Chapter 6

Regular Language and Right-Invariant Equivalence Relations

6.1 Right-Invariant Equivalence Relations on Σ^*

Let $D = (Q, \Sigma, \delta, q_0, F)$ be a DFA. The DFA D may be redundant, for example, if there are states that are not accessible from the start state.

The set Q_r of *accessible or reachable states* is the subset of Q defined as

$$Q_r = \{p \in Q \mid \exists w \in \Sigma^*,\ \delta^*(q_0, w) = p\}.$$

The set Q_r can be easily computed by stages.

If $Q \neq Q_r$, we can "clean up" D by deleting the states in $Q - Q_r$ and restricting the transition function δ to Q_r.

This way, we get an equivalent DFA D_r such that $L(D) = L(D_r)$, where all the states of D_r are reachable. From now on, we assume that we are dealing with DFA's such that $D = D_r$ (called *reachable, or trim*).

Recall that an *equivalence relation* $\simeq$ on a set A is a relation which is *reflexive*, *symmetric*, and *transitive*.

Given any $a \in A$, the set

$$\{b \in A \mid a \simeq b\}$$

is called the *equivalence class of a*, and it is denoted as $[a]_{\simeq}$, or even as $[a]$.

Recall that for any two elements $a, b \in A$, $[a] \cap [b] = \emptyset$ iff $a \not\simeq b$, and $[a] = [b]$ iff $a \simeq b$.

The set of equivalence classes associated with the equivalence relation $\simeq$ is a *partition* Π of A (also denoted as $A/\simeq$). This means that it is a family of nonempty pairwise disjoint sets whose union is equal to A itself.

The equivalence classes are also called the *blocks* of the partition Π. The number of blocks in the partition Π is called the *index* of $\simeq$ (and Π).

Given any two equivalence relations $\simeq_1$ and $\simeq_2$ with associated partitions Π_1 and Π_2,

$$\simeq_1 \subseteq \simeq_2$$

iff every block of the partition Π_1 is contained in some block of the partition Π_2. Then, every block of the partition Π_2 is the union of blocks of the partition Π_1, and we say that $\simeq_1$ is a *refinement* of $\simeq_2$ (and similarly, Π_1 is a refinement of Π_2).

Note that Π_2 has at most as many blocks as Π_1 does.

We now define an equivalence relation on strings induced by a DFA. This equivalence is a kind of "observational" equivalence, in the sense that we decide that two strings u, v are equivalent iff, when feeding first u and then v to the DFA, u and v drive the DFA to the same state. From the point of view of the observer, u and v have the same effect (reaching the same state).

Definition 6.1. Given a DFA $D = (Q, \Sigma, \delta, q_0, F)$, we define the relation $\simeq_D$ (*Myhill-Nerode equivalence*) on Σ^* as follows: for any two strings $u, v \in \Sigma^*$,

$$u \simeq_D v \quad \text{iff} \quad \delta^*(q_0, u) = \delta^*(q_0, v).$$

Example 6.1. We can figure out what the equivalence classes of $\simeq_D$ are for the following DFA:

	a	b
0	1	0
1	2	1
2	0	2

with 0 both start state and (unique) final state. For example

$$\begin{aligned} abbabbb &\simeq_D aa \\ ababab &\simeq_D \epsilon \\ bba &\simeq_D a. \end{aligned}$$

There are three equivalences classes:

$$[\epsilon]_{\simeq}, \quad [a]_{\simeq}, \quad [aa]_{\simeq}.$$

Observe that $L(D) = [\epsilon]_{\simeq}$. Also, the equivalence classes are in one–to–one correspondence with the states of D.

The relation $\simeq_D$ turns out to have some interesting properties.

In particular, it is *right-invariant*, which means that for all $u, v, w \in \Sigma^*$, if $u \simeq v$, then $uw \simeq vw$.

Proposition 6.1. *Given any trim DFA* $D = (Q, \Sigma, \delta, q_0, F)$, *the relation* $\simeq_D$ *is an equivalence relation which is right-invariant and has finite index. Furthermore, if* Q *has* n *states, then the index of* $\simeq_D$ *is* n, *and every equivalence class of* $\simeq_D$ *is a regular language. Finally,* $L(D)$ *is the union of some of the equivalence classes of* $\simeq_D$.

One should not be too optimistic and hope that every equivalence relation on strings is right-invariant.

Example 6.2. For example, if $\Sigma = \{a\}$, the equivalence relation $\simeq$ given by the partition

$$\{\epsilon, a, a^4, a^9, a^{16}, \ldots, a^{n^2}, \ldots \mid n \geq 0\}$$
$$\cup \{a^2, a^3, a^5, a^6, a^7, a^8, \ldots, a^m, \ldots \mid m \text{ is not a square}\}$$

we have $a \simeq a^4$, yet by concatenating on the right with a^5, since $aa^5 = a^6$ and $a^4a^5 = a^9$ we get

$$a^6 \not\simeq a^9,$$

that is, a^6 and a^9 are *not* equivalent. It turns out that the problem is that neither equivalence class is a regular language.

It is worth noting that a right-invariant equivalence relation is *not necessarily left-invariant*, which means that if $u \simeq v$ then $wu \simeq wv$.

Example 6.3. For example, if $\simeq$ is given by the four equivalence classes

$$C_1 = \{bb\}^*, \;\; C_2 = \{bb\}^* a, \;\; C_3 = b\{bb\}^*,$$
$$C_4 = \{bb\}^* a\{a, b\}^+ \cup b\{bb\}^* a\{a, b\}^*,$$

then we can check that $\simeq$ is right-invariant by figuring out the inclusions $C_i a \subseteq C_j$ and $C_i b \subseteq C_j$, which are recorded in the following table:

	a	b
C_1	C_2	C_3
C_2	C_4	C_4
C_3	C_4	C_1
C_4	C_4	C_4

However, both $ab, ba \in C_4$, yet $bab \in C_4$ and $bba \in C_2$, so $\simeq$ is not left-invariant.

The remarkable fact due to Myhill and Nerode, is that Proposition 6.1 has a converse.

Example 6.4. Consider the equivalence relation $\simeq$ on $\{a, b\}^*$ given by the three equivalence classes

$$C_1 = \{\epsilon\},\ \ C_2 = a\{a, b\}^*,\ \ C_3 = b\{a, b\}^*.$$

We leave it as an easy exercise to check that $\simeq$ is right-invariant.

For any subset $C \subseteq \{a, b\}^*$ and any string $w \in \{a, b\}^*$ define Cw as the set of strings

$$Cw = \{uw \mid u \in C\}.$$

There are two reasons why a DFA can be recovered from the right-invariant equivalence relation $\simeq$:

(1) For every equivalence class C_i and every string w, there is a unique equivalence class C_j such that

$$C_i w \subseteq C_j.$$

Actually, it is enough to check the above property for strings w of length 1 (*i.e.* symbols in the alphabet) because the property for arbitrary strings follows by induction.

(2) For every $w \in \Sigma^*$ and every class C_i,

$$C_1 w \subseteq C_i \quad \text{iff} \quad w \in C_i,$$

where C_1 is the equivalence class of the empty string.

We can make a table recording these inclusions.

Example 6.5. Continuing Example 6.4, we get:

	a	b
C_1	C_2	C_3
C_2	C_2	C_2
C_3	C_3	C_3

The key point is that the above table is the transition table of a DFA with start state $C_1 = [\epsilon]$. Furthermore, if C_i ($i = 1, 2, 3$) is chosen as a single final state, the corresponding DFA D_i accepts C_i. This is the converse of Myhill-Nerode!

Observe that the inclusions $C_i w \subseteq C_j$ may be strict inclusions. For example, $C_1 a = \{a\}$ is a proper subset of $C_2 = a\{a, b\}^*$

Example 6.6. Consider the equivalence relation $\simeq$ given by the four equivalence classes

$$\begin{aligned} &C_1 = \{\epsilon\},\ \ C_2 = \{a\}, \\ &C_3 = \{b\}^+,\ \ C_4 = a\{a,b\}^+ \cup \{b\}^+a\{a,b\}^*. \end{aligned}$$

We leave it as an easy exercise to check that $\simeq$ is right-invariant.

We obtain the following table of inclusions $C_ia \subseteq C_j$ and $C_ib \subseteq C_j$:

	a	b
C_1	C_2	C_3
C_2	C_4	C_4
C_3	C_4	C_3
C_4	C_4	C_4

The above table is the transition function of a DFA with four states and start state C_1. If C_i $(i = 1, 2, 3, 4)$ is chosen as a single final state, the corresponding DFA D_i accepts C_i.

Proposition 6.2. *Given any equivalence relation $\simeq$ on Σ^*, if $\simeq$ is right-invariant and has finite index n, then every equivalence class (block) in the partition Π associated with $\simeq$ is a regular language.*

Proof. Let $C_1, \ldots, C_n$ be the blocks of Π, and assume that $C_1 = [\epsilon]$ is the equivalence class of the empty string.

First, we claim that for every block C_i and every $w \in \Sigma^*$, there is a unique block C_j such that $C_i w \subseteq C_j$, where $C_i w = \{uw \mid u \in C_i\}$.

