1. **Problem:** Given a DFA $M$, find an equivalent DFA with a minimum number of states.

   We present two solutions to this problem. The first one is algorithmic. The second one is more conceptual and proves that the equivalent minimum state DFA is unique, up to the labeling of its states.

2. **First Method: Merging of Equivalent States**

   Let $M = (K, \Sigma, \delta, q_0, F)$ be a DFA. Given two states $q$ and $q'$ from $K$, we define an equivalence relation $\sim$ on the states of $M$ by:

   \[ q \sim q' \text{ means } \delta(q, w) \in F \iff \delta(q', w) \in F, \ \forall w \in \Sigma^*. \]

   The $\sim$-equivalence classes are computed by a sequence of other equivalence relations $\equiv_k$ by successive refinements.

   Let $q$ and $q'$ be two states of $M$. Then: $q \equiv_0 q'$ means $q \in F \iff q' \in F$. That is, $\equiv_0$ has two equivalence classes: the set of accepting states $F$ and the set of rejecting states $Q \setminus F$.

   For $k > 0$, define $\equiv_{k+1}$ to mean:

   \[ q \equiv_{n+1} q' \text{ as } q \equiv_k q' \text{ and } \delta(q, a) \equiv_n \delta(q', a), \ \forall a \in \Sigma. \]

   That is, $q \equiv_n q'$ means $\delta(q, w) \in F \iff \delta(q', w) \in F$, for all strings $w$ whose length is less than or equal to $k$.

   To see this, we argue by induction. It is easy to verify that $p \equiv_1 q$ is equivalent to $\delta(p, w) \equiv_0 \delta(q, w)$, for all $w \in \Sigma^*$ with $|w| \leq 1$.

   We make the induction hypothesis that $q \equiv_k q'$ implies that $\delta(q, w) \equiv_0 \delta(q', w)$, for all strings $w$ so $|w| \leq k$.

   For general $k$, consider $q \equiv_{k+1} q'$. Then $\delta(q, \sigma w) = \delta(\delta(q, \sigma), w)$ and $\delta(q', \sigma w) = \delta(\delta(q', \sigma), w)$. But $\delta(q, \sigma) \equiv_k \delta(q', \sigma)$. By the induction hypothesis, we find that $\delta(\delta(q, \sigma), w) \equiv_0 \delta(\delta(q', \sigma), w)$, for any string $w$ with $|w| \leq k$.  

   1
Hence, if \( q \equiv_k q' \) for all \( k \), then \( q \sim q' \).

The converse is shown similarly.

The equivalence classes of \( \equiv_k \) stabilize for \( k \) less than or equal to the number of states of the automaton \( M \). Further, \( q \sim q' \) if and only if \( q \equiv_n q' \), for all \( k \). Now, if all the states of \( M \) are reachable from the start state \( q_0 \) and if equivalent states are merged, then the resulting automaton has a minimum number of states.

These observations give rise to an effective algorithm to find the minimum state automaton, by successively computing the \( \equiv_k \)-equivalence classes, for \( k = 0, 1, 2, \ldots \). The process terminates when the equivalence classes for two successive values of \( k \) agree. We are guaranteed that the algorithm terminates when \( k \) achieves the value of the number of states of the DFA.

The algorithm below is based on finding pairs of inequivalent states for some value of \( k \).

**Algorithm for Merging Equivalent States:**

We first make a table of unordered pairs of distinct states. No pair is marked.

1. First, mark all pairs of inequivalent states relative to strings of length 0; so mark the pair \( \{p, q\} \) if \( p \in F, q \in Q \setminus F \) or \( p \in Q \setminus F, q \in F \).
2. Next, we mark all pairs of inequivalent states relative to strings of length \( k = 1, 2, \ldots, n \), where \( n \) is the total number of states of the original DFA.

\[
\text{for } k = 1..n \text{ do}
\]

if there is an unmarked pair \( \{p, q\} \), so that \( \{\delta(p, \sigma), \delta(q, \sigma)\} \) is marked, then mark the pair \( \{p, q\} \) od;

3. When the loop terminates, all inequivalent pairs are marked; so the unmarked pairs are equivalent states. Merge these pairs together.

**Notes:** At the beginning of each iteration of the loop, the marked pair \( \{p, q\} \) satisfy \( p \neq_{k-1} q \). Pairs are marked if \( \{\delta(p, \sigma), \delta(q, \sigma)\} \) is marked, that is, if \( \delta(p, \sigma) \neq_{k-1} \delta(q, \sigma) \) for some \( \sigma \in \Sigma \). By the definition, \( p \neq_k q \).

