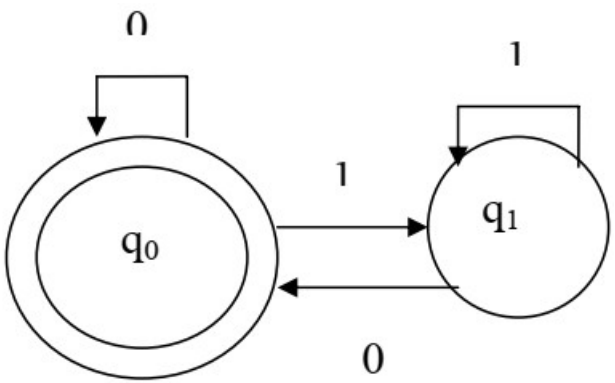


- Which of the following strings are accepted by M_2?

a. 0 b. 1 c. 00 d. 11111 e. 1000000 f. 1010011 g. λ

… Exercises

- Consider the state diagram of finite state automaton M_3, and give a formal description of M_3.
- $M_3 = <Q, \Sigma, P, q_0, F>$, where
- $q_0 = \{q_0\}$, $F = \{q_0\}$, $\Sigma = \{0, 1\}$, $Q = \{q_0, q_1\}$



- Which of the following strings are accepted by M_2?

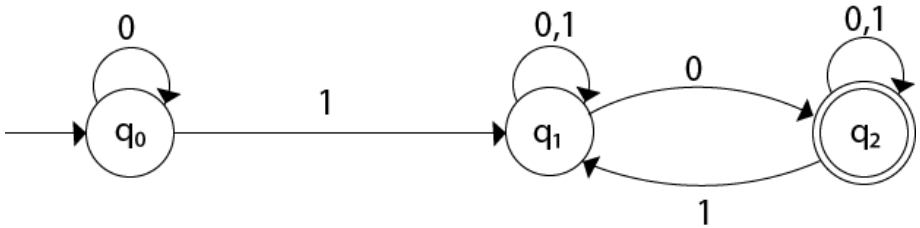
a. 0 b. 1 c. 00 d. 11111 e. 1000000 f. 1010011 g. λ

Conversion from NFA to DFA

- In NFA, when a specific input is given to the current state, the machine goes to multiple states.
- It can have zero, one or more than one move on a given input symbol
- Steps for converting NFA to DFA:
  - **Step 1:** Initially Q' = ϕ
  - **Step 2:** Add q0 of NFA to Q'. Then find the transitions from this start state.
  - **Step 3:** In Q', find the possible set of states for each input symbol. If this set of states is not in Q', then add it to Q'.
  - **Step 4:** In DFA, the final state will be all the states which contain F(final states of NFA)

… Conversion from NFA to DFA

- **Example 1**: Convert the given NFA to DFA



- Soln.
- For the given transition diagram we will first construct the transition table.

State	0	1
→q0	q0	q1
q1	{q1, q2}	q1
*q2	q2	{q1, q2}

- δ' transition for state q0:
  - δ'([q0], 0) = [q0]
  - δ'([q0], 1) = [q1]
- ' transition for state q1
  - δ'([q1], 0) = [q1, q2] (**new** state
  - δ'([q1], 1) = [q1]

… Conversion from NFA to DFA

- δ' transition for state q2:

  $\delta'([q2], 0) = [q2]$

  $\delta'([q2], 1) = [q1, q2]$

- δ' transition on [q1, q2]:

  $$\begin{aligned}\delta'([q1, q2], 0) &= \delta(q1, 0) \cup \delta(q2, 0)\\ &= \{q1, q2\} \cup \{q2\}\\ &= [q1, q2]\end{aligned}$$

  $$\begin{aligned}\delta'([q1, q2], 1) &= \delta(q1, 1) \cup \delta(q2, 1)\\ &= \{q1\} \cup \{q1, q2\}\\ &= \{q1, q2\}\\ &= [q1, q2]\end{aligned}$$

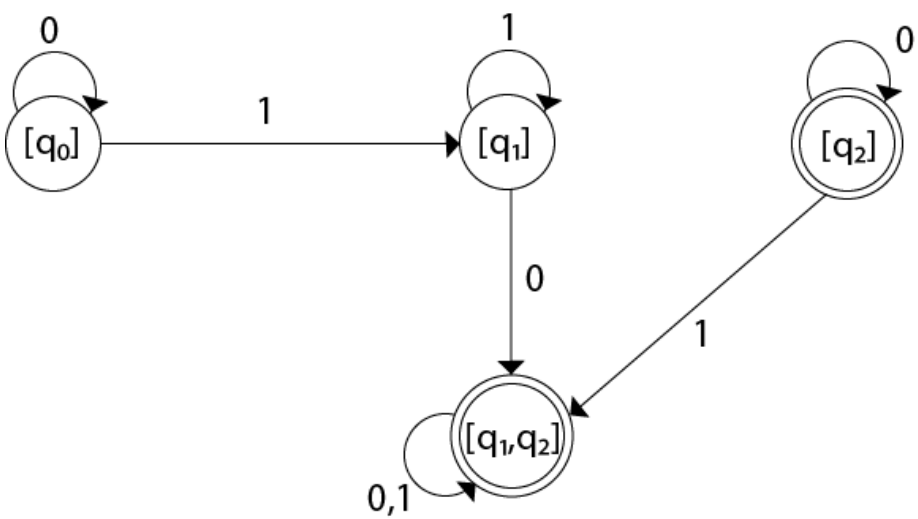
- The state [q1, q2] is the final state as well because it contains a final state q2

