Chapter II: Finite Automata

Some Terminologies (*Chapter I, Revisited*):

- **Alphabet**: It is the finite set of symbols and is represented by Σ
  $\Sigma = \{ s_1, s_2, s_3, \ldots s_n \}$
  Eg. For a binary string, $\Sigma = \{0,1\}$
- **String**. It is finite Sequence of Symbols, usually denoted by ω.
  $\omega = \alpha_1\alpha_2\alpha_3 \ldots \alpha_n$; $\alpha_i \in \Sigma$, $i \in \{1,2,3,\ldots,n\}$
  Eg. If $\Sigma = \{0,1\}$ then string, ω, can be 0, 1, 01, 011, 111, 101 1111 etc.
  - **Concatenation of strings** ω_1 and ω_2 is $\omega_1\omega_2$
    Eg. Concatenation of strings "0110" and "1100" is "01101100"
  - ω^x means the string ω is repeated x times
    Eg. if ω = "01" then ω^3 = "010101"
  - **Reversal of String** ω, denoted as ω^R, is the sequence of symbols in ω in reversed order.
    Eg. if ω = "01001" then ω^R = "10010"
  - **Kleene Star of String** ω, denoted as ω^*, is the set of strings ω^i, where $i \in \{0, 1, 2, \ldots\}$
    Eg. if ω = "010" then ω^* = {*e*, 010, 010010, 010010010, ...}
    – Here, *e* represents empty string
- **Language**: Language is the set of Strings with some property.
  i.e. It is the subset of Σ^*. [Σ^* is the set of all the strings formed by any sequence of any number of alphabets]
  $L = \{\omega : \omega \in \Sigma^*, \omega \text{ has some property}\}$
  eg. If Language, $L = \{\omega : \omega \in \{0,1\}^*, \omega \text{ contains the substring } 101\}$
  Then string 01010 $\in$ L where as 0010 $\notin$ L

Deterministic Finite State Automata (DFA)

- A Deterministic Finite State Automaton, D, is a quin-tuple (5-tuple) defines as:

  $D = (Q, \Sigma, \delta, q_0, F)$

  Where, Q is finite, non-empty set of states
  Σ is finite set of alphabets
  $\delta : Q \times \Sigma \rightarrow Q$ is transition function
  q_0 is initial state, $q_0 \in Q$
  F is set of final or accepting states, $F \subseteq Q$

- The **Configuration of DFA** $(Q, \Sigma, \delta, q_0, F)$ is $(q_i, \omega) \in Q \times \Sigma^*$ and is determined by the present state 'q_i' and input 'ω' being processed.

- A Binary Relation, represented by $\vdash_D$, exists between configurations if D can pass from one configuration to another in a single move. i.e. if (q, ω) and (q', ω') are two configurations in D then
  $(q, \omega) \vdash_D (q', \omega')$ if and only if $\omega = a\omega$ for $a \in \Sigma$ and $\delta(q, a) = q'$

- The reflexive transitive closure of $\vdash_D$ is written as $\vdash_D^*$. $(q, \omega) \vdash_D^* (q', \omega')$ means that starting from the configuration (q, ω) of D, configuration (q', ω') can be reached after zero or more number of moves.

- A string, $\omega \in \Sigma^*$ is said to be **accepted by DFA** $(Q, \Sigma, \delta, q_0, F)$ if
  $(q_0, \omega) \vdash_D^* (q_F, e)$; $q_F \in F$
  i.e. starting with initial state with string ω as the input, the configuration transition(s) in zero or more steps leads DFA to Final/Accepting State when last symbol of string is processed.

- The set of strings accepted by DFA is called the **Language, L(D), accepted by DFA.**

- **State Transition Diagram:** It consists of nodes and edges. States are represented by nodes and transition between states is represented by edge. Starting node is represented by $\rightarrow(q_0)$, any other state by (q_i) and the Final state is represented by $((q_f))$. The label on the edge represents input symbol causing state transition.

Example:

#1. Construct a DFA that accepts the Language
$L(D) = \{ \omega \in \{a, b\}^* : \omega \text{ contains at-least one a} \}$
Also Show Configuration Transition for string "bbaa". Does it belongs to L(D)?