**Example:**
The accepting states are \{1, 2, 5\}.

We can represent the results of the algorithm in terms of a series of the following tables.

If we needed to consider ordered pairs of states, we would need a square table. For unordered pairs of distinct states, we need only consider the portion of \(Q \times Q\) that lies below the main diagonal. Since we are trying to determine if states need to be merged, we do not need to consider the pairs of the form \(\{p, p\}\).

The initial table of unordered pairs of distinct states is given by:

<table>
<thead>
<tr>
<th>State</th>
<th>a</th>
<th>b</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>1</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>2</td>
<td>4</td>
<td>3</td>
</tr>
<tr>
<td>3</td>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td>4</td>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td>5</td>
<td>5</td>
<td>5</td>
</tr>
</tbody>
</table>

We now mark pairs \((p, q)\) if \(p \in F\) and \(q \notin F\) or \(p \notin F\) and \(q \in F\). This distinguishes states relative to the equivalence relation \(\equiv_0\).
This table is found according to the following information:

<table>
<thead>
<tr>
<th>Computation of Pairs</th>
<th>(p, q)</th>
<th>(δ(p, a), δ(q, a))</th>
<th>(δ(p, b), δ(q, b))</th>
</tr>
</thead>
<tbody>
<tr>
<td>(0, 3)</td>
<td>(1, 5)</td>
<td>(2, 5)</td>
<td></td>
</tr>
<tr>
<td>(0, 4)</td>
<td>(1, 5)</td>
<td>(2, 5)</td>
<td></td>
</tr>
<tr>
<td>(1, 2)</td>
<td>(3, 4)</td>
<td>(3, 4)</td>
<td></td>
</tr>
<tr>
<td>(1, 5)</td>
<td>(3, 5)</td>
<td>(4, 5)</td>
<td></td>
</tr>
<tr>
<td>(2, 5)</td>
<td>(4, 5)</td>
<td>(3, 5)</td>
<td></td>
</tr>
<tr>
<td>(3, 4)</td>
<td>(5, 5)</td>
<td>(5, 5)</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Marking Pairs</th>
<th>(p, q)</th>
<th>(δ(p, a), δ(q, a))</th>
<th>(δ(p, b), δ(q, b))</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>(0, 3)</td>
<td>- - -</td>
<td>- - -</td>
</tr>
<tr>
<td></td>
<td>(0, 4)</td>
<td>- - -</td>
<td>- - -</td>
</tr>
<tr>
<td></td>
<td>(1, 2)</td>
<td>- - -</td>
<td>- - -</td>
</tr>
<tr>
<td></td>
<td>(1, 5)</td>
<td>√</td>
<td>√</td>
</tr>
<tr>
<td></td>
<td>(2, 5)</td>
<td>√</td>
<td>√</td>
</tr>
<tr>
<td></td>
<td>(3, 4)</td>
<td>- - -</td>
<td>- - -</td>
</tr>
</tbody>
</table>

Mark pairs (p, q) if (δ(p, σ), δ(q, σ)) is marked already for some σ ∈ Σ. This distinguishes states relative to the equivalence relation ≡₁. Note: we find q₁ ≠ q₅ and q₂ ≠ q₅.
Mark pairs \((p, q)\) if \((\delta(p, \sigma), \delta(q, \sigma))\) is marked already for some \(\sigma \in \Sigma\). This distinguishes states relative to the equivalence relation \(\equiv_2\). Note: we find \(q_0 \neq q_3\) and \(q_0 \neq q_4\).

For the next iteration, we find that no new pairs of states are marked. Hence, the algorithm terminates.

The result of the algorithm is seen to be that states 1 and 2 should be merged and states 3 and 4 should be merged as well.

In the above calculation, we made use of the following information.
Second Method: Construction of the Minimum State DFA directly from the Language L

Let \( M = (K, \Sigma, \delta, q_0, F) \) be a finite deterministic automaton such that all its states are reachable from its start state. Let \( L = L(M) \) be the language it accepts.

We associate with \( M \) a special equivalence relation \( R_M \) on \( \Sigma^* \), where

\[
x R_M y \iff \delta(q_0, x) = \delta(q_0, y), \quad \text{where} \quad x, y \in \Sigma^* \quad \text{that is,} \quad \text{two strings} \ x \ \text{and} \ y \ \text{are equivalent if they terminate at the same state.} \quad \text{Hence, we can identify the} \ R_M \ \text{equivalence classes} \ [x]_M \ \text{with the states of} \ M. \quad \text{The language} \ L(M) \ \text{is the union of the} \ R_M \text{-equivalence classes which include an element} \ x, \ \text{so} \ \delta(q_0, x) \ \in F. \ \text{We may call} \ R_M, \ \text{machine equivalence.}
\]

We call an equivalence relation \( R \) on \( \Sigma^* \) right-invariant if \( xRy \Rightarrow xzRyz \), for all strings \( z \in \Sigma^* \).