…Cont'd

- The transition table for the constructed DFA will be:

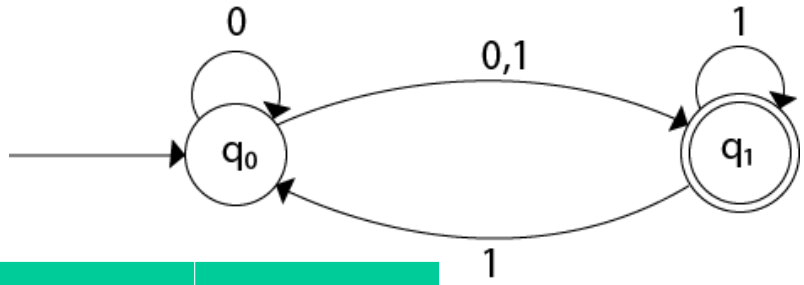
State	0	1
→[q0]	[q0]	[q1]
[q1]	[q1, q2]	[q1]
*[q2]	[q2]	[q1, q2]
*[q1, q2]	[q1, q2]	[q1, q2]

- … eliminated because q2 is an unreachable state.



… Conversion from NFA to DFA

- Example 2: Convert the given NFA to DFA.



- Constructing transition table for the given NFA

State	0	1
→ q0	{q0, q1}	{q1}
*q1	ϕ	{q0, q1}

- δ' transition for state q0.

δ'([q0], 0) = {q0, q1}

= [q0, q1] (**new** state generated)

δ'([q0], 1) = {q1} = [q1]

Cont'd...

- δ' transition for state q1 is obtained as:

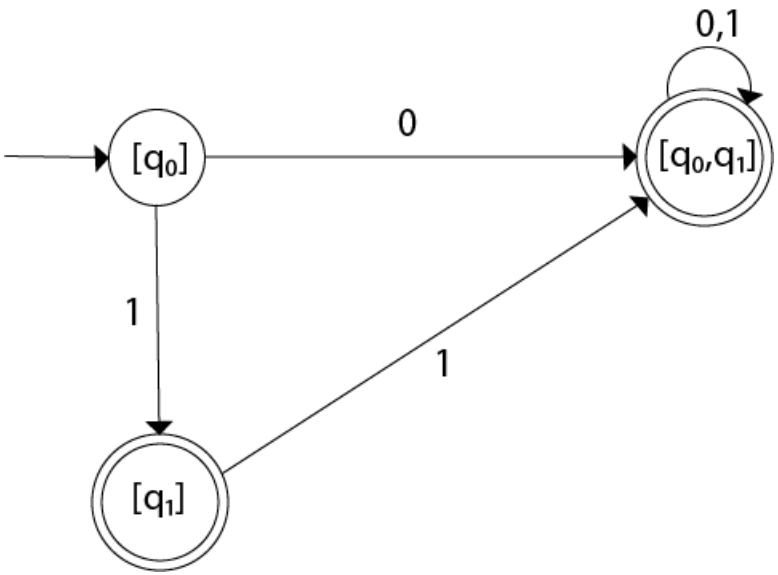
  $\delta'([q1], 0) = \phi$

  $\delta'([q1], 1) = [q0, q1]$

- Obtain δ' transition on [q0, q1]:

$$
\begin{aligned}
\delta'([q0, q1], 0) &= \delta(q0, 0) \cup \delta(q1, 0) \\
&= \{q0, q1\} \cup \phi \\
&= \{q0, q1\} \\
&= [q0, q1] \\
\delta'([q0, q1], 1) &= \delta(q0, 1) \cup \delta(q1, 1) \\
&= \{q1\} \cup \{q0, q1\} \\
&= \{q0, q1\} \\
&= [q0, q1]
\end{aligned}
$$

- The transition diagram will be:



NFA to DFA

- Example 3: Convert the following NFA to DFA

State	a	b
q0	q0, q1	q0
q1	ϕ	q2
q2	ϕ	ϕ

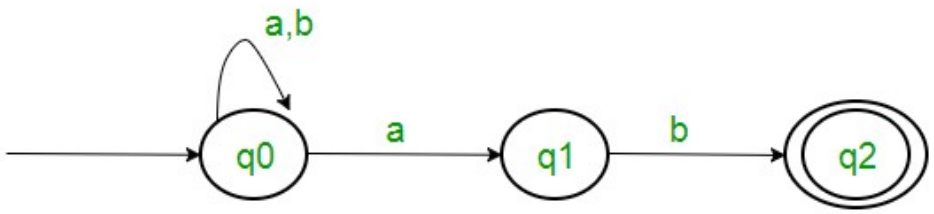


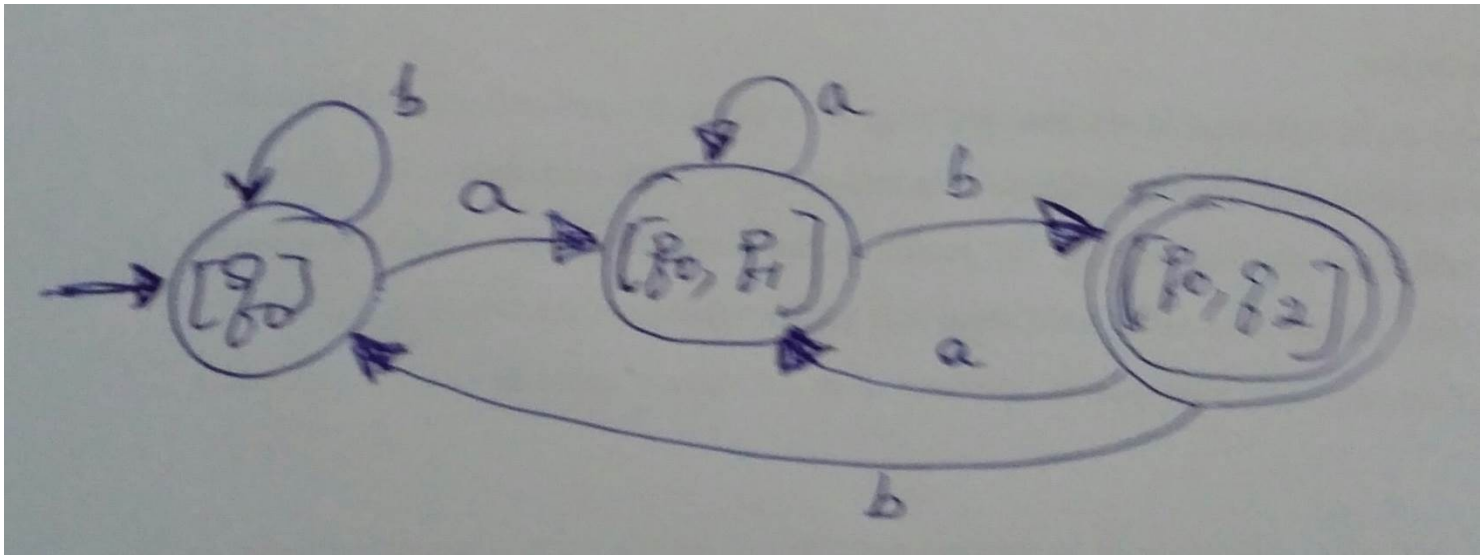
Figure 1

- δ'([q0, q1], a) = δ(q0, a) ∪ δ(q1, a)=[q0, q1]
- δ'([q0, q1], b) =δ(q0, b) ∪ δ(q1, b) =[q0, q2], new state

- δ'([q0, q2], a)= δ(q0, a) ∪ δ(q2, a)=[q0, q1]
- δ'([q0, q2], b)= δ(q0, b) ∪ δ(q2, b)=[q0]

… Cont'd

- The corresponding DFA



Regular Expressions

- Definition: A regular expression is a string over $\sum$ if the following conditions hold:
  1. λ, $\emptyset$, and $a \in \sum$ are regular expressions
  2. If α and β are regular expressions, so is $\alpha\beta$
  3. If α and β are regular expressions, so is $\alpha+\beta$
  4. If α is a regular expression, so is α^*
  5. Nothing else is a regular expression if it doesn't follow from (1) to (4)
- Let α be a regular expression, the language represented by α is denoted by $L(\alpha)$.

Regular Expressions: cont'd

- L satisfies:
  1. $L(\emptyset) = \emptyset$, $L(\lambda) = \{\lambda\}$, $L(a) = \{a\}$, $a \in \Sigma$
  2. If α and β are regular expressions, then:
     - $L(\alpha\beta) = L(\alpha)L(\beta)$
     - $L(\alpha+\beta) = L(\alpha)+L(\beta)$
     - $L(\alpha^*) = L(\alpha)^*$

- Example:

  $\alpha = a^*(b+c)$

  $\rightarrow L(\alpha) = L(a^*(b+c))$

  $= L(a^*)L(b+c)$

  $= L(a)^*(L(b) \cup L(c))$

  $= \{a\}^*(\{b\} \cup \{c\})$

  $= \{a\}^*(\{b,c\})$

<u>Note</u>: In the absence of parentheses, the hierarchy of operations is as follows: iteration, concatenation, and union.

