CENG 280 - Chapter 2: Finite Automata

Burak Metehan Tunçel - April 2022

1 Deterministic Finite Automata

Finite automaton, or **finite-state machine** shares with a real computer the fact that it has a "central processing unit" of fixed, finite capacity. It receives its input as a string, delivered to it on an input tape. It delivers no output at all, except an indication of whether or not the input is considered acceptable. It is, in other words, a language recognition device.

What makes the finite automaton such a restricted model of real computers is the *complete absence of memory* outside its fixed central processor.

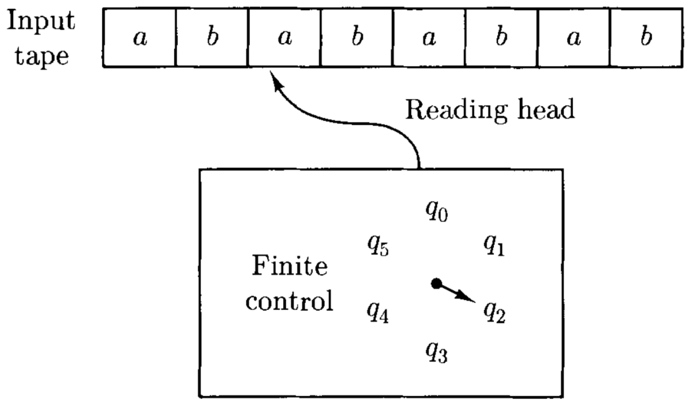


Figure 1: Finite Automata

Let's describe the operation of a finite automaton in more detail.

- Strings are fed into the device by means of an **input tape**, which is divided into squares, with one symbol inscribed in each tape square (see Figure 1).
- The main part of the machine itself is a "black box", with innards that can be, at any specified moment, in one of a finite number of distinct internal states.
- The black box -called the **finite control**- can sense what symbol is written at any position on the input tape by means of a movable **reading head**.
- Initially, the reading head is placed at the leftmost square of the tape and the finite control is set in a designated **initial state**.
- At regular intervals the automaton reads one symbol from the input tape and then enters a new state that *depends only on the current state and the symbol just read.*
- After reading an input symbol, the reading head moves one square to the right on the input tape so that on the next move it will read the symbol in the next tape square. This process is repeated again and again; a symbol is read, the reading head moves to the right, and the state of the finite control changes. Eventually the reading head reaches the end of the input string.
- The automaton then indicates its approval or disapproval of what it has read by the state it is in at the end: if it winds up in one of a set of **final states** the input string is considered to be **accepted**.
- The language accepted by the machine is the set of strings it accepts.

Definition 1

A **deterministic finite automaton** is a quintuple $M = (K, \Sigma, \delta, s, F)$ where

- K is a finite set of **states**
- Σ is an alphabet,
- $s \in K$ is the **initial state**
- $F \subseteq K$ is the set of **final states**
- δ, the **transition function**, is a function from $K \times \Sigma$ to K.

The **configuration** of the machine is the *current state and the unread part of the input string*, i.e., a configuration is an element of $K \times \Sigma^*$.

The binary $\vdash_M$ (yields) relation holds between two configurations of M if and only if the machine can pass from one configuration to another one as a result of a single move. Let (q, w) and (q', w') be two configurations of M. Then $(q, w) \vdash_M (q', w')$ if and only if $w = aw'$ for some $a \in \Sigma$ and $q' = \delta(q, a)$. $(q, w) \vdash_M (q', w')$ reads (q, w) **yields** (q', w') in **one step**. Note that $\vdash_M$ is a function from $K \times \Sigma^+$ to $K \times \Sigma^*$, hence, for every configuration except (q, e) there exists a uniquely determined next configuration.

$\vdash_M^*$ is the **reflexive transitive closure** of $\vdash_M$. $(q, w) \vdash_M^* (q', w')$ reads (q, w) **yields** (q', w'). A string $w \in \Sigma^*$ is **accepted** by M if and only if $(s, w) \vdash_M^* (f, e)$ for some $f \in F$. The **language** of M, $L(M)$, is the set of strings *accepted* by M.

Example 1

Let M be the deterministic finite automaton $(K, \Sigma, \delta, s, F)$, where

- $K = \{q_0, q_1\}$
- $\Sigma = \{a, b\}$
- $s = q_0$
- $F = \{q_0\}$

and δ is the function tabulated below.

q	σ	$\delta(q, \sigma)$
q_0	a	q_0
q_0	b	q_1
q_1	a	q_1
q_1	b	q_0

It is then easy to see that $L(M)$ is the set of all strings in $\{a, b\}^*$ that have an even number of b's. For M passes from state q_0 to q_1 or from q_1 back to q_0 when a b is read, but M essentially ignores a's, always remaining in its current state when an a is read. Thus M counts b's modulo 2, and since q_0 (the initial state) is also the sole final state, M accepts a string if and only if the number of b's is even.
If M is given the input $aabba$, its initial configuration is $(q_0, aabba)$. Then

$$\begin{aligned}
(q_0, aabba) &\vdash_M (q_0, abba)\\
&\vdash_M (q_0, bba)\\
&\vdash_M (q_1, ba)\\
&\vdash_M (q_0, a)\\
&\vdash_M (q_0, e)
\end{aligned}$$

Therefore $(q_0, aabba) \vdash_M^* (q_0, e)$, and so $aabba$ is accepted by M.

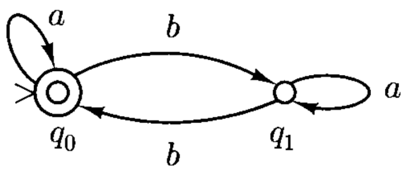


Figure 2: State Diagram

The tabular representation of the transition function used in this example is not the clearest description of a machine. We generally use a more convenient graphical representation called the **state diagram** (Figure 2).

2 Nondeterministic Finite Automata

In this section we add a powerful and intriguing feature to finite automata. This feature is called **nondeterminism**, and is essentially the ability to change states in a way that is only partially determined by the current state and input symbol. That is, we shall now permit several possible "next states" for a given combination of current state and input symbol.

The automaton, as it reads the input string, may choose at each step to go into anyone of these legal next states; the choice is not determined by anything in our model, and is therefore said to be *nondeterministic*. On the other hand, the choice is not wholly unlimited either; only those next states that are legal from a given state with a given input symbol can be chosen.