We also claim that for every $w \in \Sigma^*$, for every block C_i,

$$C_1 w \subseteq C_i \quad \text{iff} \quad w \in C_i.$$

For every class C_k, let

$$D_k = (\{1, \ldots, n\}, \Sigma, \delta, 1, \{k\}),$$

where $\delta(i, a) = j$ iff $C_i a \subseteq C_j$.

Using induction, it can be shown that

$$\delta^*(i, w) = j \quad \text{iff} \quad C_i w \subseteq C_j. \tag{$*$}$$

For this, we prove by induction on $|w|$ that

(a) If $\delta^*(i, w) = j$, then $C_i w \subseteq C_j$.

(b) If $C_i w \subseteq C_j$, then $\delta^*(i, w) = j$.

Proving (b) is a little harder than proving (a).

Using $(*)$ and claim 2, it is not hard to verify that $L(D_k) = C_k$, proving that every block C_k is a regular language. $\square$

We can combine Proposition 6.1 and Proposition 6.2 to get the following characterization of a regular language due to Myhill and Nerode:

Theorem 6.3. *(Myhill-Nerode) A language L (over an alphabet Σ) is a regular language iff it is the union of some of the equivalence classes of an equivalence relation $\simeq$ on Σ^*, which is right-invariant and has finite index.*

Theorem 6.3 can also be used to prove that certain languages are not regular.

The scenario is this: to prove that L is not regular, first we check that L is infinite.

If so, we try finding three strings x, y, z, where and x and $y \neq x$ are prefixes of strings in L such that

$$x \simeq y,$$

where $\simeq$ is a right-invariant relation of finite index such that L is the union of equivalence of L (which must exist by Myhill–Nerode since we are assuming by contradiction that L is regular), and where z is chosen so that

$$xz \in L \quad \text{and} \quad yz \notin L.$$

Example 6.7. We prove that $L = \{a^n b^n \mid n \geq 1\}$ is not regular.

Assuming for the sake of contradiction that L is regular, there is some equivalence relation $\simeq$ which is right-invariant and of finite index and such that L is the union of some of the classes of $\simeq$.

Since the sequence

$$a, aa, aaa, \ldots, a^i, \ldots$$

is infinite and $\simeq$ has a finite number of classes, two of these strings must belong to the same class, which means that

$$a^i \simeq a^j \quad \text{for some } i \neq j.$$

But since $\simeq$ is right invariant, by concatenating with $z = b^i$ on the right, we see that

$$a^i b^i \simeq a^j b^i \quad \text{for some } i \neq j.$$

However $a^i b^i \in L$, and $a^j b^i \notin L$ since $i \neq j$, which is absurd. Thus, in fact, L is not regular.

Here is another illustration of the use of the Myhill-Nerode Theorem to prove that a language is not regular.

Example 6.8. We claim that the language,

$$L' = \{a^{n!} \mid n \geq 1\},$$

is not regular, where $n!$ (n factorial) is given by $0! = 1$ and $(n+1)! = (n+1)n!$.

Assume L' is regular. Then, there is some equivalence relation $\simeq$ which is right-invariant and of finite index and such that L' is the union of some of the classes of $\simeq$.

Since the sequence

$$a, a^2, \ldots, a^n, \ldots$$

is infinite, two of these strings must belong to the same class, which means that

$$a^p \simeq a^q \quad \text{for some } p, q \text{ with } 1 \leq p < q.$$

As $q! \geq q$ for all $q \geq 0$ and $q > p$, we can concatenate on the right with $a^{q!-p}$ and we get

$$a^p a^{q!-p} \simeq a^q a^{q!-p},$$

that is,

$$a^{q!} \simeq a^{q!+q-p}.$$

Since $p < q$ we have $q! < q! + q - p$, so if we can prove that

$$q! + q - p < (q+1)!,$$

then we have a contradiction ($q!+q-p$ is not a factorial). Since $p \geq 1$, it suffices to show that

$$q! + q - 1 < (q+1)!,$$

which is easily done (this reduces to showing that

$$q - 1 < qq!).$$

There is another version of the Myhill-Nerode Theorem involving congruences which is also quite useful.

An equivalence relation, $\simeq$, on Σ^* is *left and right-invariant* iff for all $x, y, u, v \in \Sigma^*$,

$$\text{if} \quad x \simeq y \quad \text{then} \quad uxv \simeq uyv.$$

An equivalence relation, $\simeq$, on Σ^* is a *congruence* iff for all $u_1, u_2, v_1, v_2 \in \Sigma^*$,

$$\text{if} \quad u_1 \simeq v_1 \quad \text{and} \quad u_2 \simeq v_2 \quad \text{then} \quad u_1u_2 \simeq v_1v_2.$$

It is easy to prove that an equivalence relation is a congruence iff it is left and right-invariant.

For example, assume that $\simeq$ is a left and right-invariant equivalence relation, and assume that

$$u_1 \simeq v_1 \quad \text{and} \quad u_2 \simeq v_2.$$

By right-invariance applied to $u_1 \simeq v_1$, we get

$$u_1u_2 \simeq v_1u_2$$

and by left-invariance applied to $u_2 \simeq v_2$ we get

$$v_1u_2 \simeq v_1v_2.$$

By transitivity, we conclude that

$$u_1u_2 \simeq v_1v_2.$$

which shows that $\simeq$ is a congruence.

Proving that a congruence is left and right-invariant is even easier.

There is a version of Proposition 6.1 that applies to congruences and for this we define the relation $\sim_D$ as follows: For any (trim) DFA, $D = (Q, \Sigma, \delta, q_0, F)$, for all $x, y \in \Sigma^*$,

$$x \sim_D y \quad \text{iff} \quad (\forall q \in Q)(\delta^*(q, x) = \delta^*(q, y)).$$

Proposition 6.4. *Given any (trim) DFA $D = (Q, \Sigma, \delta, q_0, F)$, the relation $\sim_D$ is an equivalence relation which is left and right-invariant and has finite index. Furthermore, if Q has n states, then the index of $\sim_D$ is at most n^n and every equivalence class of $\sim_D$ is a regular language. Finally, $L(D)$ is the union of some of the equivalence classes of $\sim_D$.*

Using Proposition 6.4 and Proposition 6.2, we obtain another version of the Myhill-Nerode Theorem.

Theorem 6.5. *(Myhill-Nerode, Congruence Version) A language L (over an alphabet Σ) is a regular language iff it is the union of some of the equivalence classes of an equivalence relation $\simeq$ on Σ^*, which is a congruence and has finite index.*

6.2 Finding Minimal DFA's

Given any language L (not necessarily regular), we can define an equivalence relation ρ_L on Σ^* which is right-invariant, but not necessarily of finite index.

The equivalence relation ρ_L is such that L is the union of equivalence classes of ρ_L.

Furthermore, when L is regular, the relation ρ_L has finite index. In fact, this index is the size of a smallest DFA accepting L.

As a consequence, if L is regular, a simple modification of the proof of Proposition 6.2 applied to $\simeq\, = \,\rho_L$ yields a minimal DFA D_{ρ_L} accepting L.

Then, given any trim DFA D accepting L, the equivalence relation ρ_L can be translated to an equivalence relation $\equiv$ on states, in such a way that for all $u, v \in \Sigma^*$,

$$u \rho_L v \quad \text{iff} \quad \varphi(u) \equiv \varphi(v),$$

where $\varphi\colon \Sigma^* \to Q$ is the function (run the DFA D on u from q_0) given by

$$\varphi(u) = \delta^*(q_0, u).$$

One can then construct a quotient DFA $D/\equiv$ whose states are obtained by merging all states in a given equivalence class of states into a single state, and the resulting DFA $D/\equiv$ is a mininal DFA.

Even though $D/\equiv$ appears to depend on D, it is in fact unique, and isomorphic to the abstract DFA D_{ρ_L} induced by ρ_L.

The last step in obtaining the minimal DFA $D/\equiv$ is to give a constructive method to compute the state equivalence relation $\equiv$.

This can be done by constructing a sequence of approximations $\equiv_i$, where each $\equiv_{i+1}$ refines $\equiv_i$.

It turns out that if D has n states, then there is some index $i_0 \leq n-2$ such that

$$\equiv_j = \equiv_{i_0} \quad \text{for all } j \geq i_0 + 1,$$

and that

$$\equiv = \equiv_{i_0} .$$

Furthermore, $\equiv_{i+1}$ can be computed inductively from $\equiv_i$. In summary, we obtain a iterative algorithm for computing $\equiv$ that terminates in at most $n-2$ steps.

Definition 6.2. Given any language L (over Σ), we define the *right-invariant equivalence ρ_L associated with L* as the relation on Σ^* defined as follows: for any two strings $u, v \in \Sigma^*$,

$$u \rho_L v \quad \text{iff} \quad \forall w \in \Sigma^* (uw \in L \quad \text{iff} \quad vw \in L).$$

We leave as an easy exercise to prove that ρ_L is an equivalence relation which is right-invariant. It is also easy to see that L is the union of the equivalence classes of strings in L.