Regular Expressions: cont'd

- Two regular expression P and Q are equivalent ($P \approx Q$) if P and Q represent the same set of strings.
- Identities for regular expressions
  - $\emptyset + R = R$
  - $\emptyset R = R\emptyset = R$
  - $\lambda^* = \lambda$, $\emptyset^* = \lambda$
  - $R + R = R$
  - $R^*R^* = R^*$
  - $RR^* = R^*R$
  - $(R^*)^* = R^*$
  - $\lambda + RR^* = R^* = \lambda + R^*R$
  - $(PQ)^*P = P(QP)^*$
  - $(P+Q)^* = (P^*Q^*)^* = (P^*+Q^*)^*$
  - $(P+Q)R = PR + QR$, $R(P+Q) = RP + RQ$

Regular Expressions: cont'd

- **Example**: Give a regular expression for representing the set L of strings in which every 0 is immediately followed by at least two 1's.
  - **Soln**. (1*0111*)*
- Theorem: If r is a regular expression, then L(r) is a regular language.

  proof: construct NFSA with λ-moves that accepts L(r).

  i. r = Ø, r = λ, r = a
  ii. r1 + r2
  iii. r1r2
  iv. r1*

  Let r be a regular expression, M(r) is a NFSA with λ-moves that accepts L(r)

Exercises

- **Question 1 :**
- Which one of the following languages over the alphabet {0,1} is described by the regular expression? (0+1)*0(0+1)*0(0+1)*

  (A) The set of all strings containing the substring 00.

  (B) The set of all strings containing at most two 0's.

  (C) The set of all strings containing at least two 0's.

  (D) The set of all strings that begin and end with either 0 or 1.

  Ans. C.
- **Question 2 :**
- Which of the following languages is generated by given grammar? S -> aS | bS | λ

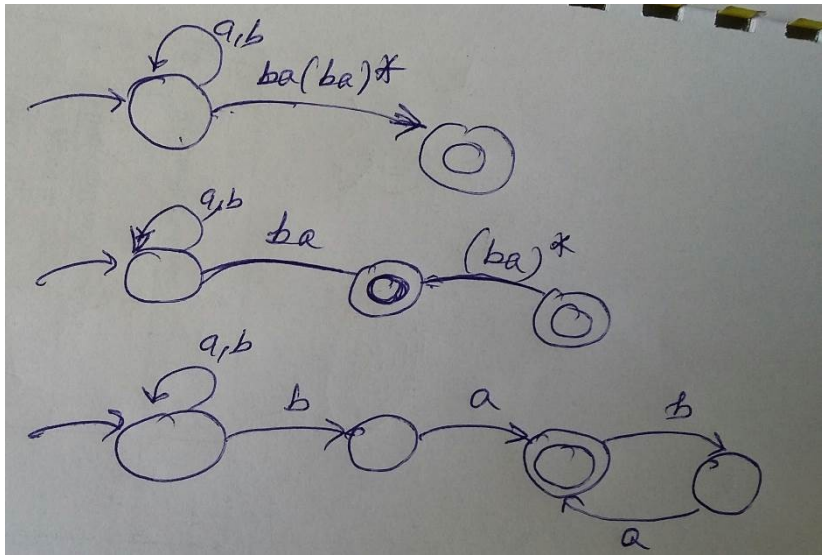
  (A) $\{a^n b^m \mid n,m \geq 0\}$ (B) $\{w \in \{a,b\}^* \mid$ w has equal number of a's and b's$\}$

Exer…

- (C) $\{a^n | n \geq 0\} \cup \{b^n | n \geq 0\} \cup \{a^n b^n | n \geq 0\}$ (D) $\{a,b\}^*$
- **<u>Solution</u>**: Option (A) says that it will have 0 or more a followed by 0 or more b. But S -> bS => baS => ba is also a part of language. So (A) is not correct. Option (B) says that it will have equal no. of a's and b's. But But S -> bS => b is also a part of language. So (B) is not correct. Option (C) says either it will have 0 or more a's or 0 or more b's or a's followed by b's. But as shown in option (A), ba is also part of language. So (C) is not correct. Option (D) says it can have any number of a's and any numbers of b's in any order. So (D) is correct.

Regular Expressions: cont'd

- <u>Example</u>: Construct NFSA that accepts r=(a+b)*ba(ba)*



- <u>Exercise</u>: Construct NFSAs equivalent to the following regular expressions:
  1. (1+0)*(00+11)(0+1)*
  2. 10+(0+11)0*1

Regular Expressions: cont'd

- Algebraic method for finding the regular expression recognized by a FSA

  <u>Assumptions</u>

  1. No λ-moves
  2. There is only one initial state, say v_1
  3. Vertices are $v_1, \ldots, v_n$
  4. V_i is the regular expression representing the set of strings accepted by the system
  5. α_{ij} denotes the regular expression representing the set of labels of edges from v_i to v_j

  Consequently, we can get the following set of equations in $v_1, \ldots, v_n$

  $v_1 = v_1\alpha_{11} + v_2\alpha_{21} + \ldots + v_n\alpha_{n1} + \lambda$

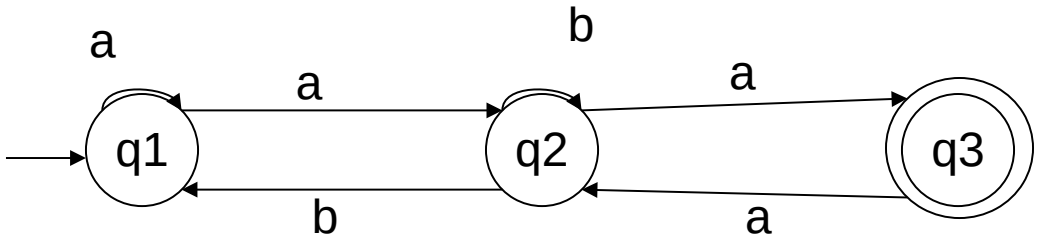
  $v_2 = v_1\alpha_{12} + v_2\alpha_{22} + \ldots + v_n\alpha_{n2}$