Solution:

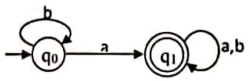


Fig: State Transition Diagram for D

The Required DFA is
$D = (Q, \Sigma, \delta, q_0, F)$
Where, $Q = \{q_0, q_1\}$ is set of states
$\Sigma = \{a, b\}$ is set of alphabets (input symbols)
$\delta : Q \times \Sigma \rightarrow Q$ is transition function defined by following transition table

	δ	
Q \ Σ	a	b
q_0	q_1	q_0
q_1	q_1	q_1

q_0 is the initial state
$F = \{q_1\}$ is the set of Final/Accepting States

Configuration Transition for string "bbaa" :
$(q_0, bbaa) \vdash_D (q_0, baa) \vdash_D (q_0, aa) \vdash_D (q_1, a) \vdash_D (q_1, e)$ i.e. $(q_0, bbaa) \vdash_D^* (q_1, e)$
$q_1 \in F$, Hence, string "bbaa" $\in$ L(D)

#2. Construct a DFA that accepts the Language
$L(D) = \{ \omega \in \{a, b\}^* : \omega \text{ contains substring "abaab"} \}$
Also Show Configuration Transition for string "abab". Does it belongs to L(D)?

Solution:

Fig: State Transition Diagram for D

The Required DFA is

$D = (Q, \sum, \delta, q_0, F)$

Where, $Q = \{q_0, q_1, q_2, q_3, q_4, q_5\}$ is set of states

$\sum = \{a, b\}$ is set of alphabets (input symbols)

$\delta : Q \times \sum \rightarrow Q$ is transition function defined by following transition table

	δ	
Q \ Σ	a	b
q_0	q_1	q_0
q_1	q_1	q_2
q_2	q_3	q_0
q_3	q_4	q_2
q_4	q_1	q_5
q_5	q_5	q_5

q_0 is the initial state

$F = \{q_5\}$ is the set of Final States

Configuration Transition for string "abab" :

$(q_0, abab) \vdash_D (q_1, bab) \vdash_D (q_2, ab) \vdash_D (q_3, b) \vdash_D (q_2, e)$ i.e. $(q_0, abab) \vdash_D^* (q_2, e)$

$q_2 \notin F$, Hence, string "abab" $\notin L(D)$

Few Notes on DFA:

- A DFA must have finite number of states
- From every states of a DFA, for each symbol in alphabet set, there must be one and only one outgoing arrow. i.e the transition of states for a symbol must be deterministic.
- There must be one and only one initial state.
- There can be zero or more final states.
- DFA with no final state represents the Language, $L = \phi$. Such a DFA do not accepts any string.

Non-Deterministic Finite State Automata (NFA)

A Non-Deterministic Finite State Automaton, N, is a quin-tuple (5-tuple) defines as:

$N = (Q, \sum, \delta, q_0, F)$

Where, Q is finite, non-empty set of states

$\sum$ is finite set of alphabets

$\delta : Q \times (\sum \cup \{e\}) \rightarrow P(Q)$ is transition function ; [P(Q) is Power Set of Q]

q_0 is initial state, $q_0 \in Q$

F is set of final or accepting states, $F \subseteq Q$

Note that, in contrast to DFA, an NFA can have no or more than one possible state transitions or same symbol read. The '*e*-move' (transition from one state to another without reading any input string) is also possible in a Non-Deterministic Finite State Automaton. A string is said to be accepted by NFA if there exist at-least one possible path from Initial State to Accepting State for that string.

Example:

Construct a NFA that accepts the Language

$L(D) = \{ \omega \in \{a, b\}^* : \omega$ contains substring "abaab" $\}$

Solution:

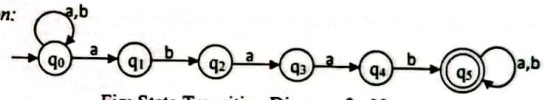


Fig: State Transition Diagram for N

The Required NFA is

$N = (Q, \sum, \delta, q_0, F)$

Where, $Q = \{q_0, q_1, q_2, q_3, q_4, q_5\}$ is set of states

$\sum = \{a, b\}$ is set of alphabets (input symbols)

$\delta: Q \times (\sum \cup \{e\}) \rightarrow P(Q)$ is transition function defined by following transition table

	δ	
Q \ Σ	a	b
q_0	$\{q_0, q_1\}$	$\{q_0\}$
q_1	Φ	$\{q_2\}$
q_2	$\{q_3\}$	Φ
q_3	$\{q_4\}$	Φ
q_4	Φ	$\{q_5\}$
q_5	$\{q_5\}$	$\{q_5\}$

q_0 is the initial state

$F = \{q_5\}$ is the set of Final States

Here, clearly, any string containing "abaab" will be able to reach the final state 'q_5' as any symbol before "aabab" will be consumed in 'q_0' state, "abaab" will take NFA from 'q_0' to 'q_5' and any remaining symbol will be consumed in 'q_5'. All other strings will end up in one of the non-accepting states for any chosen path.