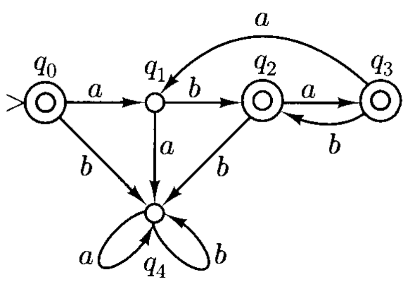


Figure 3

To see that a nondeterministic finite automaton can be a much more convenient device to design than a deterministic finite automaton, consider the language $L = (ab \cup aba)^*$, which is accepted by the deterministic finite automaton illustrated in Figure 3.

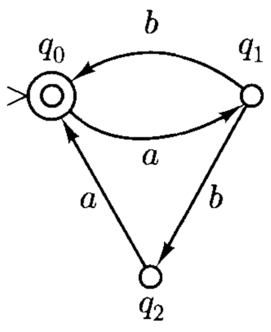


Figure 4

L is accepted by the simple nondeterministic device shown in Figure 4. When this device is in state q_l and the input symbol is b, there are two possible next states, q_0 and q_2. Thus Figure 4 does not represent a deterministic finite automaton.

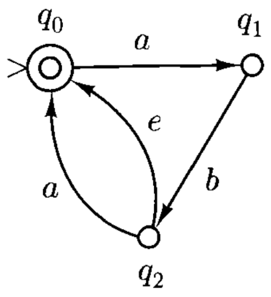


Figure 5

We also allow in the state diagram of a nondeterministic automaton arrows that are labeled by the empty string e. For example, the device of Figure 5 accepts the same language L. From q2 this machine can return to qo either by reading an a or immediately, without consuming any input.

Definition 2

A **nondeterministic finite automaton** is a quintuple $M = (K, \Sigma, \Delta, s, F)$, where

- K is a finite set of **states**
- Σ is an alphabet
- $s \in K$ is the **initial state**
- $F \subseteq K$ is the set of **final states**, and
- Δ, the **transition relation**, is a subset of $K \times (\Sigma \cup \{e\}) \times K$

$(q, a, p) \in \Delta$ is called a **transition** of M. (q, e, p) indicates that the machine can pass to state p from state q without reading an input symbol.

The configuration of the machine is the current state and the unread part of the input string, i.e., a configuration is an element of $K \times \Sigma^*$.

The binary $\vdash_M$ (**yields**) relation holds between two configurations of M if and only if the machine can pass from one configuration to another one as a result of a *single move*. Let (q, w) and (q', w') be two configurations of M. Then $(q, w) \vdash_M (q', w')$ if and only if $w = aw'$ for some $a \in \Sigma \cup \{e\}$ and $(q, a, q') \in \Delta$. $(q, w) \vdash_M (q', w')$ reads (q, w) **yields** (q', w') **in one step**. Note that $\vdash_M$ might not be a function, i.e., there might be several pairs (q', w') (or none at all) such that $(q, w) \vdash_M (q', w')$.

$\vdash^*_M$ is the **reflexive transitive closure** of $\vdash_M$. $(q, w) \vdash^*_M (q', w')$ reads (q, w) yields (q', w'). A string $w \in \Sigma^*$ is **accepted** by M if and only if there is a state $f \in F$ such that $(s, w) \vdash^*_M (f, e)$. The **language** of M, $L(M)$, is the set of strings accepted by M.

A deterministic finite state automaton is just a special type of nondeterministic finite state automaton. We obtain a DFA when Δ defines a function from $K \times \Sigma$ to K. In other words, an NFA $M = (K, \Sigma, \Delta, s, F)$ is deterministic if there are no transitions of the form (q, e, p) and for each $q \in K$ and $a \in \Sigma$, there exists *exactly one* $p \in K$ such that $(q, a, p) \in \Delta$.

We can conclude that the class of languages recognized by deterministic finite state automaton is a *subset* of the class of languages recognized by nondeterministic finite state automaton. Essentially, these classes are the same: A nondeterministic finite automaton can always be converted to an *equivalent* deterministic finite state automaton.

Two automaton M_1 and M_2 are said to be **equivalent** when $L(M_1) = L(M_2)$.

Theorem 1

For each nondeterministic finite automaton, there exists an equivalent deterministic finite automaton.

Proof. The proof of the theorem is constructive: use **subset construction** algorithm to construct a DFA from an NFA and then show they are equivalent. Given an NFA $M = (K, \Sigma, \Delta, s, F)$, the algorithm constructs an equivalent DFA $M' = (K', \Sigma, \delta, s', F')$ as follows. For each state $q \in K$, the set of states that can be reached without reading an input symbol is defined as

$$E(q) = \{p \in K \mid (q, e) \vdash^*_M (p, e)\} \qquad \text{(Check Example 2)}$$

Essentially, $E(q)$ is the reflexive transitive closure of the set $\{q\}$ under the relation $\{(p, r) \mid (p, e, r) \in \Delta\}$. The DFA is defined as:

$$\begin{aligned}
K' &= 2^K \\
s' &= E(s) \\
F' &= \{Q \subseteq K \mid Q \cap F \neq \emptyset\} \\
\delta'(Q, a) &= \{E(p) : p \in K, (q, a, p) \in \Delta \text{ for some } q \in Q\} \text{ for each } Q \in K' \text{ and } a \in \Sigma \\
&= \bigcup \{E(p) : p \in K, (q, a, p) \in \Delta \text{ for some } q \in Q\}
\end{aligned}$$

To prove that M and M' are equivalent, show that for any string $w \in \Sigma^*$

$$(s, w) \vdash^*_M (f, e) \text{ for some } f \in F \text{ iff } (E(s), q) \vdash^*_{M'} (P, e) \text{ for some } P \in F'$$

Thus they recognize the same language. □

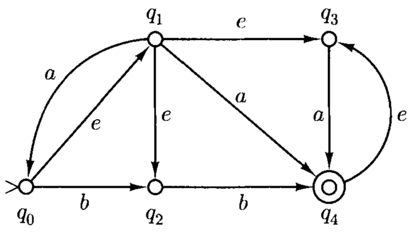


Figure 6

Example 2

In the automaton of Figure 6, we have $E(q_0) = \{q_0, q_1, q_2, q_3\}$, $E(q_1) = \{q_1, q_2, q_3\}$, $E(q_2) = \{q_2\}$, $E(q_3) = \{q_3\}$, and $E(q_4) = \{q_3, q_4\}$.