  …

  $v_n = v_1\alpha_{1n} + v_2\alpha_{2n} + \ldots + v_n\alpha_{nn}$

Regular Expressions: cont'd

- By repeatedly applying substitution and Arden's Theorem, we can express v_i in terms of α_{ij}'s
- The set of strings recognized by the system is found by taking the 'union' of all v_i's corresponding to the final states.
- <u>Example</u>:



Find the regular expression equivalent to the given NFSA

Regular Expressions: cont'd

- <u>Exercise</u>: Find the regular expressions equivalent to the following NFSAs

Minimization of DFSA

- A DFSA with possible minimum states is called a minimal DFSA.
- Given M, DFSA that generates a language L, then we denote its minimal DFSA by M_L.

Construction of Minimal DFSA

Let $M = (Q, \sum, t, q_o, F)$ be DFSA such that $L = L(M)$

To construct M_L, we proceed as follows:

I. Find all disjoint indistinguishable states of M

Define a number of relations D_o, D_1, …

- Given states q and q', q is distinguishable from q', by a string of length 0, denoted by

qDoq' iff either:

q∈F and q' not ∈F OR

q not ∈F and q'∈F

- If i>0, q is distinguishable from q' by a string of length ≤ i, denoted by

q D_i q' iff

$qD_{i-1}q'$ OR

if there exists a∈∑ such that $t(q, a)D_{i-1}t(q', a)$

Minimization of DFSA: cont'd

II. $M_L = (Q_L, \sum, t_L, q_{oL}, F_L)$ where

Q_L= indistinguishable states

q_{oL}= indistinguishable states containing q_o

F_L = Final states of M

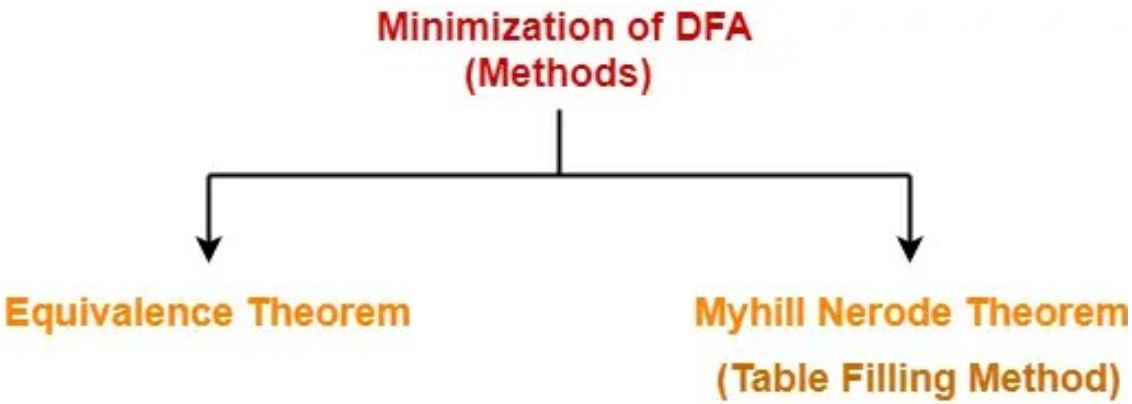
t_L is given by the following:

Let $Q' \subseteq Q$, where Q' is indistinguishable states

tL(Q', a) = Q'', indistinguishable states containing t(q, a), $q \in Q'$

Minimization of DFA

- The process of reducing a given DFA to its minimal form is called as minimization of DFA.
- It contains the minimum number of states.
- The DFA in its minimal form is called as a **Minimal DFA**.
- How to minimize DFA?
- The two popular methods for minimizing a DFA are-