Note: \( R_M \) is right invariant.

Let \( L \) be any language over the alphabet \( \Sigma \), that is, \( L \subset \Sigma^* \). We can associate an equivalence relation \( R_L \) on \( \Sigma^* \) directly from \( L \), without using a finite automaton.

Given any two strings \( x, y \in \Sigma^* \), we say \( x R_L y \iff xz \in L \) exactly when \( yz \in L \), for all \( z \in \Sigma^* \).
Note: the equivalence relation $R_L$ is right invariant and $R_L$ is a refinement of $R_M$, if $L = L(M)$, for a DFA $M$. From this, it follows that there are only finitely many $R_L$-equivalence classes for any regular language since $L$ is accepted by some DFA.

Example: (1) Let $L = L((ab \cup ba)^*)$. We consider the equivalence classes of $L$ relative to $R_L$.

1. The equivalence class $[\varepsilon]$ equals $L$ itself.
   
   To see this, consider the definition of $R_L$. Let $w \in \Sigma^*$ be any fixed string. Suppose $w \in [\varepsilon]$. Now, $\varepsilon z = z \in L \iff wz \in L$, by the definition of $R_L$. But $wz \in L$, for all $z \in L$, $\iff w \in L$. (To see this, choose $z = \varepsilon$.)

2. $[a] = La$. To see this, consider: $ax \in L \iff z \in bL$. Further, for $w \in L$, $waz \in L \iff z \in bL$, as well.

3. $[b] = Lb$. This follows by a similar argument as (2).

Example: (2) Let $L = \{a^n b^n : n \geq 1 \}$. Then the equivalence classes $[a^k]$ are distinct for all $k$. In particular, $L$ has infinitely many equivalence classes. This gives a new proof that $L$ is not regular.

4. We can construct a deterministic finite automaton $M_L$ directly from the equivalence relation $R_L$, if $R_L$ has FINITE index; that is, if the number of $R_L$ equivalence classes is finite.

   We set $M_L = (K_L, \Sigma, \delta_L, s_L, F_L)$.

   Let $K_L$, the states of the machine $M_L$, be the collection of all $R_L$-equivalence classes; write them as $[x]_L$, for a string $x$.

   The transition function $\delta_L : K_L \times \Sigma \rightarrow K_L$ is given as: $\delta_L([x]_L, a) = [xa]_L$. Note: $\delta_L$ is well defined.

   Set $s_L = [\varepsilon]_L$ and $F_L = \{[x]_L : x \in L \}$.

   The minimum state automaton accepting $L$ is given by $M_L$, further, any other minimum state automaton that accepts $L$ can be identified with $M_L$, by a re-labeling of its states.
Myhill-Nerode Theorem: Let $L \subseteq \Sigma^*$. Then $L$ is regular if and only if $R_L$ has at most finitely many equivalence classes.

The regular languages are closed under homomorphisms.

A homomorphism is a map $h : \Sigma^* \to \cdot$ such that for all $x, y \in \Sigma^*$ we have that

$$h(xy) = h(x)h(y) \text{ and } h(\epsilon) = \epsilon.$$  

It follows at once that the values of $h$ are determined on any string once they are known for the letters of $\Sigma^*$.

**Proposition.** Let $h : \Sigma^* \to \cdot$ be a homomorphism. Let $L \subseteq \Sigma^*$ be a regular language. Then $h(L)$ is also regular.

**Sketch of Proof:** We use the fact that $L$ is denoted by some regular expression. Then the argument reduces to the following formula:

$$L(h(\beta)) = h(L(\beta)),$$

for any regular expression $\beta$. This formula can be established by induction on the number of operators in the regular expression $\beta$.

The base case of zero operators corresponds to $\beta = \sigma \in \Sigma, \epsilon$ or $\emptyset$. In all three cases, the desired formula holds.

We next need to make the observations that

1. $h(L_1 \cdot L_2) = h(L_1) \cdot h(L_2)$,
2. $h(\bigcup L_k) = \bigcup h(L_k)$,
3. $h(L^*) = h(L^*)$.

The result now follows in a routine fashion.

**Proposition.** Let $h : \Sigma^* \to \cdot$ be a homomorphism. Let $L' \subseteq \cdot$ be a regular language. Then so is $h^{-1}(L')$.  