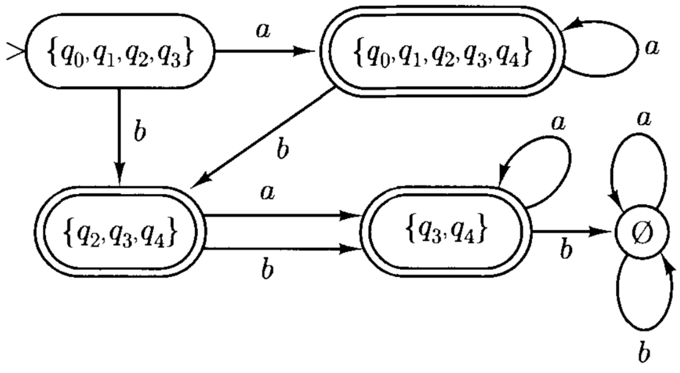


Figure 7

Example 3: NFA to DFA

Let's apply the algorithm then to the nondeterministic automaton in Figure 6. Since M has 5 states, M' will have $2^5 = 32$ states. However, only a few of these states will be relevant to the operation of M' -namely, those states that can be reached from state s' by reading some input string. Obviously, any state in K' that is not reachable from s' is irrelevant to the operation of M' and to the language accepted by it. We shall build this reachable part of M' by starting from s' and introducing a new state only when it is needed as the value of $\delta'(q, a)$ for some state $q \in K'$ already introduced and some $a \in \Sigma$.

We have already defined $E(q)$ for each state q of M (from Example 2). Since $s' = E(q_0) = \{q_0, q_1, q_2, q_3\}$,

$$(q_1, a, q_0),\ (q_1, a, q_4),\ \text{and}\ (q_3, a, q_4)$$

are all the transitions (q, a, p) for some $q \in s'$. It follows that

$$\delta'(s', a) = E(q_0) \cup E(q_4) = \{q_0, q_1, q_2, q_3, q_4\}$$

Similarly,

$$(q_0, b, q_2)\ \text{and}\ (q_2, b, q_4)$$

are all the transitions of the form (q, b, p) for some $q \in s'$, so

$$\delta'(s', b) = E(q_2) \cup E(q_4) = \{q_2, q_3, q_4\}$$

Repeating this calculation for the newly introduced states, we have the following,,

$$\begin{aligned}
\delta'(\{q_0, q_1, q_2, q_3, q_4\}, a) &= \{q_0, q_1, q_2, q_3, q_4\} \\
\delta'(\{q_0, q_1, q_2, q_3, q_4\}, b) &= \{q_2, q_3, q_4\} \\
\delta'(\{q_2, q_3, q_4\}, a) &= E(q_4) = \{q_3, q_4\} \\
\delta'(\{q_2, q_3, q_4\}, b) &= E(q_4) = \{q_3, q_4\}
\end{aligned}$$

Next,

$$\begin{aligned}
&\delta'(\{q_3, q_4\}, a) = E(q_4) = \{q_3, q_4\} \\
&\delta'(\{q_3, q_4\}, b) = 0
\end{aligned}$$

and finally

$$\delta'(\emptyset, a) = \delta'(\emptyset, b) = \emptyset$$

The relevant part of M' is illustrated in Figure 7. F', the set of final states, contains each set of states of which q_4 is a member, since q_4 is the sole member of F; so in the illustration, the three states $\{q_0, q_1, q_2, q_3, q_4\}$, $\{q_2, q_3, q_4\}$, and $\{q_3, q_4\}$ of M' are final.

Basically, first look the initial state of NFA. E(s), a set, will be the initial state of DFA. It was $\{q_0, q_1, q_2, q_3\}$ for the example 3. Then look the each element of the set, if there is a way to any other state by reading an input inside alphabet (in example it is (a, b)), note that state(s) and combine their E(s'). In the first part of example $\delta'(s', a) = E(q_0) \cup E(q_4) = \{q_0, q_1, q_2, q_3, q_4\}$. When introduce new state, repeat that.

3 Finite Automata and Regular Expressions

The main result of the last section was that the class of languages accepted by finite automata remains the same even if a new and seemingly powerful feature -*nondeterminism*- is allowed. This suggests that the class of languages accepted by finite automata has a sort of *stability*: Two different approaches, one apparently more powerful than the other, end up defining the same class. In this section we shall prove another important characterization of this class of languages, further evidence of how remarkably stable it is: The class of languages accepted by finite automata, deterministic or nondeterministic, is the same as the class of *regular languages* -those that can be described by regular expressions.

Theorem 2

The class of languages accepted by finite automata is closed under

a) union

b) concatenation

c) Kleene star

d) complementation

e) intersection

Proof. In each case we show how to construct an automaton M that accepts the appropriate language, given two automata M_1 and M_2 (only M_1 in the cases of Kleene star and complementation).

(a) *Union.* Let $M_1 = (K_1, \Sigma, \Delta_1, s_1, F_1)$ and $M_2 = (K_2, \Sigma, \Delta_2, s_2, F_2)$ be nondeterministic finite automata; we shall construct a nondeterministic finite automaton M such that $L(M) = L(M_1) \cup L(M_2)$. The construction of M is rather simple and intuitively clear, illustrated in Figure 8. Basically, M uses nondeterminism to guess whether the input is in $L(M_1)$ or in $L(M_2)$, and then processes the string exactly as the corresponding automaton would; it follows that $L(M) = L(M_1) \cup L(M_2)$. But let us give the formal details and proof for this case. Without

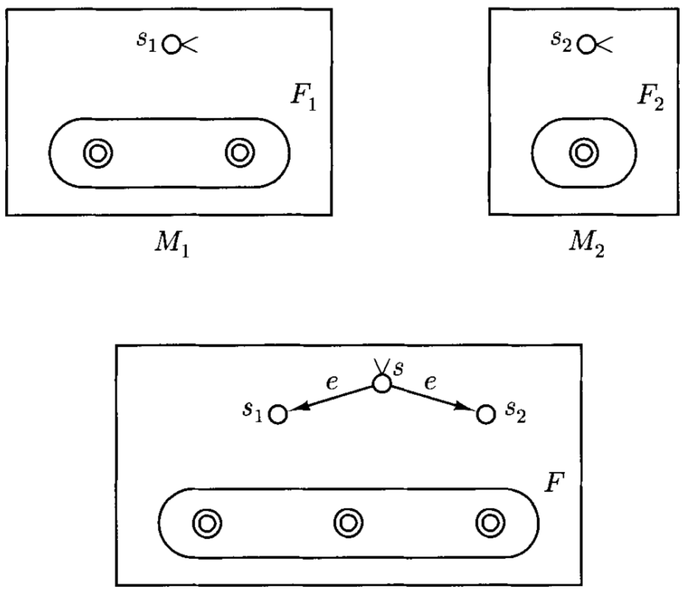


Figure 8

loss of generality, we may assume that K_1 and K_2 are disjoint sets. Then the finite automaton M that accepts $L(M_1) \cup L(M_2)$ is defined as follows

