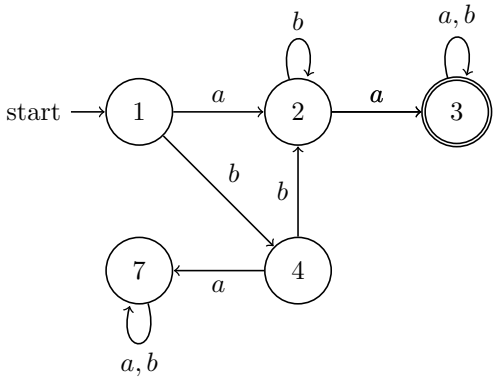
Minimization of DFAs

At this point we turn to the minimization algorithm. There is a beautiful theory behind why this all works, and we will see how this theory works in the next few lectures.

A minimal DFA is one in which every state is *useful* to the acceptance of its language. Thus, a minimization algorithm must decide which states are useful and which are not. It will do this by determining which states are *equivalent* and which are *distinct*.

We will use the following DFA as our running example through this section:

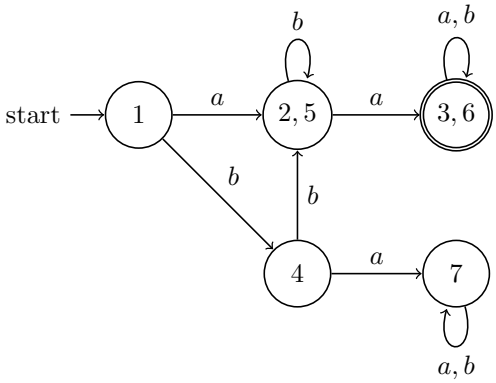


It may not be obvious, but this DFA actually is not minimal. DFAs always have a particular start state, but we could just as well start from anywhere. Suppose we call the above DFA M, then our current start state in M is 1, and $L(M)$ can be described by the regular expression $(ab^*a(a \cup b)^*) \cup (bbb^*a(a \cup b)^*)$. If we change the start state to, say, state 4, then $L(M)$ can be described by $(bb^*a(a \cup b)^*)$. If we start at state 7, then $L(M) = \emptyset$.

Suppose we decided to start in state 3, then $L(M)$ can be described by $(a \cup b)^*$, but the same can be said for state 6. Similarly, if we start in state 2, then $L(B)$ can be described by $b^*a(a \cup b)^*$, but the same can be said for state 5. Thus, when we get to state 3 or 6, while running M, it will accept or reject the exact same strings. The same can be said for states 2 and 5.

Denote by $L(M)_q$ the language accepted by M when the start state of M is q. If $L(M)_q = L(M)_r$ for two states q and r of M. Then we can actually merge the two states q and r into a single state, because once we reach either q or r the machine will accept the exact same language.

The new, and minimal DFA, is as follows:



Definition 1. *Call two states, q and r, of a DFA, M,* equivalent *when $L(M)_q = L(M)_r$.*

The minimization algorithm then has to determine which states are equivalent and which are distinct. Then we construct a new DFA by merging equivalent states. Thus, the minimization algorithm can be broken down into two phases.

1 Myhill-Nerode theorem for regular languages

Definition 2. *A DFA M for a regular language L is* minimal *if and only if M has the smallest number of states possible.*

Minimal DFAs are the most efficient in both time and space.

Definition 3. *Suppose $M = (Q, \Sigma, \delta, q_0, F)$ be a DFA. Given two words $w_1, w_2 \in \Sigma^*$, define $w_1 \approx_M w_2$ if and only if $\hat{\delta}(w_1, q_0) = \hat{\delta}(w_2, q_0)$.*

Lemma 4 (Properties of $\approx_M$)**.** *Suppose $M = (Q, \Sigma, \delta, q_0, F)$ be a DFA.*

1. *(Reflexive) For any $w \in \Sigma^*$, $w \approx_M w$.*
2. *(Symmetric) For any $w_1, w_2 \in \Sigma^*$, if $w_1 \approx_M w_2$, then $w_2 \approx_M w_1$.*
3. *(Transitive) For any $w_1, w_2, w_3 \in \Sigma^*$, if $w_1 \approx_M w_2$ and $w_2 \approx_M w_3$, then $w_1 \approx_M w_3$.*

Definition 5. *Suppose L is a regular language. Given two words $w_1, w_2 \in \Sigma^*$, define $w_1 \approx_L w_2$ if and only if for any $w \in \Sigma^*$, $w_1 w \in L$ if and only if $w_2 w \in L$.*

We read $w_1 \approx_L w_2$ as w_1 is indistinguishable from w_2.

Lemma 6 (Properties of $\approx_L$)**.** *Suppose L is a regular language.*

1. *(Reflexive) For any $w \in \Sigma^*$, $w \approx_L w$.*
2. *(Symmetric) For any $w_1, w_2 \in \Sigma^*$, if $w_1 \approx_L w_2$, then $w_2 \approx_L w_1$.*
3. *(Transitive) For any $w_1, w_2, w_3 \in \Sigma^*$, if $w_1 \approx_L w_2$ and $w_2 \approx_L w_3$, then $w_1 \approx_L w_3$.*

Lemma 7 (DFA and Word Indistinguishablity)**.** *Suppose $M = (Q, \Sigma, \delta, q_0, F)$ be a DFA accepting L. Then for any $w_1, w_2 \in \Sigma^*$, if $w_1 \approx_M w_2$, then $w_1 \approx_L w_2$.*

Theorem 8 (Myhill-Nerode Theorem)**.** *A language L is regular if and only if there are a finite number of equivalence classes of $\approx_L$.*

Definition 9. *Suppose $\approx$ is an equivalence relation on a set A. Then an* equivalence class *of an element $x \in A$ with respect to $\approx$ is a set $[x] = \{y \in A \mid x \approx y\}$.*

Definition 10. *Suppose L is a language over Σ such that there exists a finite number of indistinguishablity relations $\approx_L$. Then L's DFA is defined as follows:*

$$Q = \{[w] \mid w \in \Sigma^*\}$$
$$q_0 = [\varepsilon]$$
$$F = \{[w] \mid w \in L\}$$
$$\delta(a, [w]) = [wa]$$

2 The Minimization Algorithm

Phase 1: State Distinction

In this phase we build a table, called distinct, with an entry for each pair of states. Initially, Oevery cell in the table is set to 0. We will denote a cell in the table by $\mathsf{distinct}(p, q)$. Suppose $M = (Q, \Sigma, \delta, q_0, F)$ is a DFA. Then the following algorithm describes how to fill each cell of the table:

Step 1. For each pair of states $(p, q) \in Q \times Q$

If $p \in F$ and $q \notin F$ (or vice versa), then set $\mathsf{distinct}(p, q) = 1$.

Step 2. Loop until there is no change in the table contents:

For each pair of states $(p, q) \in Q \times Q$:

For each alphabet symbol $a \in \Sigma$:

If $\mathsf{distinct}(p, q) = 0$ and $\mathsf{distinct}(\delta(p, a), \delta(q, a)) = 1$, then set $\mathsf{distinct}(p, q) = 1$.

The algorithm works by first running step 1, and then after it completes, running step 2. When the algorithm is finished, then distinct tells us which states in M are distinct. We say p and q are distinct iff $\mathsf{distinct}(p, q) = 1$.

The following table is distinct for our example DFA after running step 1 of the above algorithm:

1							
2	0						
3	1	1					
4	0	0	1				
5	0	0	1	1			
6	1	1	0	1	1		
7	0	0	1	1	1	1	
	1	2	3	4	5	6	7

The bottom of the table is labeled with the first projection, and the vertical is the second projection. Thus, we can see that $\mathsf{distinct}(3, 4) = 1$. Notice that by definition $\mathsf{distinct}(p, q) = \mathsf{distinct}(q, p)$, and so we do not need to track both cells. Also, notice that $\mathsf{distinct}(p, p) = 0$ for all states in the DFA, and so we do not need to track the diagonal.

The following table is after the first iteration of the second step of the algorithm above:

1							
2	1						
3	1	1					
4	1	1	1				
5	1	0	1	1			
6	1	1	0	1	1		
7	1	1	1	1	1	1	
	1	2	3	4	5	6	7

On the second iteration there is no change to the table, and the algorithm terminates. The table now tells us that the only equivalent states are $(2, 5)$ and $(3, 6)$. Thus, these will be merged in the second phase of the algorithm.

Phase 2: Constructing the Minimal DFA

Once we have computed distinct we can define an equivalence relation:

$$p \equiv q \text{ iff } \mathsf{distinct}(p, q) = 0$$

Thus, the previous example table tells us that $2 \equiv 5$ and $3 \equiv 6$.

Given any equivalence relation like $\equiv$ we can define what are called equivalence classes. The equivalence class for the state p, denoted $[p]$, is the following set:

$$[p] = \{q \in Q \mid p \equiv q\}$$

Thus, $[p]$ is the set of all states that are equivalent to p. Thus, $[2] = [5] = \{2, 5\}$ and $[3] = [6] = \{3, 6\}$ for the equivalence relation induced by our example. Note that all of the other equivalence classes are singleton sets.

We now can use the equivalence relation obtained from distinct to construct a minimal DFA by constructing a new DFA called the quotient DFA. Suppose $M = (Q, \Sigma, \delta, q_0, F)$ is a DFA and we have already computed the table distinct for M, and hence, we have an equivalence relation, $\equiv \subseteq Q \times Q$ telling us which states are equivalent in M. Then we can construct the quotient DFA $M_{\equiv} = (Q_{\equiv}, \Sigma_{\equiv}, \delta_{\equiv}, q_{\equiv}, F_{\equiv})$ as follows:

$$\begin{array}{lcl} Q_{\equiv} & = & \{[p] \mid [p] \text{ is an equivalence class with respect to } \equiv\} \\ \Sigma_{\equiv} & = & \Sigma \\ \delta_{\equiv}([q], a) & = & [\delta(q, a)] \\ q_{\equiv} & = & [q_0] \\ F_{\equiv} & = & \{[p] \mid [p] \cap F \neq \emptyset\} \end{array}$$

This construction automatically places mergable states into the same equivalence class, and hence, the new DFA will remove those, and replace them with a single state.

We can now compute the final minimal DFA for our running example. We will denote each equivalence class by its set of elements. The final DFA is as follows:

$$\begin{array}{lcl} Q_{\equiv} & = & \{\{1\}, \{2, 5\}, \{3, 6\}, \{4\}, \{7\}\} \\ \Sigma_{\equiv} & = & \{a, b\} \\ q_{\equiv} & = & \{1\} \\ F_{\equiv} & = & \{\{3, 6\}\} \end{array}$$

The transition function is depicted in graphical form at the end of page 4.