Theorm:

Each Non-Deterministic Finite State Automaton (NFA) has an equivalent Deterministic Finite State Automaton (DFA).

Let N be NFA, $N = (Q, \Sigma, \delta, q_0, F)$

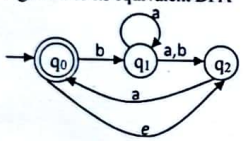
and, $D = (Q', \Sigma', \delta', q_0', F')$ be DFA, Such that,

- States of D be power set of states of N, i.e. $Q' = P(Q)$
- $\Sigma' = \Sigma$
- $q_0' = E(q_0) = \{q \in Q: (q_0, e) \vdash_N^* (q, e)\}$, i.e. initial state of D is set of initial states of N
- Transition Function, $\delta'(R', a) = \{q: q \in Q, q \in E(\delta(r', a)), r' \in R'\}$, i.e. δ' is a unique mapping.
- $F' = \{R' \in Q': R' \cap F \neq \Phi\}$, i.e. the machine accepts at R' if any element of R' is accepting state in F.

The so constructed DFA follows the Sequences of states upon reading input, such that each state corresponds to subset of set of states NFA would occupy. Every string that N would accept will be accepted by D and any string that N would not accept will not be accepted by D. i.e. D is equivalent Deterministic Finite State Automaton for Non-Deterministic Finite State Automaton, N.

Example:

Convert the following NFA to its equivalent DFA



Here, Given NFA is

$N = (Q, \Sigma, \delta, q_0, F)$

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

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

$\delta: Q \times (\Sigma \cup \{e\}) \rightarrow P(Q)$ is transition function defined by following transition table

	δ		
Q \ Σ	a	b	e
q_0	Φ	$\{q_1\}$	$\{q_2\}$
q_1	$\{q_1, q_2\}$	$\{q_2\}$	Φ
q_2	$\{q_0\}$	Φ	Φ

q_0 is the initial state

$F = \{q_0\}$ is the set of Final States

Now, The equivalent DFA is

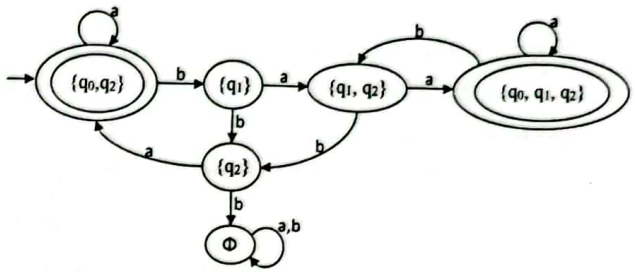


Fig: Equivalent DFA for given NFA

Hence, the required Equivalent DFA is

$D = (Q, \Sigma, \delta, q_0, F)$

Where, $Q = \{\Phi, \{q_1\}, \{q_2\}, \{q_0, q_2\}, \{q_1, q_2\}, \{q_0, q_1, q_2\}\}$ is set of states

$\Sigma = \{a, b\}$ is set of alphabets (input symbols)

$\delta: Q \times \Sigma \rightarrow Q$ is transition function defined by following transition table

	δ	
Q \ Σ	a	b
$\{q_0, q_2\}$	$\{q_0, q_2\}$	$\{q_1\}$
$\{q_1\}$	$\{q_1, q_2\}$	$\{q_2\}$
$\{q_1, q_2\}$	$\{q_0, q_1, q_2\}$	$\{q_2\}$
$\{q_2\}$	$\{q_0, q_2\}$	Φ
$\{q_0, q_1, q_2\}$	$\{q_0, q_1, q_2\}$	$\{q_1, q_2\}$
Φ	Φ	Φ

$\{q_0, q_2\}$ is the initial state

$F = \{\{q_0, q_2\}, \{q_0, q_1, q_2\}\}$ is the set of Final States

Regular Language

Regular Language is a Formal Language that can be expressed using a Regular Expression. i.e. Regular Language is a set that can be Φ, $\{e\}$, $\{a: a \in \Sigma\}$ or any set obtained by operation of union, concatenation or Kleene Star on Regular sets.