Example 6.9. For example, consider the regular language

$$L = \{a\} \cup \{b^m \mid m \geq 1\}.$$

We leave it as an exercise to show that the equivalence relation ρ_L consists of the four equivalence classes

$$\begin{aligned} &C_1 = \{\epsilon\}, \;\; C_2 = \{a\}, \\ &C_3 = \{b\}^+, \;\; C_4 = a\{a, b\}^+ \cup \{b\}^+ a\{a, b\}^* \end{aligned}$$

encountered earlier in Example 6.6. Observe that

$$L = C_2 \cup C_3.$$

When L is regular, we have the following remarkable result:

Proposition 6.6. *Given any regular language L, for any (trim) DFA $D = (Q, \Sigma, \delta, q_0, F)$ such that $L = L(D)$, ρ_L is a right-invariant equivalence relation, and we have $\simeq_D \subseteq \rho_L$. Furthermore, if ρ_L has m classes and Q has n states, then $m \leq n$.*

Proposition 6.6 shows that when L is regular, the index m of ρ_L is finite, and it is a lower bound on the size of all DFA's accepting L.

It remains to show that a DFA with m states accepting L exists.

However, going back to the proof of Proposition 6.2 starting with the right-invariant equivalence relation ρ_L of finite index m, if L is the union of the classes $C_{i_1}, \ldots, C_{i_k}$, the DFA

$$D_{\rho_L} = (\{1, \ldots, m\}, \Sigma, \delta, 1, \{i_1, \ldots, i_k\}),$$

where $\delta(i, a) = j$ iff $C_i a \subseteq C_j$, is such that $L = L(D_{\rho_L})$. Thus, D_{ρ_L} is a minimal DFA accepting L.

Example 6.10. For example, if

$$L = \{a\} \cup \{b^m \mid m \geq 1\}.$$

then we saw in Example 6.9 that ρ_L consists of the four equivalence classes

$$\begin{aligned} &C_1 = \{\epsilon\},\;\; C_2 = \{a\}, \\ &C_3 = \{b\}^+,\;\; C_4 = a\{a,b\}^+ \cup \{b\}^+ a\{a,b\}^*, \end{aligned}$$

and we showed in Example 6.6 that the transition table of D_{ρ_L} is given by

	a	b
C_1	C_2	C_3
C_2	C_4	C_4
C_3	C_4	C_3
C_4	C_4	C_4

By picking the final states to be C_2 and C_3, we obtain the minimal DFA D_{ρ_L} accepting $L = \{a\} \cup \{b^m \mid m \geq 1\}$.

In the next section, we give an algorithm which allows us to find D_{ρ_L}, given any DFA D accepting L. This algorithms finds which states of D are equivalent.

6.3 State Equivalence and Minimal DFA's

The proof of lemma 6.6 suggests the following definition of an equivalence between states.

Definition 6.3. Given any DFA $D = (Q, \Sigma, \delta, q_0, F)$, the relation $\equiv$ on Q, called *state equivalence*, is defined as follows: for all $p, q \in Q$,

$$p \equiv q \text{ iff } \forall w \in \Sigma^*(\delta^*(p, w) \in F \text{ iff } \delta^*(q, w) \in F). \quad (*)$$

When $p \equiv q$, we say that *p and q are indistinguishable*.

It is trivial to verify that $\equiv$ is an equivalence relation, and that it satisfies the following property:

$$\text{if } p \equiv q \text{ then } \delta(p, a) \equiv \delta(q, a),$$

for all $a \in \Sigma$.

Since condition $(*)$ in Definition 6.3 must hold for $w = \epsilon$, in this case we get

$$p \in F \quad \text{iff} \quad q \in F.$$

Therefore, if two states p, q are equivalent, then either both $p, q \in F$ or both $p, q \in \overline{F}$. This implies that a final state and a rejecting states are *never* equivalent.

Example 6.11. In the DFA of Figure 6.1, states A and C are equivalent. No other two states are equivalent.

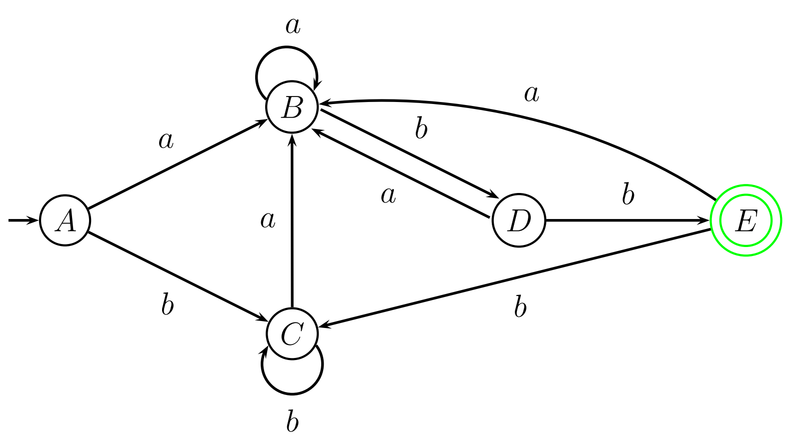

Figure 6.1: A non-minimal DFA for $\{a, b\}^*\{abb\}$

It is illuminating to express state equivalence as the equality of two languages.

Given the DFA $D = (Q, \Sigma, \delta, q_0, F)$, let $D_p = (Q, \Sigma, \delta, p, F)$ be the DFA obtained from D by redefining the start state to be p. Then, it is clear that

$$p \equiv q \quad \text{iff} \quad L(D_p) = L(D_q).$$

If $L = L(D)$, Theorem 6.7 below shows the relationship between ρ_L and $\equiv$ and, more generally, between the DFA D_{ρ_L} and the DFA $D/\equiv$, obtained as the quotient of the DFA D modulo the equivalence relation $\equiv$ on Q.

The minimal DFA $D/\equiv$ is obtained by merging the states in each block C_i of the partition Π associated with $\equiv$, forming states corresponding to the blocks C_i, and drawing a transition on input a from a block C_i to a block C_j of Π iff there is a transition $q = \delta(p, a)$ from any state $p \in C_i$ to any state $q \in C_j$ on input a.

The start state is the block containing q_0, and the final states are the blocks consisting of final states.

Example 6.12. For example, consider the DFA D_1 accepting $L = \{ab, ba\}^*$ shown in Figure 6.2.

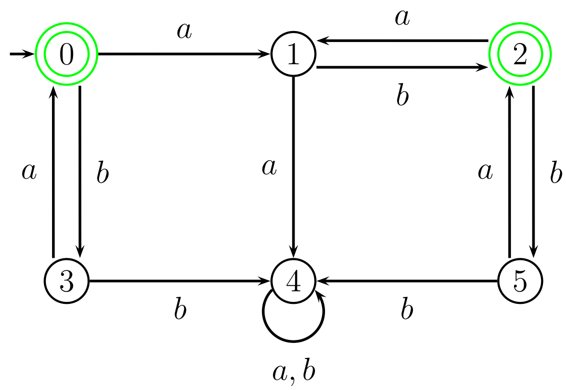

Figure 6.2: A nonminimal DFA D_1 for $L = \{ab, ba\}^*$

This is not a minimal DFA. In fact,

$$0 \equiv 2 \quad \text{and} \quad 3 \equiv 5.$$

Here is the minimal DFA for L:

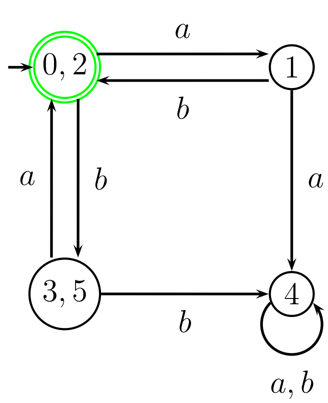

Figure 6.3: A minimal DFA D_2 for $L = \{ab, ba\}^*$

The minimal DFA D_2 is obtained by merging the states in the equivalence class $\{0, 2\}$ into a single state, similarly merging the states in the equivalence class $\{3, 5\}$ into a single state, and drawing the transitions between equivalence classes. We obtain the DFA shown in Figure 6.3.

Formally, the quotient DFA $D/\equiv$ is defined such that

$$D/\equiv = (Q/\equiv, \Sigma, \delta/\equiv, [q_0]_\equiv, F/\equiv),$$

where

$$\delta/\equiv([p]_\equiv, a) = [\delta(p, a)]_\equiv.$$

Theorem 6.7. *For any (trim) DFA $D = (Q, \Sigma, \delta, q_0, F)$ accepting the regular language $L = L(D)$, the function $\varphi\colon \Sigma^* \to Q$ defined such that*

$$\varphi(u) = \delta^*(q_0, u)$$

satisfies the property

$$u\rho_L v \quad \textit{iff} \quad \varphi(u) \equiv \varphi(v) \quad \textit{for all } u, v \in \Sigma^*,$$

and induces a bijection $\widehat{\varphi}\colon \Sigma^*/\rho_L \to Q/\equiv$, defined such that

$$\widehat{\varphi}([u]_{\rho_L}) = [\delta^*(q_0, u)]_{\equiv}.$$

Furthermore, we have

$$[u]_{\rho_L} a \subseteq [v]_{\rho_L} \quad \textit{iff} \quad \delta(\varphi(u), a) \equiv \varphi(v).$$

Consequently, $\widehat{\varphi}$, induces an isomorphism of DFA's. $\widehat{\varphi}\colon D_{\rho_L} \to D/\equiv$

Theorem 6.7 shows that the DFA D_{ρ_L} is isomorphic to the DFA $D/\equiv$ obtained as the quotient of the DFA D modulo the equivalence relation $\equiv$ on Q. Since D_{ρ_L} is a minimal DFA accepting L, so is $D/\equiv$.