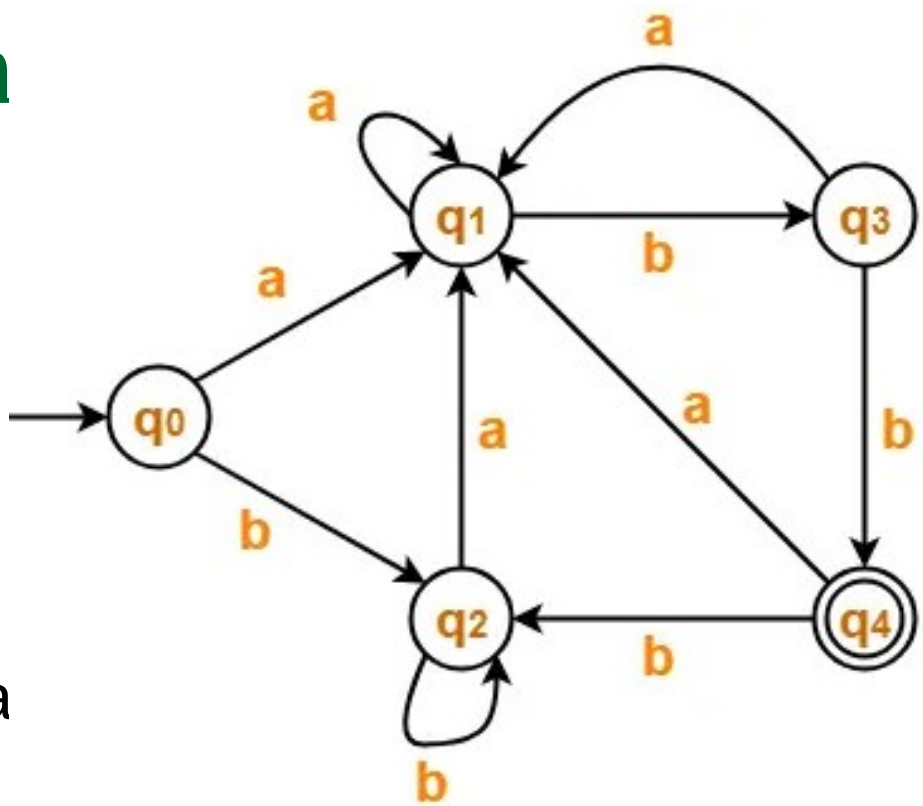
- We will discuss it using Equivalence theorem

…Using the Equivalence Theorem-

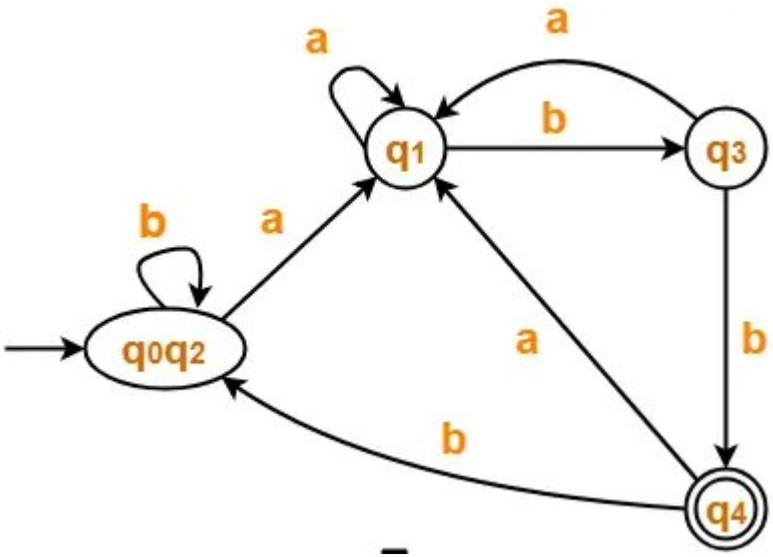
- Eliminate all the dead states and inaccessible states from the given DFA (if any).
  - **Dead state**: All those non-final states which transit to itself for all input symbols in ∑ are called as dead states.
  - **Inaccessible/Non-reachable state**: All those states which can never be reached from the initial state are called as inaccessible states.
- Divide Q (set of states) into two sets such that one set contains all the non-final states and other set contains all the final states.
- Two states q_1 and q_2 are distinguishable in partition P_k for any input symbol 'a', if $\delta(q_1, a)$ and $\delta(q_2, a)$ are in different sets
- All those states which belong to the same set are equivalent.
- The equivalent states are merged to form a single state in the minimal DFA.

Practice problem

- Minimize the given DFA-

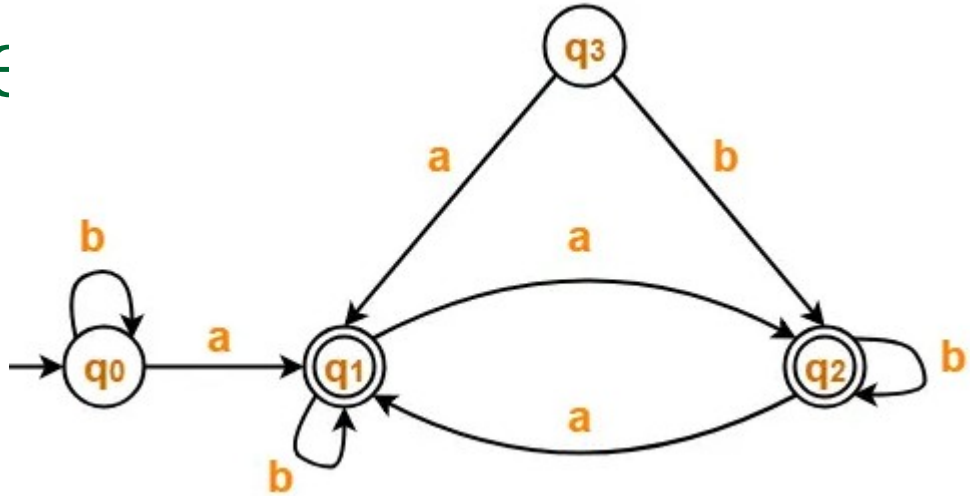


- **Soln**:
- The given DFA contains no dead states and inaccessible states.
- states q_0 and q_2 are equiva together
  - Minimal dfa