Properties of Regular Language

In next section, we will show that any Regular Language will be accepted by some Finite automaton and every language accepted by Finite Automaton is regular. So following properties for the languages accepted by finite automaton are also the properties of Regular Language.

Theorem:

Class of languages accepted by finite automaton is closed under union operation

Let $N' = (Q', \Sigma, \delta', q_0', F')$ and

$N'' = (Q'', \Sigma, \delta'', q_0'', F'')$

be two NFAs accepting languages L(N') and L(N'') respectively.

Let's construct NFA

$N = (Q, \sum, \delta, q_0, F)$

Such that

$Q = Q' \cup Q'' \cup \{q_0\}$

$\delta = \delta' \cup \delta'' \cup \{((q_0,e), q_0'), ((q_0,e), q_0'')\}$

$F = F' \cup F''$

For string $\omega \in L(N') \cup L(N'')$,

$(q_0, \omega) \vdash_N^* (q_0', \omega)$ OR, $(q_0, \omega) \vdash_N^* (q_0'', \omega)$

$(q_0', \omega) \vdash_N^* (q', e)$; $q' \in F$ OR, $(q_0'', \omega) \vdash_N^* (q'', e)$; $q'' \in F$

because, because,

$(q_0', \omega) \vdash_{N'}^* (q', e)$; $q' \in F'$, $\omega \in L(N')$ $(q_0'', \omega) \vdash_{N''}^* (q'', e)$; $q'' \in F''$, $\omega \in L(N'')$

Thus, Class of languages accepted by finite automaton is closed under union operation.

Theorem:

#Class of languages accepted by finite automaton is closed under concatenation operation

Let $N' = (Q', \sum, \delta', q_0', F')$ and

$N'' = (Q'', \sum, \delta'', q_0'', F'')$

be two NFAs accepting languages L(N') and L(N'') respectively.

Let's construct NFA

$N = (Q, \sum, \delta, q_0, F)$

Such that

$q_0 = q_0'$

$Q = Q' \cup Q''$

$\delta = \delta' \cup \delta'' \cup \{((q,e), q_0''): q \in F'\}$

$F = F''$

Let, For string $\omega = \omega_1\omega_2$ such that $\omega_1 \in L(N')$ and $\omega_2 \in L(N'')$,

$(q_0, \omega) \vdash_N^* (q_0', \omega)$

$(q_0, \omega_1\omega_2) \vdash_N^* (q, \omega_2)$, $q \in F'$

because,

$q_0 = q_0'$, $(q_0', \omega_1) \vdash_{N'}^* (q,e)$

but

$(q, \omega_2) \vdash_N^* (q_0'', \omega_2)$, $q \in F'$ because $\delta(q,e) = q_0''$ if $q \in F'$

then

$(q_0'', \omega_2) \vdash_N^* (q,e)$; $q \in F$

because

$(q_0'', \omega_2) \vdash_{N''}^* (q,e)$; $q \in F''$

Thus, Class of languages accepted by finite automaton is closed under Concatenation operation.

Theorem:

#Class of languages accepted by finite automaton is closed under Kleene Star operation

Let $N' = (Q', \sum, \delta', q_0', F')$ an NFA accepting languages L(N').

Let's construct NFA

$N = (Q, \sum, \delta, q_0, F)$

Such that

$Q = Q' \cup \{q_0\}$

$\delta = \delta' \cup \{((q_0,e), q_0'), ((q,e), q_0'): q \in F'\}$

$F = \{q_0\} \cup F'$

Let, For string $\omega = \omega_1\omega_2 \ldots \omega_n$ such that $\omega_i \in L(N')$,

$(q_0, \omega) \vdash_N^* (q_0', \omega)$; $\delta(q_0,e) = q_0'$,

$(q_0', \omega_1\omega_2\ldots\omega_n) \vdash_N^* (q, \omega_2\ldots\omega_n)$, $q \in F$ because, $(q_0', \omega_1) \vdash_{N'}^* (q,e)$; $q \in F$, $\omega_1 \in L(N')$