Example 6.13. Consider the following DFA D,

	a	b
1	2	3
2	4	4
3	4	3
4	5	5
5	5	5

with start state 1 and final states 2 and 3. It is easy to see that

$$L(D) = \{a\} \cup \{b^m \mid m \geq 1\}.$$

It is not hard to check that states 4 and 5 are equivalent, and no other pairs of distinct states are equivalent.

The quotient DFA $D/\equiv$ is obtained my merging states 4 and 5, and we obtain the following minimal DFA:

	a	b
1	2	3
2	4	4
3	4	3
4	4	4

with start state 1 and final states 2 and 3. This DFA is isomorphic to the DFA D_{ρ_L} of Example 6.10.

There are other characterizations of the regular languages.

Among those, the characterization in terms of right derivatives is of particular interest because it yields an alternative construction of minimal DFA's.

Definition 6.4. Given any language, $L \subseteq \Sigma^*$, for any string, $u \in \Sigma^*$, the *right derivative of L by u*, denoted L/u, is the language

$$L/u = \{w \in \Sigma^* \mid uw \in L\}.$$

Theorem 6.8. *If $L \subseteq \Sigma^*$ is any language, then L is regular iff it has finitely many right derivatives. Furthermore, if L is regular, then all its right derivatives are regular and their number is equal to the number of states of the minimal DFA's for L.*

Note that if $F = \emptyset$, then $\equiv$ has a single block (Q), and if $F = Q$, then $\equiv$ has a single block (F). In the first case, the minimal DFA is the one state DFA rejecting all strings. In the second case, the minimal DFA is the one state DFA accepting all strings.

When $F \neq \emptyset$ and $F \neq Q$, there are at least two states in Q, and $\equiv$ also has at least two blocks, as we shall see shortly.

It remains to compute $\equiv$ explicitly. This is done using a sequence of approximations. In view of the previous discussion, we are assuming that $F \neq \emptyset$ and $F \neq Q$, which means that $n \geq 2$, where n is the number of states in Q.

Definition 6.5. Given any DFA $D = (Q, \Sigma, \delta, q_0, F)$, for every $i \geq 0$, the relation $\equiv_i$ on Q, called *i-state equivalence*, is defined as follows: for all $p, q \in Q$,

$$\begin{aligned} p \equiv_i q \quad \text{iff} \quad & \forall w \in \Sigma^*,\ |w| \leq i \\ & (\delta^*(p, w) \in F \quad \text{iff} \quad \delta^*(q, w) \in F). \end{aligned}$$

When $p \equiv_i q$, we say that
p and q are i-indistinguishable.

It remains to compute $\equiv_{i+1}$ from $\equiv_i$, which can be done using the following proposition. The proposition also shows that

$$\equiv = \equiv_{i_0} .$$

Proposition 6.9. *For any (trim) DFA $D = (Q, \Sigma, \delta, q_0, F)$, for all $p, q \in Q$,*

$p \equiv_{i+1} q$ iff $p \equiv_i q$ and $\delta(p, a) \equiv_i \delta(q, a)$, for every $a \in \Sigma$.

Furthermore, if $F \neq \emptyset$ and $F \neq Q$, there is a smallest integer $i_0 \leq n - 2$, such that

$$\equiv_{i_0+1} = \equiv_{i_0} = \equiv .$$

Note that if $F = Q$ or $F = \emptyset$, then $\equiv = \equiv_0$, and the inductive characterization of Lemma 6.9 holds trivially.

Using proposition 6.9, we can compute $\equiv$ inductively, starting from $\equiv_0 = (F, Q - F)$, and computing $\equiv_{i+1}$ from $\equiv_i$, until the sequence of partitions associated with the $\equiv_i$ stabilizes.

There are a number of algorithms for computing $\equiv$, or to determine whether $p \equiv q$ for some given $p, q \in Q$.

A simple method to compute $\equiv$ is described in Hopcroft and Ullman. The basic idea is to *propagate inequivalence, rather than equivalence*.

The method consists in forming a triangular array corresponding to all unordered pairs (p, q), with $p \neq q$ (the rows and the columns of this triangular array are indexed by the states in Q, where the entries are below the descending diagonal).

Initially, the entry (p, q) is marked iff p and q are **not 0-equivalent**, which means that p and q are not both in F or not both in $Q - F$.

Then, we process every unmarked entry on every row as follows: for any unmarked pair (p, q), we consider pairs $(\delta(p, a), \delta(q, a))$, for all $a \in \Sigma$. If any pair $(\delta(p, a), \delta(q, a))$ is already marked, this means that $\delta(p, a)$ and $\delta(q, a)$ are *inequivalent*, and thus p and q are *inequivalent*, and we mark the pair (p, q).

We continue in this fashion, until at the end of a round during which all the rows are processed, nothing has changed. When the algorithm stops, all marked pairs are inequivalent, and all unmarked pairs correspond to equivalent states.

Example 6.14. Let us illustrates the above method. Consider the following DFA accepting $\{a, b\}^*\{abb\}$.

	a	b
A	B	C
B	B	D
C	B	C
D	B	E
E	B	C

The start state is A, and the set of final states is $F = \{E\}$.

The initial (half) array is as follows, using $\times$ to indicate that the corresponding pair (say, (E, A)) consists of inequivalent states, and $\square$ to indicate that nothing is known yet.

$$
\begin{array}{ccccc}
B & \square & & & \\
C & \square & \square & & \\
D & \square & \square & \square & \\
E & \times & \times & \times & \times \\
 & A & B & C & D
\end{array}
$$

After the first round, we have

$$
\begin{array}{ccccc}
B & \square & & & \\
C & \square & \square & & \\
D & \times & \times & \times & \\
E & \times & \times & \times & \times \\
 & A & B & C & D
\end{array}
$$

After the second round, we have

$$\begin{array}{ccccc}
B & \times & & & \\
C & \square & \times & & \\
D & \times & \times & \times & \\
E & \times & \times & \times & \times \\
 & A & B & C & D
\end{array}$$

Finally, nothing changes during the third round, and thus, only A and C are equivalent, and we get the four equivalence classes

$$(\{A, C\}, \{B\}, \{D\}, \{E\}).$$

We obtain the minimal DFA showed in Figure 6.4.

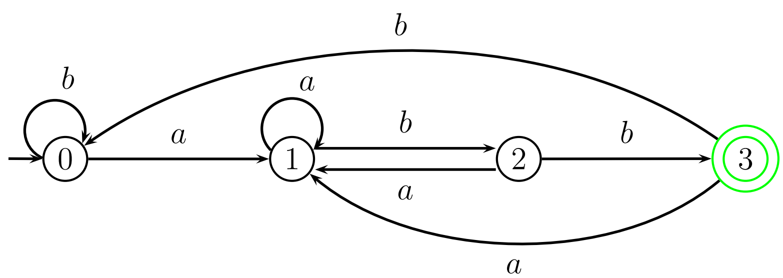

Figure 6.4: A minimal DFA acepting $\{a, b\}^*\{abb\}$

There are ways of improving the efficiency of this algorithm, see Hopcroft and Ullman for such improvements.

Fast algorithms for testing whether $p \equiv q$ for some given $p, q \in Q$ also exist. One of these algorithms is based on "*forward closures*," an idea due to Knuth. Such an algorithm is related to a fast unification algorithm; see Section 6.5.

6.4 The Pumping Lemma

Another useful tool for proving that languages are not regular is the so-called *pumping lemma*.

Lemma 6.10. *Given any DFA* $D = (Q, \Sigma, \delta, q_0, F)$ *there is some* $m \geq 1$ *such that for every* $w \in \Sigma^*$*, if* $w \in L(D)$ *and* $|w| \geq m$*, then there exists a decomposition of* w *as* $w = uxv$*, where*

(1) $x \neq \epsilon$*,*

(2) $ux^iv \in L(D)$*, for all* $i \geq 0$*, and*

(3) $|ux| \leq m$*.*

Moreover, m *can be chosen to be the number of states of the DFA* D*.*

An important consequence of the pumping lemma is that if a DFA D has m states and if there is some string $w \in L(D)$ such that $|w| \geq m$, then $L(D)$ is infinite.

As a consequence, if $L(D)$ is finite, there are *no* strings w in $L(D)$ such that $|w| \geq m$. In this case, since the premise of the pumping lemma is false, the pumping lemma holds vacuously; that is, if $L(D)$ is finite, the pumping lemma yields no information.

Another corollary of the pumping lemma is that there is a test to decide whether a DFA D accepts an infinite language $L(D)$.

Proposition 6.11. *Let D be a DFA with m states, The language $L(D)$ accepted by D is infinite iff there is some string $w \in L(D)$ such that $m \leq |w| < 2m$.*

If $L(D)$ is infinite, there are strings of length $\geq m$ in $L(D)$, but a prirori there is no guarantee that there are "short" strings w in $L(D)$, that is, strings whose length is uniformly bounded by some function of m independent of D.

The pumping lemma ensures that there are such strings, and the function is $m \mapsto 2m$.

Typically, the pumping lemma is used to prove that a language is not regular.

The method is to proceed by contradiction, i.e., to assume (contrary to what we wish to prove) that a language L is indeed regular, and derive a contradiction of the pumping lemma.

Thus, it would be helpful to see what the negation of the pumping lemma is, and for this, we first state the pumping lemma as a logical formula.

