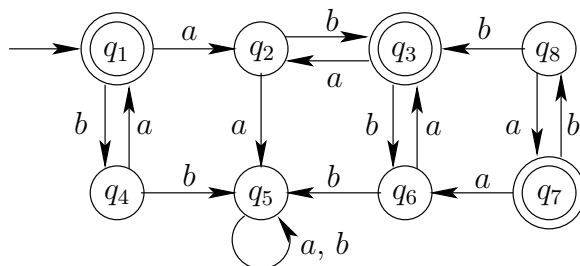


## State minimization

A DFA can be very useful as a component of a larger system, in which case we want it to be efficient with a minimal number of states. Also, conversions (NFA to DFA, regular expressions to NFA to DFA) result in an increased number of states and often sub-optimal automata, in which case optimization is desirable.

The simplest type of optimization that can be performed is removal of *unreachable states*. A state is *unreachable* if there is no path to it from the start state.

Consider the following DFA, which recognizes the language  $L = (ab \cup ba)^*$ :



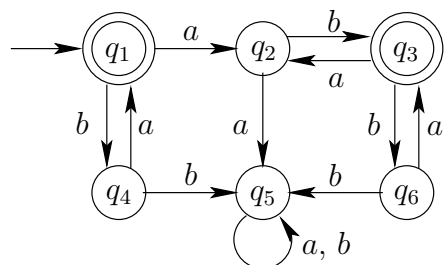
In this DFA, states  $q_7$  and  $q_8$  are unreachable. To efficiently identify and remove all unreachable states, we can use the following algorithm that outputs the set of all reachable states (from which we compute unreachable states). Let  $M = (Q, \Sigma, \delta, q_0, F)$ .

$$R := \{q_0\}$$

while there is a state  $q \in R$  and  $a \in \Sigma$  such that  $\delta(q, a) \notin R$  do

$$R := R \cup \{\delta(q, a)\}$$

After identifying and removing unreachable states together with the arrows touching (i.e., leaving) them, the DFA above becomes:



The next phase of optimization involves identifying and merging *equivalent states*. Informally, two states of a DFA are equivalent if exactly the same set of strings leads the machine to acceptance from either state.

**Definition 1** Let  $L \subseteq \Sigma^*$  be a language and  $x, y \in \Sigma^*$ . We say that  $x$  and  $y$  are equivalent with respect to  $L$ , denoted  $x \approx_L y$ , if for all  $z \in \Sigma^*$ ,  $xz \in L$  if and only if  $yz \in L$ .

This definition states that both  $x$  and  $y$  belong to  $L$  or neither. So is true for each  $xz$  and  $yz$ . There are four equivalence classes with respect to our example language  $L = (ab \cup ba)^*$ . They are: (i)  $w \in L$ , (ii)  $wa$  for some  $w \in L$  (need additional  $bv$  for some  $v \in L$  to be in the language); (iii)  $wb$  for some  $w \in L$  (need additional  $av$  for some  $v \in L$  to be in the language); and (iv)  $w(aa \cup bb)\Sigma^*$  (nothing can make this a string in  $L$ ). All strings in a class have the same fate with respect to inclusion in  $L$ .

The above definition was in terms of a language, and not in terms of an automaton. We can also define classes of equivalent string with respect to machine  $M$ .

**Definition 2** Let  $M = (Q, \Sigma, \delta, q_0, F)$  be a DFA. We say that two strings  $x, y \in \Sigma^*$  are equivalent with respect to  $M$ , denoted  $x \sim_M y$ , if they both drive  $M$  from  $q_0$  to the same state.

The number of equivalence classes with respect to  $M$  is not the same as the number of equivalence classes with respect to  $L(M)$ . For example, there are 6 equivalence classes with respect to the DFA above. They are  $E_{q_1} = (ba)^*$ ,  $E_{q_2} = La \cup a$ ,  $E_{q_3} = abL$ ,  $E_{q_4} = b(ab)^*$ ,  $E_{q_5} = L(bb \cup aa)\Sigma^*$ , and  $E_{q_6} = abLb$  (here  $L$  denotes all strings in the language). But there is a relation between these two notions.

**Theorem 1** For any DFA  $M = (Q, \Sigma, \delta, q_0, F)$  and  $x, y \in \Sigma^*$ , if  $x \sim_M y$ , then  $x \equiv_{L(M)} y$ .