$(q, \omega_2\ldots\omega_n) \vdash_N^* (q_0', \omega_2\ldots\omega_n)$; $q \in F$ because, $\delta(q,e) = q_0'$, $q \in F'$

Again,

$(q_0', \omega_2\ldots\omega_n) \vdash_N^* (q, \omega_3\ldots\omega_n)$, $q \in F$ because, $(q_0', \omega_2) \vdash_{N'}^* (q,e)$; $q \in F$, $\omega_2 \in L(N')$

$(q, \omega_3\ldots\omega_n) \vdash_N^* (q_0', \omega_3\ldots\omega_n)$; $q \in F$ because, $\delta(q,e) = q_0'$, $q \in F'$

Simillarly,

$(q_0', \omega_n) \vdash_N^* (q, e)$, $q \in F$

i.e. $(q_0', \omega) \vdash_N^* (q, e)$, $q \in F$

Thus, Class of languages accepted by finite automaton is closed under Kleene Star operation

Theorem:

#Class of languages accepted by finite automaton is closed under Complementation

Let $N' = (Q', \sum, \delta', q_0', F')$ an NFA accepting languages L(N').

Let's construct NFA

$N = (Q, \sum, \delta, q_0, F)$

Such that

$Q = Q'$

$\delta = \delta'$

$q_0 = q_0'$

$F = Q' - F'$

Then, by construction, N would accept the language,

$L(N) = \sum^* - L(N')$

Thus, Class of languages accepted by finite automaton is closed under Complementation

Theorem:

#Class of languages accepted by finite automaton is closed under intersection operation

For any Languages, L_1 and L_2

$L_1 \cap L_2 = \sum^* - ((\sum^* - L_1) \cup (\sum^* - L_2))$

The RHS involves operations of union and complementation only. As languages accepted by finite automata are closed under operation of union and complementation, hence they are also closed under intersection operation.

Equivalence of Finite Automata and Regular Language

Theorem:

A Language is Regular if and only if it is accepted by Finite Automata.

To prove this theorem, we have to show:

i. If a Language is Regular then it is accepted by a Finite Automata.

ii. If a Language is accepted by Finite Automata then it is Regular.

i. Proof for "If a Language is Regular then it is accepted by a Finite Automata."

We know, Regular Language is a set that can be Φ, $\{e\}$, $\{a: a \in \sum\}$ or any set obtained by operation of union, concatenation or Kleene Star on Regular sets.

Since finite automaton

recognizes Φ,

recognizes $\{e\}$,

recognizes {a}, and since class of languages accepted by finite automaton is closed under union, concatenation and Kleene Star operations, hence, any regular language is accepted by some finite automaton.

ii. Proof for "If a Language is accepted by Finite Automata then it is Regular."

Let, $D = (Q, \sum, \delta, q_0, F)$ be some DFA such that L(D) is language accepted by D. Such that

$Q = \{q_0, q_1, \ldots, q_n\}$

Let i,j $\in \{0,1,2,\ldots,n\}$ and k $\in \{0,1,2,\ldots,n,n+1\}$ and

$R_{ij}^k = \{\omega \in \sum^* : (q_i, \omega) \vdash_D^* (q_j, e)$, the configuration does not pass through intermediate state $q_{k'}, k' \geq k\}$

then, $L(D) = \cup_i \{R_{0j}^{n+1} : q_j \in F\}$

For k=0

$R_{ij}^0 = \{s : (q_i, s) \vdash_D^* (q_j, e)\}$; $i \neq j$

$= \{s : (q_i, s) \vdash_D^* (q_j, e)\} \cup \{e\}$; $i = j$

Here, {s} is singleton set, so it is regular language and set of empty string, {e}, is also regular.

Thus, R_{ij}^0 is Regular Language.

Induction Hypothesis:

Let, R_{ij}^k be Regular Language.

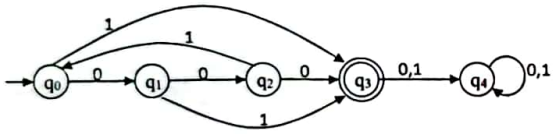
then, for k+1,

$R_{ij}^{k+1} = R_{ij}^k \cup R_{ik}^k (R_{kk}^k)^* R_{kj}^k$

Here, Each of R_{ij}^k, R_{ik}^k, R_{kk}^k and R_{kj}^k are Regular. So, their union, Concatenation and kleene star are also Regular. Hence, If a Language is accepted by Finite Automata then it is Regular.