$M = (K, \Sigma, \Delta, s, F)$, where s is a new state not in K_1 or K_2,

- $K = K_1 \cup K_2 \cup \{s\}$
- $F = F_1 \cup F_2$
- $\Delta = \Delta_1 \cup \Delta_2 \cup \{(s, e, s_1), (s, e, s_2)\}$

That is, M begins any computation by nondeterministically choosing to enter either the initial state of M_1 or the initial state of M_2 ,and thereafter, M imitates either M_1 or M_2. Formally, if $w \in \Sigma^*$, then $(s, w) \vdash_M^* (q, e)$ for some $q \in F$ if and only if either $(s_1, w) \vdash_{M_1}^* (q, e)$ for some $q \in F_2$, or $(s_2, w) \vdash_{M_2}^* (q, e)$ for some $q \in F_2$. Hence M accepts w if and only if M_1 accepts w or M_2 accepts w, and $L(M) = L(M_1) \cup L(M_2)$.

(b) *Concatenation.* Again, let M_1 and M_2 be nondeterministic finite automata; we construct a nondeterministic finite automaton M such that $L(M) = L(M_1)L(M_2)$. The construction is shown schematically in Figure 9; M now operates by simulating M_1 for a while, and then "*jumping*" nondeterministically from a final state of M_1 to the initial state of M_2. Thereafter, M imitates M_2.

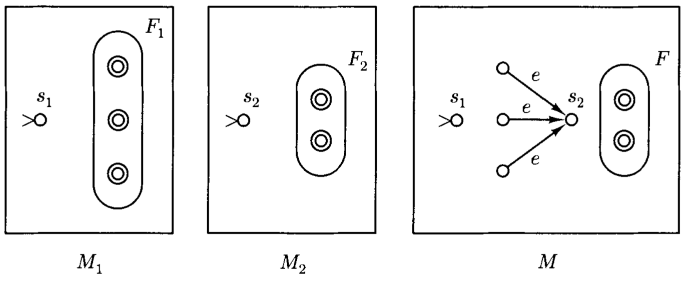


Figure 9

Then the finite automaton M that accepts $L(M_1)L(M_2)$ is defined as follows (*May include mistakes*)

$M = (K, \Sigma, \Delta, s, F)$

- $s = s_1$
- $K = K_1 \cup K_2$
- $F = F_2$
- $\Delta = \Delta_1 \cup \Delta_2 \cup \{(f, e, s_2) \mid f \in F_1\}$

(c) *Kleene star.* Let M_1 be a nondeterministic finite automaton; we construct a nondeterministic finite automaton M such that $L(M) = L(M_1)^*$. The construction is similar to that for concatenation, and is illustrated in Figure 10. M consists of the states of M_1 and all the transitions of M_1; any final state of M_1 is a final state of M. In addition, M has a new initial state s. This new initial state is also final, so that e is accepted. From s there is an e-transition to the initial state s_1 of M_1, so that the operation of M_1 can be initiated after M has been started in state s. Finally, e-transitions are added from each final state of M_1 back to s_1; this way, once a string in $L(M_1)$ has been read, computation can resume from the initial state of M_1.

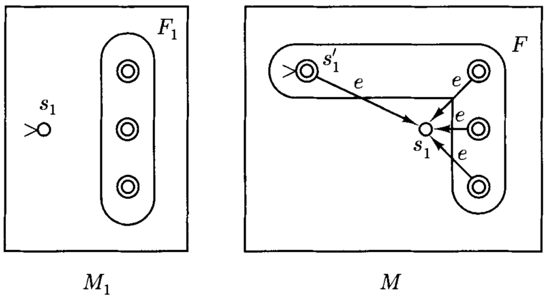


Figure 10

Then the finite automaton M that accepts $L(M_1)^*$ is defined as follows (*May include mistakes*)

$M = (K, \Sigma, \Delta, s, F)$, where s is not in M_1

- $K = K_1 \cup \{s\}$
- $F = F_1 \cup \{s\}$
- $\Delta = \Delta_1 \cup \{(s, e, s_1)\}$

(d) *Complementation.* Let $M = (K, \Sigma, \delta, s, F)$ be a *deterministic* finite automaton. Then the complementary language $\overline{L} = \Sigma^* - L(M)$ is accepted by the deterministic finite automaton $M = (K, \Sigma, \delta, s, K - F)$. That is, $\overline{M}$ is identical to M except that final and non final states are interchanged.

Then the finite automaton M that accepts $\overline{L(M_1)}$ is defined as follows (*May include mistakes*)

$M = (K, \Sigma, \Delta, s, F)$.

- $s = s_1$
- $K = K_1$
- $F = K \setminus F_1$
- $\Delta = \Delta_1$

(e) *Intersection.* Just recall that

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

and so closedness under intersection follows from closedness under union and complementation ((a) and (d) above). So, basically

$$L(M) = L(M_1) \cap L(M_2) = \overline{\overline{L(M_1)} \cup \overline{L(M_2)}}$$

□

Theorem 3

A language is regular if and only if it is accepted by a finite automaton.

Proof. • *Only if.* Recall that the class of regular languages is the smallest class of languages containing the empty set $\emptyset$ and the singletons a, where a is a symbol, and *closed under union, concatenation, and Kleene star.*

It is evident (see Figure 11) that the empty set and all singletons are indeed accepted by finite automata; and by Theorem 2 the finite automaton languages are closed under *union*, *concatenation*, and *Kleene star*. Hence *every regular language is accepted by some finite automaton.*

Example 4

Consider the regular expression $(ab \cup aab)^*$. A nondeterministic finite automaton accepting the language denoted by this regular expression can be built up using the methods in the proof of the various parts of Theorem 2, as illustrated in Figure 11.