We will use the following abbreviations:

$$
\begin{aligned}
nat &= \{0, 1, 2, \ldots\}, \\
pos &= \{1, 2, \ldots\}, \\
A &\equiv w = uxv, \\
B &\equiv x \neq \epsilon, \\
C &\equiv |ux| \leq m, \\
P &\equiv \forall i\colon nat\, (ux^i v \in L(D)).
\end{aligned}
$$

The pumping lemma can be stated as

$$
\forall D\colon \mathrm{DFA}\, \exists m\colon pos\, \forall w\colon \Sigma^*
$$
$$
\Big((w \in L(D) \wedge |w| \geq m) \supset (\exists u, x, v\colon \Sigma^* A \wedge B \wedge C \wedge P)\Big).
$$

Recalling that

$$\begin{aligned}\neg(A \wedge B \wedge C \wedge P) &\equiv \neg(A \wedge B \wedge C) \vee \neg P \\ &\equiv (A \wedge B \wedge C) \supset \neg P\end{aligned}$$

and

$$\neg(R \supset S) \equiv R \wedge \neg S,$$

the negation of the pumping lemma can be stated as

$$\begin{aligned}&\exists D\colon \mathrm{DFA}\,\forall m\colon pos\,\exists w\colon \Sigma^* \\ &\qquad\Big((w \in L(D) \wedge |w| \geq m) \\ &\qquad\quad \wedge (\forall u, x, v\colon \Sigma^*\,(A \wedge B \wedge C) \supset \neg P)\Big).\end{aligned}$$

Since

$$\neg P \equiv \exists i\colon nat\,(ux^i v \notin L(D)),$$

in order to show that the pumping lemma is contradicted, one needs to show that for some DFA D, for every $m \geq 1$, there is some string $w \in L(D)$ of length at least m, such that *for every possible decomposition $w = uxv$ satisfying the constraints $x \neq \epsilon$ and $|ux| \leq m$, there is some $i \geq 0$ such that $ux^i v \notin L(D)$.*

6.5 A Fast Algorithm for Checking State Equivalence Using a "Forward-Closure"

Given two states $p, q \in Q$, if $p \equiv q$, then we know that $\delta(p, a) \equiv \delta(q, a)$, for all $a \in \Sigma$.

This suggests a method for testing whether two distinct states p, q are equivalent.

Starting with the relation $R = \{(p, q)\}$, construct the smallest equivalence relation $R^\dagger$ containing R with the property that whenever $(r, s) \in R^\dagger$, then $(\delta(r, a), \delta(s, a)) \in R^\dagger$, for all $a \in \Sigma$.

If we ever encounter a pair (r, s) such that $r \in F$ and $s \in \overline{F}$, or $r \in \overline{F}$ and $s \in F$, then r and s are inequivalent, and so are p and q.

Otherwise, it can be shown that p and q are indeed equivalent.

Thus, testing for the equivalence of two states reduces to finding an efficient method for computing the "forward closure" of a relation defined on the set of states of a DFA.

Such a method was worked out by John Hopcroft and Richard Karp and published in a 1971 Cornell technical report.

This method is based on an idea of Donald Knuth for solving Exercise 11, in Section 2.3.5 of *The Art of Computer Programming*, Vol. 1, second edition, 1973. A sketch of the solution for this exercise is given on page 594.

As far as I know, Hopcroft and Karp's method was never published in a journal, but a simple recursive algorithm does appear on page 144 of Aho, Hopcroft and Ullman's *The Design and Analysis of Computer Algorithms*, first edition, 1974.

Essentially the same idea was used by Paterson and Wegman to design a fast unification algorithm (in 1978).

A relation $S \subseteq Q \times Q$ is a *forward closure* iff it is an equivalence relation and whenever $(r, s) \in S$, then $(\delta(r, a), \delta(s, a)) \in S$, for all $a \in \Sigma$.

The *forward closure* of a relation $R \subseteq Q \times Q$ is the smallest equivalence relation $R^{\dagger}$ containing R which is forward closed.

We say that a forward closure S is *good* iff whenever $(r, s) \in S$, then $good(r, s)$, where $good(r, s)$ holds iff either both $r, s \in F$, or both $r, s \notin F$. Obviously, $bad(r, s)$ iff $\neg good(r, s)$.

Given any relation $R \subseteq Q \times Q$, recall that the smallest equivalence relation $R_{\approx}$ containing R is the relation $(R \cup R^{-1})^*$ (where $R^{-1} = \{(q, p) \mid (p, q) \in R\}$, and $(R \cup R^{-1})^*$ is the reflexive and transitive closure of $(R \cup R^{-1})$).

The forward closure of R can be computed inductively by defining the sequence of relations $R_i \subseteq Q \times Q$ as follows:

$$
\begin{aligned}
R_0 &= R_{\approx} \\
R_{i+1} &= (R_i \cup \{(\delta(r,a), \delta(s,a)) \mid (r,s) \in R_i,\ a \in \Sigma\})_{\approx}.
\end{aligned}
$$

It is not hard to prove that $R_{i_0+1} = R_{i_0}$ for some least i_0, and that $R^{\dagger} = R_{i_0}$ is the smallest forward closure containing R.

The following two facts can also been established.

(a) if $R^{\dagger}$ is good, then

$$R^{\dagger} \subseteq \equiv . \tag{6.1}$$

(b) if $p \equiv q$, then

$$R^{\dagger} \subseteq \equiv;$$

that is, equation (6.1) holds. This implies that $R^{\dagger}$ is good.

As a consequence, we obtain the correctness of our procedure: $p \equiv q$ iff the forward closure $R^{\dagger}$ of the relation $R = \{(p, q)\}$ is good.

In practice, we maintain a partition Π representing the equivalence relation that we are closing under forward closure.

We add each new pair $(\delta(r, a), \delta(s, a))$ one at a time, and immediately form the smallest equivalence relation containing the new relation.

If $\delta(r, a)$ and $\delta(s, a)$ already belong to the same block of Π, we consider another pair, else we merge the blocks corresponding to $\delta(r, a)$ and $\delta(s, a)$, and then consider another pair.

The algorithm is recursive, but it can easily be implemented using a stack.

To manipulate partitions efficiently, we represent them as lists of trees (forests).

Each equivalence class C in the partition Π is represented by a tree structure consisting of nodes and parent pointers, with the pointers from the sons of a node to the node itself.

The root has a null pointer. Each node also maintains a counter keeping track of the number of nodes in the subtree rooted at that node.

Note that pointers can be avoided. We can represent a forest of n nodes as a list of n pairs of the form $(father, count)$. If $(father, count)$ is the ith pair in the list, then $father = 0$ iff node i is a root node, otherwise, $father$ is the index of the node in the list which is the parent of node i.

The number $count$ is the total number of nodes in the tree rooted at the ith node.

For example, the following list of nine nodes

$$((0,3),(0,2),(1,1),(0,2),(0,2),(1,1),(2,1),(4,1),(5,1))$$

represents a forest consisting of the following four trees:

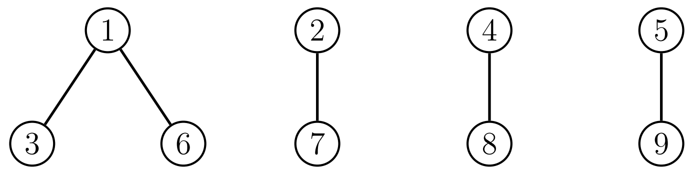

Figure 6.5: A forest of four trees

Two functions *union* and *find* are defined as follows.

Given a state p, $\mathit{find}(p,\Pi)$ finds the root of the tree containing p as a node (not necessarily a leaf).

Given two root nodes p, q, $\mathit{union}(p,q,\Pi)$ forms a new partition by merging the two trees with roots p and q as follows: if the counter of p is smaller than that of q, then let the root of p point to q, else let the root of q point to p.

For example, given the two trees shown on the left in Figure 6.6, *find*$(6, \Pi)$ returns 3 and *find*$(8, \Pi)$ returns 4. Then $union(3, 4, \Pi)$ yields the tree shown on the right in Figure 6.6.

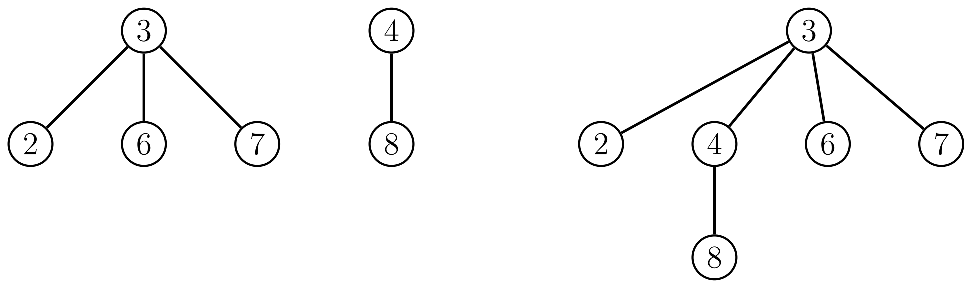

Figure 6.6: Applying the function *union* to the trees rooted at 3 and 4

In order to speed up the algorithm, using an idea due to Tarjan, we can modify *find* as follows:

during a call *find*(p, Π), as we follow the path from p to the root r of the tree containing p, we redirect the parent pointer of every node q on the path from p (including p itself) to r (we perform *path compression*).