Example:

Find Regular Expression for Following Automaton



$R_{03}^5 = R_{03}^4 \cup R_{04}^4 (R_{44}^4)^* R_{43}^4 = R_{03}^4$ $\quad [R_{04}^4 (R_{44}^4)^* R_{43}^4 = \Phi]$

$R_{03}^4 = R_{03}^3 \cup R_{03}^3 (R_{33}^3)^* R_{33}^3 = R_{03}^3$ $\quad [R_{03}^3 (R_{33}^3)^* R_{33}^3 = R_{33}^3]$

$R_{33}^3 = R_{03}^2 \cup R_{02}^2 (R_{22}^2)^* R_{23}^2$

Here, $R_{03}^2 = 1 \cup 01$

$R_{02}^2 = 00$

$R_{22}^2 = 100$

$R_{23}^2 = 0 \cup 11 \cup 101$

Hence, Required Regular Expression,

$R_{03}^5 = 1 \cup 01 \cup (00(100)^*(0 \cup 11 \cup 101))$

Pumping Lemma for Regular Language

For regular language L, there is a number $n \geq 1$, such that a string $\omega \in L$ with length at-least n can be divided into three pieces, $\omega = xyz$, satisfying the conditions

$xy^iz \in L$; $i \geq 0$

$|y| > 0$

$|xy| \leq n$

Proof:

Let $D = (Q, \sum, \delta, q_0, F)$ be a DFA that recognizes language L and n be number of states in D.

Let, $\omega = \alpha_1\alpha_2\alpha_3 \ldots \alpha_{n'}$, $n' \geq n$, be a string of length n', $\omega \in L$. Then DFA passes through the sequence of states-

q_1', q_2', ... $q_{n'+1}'$ in processing ω

i.e. $(q_0, \omega) \vdash_D^* (q_F, e)$, $q_F \in F$

Since, n' > n, by pigeon hole principle, there must be states that is repeated.

Let, $q_l' = q_m'$, then the string, ω, can be divided into three parts,

$\omega = xyz$

Such that, $x = \alpha_1\alpha_2\alpha_3 \ldots \alpha_{l-1}$

$y = \alpha_l\alpha_{l+1} \ldots \alpha_{m-1}$

$z = \alpha_m\alpha_{m+1} \ldots \alpha_{n'}$

Where, α_{l-1} causes DFA to go to state q_l' and α_{m-1} causes DFA to go to state q_m'. The substring 'z' takes the DFA to accept state.

Therefore, as $q_l' = q_m'$, y can be repeated any i times, y^i , $i \geq 0$ Such that, xy^iz still will belong to L, |y| will be greater than zero and $|xy| \leq n$.

Note: If a language **Do Not Satisfy Pumping Lemma,** then it is definitely **not Regular.** But if it does **satisfies Pumping lemma,** then it **might or might not be Regular**.

Example:

Check if the following language is regular.

$\sum = \{a, b\}$, $L = \{a^i, b^i : i \geq 0\}$

Solution:

Let the language L be regular. Let a string ω be a^nb^n where n is pumping length (pumping length is actually the number of states in the DFA that accepts L) i.e. $\omega = a^nb^n$

Then, by pumping lemma,

ω can be represented as $\omega = xyz$

Such that, $xy^iz \in L$; $i \geq 0$

$|y| > 0$

$|xy| \leq n$

Let $x = a^p$, $y = a^q$, $z = a^{(n-(p+q))}b^n$

Now, $xy^iz = a^pa^{qi}a^{(n-(p+q))}b^n$

$= a^{p+qi+(n-(p+q))}b^n$

$= a^{n+q(i-1)}\, b^n$

For i = 2,

$xy^iz = xy^2z = a^{n+q(2-1)}\, b^n = a^{n+q}\, b^n \neq a^nb^n$ [because, as $|y| > 0$ so, $q > 0$]

i.e. $xy^iz \notin L$

Hence, the contradiction.

Therefore, Language, $L = \{a^i, b^i : i \geq 0\}$ is not Regular.