8
Proof: We use the fact that \( L' \) is accepted by some DFA \( M' = (K', \delta', s', F') \). Let \( M \) denote the DFA that will accept \( h^{-1}(L') \), where \( M = (K, \Sigma, \delta, s, F) \) where \( K = K' \), \( F = F' \), and \( s = s' \). We define
\[
\delta(q, a) = \delta'(q, h(a)).
\]
We can establish by induction that
\[
\delta(q, x) = \delta'(q, h(x)), \text{ where } x \in \Sigma^*.
\]
Finally, we observe that \( x \in L(M) \iff \delta(s, x) \in F \iff \delta(s, h(x)) \in F \iff h(x) \in L(M') \iff x \in h^{-1}(L(M')). \)

Showing Languages are Not Regular:

Example (1) Let \( L = \{a^n b^m : m \neq n \} \). Show \( L \) is not regular. We consider the language \( \mathcal{T} \cap L(a^* b^*) \). Then this intersection can be computed to be \( \{a^n b^n : n \geq 0 \} \). Of course, this last language is not regular. Suppose \( L \) is regular. Then so is \( \mathcal{T} \). Also, \( L(a^* b^*) \) is a regular language. So, by the closure of regular languages under set operations, we find that \( \{a^n b^n : n \geq 0 \} \) is regular. Contradiction.

It is possible to apply the pumping lemma to this example but it is messy. The usual choice of the string to pump is \( a^{|i|} b^{|i|+1} \).

Example (2) Let \( L = \{a^m b^k c^{n+k} : n \geq 0, k \geq 0 \} \). Show that \( L \) is not regular. Suppose \( L \) is regular. Then any homomorphic image of \( L \) is also regular. Consider the homomorphism \( h \) defined by \( h(a) = a, h(b) = a, h(c) = c \). Then we find that \( h(L) = \{a^i b^i : i \geq 0 \} \), which is not regular.

Characterization of Regular Languages over One Letter Alphabets: Let \( L \) be a language over the one letter alphabet \( \{a\} \). Then \( L \) is regular if and only if the set of non-negative integers \( U = \{m : a^m \in L\} \) is ultimately periodic; that is there are integers \( n \geq 0 \) and \( p > 0 \) such that for all \( m \geq n, m \in U \) if and only if \( m + p \in U \). We call the number \( p \) the period of \( U \).
Proof: We use the fact that a regular language is accepted by a DFA. Suppose $L = L(M)$, for some DFA $M$. By construction, there is exactly one edge out of any state, which must be labeled by $a$. Without loss of generality, we shall assume all states of $M$ are accessible from the start state. Suppose the states of $M$ are $\{q_1, q_2, \ldots, q_n\}$ where $q_1$ is the start state and that the indexing is chosen so $\delta(q_i, a) = q_{i+1}, 1 \leq i \leq n - 1$. Note: we must have $\delta(q_n, a) = q_k$ for some $k$ with $1 \leq k \leq n$. In particular, the state space consists of an initial segment of length $k - 1$ and a loop of length $p = n - k + 1$.

**Observation:** For $a^i, i \geq k - 1$, $a^i \in L(M) \iff a^{i+p} \in L(M)$. This follows since $\delta(q_1, a^i) = \delta(q_1, a^{i+p})$.

Hence, the set of integers $\{i : a^i \in L\}$ is ultimately periodic.

We leave the converse to the reader. The proof rests on constructing the appropriate DFA.

Complexity of Algorithms for Finite Automata and Regular Languages:

(a) There is an exponential algorithm which constructs an equivalent deterministic finite automaton for a given non-deterministic finite automaton.

(b) There is an exponential algorithm that constructs an equivalent regular expression for a given non-deterministic automaton.

(c) There is a polynomial algorithm which constructs an equivalent non-deterministic finite automaton for a given regular expression.

(d) There is a polynomial algorithm that constructs a minimum state deterministic finite automaton for a given deterministic finite automaton.

(e) There is a polynomial algorithm which decides when two deterministic finite automata are equivalent.

(f) There is an exponential algorithm which decides when two non-deterministic finite automata are equivalent.

(g) For a given regular language $L$ and string $w$, there is an algorithm that decides if $w$ belongs to $L$ whose complexity is linear in the length of $w$. 

10
(h) For a given regular language $L = L(M)$, where $M$ is a deterministic finite automaton, and string $w$, there is an algorithm that decides if $w$ belongs to $L$ whose complexity is linear in the length of $w$.

For a given regular language $L = L(M)$, where $M$ is a non-deterministic finite automaton, and string $w$, there is an algorithm that decides if $w$ belongs to $L$ whose complexity is $O(|K|^2 |w|)$. 