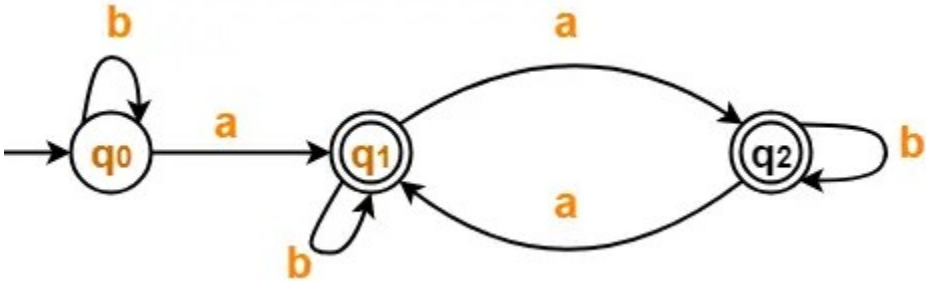


… Practice proble

- Minimize the given DFA-

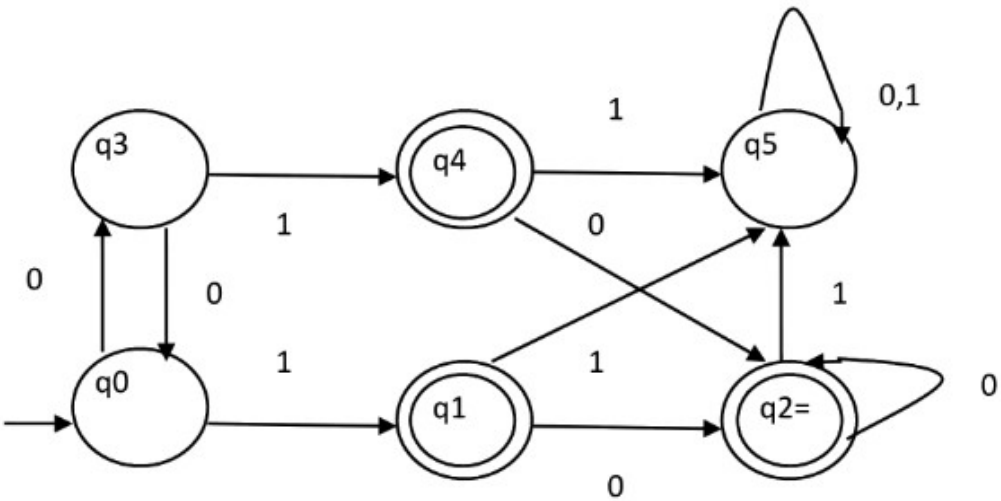


- Soln.
- State q_3 is inaccessible from the initial state.
- So, we eliminate it and its associated edges from the DFA.
- The resulting dfa is:

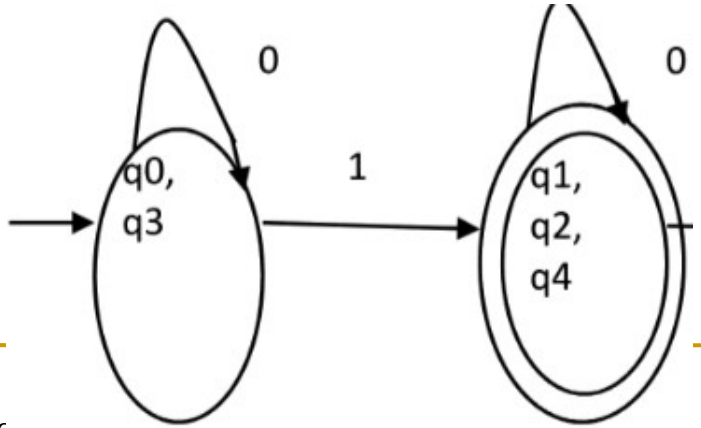


- States q_1 and q_2 are equivalent and can be merged together.
- So, Our minimal DFA is-

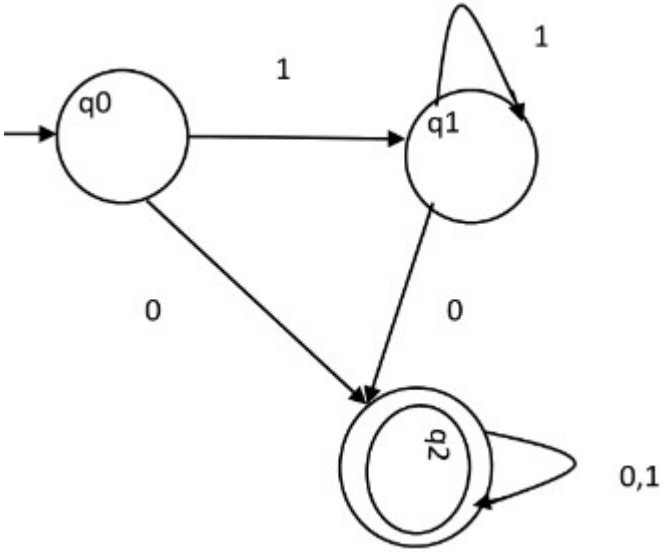
- Optimize the given dfa:
- Two partitions:
- {q1, q2, q4}, {q0, q3, q5}
- Q5 is inaccessible