For example, applying $\mathit{find}(8, \Pi)$ to the tree shown on the right in Figure 6.6 yields the tree shown in Figure 6.7

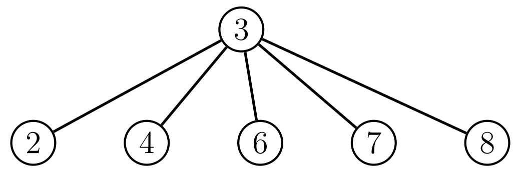

Figure 6.7: The result of applying *find* with path compression

Then, the algorithm is as follows:

function $\mathit{unif}[p, q, \Pi, dd]$: flag;
  begin
    $\mathit{trans} := \mathit{left}(dd)$; $\mathit{ff} := \mathit{right}(dd)$; $pq := (p, q)$;
    $st := (pq)$; $\mathit{flag} := 1$;
    $k := \mathit{Length}(\mathit{first}(\mathit{trans}))$;
    while $st \neq () \wedge \mathit{flag} \neq 0$ **do**
      $uv := \mathit{top}(st)$; $uu := \mathit{left}(uv)$; $vv := \mathit{right}(uv)$;
      $\mathit{pop}(st)$;
      if $\mathit{bad}(\mathit{ff}, uv) = 1$ **then** $\mathit{flag} := 0$
      else
        $u := \mathit{find}(uu, \Pi)$; $v := \mathit{find}(vv, \Pi)$;
        if $u \neq v$ **then**
          $\mathit{union}(u, v, \Pi)$;
          for $i = 1$ **to** k **do**
            $u1 := \mathit{delta}(\mathit{trans}, uu, k - i + 1)$;
            $v1 := \mathit{delta}(\mathit{trans}, vv, k - i + 1)$;
            $uv := (u1, v1)$; $\mathit{push}(st, uv)$
          endfor
        endif
      endif
    endwhile
  end

The initial partition Π is the identity relation on Q, i.e., it consists of blocks $\{q\}$ for all states $q \in Q$.

The algorithm uses a stack st. We are assuming that the DFA dd is specified by a list of two sublists, the first list, denoted $left(dd)$ in the pseudo-code above, being a representation of the transition function, and the second one, denoted $right(dd)$, the set of final states.

The transition function itself is a list of lists, where the i-th list represents the i-th row of the transition table for dd.

The function $delta$ is such that $delta(trans, i, j)$ returns the j-th state in the i-th row of the transition table of dd.

For example, we have the DFA

$$\begin{aligned} dd = {} & (((2,3),(2,4),(2,3),(2,5),(2,3), \\ & (7,6),(7,8),(7,9),(7,6)),(5,9)) \end{aligned}$$

consisting of 9 states labeled $1, \ldots, 9$, and two final states 5 and 9 shown in Figure 6.8.

Also, the alphabet has two letters, since every row in the transition table consists of two entries.

For example, the two transitions from state 3 are given by the pair $(2, 3)$, which indicates that $\delta(3, a) = 2$ and $\delta(3, b) = 3$.

The sequence of steps performed by the algorithm starting with $p = 1$ and $q = 6$ is shown below.

At every step, we show the current pair of states, the partition, and the stack.

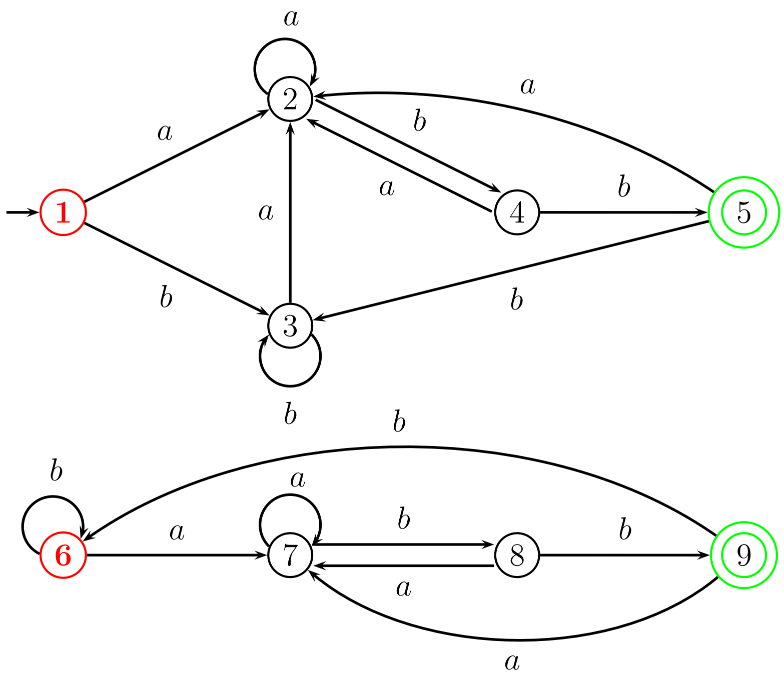

Figure 6.8: Testing state equivalence in a DFA

$p = 1, q = 6,\ \Pi = \{\{1,6\},\{2\},\{3\},\{4\},\{5\},\{7\},\{8\},\{9\}\},\ st = \{\{1,6\}\}$

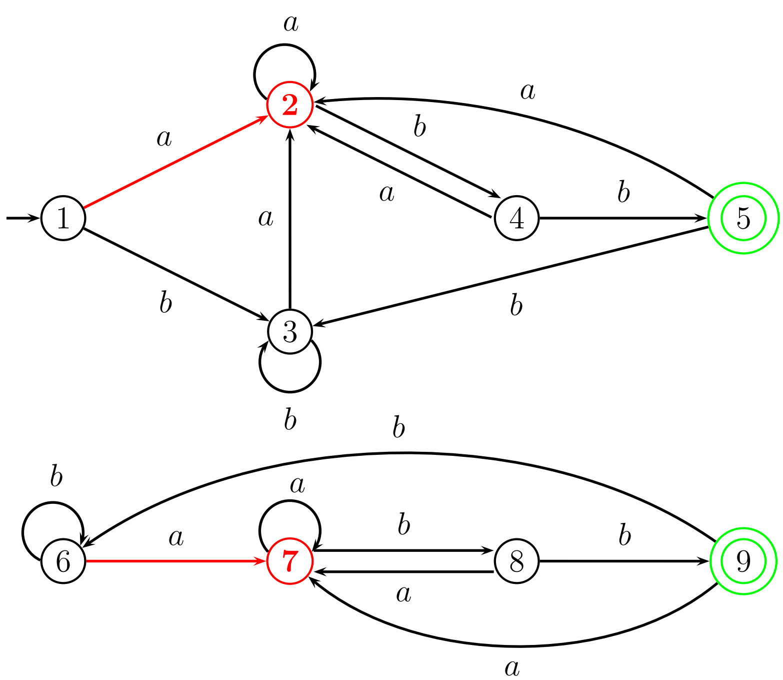

Figure 6.9: Testing state equivalence in a DFA

$p = 2, q = 7,\ \Pi = \{\{1,6\},\{2,7\},\{3\},\{4\},\{5\},\{8\},\{9\}\},\ st = \{\{3,6\},\{2,7\}\}$

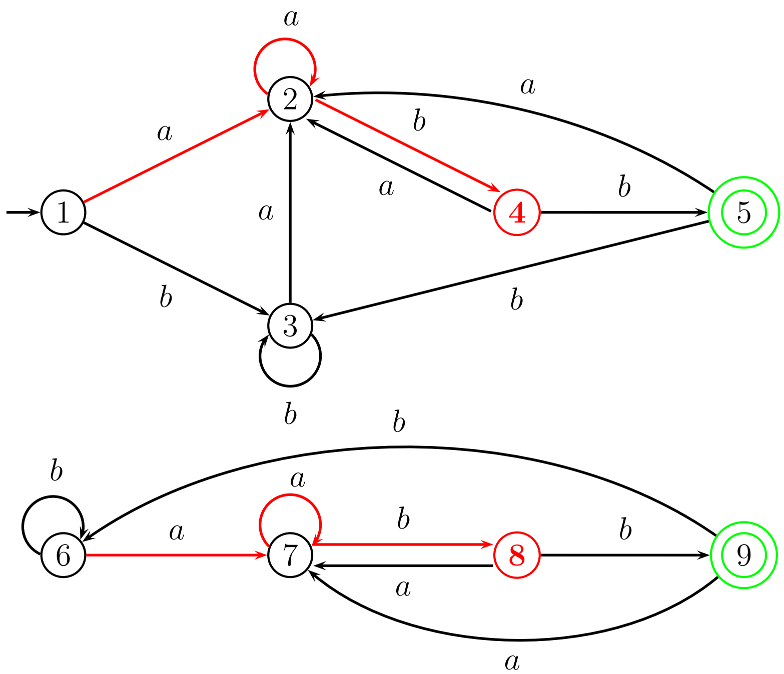

Figure 6.10: Testing state equivalence in a DFA

$p = 4, q = 8,\ \Pi = \{\{1,6\},\{2,7\},\{3\},\{4,8\},\{5\},\{9\}\},\ st = \{\{3,6\},\{4,8\}\}$