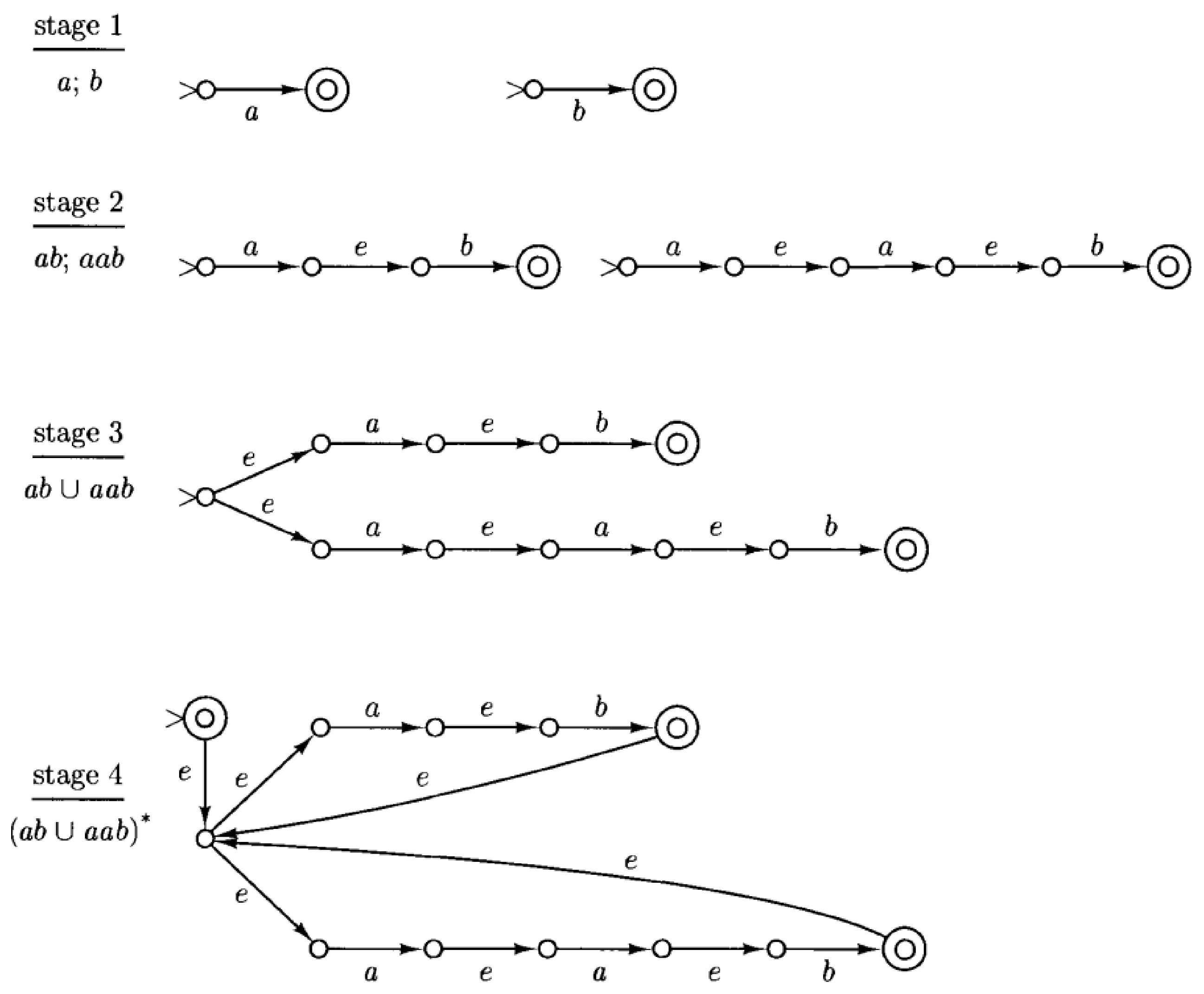


Figure 11

- *If.* Let $M = (K, \Sigma, \Delta, s, F)$ be a finite automaton (not necessarily deterministic). We shall construct a regular expression R such that $L(R) = L(M)$. We shall represent $L(M)$ as the union of many (but a finite number of) simple languages. Let $K = \{q_1, ..., q_n\}$ and $s = q_1$. For $i, j = 1, ..., n$ and $k = 0, ..., n$:

  $R(i, j, k)$: the set of strings $w \in \Sigma^*$ that can be read by M starting from q_i to q_j without passing through an intermediate state from $\{q_{k+1}, ..., q_n\}$ -the endpoints q_i and q_j are allowed to be numbered higher than k.

  That is, $R(i, j, k)$ is the set of strings spelled by all paths from q_i to q_j of rank k (recall the similar maneuver in the computation of the reflexive-transitive closure of a relation, in which we again considered paths with progressively higher and higher-numbered intermediate nodes). When $k = n$, it follows that

  $$R(i, j, n) = \{w \in \Sigma^* \mid (q_i, w) \vdash_M^* (q_j, e)\}$$

  Therefore,

  $$L(M) = \bigcup \{R(1, j, n) \mid q_j \in F\}$$

  The crucial point is that all of these sets $R(i, j, k)$ are regular, and hence so is $L(M)$.

  The proof that each $R(i, j, k)$ is regular is by induction on k. For $k = 0$, $R(i, j, 0)$:

  - $\{a \in \Sigma \cup \{e\} \mid (q_i, a, q_j) \in \Delta\}$ if $i \neq j$
  - $\{e\} \cup \{a \in \Sigma \cup \{e\} \mid (q_i, a, q_j) \in \Delta\}$ if $i = j$.

  $R(i, j, 0)$ is either singleton or empty for each i, j, hence regular.

  For the induction step, suppose that $R(i, j, k-1)$ for all i, j have been defined as regular languages for all i, j. Then each set $R(i, j, k)$ can be defined combining previously defined regular languages by the regular operations of union, Kleene star, and concatenation, as follows:

  $$R(i, j, k) = R(i, j, k-1) \cup R(i, k, k-1)R(k, k, k-1)^* R(k, j, k-1)$$

  This equation states that to get from q_i to q_j without passing through a state numbered greater than k, M may either

  1. go from q_i to q_j without passing through a state numbered greater than $k-1$; or
  2. go (a) from q_i to q_k; then (b) from q_k to q_k zero or more times; then (c) from q_k to q_j; in each case without passing through any intermediate states numbered greater than $k-1$

  Consequently, if $R(i, j, k-1)$ is regular for each i, j, then $R(i, j, k)$ is also regular for all i, j, k. By induction, $R(i, j, n)$ is regular.