State Minimization in DFA

States of any DFA can be partitioned in groups of mutually indistinguishable states such that members of two different groups are always distinguishable. If each group is replaced by single state, the resulting DFA is Minimum DFA for that Language.

q_i and q_j are indistinguishable (i.e. $q_i \equiv q_j$) if for string ω of any length, either both q_i and q_j will reach to final state or both will not.

q_i and q_j are distinguishable if there exists an string ω that leads either q_i or q_j but not both, to final state.

$q_i \equiv_n q_j$ means q_i and q_j are not distinguishable by string of length less or equal to length n.

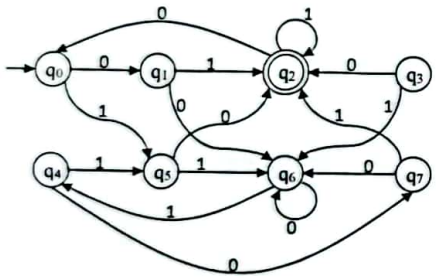
Let us define '$\equiv_n$' as partition of set of states such that any two states of different groups are distinguishable by some string of length less or equal to n.

Steps for DFA State Minimization:

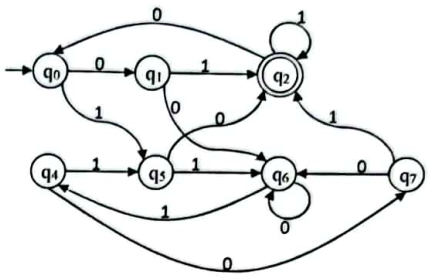
1. Remove any unreachable states.
2. Determine $\equiv_0$ = {{Set of Non-Final States}, {Set of Final States}}
   - Final and Non-Final states are distinguishable by empty string
3. Determine $\equiv_1, \equiv_2 \ldots \equiv_n$ by further splitting groups in previous partitions such that any two states in a group, upon seeing same alphabet, transition to states that belong to same group. (i.e. Split the group if there exist an alphabet that takes states in same group to states in different groups)
4. Terminate at n^{th} iteration if $\equiv_n = \equiv_{n-1}$

Example:

Minimize Following DFA:



- We see that, starting from q_0, we can never reach to q_3. So, q_5 is unreachable and can be removed. Redrawing DFA without unreachable states,



- Let $\equiv_n$ be the partition of set of states $\{q_0, q_1, q_2, q_4, q_5, q_6, q_7\}$ in two groups such that any two states of different groups are distinguishable by some string of length $\leq n$.

Then,

$\equiv_0 = \{\{q_0, q_1, q_4, q_5, q_6, q_7\}, \{q_2\}\}$

Group	State	0	1
G_1	q_0	G_1	G_1
	q_1	G_1	G_2
	q_4	G_1	G_1
	q_5	G_2	G_1
	q_6	G_1	G_1
	q_7	G_1	G_2
G_2	q_2	G_1	G_2

From above table we see that states q_0, q_4, q_6 are not distinguishable from string of length 1 where as q_0 and q_1 are distinguishable simply because upon seeing "1" at state q_0, DFA will go to some state in group G_1 whereas "1" will take DFA from q_1 to some state in G_2. We know that states in G_1 and G_2 are distinguishable with the string of length zero. So, further splitting the group G_1:

$\equiv_1 = \{\{q_0, q_4, q_6\}, \{q_1, q_7\}, \{q_5\}, \{q_2\}\} \neq \equiv_0$

Group	State	0	1
G_{11}	q_0	G_{12}	G_{13}
	q_4	G_{12}	G_{13}
	q_6	G_{11}	G_{11}
G_{12}	q_1	G_{11}	G_2
	q_7	G_{11}	G_2
G_{13}	q_5	G_2	G_{11}
G_2	q_2	G_{11}	G_2

$\equiv_2 = \{\{q_0, q_4\}, \{q_6\}, \{q_1, q_7\}, \{q_5\}, \{q_2\}\} \neq \equiv_1$

We see that Group 'G_{11}' further splits up.