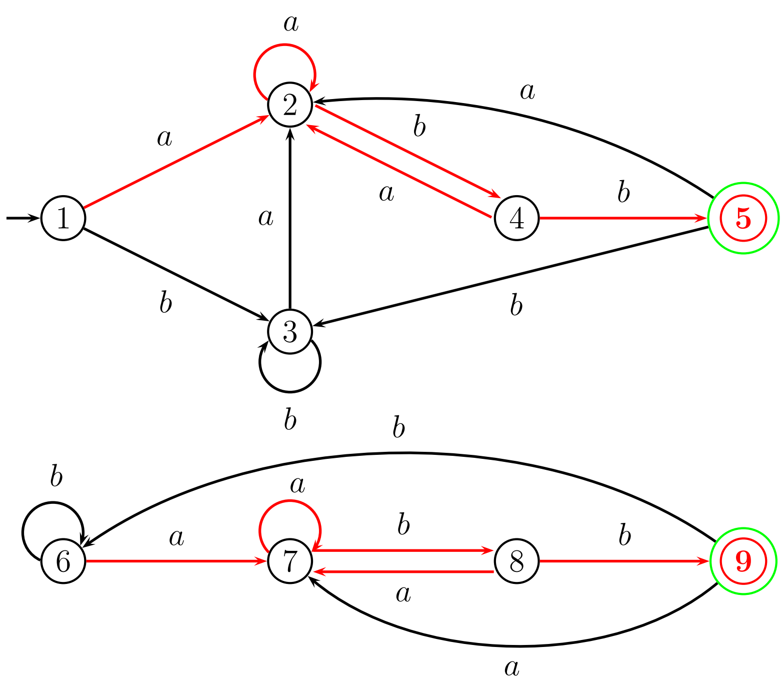

Figure 6.11: Testing state equivalence in a DFA

$p = 5, q = 9,\ \Pi = \{\{1,6\},\{2,7\},\{3\},\{4,8\},\{5,9\}\},\ st = \{\{3,6\},\{5,9\}\}$

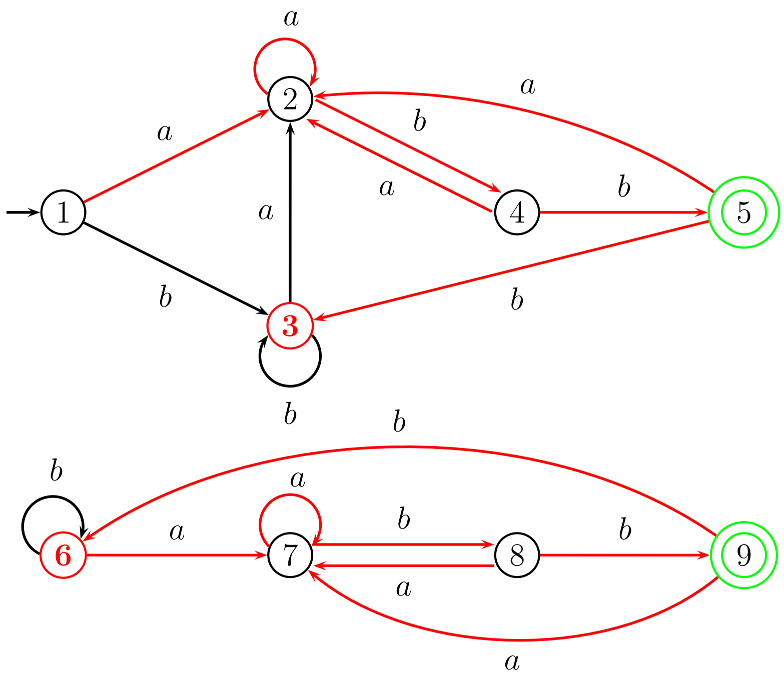

Figure 6.12: Testing state equivalence in a DFA

$p = 3, q = 6,\ \Pi = \{\{1, 3, 6\}, \{2, 7\}, \{4, 8\}, \{5, 9\}\},\ st = \{\{3, 6\}, \{3, 6\}\}$

Since states 3 and 6 belong to the first block of the partition, the algorithm terminates. Since no block of the partition contains a bad pair, the states $p = 1$ and $q = 6$ are equivalent.

Let us now test whether the states $p = 3$ and $q = 7$ are equivalent.

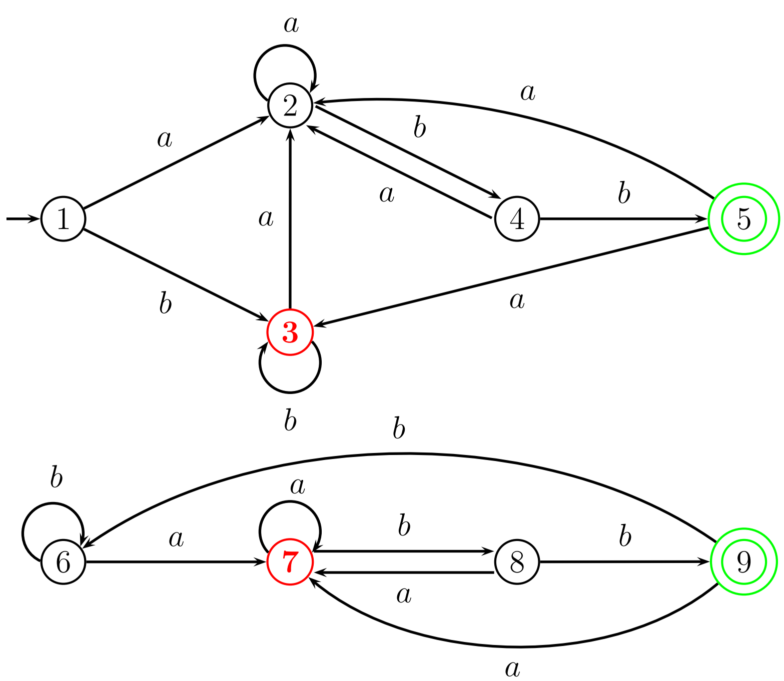

Figure 6.13: Testing state equivalence in a DFA

$p = 3, q = 7,\ \Pi = \{\{1\}, \{2\}, \{3, 7\}, \{4\}, \{5\}, \{6\}, \{8\}, \{9\}\},\ st = \{\{3, 7\}\}$

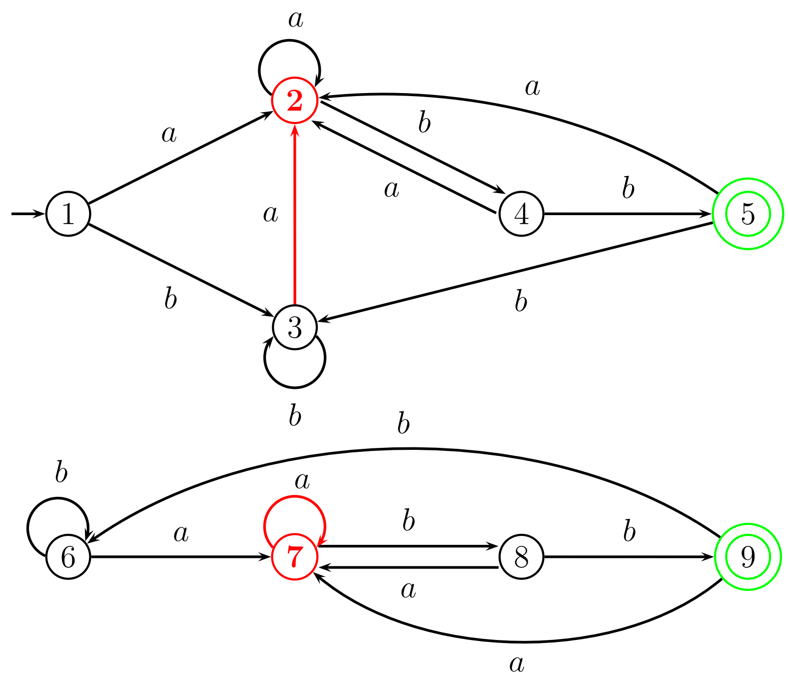

Figure 6.14: Testing state equivalence in a DFA

$p = 2, q = 7,\ \Pi = \{\{1\}, \{2, 3, 7\}, \{4\}, \{5\}, \{6\}, \{8\}, \{9\}\},\ st = \{\{3, 8\}, \{2, 7\}\}$

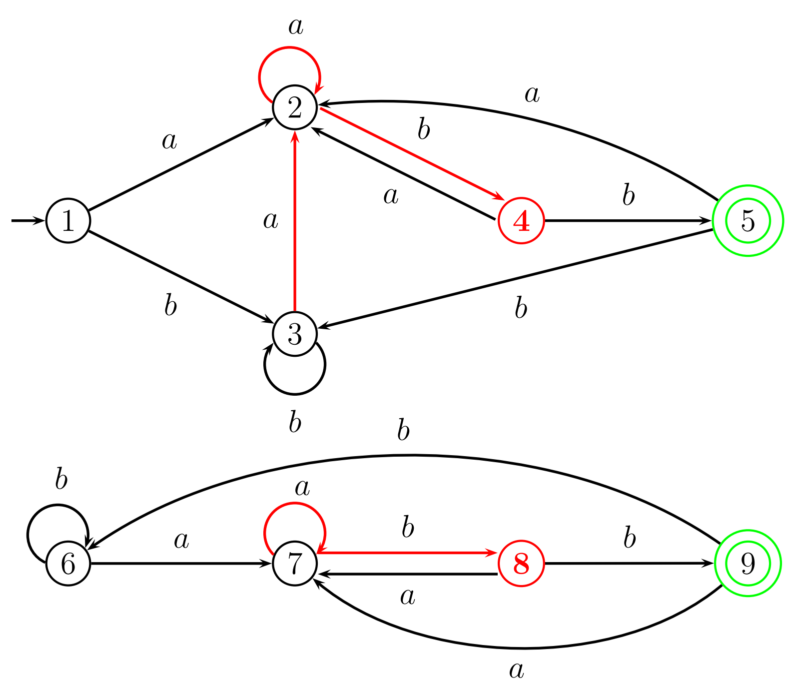

Figure 6.15: Testing state equivalence in a DFA

$p = 4, q = 8,\ \Pi = \{\{1\}, \{2, 3, 7\}, \{4, 8\}, \{5\}, \{6\}, \{9\}\},\ st = \{\{3, 8\}, \{4, 8\}\}$

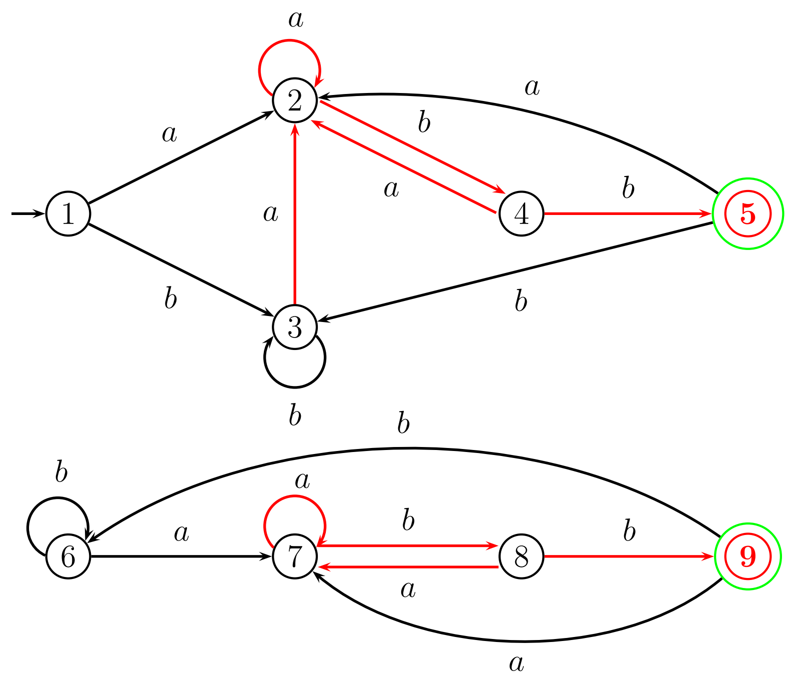

Figure 6.16: Testing state equivalence in a DFA

$p = 5, q = 9,\ \Pi = \{\{1\}, \{2, 3, 7\}, \{4, 8\}, \{5, 9\}, \{6\}\},\ st = \{\{3, 8\}, \{5, 9\}\}$

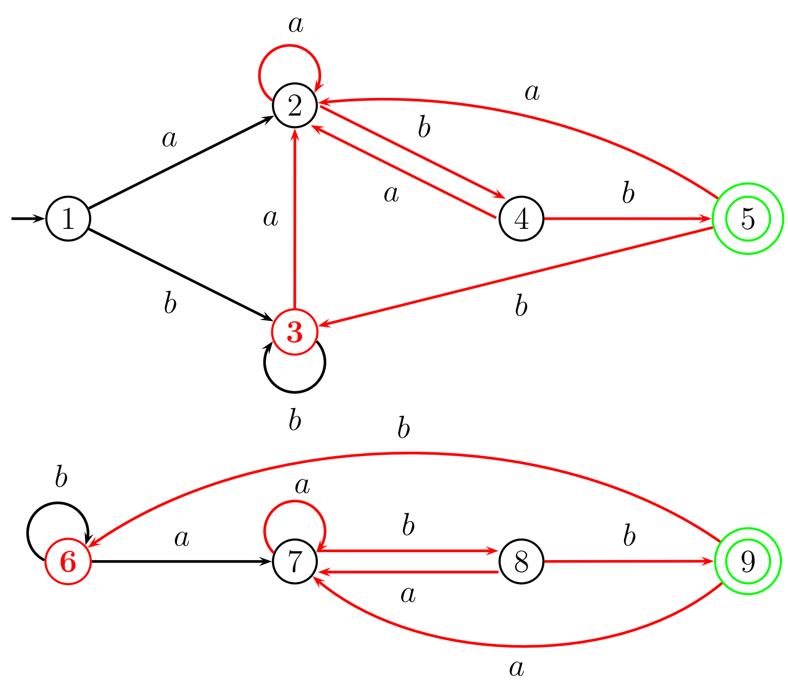

Figure 6.17: Testing state equivalence in a DFA

$p = 3, q = 6,\ \Pi = \{\{1\}, \{2, 3, 6, 7\}, \{4, 8\}, \{5, 9\}\},\ st = \{\{3, 8\}, \{3, 6\}\}$

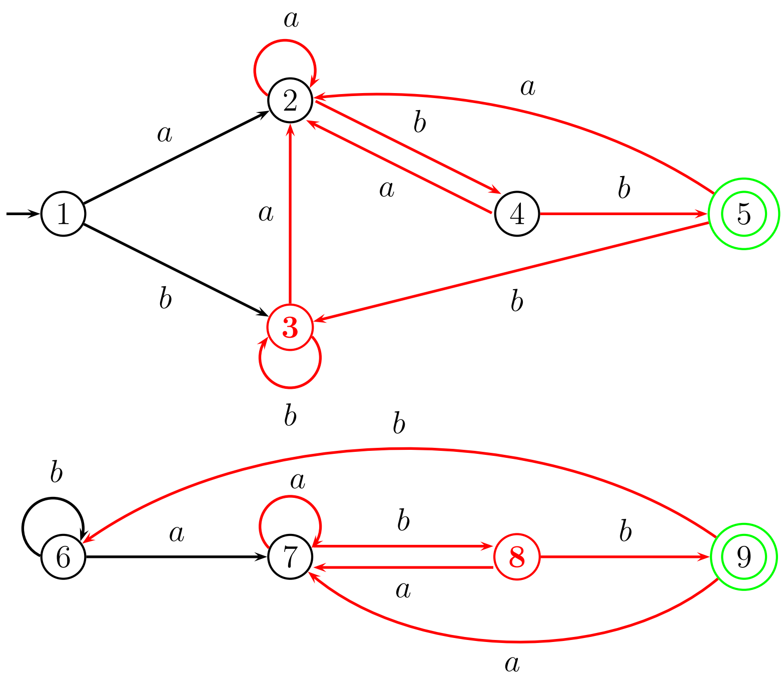

Figure 6.18: Testing state equivalence in a DFA

$p = 3, q = 8,\ \Pi = \{\{1\}, \{2, 3, 4, 6, 7, 8\}, \{5, 9\}\},\ st = \{\{3, 8\}\}$

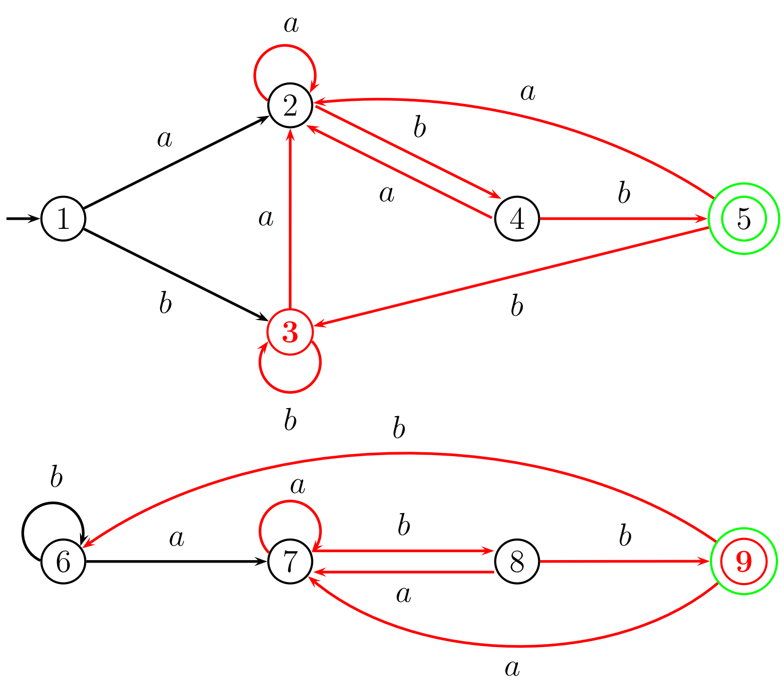

Figure 6.19: Testing state equivalence in a DFA

$$p = 3, q = 9,\ \Pi = \{\{1\}, \{2, 3, 4, 6, 7, 8\}, \{5, 9\}\},\ st = \{\{3, 9\}\}$$

Since the pair $(3, 9)$ is a bad pair, the algorithm stops, and the states $p = 3$ and $q = 7$ are inequivalent.