  □

The proof of the theorem is used to generate a regular expression from a finite state automaton. Its application is simpler when the automaton is in the following special form

- the automaton has a single final state, $F = \{f\}$
- the initial state does not have an incoming transition
- the final state does not have an outgoing transition

Every automaton can be converted to an equivalent automaton in special form.

RE construction from FA:

1. Convert FA to an NFA in special form.
2. For each $k = 0, ..., n$, and for each $i, j = 1, ..., n$: Compute $R(i, j, k)$
3. Return $R(s, f, n)$

Example 5

Let us construct a regular expression for the language accepted by the deterministic finite automaton of Figure 12. This automaton accepts the language

$$\left\{w \in \{a, b\}^* \mid w \text{ has } 3k + 1\ b\text{'s for some } k \in \mathbb{N}\right\}$$

Carrying out explicitly the construction of the proof of the *if* part can be very tedious (in this simple case, thirty-six regular expressions would have to be constructed!).

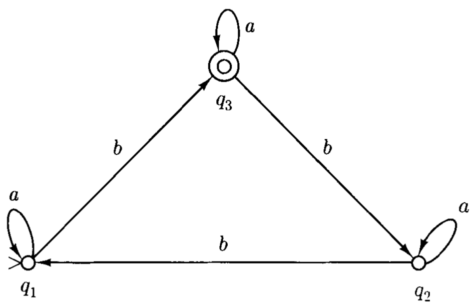


Figure 12

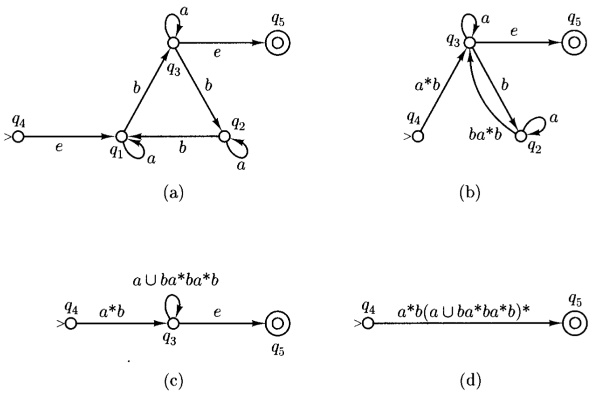


Figure 13

Things are simplified considerably if we assume that the nondeterministic automaton M has two simple properties:

- It has a single final state, $F = \{f\}$
- Furthermore, if $(q, u, p) \in \Delta$, then $q \neq f$ and $p \neq s$; that is, there are no transitions into the initial state, nor out of the final state

This "special form" is not a loss of generality, because we can add to any automaton M a new initial state s and a new final state f, together with e-transitions from s to the initial state of M and from all final states of M to f (see Figure 13(a), where the automaton of Figure 12 is brought into this "special form"). Number now the states of the automaton $q_1, q_2, ..., q_n$, as required by the construction, so that $s = q_{n-1}$ and $f = q_n$. Obviously, the regular expression sought is $R(n-1, n, n)$.

We shall compute first the $R(i, j, 0)$'s, from them the $R(i, j, 1)$'s, and so on, as suggested by the proof. At each stage we depict each $R(i, j, k)$'s as a label on an arrow going from state q_i to state q_j. We omit arrows labeled by $\emptyset$, and self-loops labeled $\{e\}$. With this convention, the initial automaton depicts the correct values of the $R(i, j, 0)$'s -see Figure 13(a). (This is so because in our initial automaton it so happens that, for each pair of states (q_i, q_j) there is at most one transition of the form (q_i, u, q_j) in Δ. In another automaton we might have to combine by union all transitions from q_i to q_j, as suggested by the proof.)

Now we compute the $R(i, j, 1)$'s; they are shown in Figure 13(b). Notice immediately that state q_1 need not be considered in the rest of the construction; all strings that lead M to acceptance passing through state q_1 have been considered and taken into account in the $R(i, j, 1)$'s. We can say that state q_1 has been *eliminated.* In some sense, we have transformed the finite automaton of Figure 13(a) to an equivalent *generalized finite automaton*, with transitions that may be labeled not only by symbols in Σ or e, but by entire *regular expressions.* The resulting generalized finite automaton has one less state than the original one, since q_1 has been eliminated.

Let us examine carefully what is involved in general in eliminating a state q (see Figure 14). For each pair of states $q_i \neq q$ and $q_j \neq q$, such that there is an arrow labeled α: from q_i to q and an arrow labeled β from q to q_j, we add an arrow from q_i to q_j labeled $\alpha\gamma^0\beta$, where γ is the label of the arrow from q to itself (if there is no such arrow, then $\gamma = \emptyset$, and thus $\gamma^* = \{e\}$, so the label becomes $\alpha\beta$. If there was already an arrow from q_i to q_j labeled δ, then the new arrow is labeled $\delta \cup \alpha\gamma^*\beta$.

Continuing like this, we eliminate state q_2 to obtain the $R(i, j, 2)$'s in Figure 13(c), and finally we eliminate q_3. We have now deleted all states except the initial and final ones, and the generalized automaton has been reduced to a single transition from the initial state to the final state. We can now read the regular expression for M as the label of this transition:

$$R = R(4, 5, 5) = R(4, 5, 3) = a^*b(ba^*ba^*b \cup a)^*$$

which is indeed $\{w \in \{a, b\}^* \mid w \text{ has } 3k + 1\ b\text{'s for some } k \in \mathbb{N}\}$.

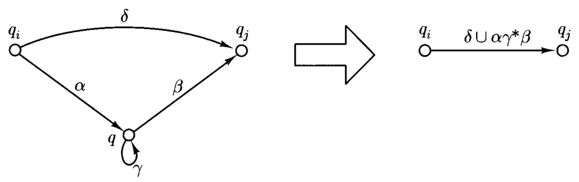


Figure 14

4 Languages that are and are not Regular

The regular languages are closed under a variety of operations and that regular languages may be specified either by regular expressions or by deterministic or nondeterministic finite automata. These facts, used singly or in combinations, provide a variety of techniques for showing languages to be regular.

Example 6

Let $\Sigma = \{0, 1, ..., 9\}$ and let $L \subseteq \Sigma^*$ be the set of decimal representations for nonnegative integers (without redundant leading O's) divisible by 2 or 3. For example, $0, 3, 6, 244 \in L$, but $1, 03, 00 \notin L$. Then L is regular. We break the proof into four parts.