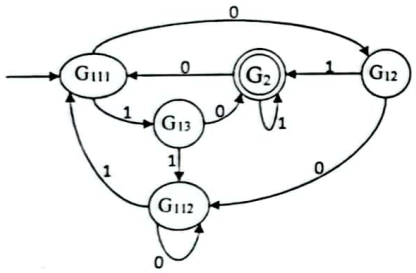
Group	State	0	1

G_{111}	q_0	G_{12}	G_{13}
	q_4	G_{12}	G_{13}
G_{112}	q_6	G_{112}	G_{111}
G_{12}	q_1	G_{112}	G_2
	q_7	G_{112}	G_2
G_{13}	q_5	G_2	G_{112}
G_2	q_2	G_{111}	G_2

$\equiv_3 = \{\{q_0, q_4\}, \{q_6\}, \{q_1, q_7\}, \{q_5\}, \{q_2\}\} = \equiv_2$

Now, as we cannot further split the group, we stop here.

Hence, the required Minimum DFA is:



Decision Algorithms for Regular Languages

Some Decision Questions and Algorithms:

- Given a DFA 'D' and string '*w*', does D accepts *w*?

Algorithm:

1. Simply Simulate D on *w*
2. Answer is *yes* if after consuming entire string *w*, D is in Accepting state.

- Given a DFA 'D' and Language it accepts be L(D) then is L(D) = Φ?

Algorithm:

1. Simulate D on all strings of length between *0* and *n-1*, where *n* is the no. os states in D.
2. Output '*yes*' if and only if all strings are rejected.
3. Otherwise output '*no*'

- Is L(D) finite?

Algorithm:

1. Simulate D on all strings of length between *n* and *2n-1* where *n* is the no. os states in D.
2. Output '*yes*' if and only if no string is accepted.
3. Otherwise output '*no*'.

- Given two FDSs D_1 and D_2, is $L(D_1) = L(D_2)$, ie. Are D_1 and D_2 equivalent?

Algorithm:

1. Construct DFA for $L(D_1) - L(D_2)$
2. Test if $L(D_1) - L(D_2)$ is empty
3. Construct DFA for $L(D_2) - L(D_1)$
4. Test if $L(D_2) - L(D_1)$ is empty
5. The answer is '*yes*' iff both $(L(D_1) - L(D_2))$ and $(L(D_2) - L(D_1))$ are empty.

Subjective Questions

1. Construct a DFA that accepts the Language
   $L(D) = \{ \omega \in \{a, b\}^* : \omega$ contains substring "ababa" $\}$
   Also Show Configuration Transition for string "baababaa". Does it belongs to L(D)?
2. Construct a DFA that accepts the Language
   $L(D) = \{ \omega \in \{a, b\}^* : \omega$ contains ends with "abb" $\}$
   Also Show Configuration Transition for string "abbabb". Does it belongs to L(D)?
3. Construct a DFA that accepts the Language
   $L(D) = \{ \omega \in \{0, 1\}^* :$ binary string 'ω' is exactly divisible by 5 $\}$
   [eg. $(1010)_2 = (10)_{10}$ is exactly divisible by 5, so it should be accepted]
   Also Show Configuration Transition for string "01111".
4. Construct a NFA that accepts the Language
   $L(D) = \{ \omega \in \{a, b\}^* : \omega$ contains exactly two a's $\}$
   Then, convert it to corresponding DFA.
5. Construct a NFA that accepts the Language
   $L(D) = \{ \omega \in \{a, b\}^* : \omega$ contains substring "aba" or "bba" $\}$
   Then, convert it to corresponding DFA.
6. Construct a NFA that accepts the Language
   $L(D) = a(ab \cup bba)^*b$
   Then, convert it to corresponding DFA.
7. Prove that the class of Regular language is closed under the Union and Concatenation operations.
8. State Pumping Lemma for Regular Language and check whether following language is regular or not:
   $L = \{ a^{n!} : n > 0 \}$
9. Use Pumping Lemma to check whether following language is regular or not:
   $L = \{ a^n b^m a^n : n > 0, m > 0 \}$
10. Construct a DFA that accepts the Language
    $L(D) = \{ \omega \in \{a, b\}^* : \omega$ Starts with "ab" $\}$
    Then, determine Regular Expression Corresponding to this DFA.
11. Minimize the DFA given as transition table below

Q \ Σ	a	b
→q_0	q_1	q_3
q_1	q_2	q_2
q_2	q_1	q_4
*q_3	q_3	q_4
*q_4	q_3	q_4
q_5	q_4	q_2

Here, '→' represents starting state and ' * ' represents Final States.

12. Minimize the DFA given as transition diagram below