- q1, q2, q4 are indistinguishable
- q0 and q3 are indistinguishable
- Merge each of the indistinguishable groups

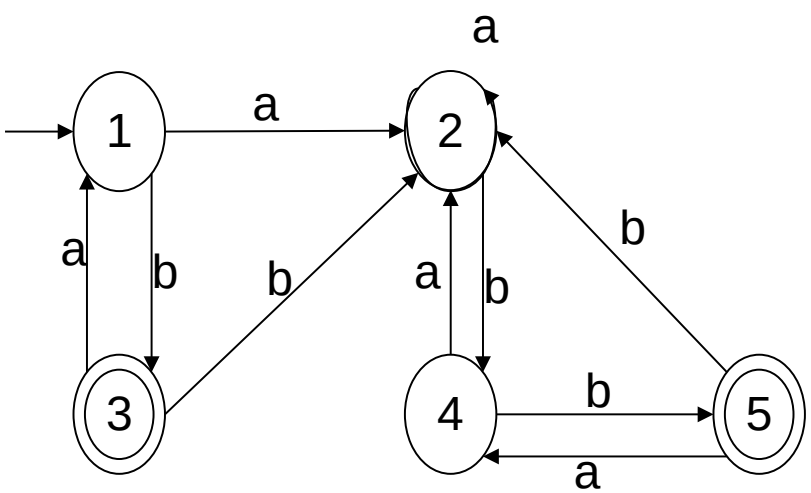


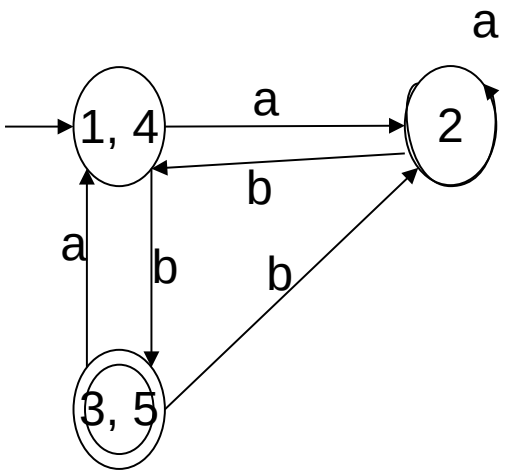
- Which statement is true about the dfa given?
  a. For the language accepted by A, A is the minimal DFA.
  b. A accepts all strings over { 0, 1 } of length at least two.
  c. The dfa accepts strings of the form 1*0(0+1)*



Minimization of DFSA: cont'd

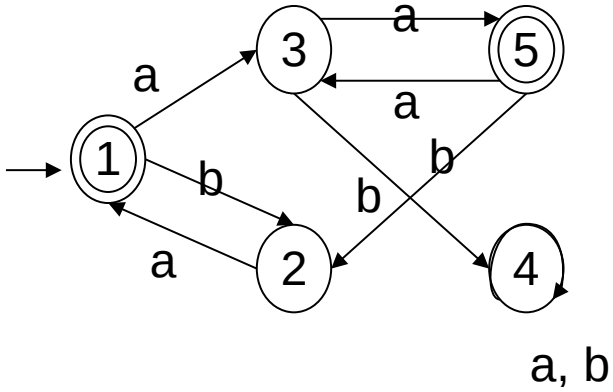
- <u>Illustration</u>: Construct a minimal DFSA for the following DFSA





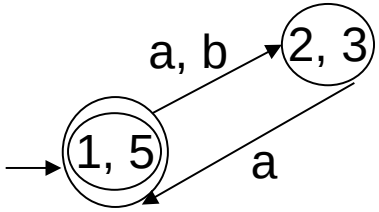
Minimization of DFSA: cont'd

- <u>Exercise</u>: Minimize the following DFSA



- 4 is inaccessible or dead state, so remove
- Partition: {1, 5}, {2, 3}
- Merge 1, 5 and 2, 3

- The dfa accepts strings of the form ((a + b)a)*



Identification of non-regular languages

- <u>Lemma</u> (Pumping lemma for regular languages)

  Let L be a regular language and $w \in L$. Then there exist substrings x, y, and z of w with:

  a. $w = xyz$, $|w| \geq m$
  b. $|xy| \leq m$, for some $m \in Z^+$
  c. $y \neq \lambda$

  such that $xy^nz \in L$ for $n \geq 0$

Non-regular languages: cont'd

- <u>Example</u>: Show that the following languages are not regular.

1. $L = \{a^n b^n \mid n \geq 0\}$
2. $L = \{a^n : n \text{ is prime}\}$
3. $L = \{ww^r \mid w \text{ in } \sum^*\}$, $\sum=\{a, b\}$

Soln

1. $G=(Q, \{a, b\}, P, S)$, where P: $S \rightarrow aSb \mid \lambda$, is not Regular

3. $G=(Q, \{a, b\}, P, S)$, where P: $S \rightarrow aSa \mid bSb \mid a \mid b \mid \lambda$ is not regular