Let L_1 be the set of decimal representations of nonnegative integers. Then it is easy to see that

$$L_1 = 0 \cup \{1, 2, ..., 9\}\, \Sigma^*$$

which is regular since it is denoted by a regular expression.

Let L_2 be the set of decimal representations of nonnegative integers divisible by 2. Then L_2 is just the set of members of L, ending in 0, 2, 4, 6, or 8; that is,

$$L_2 = L_1 \cap \Sigma^*\, \{0, 2, 4, 6, 8\}$$

which is regular by Theorem 2(e).

Let L_3 be the set of decimal representations of nonnegative integers divisible by 3. Recall that a number is divisible by 3 if and only if the sum of its digits is divisible by 3. We construct a finite automaton that keeps track in its finite control of the sum modulo 3 of a string of digits. L_3 will then be the intersection with L_1 of the language accepted by this finite automaton. The automaton is pictured in Figure 15.

Finally, $L = L_2 \cup L_3$, surely a regular language.

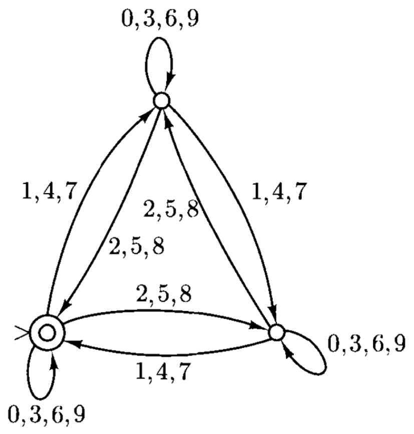


Figure 15

From Ebru Hoca's Notes

To show that a language L is regular, one of the following methods can be used:

- write a regular expression α such that $L = L(\alpha)$
- construct an NFA M such that $L = L(M)$
- use closure properties, e.g., for regular languages L_1 and L_2, show $L = L_1 \cap L_2$, $L = L_1 \cup L_2$, $L = L_1 L_2$, or $L = \Sigma^* \setminus L_1$ (more examples can be added)

Although we now have a variety of powerful techniques for showing that languages are regular, as yet we have none for showing that languages are not regular. Two properties shared by all regular languages, but not by certain nonregular languages, may be phrased intuitively as follows:

1. As a string is scanned left to right, the amount of memory that is required in order to determine at the end whether or not the *string is in the language must be bounded*, fixed in advance and dependent on the language, not the particular input string. For example, we would expect that $\{a^n b^n \mid n \geq 0\}$ is not regular, since it is difficult to imagine how a finite-state device could be constructed that would correctly remember, upon reaching the border between the a's and the b's, how many a's it had seen, so that the number could be compared against the number of b's.

2. Regular languages with an infinite number of strings are represented by automata with cycles and regular expressions involving the Kleene star. Such languages must have infinite subsets with a certain simple repetitive structure that arises from the Kleene star in a corresponding regular expression or a cycle in the state diagram of a finite automaton. This would lead us to expect, for example, that $\{a^n \mid n \geq 1 \text{ is a prime}\}$ is not regular, since there is no simple periodicity in the set of prime numbers.

In brief,

From Ebru Hoca's Notes

To show that a language L is not regular, we use the following property of regular languages:

- as a string is scanned from left to right, the amount of memory required to determine if $w \in L$ or $w \notin L$ must be bounded.
- In RL, infinite languages can be represented with Kleene star (cycle in automata), which induce a periodicity/pattern.

These intuitive ideas are formalized in the following theorem known as *Pumping Lemma.*

Theorem 4: Pumping Lemma

Let L be a regular language. There is an integer $n \geq 1$ such that any string $w \in L$ with $|w| \geq n$ can be rewritten as $w = xyz$ such that $y \neq e$, $|xy| \leq n$ and $xy^iz \in L$ for each $i \geq 0$.

Proof. Since L is regular, L is accepted by a deterministic finite automaton M. Suppose that n is the number of states of M, and let w be a string of length n or greater. Consider now the first n steps of the computation of M on w:

$$(q_0, w_1w_2...w_n) \vdash_M (q_1, w_2...w_n) \vdash_M ... \vdash_M (q_n, e)$$

where q_0 is the initial state of M, and $w_1...w_n$ are the n first symbols of w. Since M has only n states, and there are $n+1$ configurations $(q_i, w_{i+1}..., w_n)$ appearing in the computation above, by the *pigeonhole principle* there exist i and j, $0 \leq i < j \leq n$, such that $q_i = q_j$. That is, the string $y = w_iw_{i+1}...w_j$ drives M from state q_i back to state q_i, and this string is nonempty since $i < j$. But then this string could be removed from w, or repeated any number of times in w just after the jth symbol of w, and M would still accept this string. That is, M accepts $xy^iz \in L$ for each $i \geq 0$, where $x = w_1...w_i$, and $z = w_{j+1}...w_m$. Notice finally that the length of xy, the number we called j above, is by definition at most n, as required.

Also, watch the following video: What is the Pumping Lemma

Assume L is a regular language and there is a string $w \in L$. Now, imagine, not find or determine, that there is a integer n that is $n \geq 1$, $n > 0$. Also, this integer is smaller than the length of the string w, $n \leq |w|$. So,

$$|w| \geq n \geq 1 \qquad \text{or} \qquad |w| \geq n > 0$$

Also, notice that the string can be written as three parts: $w = xyz$.
If the followings are satisfied, it shows that $w \in L$ and L is regular:

- $|y| > 0$: y is not empty string
- $|xy| \leq n$: the length of the part of the string xy is less than n
- $xy^iz \in L$, for each $i \geq 0$: When part y is repeated, or pumped, as many times we want, the expression is still in language

In here, x or z can be empty string, e, but y cannot be the empty string, e. Notice that the chosen part of string that will be compared with n should be located in first n.

□

Pumping lemma is also used to show that a language is not regular: start applying the pumping lemma, then find a contradiction.

For each regular language L,
  there exists $n \geq 1$ such that (this is a general term do not pick a number!!!)
    for each $w \in L$ with $|w| \geq n$ (write a string with respect to n, that can be used to reach a contradiction)
      there exists x, y, z with $w = xyz$, $y \neq e$, $|xy| \leq n$ (consider each possible split satisfying these constraints)
        for each $i \geq 0$, $xy^iz \in L$ (show that there exists an i such that $xy^iz \neq L$, contradiction)

Applying the theorem correctly can be subtle. It is often useful to think of the application of this result as a *game* between yourself, the prover, who is striving to establish that the given language L is not regular, and an adversary who is insisting that L is regular.

The theorem states that, once L has been fixed,
the adversary must start by providing a number n;
then you come up with a string w in the language that is longer than n, $|w| \geq n$;
the adversary must now supply an appropriate decomposition of w into xyz;
and, finally, you triumphantly point out i for which xy^iz is not in the language.

If you have a strategy that always wins, no matter how brilliantly the adversary plays, then you have established that L is not regular.

Example 7

The language $L = \{a^ib^i \mid i \geq 0\}$ is not regular, for if it were regular, Theorem 4 would apply for some integer n. Consider then the string $w = a^nb^n \in L$. By the theorem, it can be rewritten as $w = xyz$ such that $|xy| \leq n$ and $y \neq e$ -that is, $y = a^i$ for some $i > 0$. But then $xz = a^{n-i}b^n \notin L$, $y^0 = e$, contradicting the theorem.

Example 8

The language $L = \{a^n \mid n \text{ is prime}\}$ is not regular. For suppose it were, and let x, y, and z be as specified in Theorem 4. Then $x = a^p$, $y = a^q$, and $z = a^r$, where $p, r \geq 0$ and $q > 0$. By the theorem, $xy^iz \in L$ for each $i \geq 0$; that is, $p + iq + r$ is prime for each $i \geq 0$. But this is impossible; for let $i = p + 2q + r + 2$; then $p + iq + r = (q + 1) \cdot (p + 2q + r)$, which is a product of two natural numbers, each greater than 1.

Example 9

Sometimes it pays to use closure properties to show that a language is not regular. Take for example

$$L = \left\{w \in \{a, b\}^* \mid w \text{ has an equal number of } a\text{'s and } b\text{'s}\right\}$$

L is not regular, because if L were indeed regular, then so would be $L \cap a^*b^*$ -by closure under intersection. However, this latter language is precisely $a^nb^n \mid n \geq 0$, which we just showed is not regular.

5 State Minimization

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.

The two popular methods for minimizing a DFA are

- Equivalence Theorem
- Myhill Nerode Theorem

5.1 Equivalence Theorem

Step-1

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 Σ are called as dead states.
- **Inaccessible State:** All those states which can never be reached from the initial state are called as inaccessible states.

Step-2

Draw a state transition table for the given DFA. Transition table shows the transition of all states on all input symbols in Σ.

Step-3

Now, start applying equivalence theorem.

- Take a counter variable k and initialize it with value 0.
- 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.
- This partition is called P_0.

Step-4

- Increment k by 1.
- Find P_k by partitioning the different sets of P_{k-1}.
- In each set of P_{k-1}, consider all the possible pair of states within each set and if the two states are distinguishable, partition the set into different sets in P_k.

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 in partition P_{k-1}.

Step-5

Repeat *step-4* until no change in partition occurs. In other words, when you find $P_k = P_{k-1}$, stop.

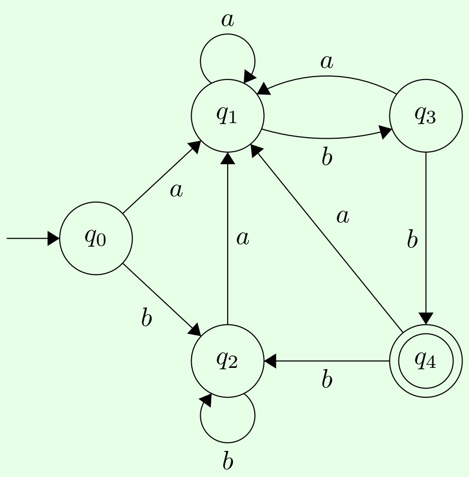
Step-6

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.

$$\textit{\# of states in Minimal DFA} = \textit{\# of sets in } P_k$$

Example 10

Minimize the following DFA:



Solution:

- Step - 1: The given DFA contains no dead states and inaccessible states.
- Step - 2: Draw a state transition table

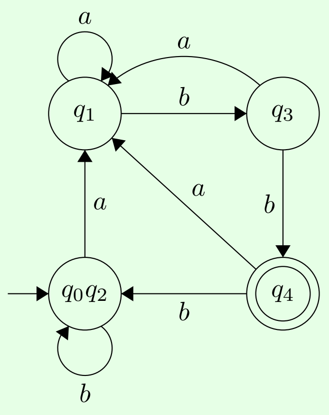
δ	a	b
q_0	q_1	q_2
q_1	q_1	q_3
q_2	q_1	q_2
q_3	q_1	q_4
q_4	q_1	q_2

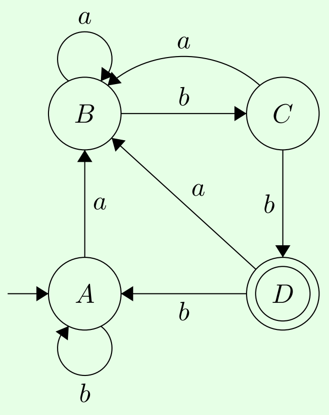
- Step - 3: Now using Equivalence Theorem, we have:

- $P_0 = \{q_0, q_1, q_2, q_3\}, \{q_4\}$
- $P_1 = \{q_0, q_1, q_2\}, \{q_3\}, \{q_4\}$
- $P_2 = \{q_0, q_2\}, \{q_1\}, \{q_3\}, \{q_4\}$
- $P_3 = \{q_0, q_2\}, \{q_1\}, \{q_3\}, \{q_4\}$

Since $P_3 = P_2$, so we stop. From P_3, we infer that states q_0 and q_2 are equivalent and can be merged together.

- So, Our minimal DFA is





5.2 Myhill Nerode Theorem

1. Create the pairs of all the states involved in the given DFA.
2. Mark all the pairs (Q_a, Q_b) such a that Q_a is Final state and Q_b is Non-Final State or vice versa.
3. If there is any unmarked pair (Q_a, Q_b) such a that $\delta(Q_a, x)$ and $\delta(Q_b, x)$ is marked, then mark (Q_a, Q_b). Here x is a input symbol. Repeat this step until no more marking can be made.
4. Combine all the unmarked pairs and make them a single state in the minimized DFA.

This way is quite cumbersome and prone to errors. Check the following links for examples:

Figure 16: https://bit.ly/3xVGyMD

Figure 17: https://bit.ly/3Oxj9Xw