In other words, if two strings drive  $M$  to the same state, they will be in the same equivalence class with respect to the language. This gives us the lower bound on the number of states. We know that we have to have four states to recognize the language. But can we actually achieve this lower bound? The answer is yes.

**Theorem 2** Let  $L \subseteq \Sigma^*$  be a regular language. Then there is a DFA that accepts  $L$  with precisely as many states as there are equivalence classes in  $\approx_L$ .

This tells us that a 4-state DFA is possible for our example language. The last question we need to answer is how actually to build a minimum-state DFA. Let's go back to the notion of equivalent states.

**Definition 3** Given DFA  $M = (Q, \Sigma, \delta, q_0, F)$ , we say that  $q \in Q$  and  $q' \in Q$  are equivalent, denoted  $q \equiv q'$ , if the following is true: on input  $x \in \Sigma^*$   $M$  transitions from  $q$  to an accept state if and only if  $M$  transitions from  $q'$  to an accept state on input  $x$ .

To build a min-state DFA, we'll go through a sequence of equivalence relations  $\equiv_0, \equiv_1, \equiv_2, \dots$ . Here  $q \equiv_i q'$  means that  $q$  and  $q'$  remain equivalent after processing a string of length  $i$  (or less). (This equivalence relation might change after processing longer strings, but are not concerned with it.)

For  $\equiv_0$ , we have that  $q \equiv_0 q'$  if both are accepting or both are not accepting. In other words, we have two equivalence classes for  $\equiv_0$ :  $F$  and  $Q - F$ . The last piece we need is to show how to compute  $\equiv_{i+1}$  from  $\equiv_i$ .

**Lemma 1** For any  $q, q' \in Q$  and any  $i \geq 0$ ,  $q \equiv_{i+1} q'$  if and only if (i)  $q \equiv_i q'$  and (ii) for all  $a \in \Sigma$ ,  $\delta(q, a) \equiv_i \delta(q', a)$ .

This gives us an algorithm for computing classes of equivalent states:

```

set equivalence classes for  $\equiv_0$  to  $F$  and  $Q - F$ 
repeat for  $i = 0, 1, 2, \dots$ 
    compute equivalence classes of  $\equiv_{i+1}$  from those of  $\equiv_i$ 
until  $\equiv_{i+1}$  is the same as  $\equiv_i$ 

```

For our example DFA, we have:

$\equiv_0$ :  $\{q_1, q_3\}$  and  $\{q_2, q_4, q_5, q_6\}$

$\equiv_1$ :  $q_1 \equiv_0 q_3$  and  $\delta(q_1, a) \equiv_0 \delta(q_3, a)$  and  $\delta(q_1, b) \equiv_0 \delta(q_3, b) \Rightarrow \{q_1, q_3\}$  remain in the same class.

$q_2 \equiv_0 q_4 \equiv_0 q_5 \equiv_0 q_6$ ,  $\delta(q_2, a) \not\equiv_0 \delta(q_4, a)$ ,  $\delta(q_2, a) \equiv_0 \delta(q_5, a)$ , and  $\delta(q_4, a) \equiv_0 \delta(q_6, a)$ , which after processing  $a$  gives us classes  $\{q_2, q_5\}$  and  $\{q_4, q_6\}$ ;  $\delta(q_2, b) \not\equiv_0 \delta(q_5, b)$  and  $\delta(q_4, b) \equiv_0 \delta(q_6, b) \Rightarrow \{q_2\}$ ,  $\{q_5\}$ , and  $\{q_4, q_6\}$ .

Thus, at the end of this step we obtain classes  $\{q_1, q_3\}$ ,  $\{q_2\}$ ,  $\{q_5\}$ , and  $\{q_4, q_6\}$ .

$\equiv_2$ :  $q_1 \equiv_1 q_3$  and  $\delta(q_1, a) \equiv_1 \delta(q_3, a)$  and  $\delta(q_1, b) \equiv_1 \delta(q_3, b) \Rightarrow \{q_1, q_3\}$ .

$q_4 \equiv_1 q_6$  and  $\delta(q_4, a) \equiv_1 \delta(q_6, a)$  and  $\delta(q_4, b) \equiv_1 \delta(q_6, b) \Rightarrow \{q_4, q_6\}$ .

This means that at the end of the step no changes occurs to the equivalence classes  $\{q_1, q_3\}$ ,  $\{q_2\}$ ,  $\{q_5\}$ , and  $\{q_4, q_6\}$ , and the algorithm terminates.

After merging states from an equivalence class into one state, we obtain min-state DFA for the language:

