Chapter 1

Finite automata minimization algorithms

1.1 Introduction

1.2 Brzozowski's algorithm

$\varepsilon - free$ FA: $M_0 = (Q_0, V, T_0, \emptyset, S_0, F_0)$
to be minimized DFA: $M_2 = (Q_2, V, T_2, \emptyset, S_2, F_2)$
intermediate NFA: $M_1 = (Q_1, V, T_1, \emptyset, S_1, F_1)$

NFA: $M_1 \to$ DFA: $M_2, M_2 = suseful_s \circ subsetopt(M_1)$

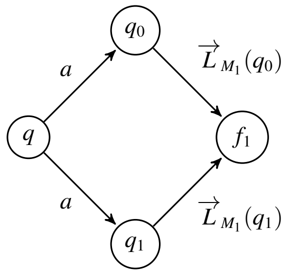
$$q_0, q_1 \in Q_1, Q_2 \subseteq \mathbb{P}(Q_1), \forall p \in Q_2, p = (q_0, q_1)$$
$$\overrightarrow{L}_{M_2}(p) = \overrightarrow{L}_{M_1}(q_0) \cup \overrightarrow{L}_{M_1}(q_1)$$
$$\Rightarrow$$
$$\overrightarrow{L}_{M_2}(p) = \bigcup_{q \in p} \overrightarrow{L}_{M_1}(q)$$
$$\Rightarrow$$

$$\begin{tikzcd}
Q_1 \arrow[r, "subsetopt"] \arrow[dr] & P(Q_1) \arrow[d, "useful_s"] \\
 & D(Q_2)
\end{tikzcd}$$

图 1.1: $M_2 = suseful_s \circ subsetopt(M_1)$

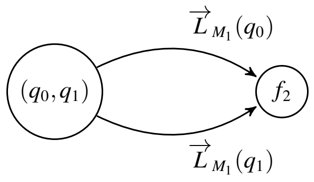
1.3 Minimization by equivalence of states

Let $A = (Q, V, T, , F)$ be a deterministic finite automaton, where Q is a finite set of states, V is a finite set of input symbols, T is a mapping from $Q \times V$ into Q, and $F \subseteq Q$ is the set of final states. No initial state



q
q_0
q_1
f_1
a
a
$\overrightarrow{L}_{M_1}(q_0)$
$\overrightarrow{L}_{M_1}(q_1)$

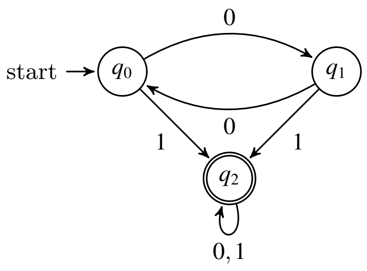
(a) $M_1 = (Q_1, V, T_1, \emptyset, S_1, F_1)$



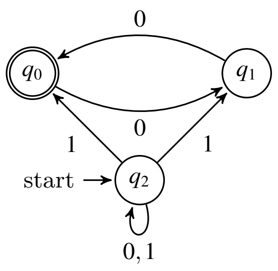
(q_0, q_1)
f_2
$\overrightarrow{L}_{M_1}(q_0)$
$\overrightarrow{L}_{M_1}(q_1)$

(b) $M_2 = (Q_2, V, T_2, \emptyset, S_2, F_2)$

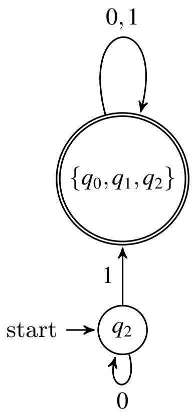
图 1.2: $M_2 = suseful_s \circ subsetopt(M_1)$



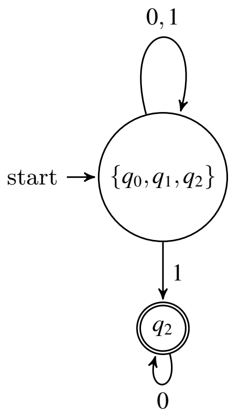
(a) $M_0 = (Q_0, V, T_0, \emptyset, S_0, F_0)$



(b) $M_0^R = (Q_0, V, T_0, \emptyset, S_0, F_0)^R = (Q_0, V, T^R, \emptyset, F_0, S_0)$



(c) $useful_s \circ subsetopt \circ R(M_0)$



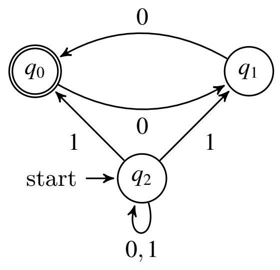
(d) $M_1 = R \circ useful_s \circ subsetopt \circ R(M_0)$

图 1.3: $M_1 = R \circ useful_s \circ subsetopt \circ R(M_0)$

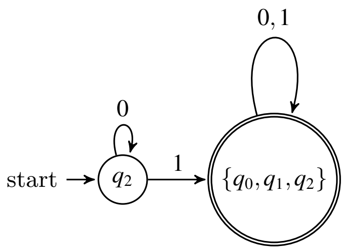
is specified since it is of no importance in what follows. The mapping T is extended to $T \times V^*$ in the usual manner where V^* denotes the set of all finite strings (including the empty string ε) of symbols from V

Definition 1.1 (equivalent states). The states s and t are said to be equivalent if for each $x \in V^*, T(s,x) \in F$ if and only if $T(t,x) \in F$.

$$\begin{aligned}
\text{start: } & U = \{q_2\} \\
u = q_2 : & T(q_2,0) = \{q_2\}, T(q_2,1) = \{q_0,q_1,q_2\} \\
& \text{add new start to } D,\ D = \{q_2,\{q_0,q_1,q_2\}\} \\
u = \{q_0,q_1,q_2\} : & T(\{q_0,q_1,q_2\},0) = T(q_0,0) \cup T(q_1,0) \cup T(q_2,0) = \{q_1\} \cup \{q_0\} \cup \{q_2\} = \{q_0,q_1,q_2\} \\
& T(\{q_0,q_1,q_2\},1) = T(q_0,1) \cup T(q_1,1) \cup T(q_2,1) = \emptyset \cup \emptyset \cup \{q_0,q_1,q_2\} = \{q_0,q_1,q_2\}
\end{aligned}$$



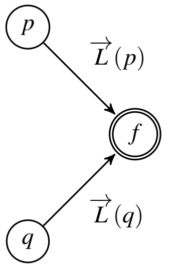
(a) M



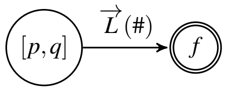
(b) $useful_s \circ subsetopt(M)$

图 1.4: $useful_s \circ subsetopt(M)$

Equivalence relation $E \subseteq Q \times Q$
$(p,q) \in E \equiv (\overrightarrow{L}(p) = \overrightarrow{L}(q))$



(a) $(p,q) \in E$



(b) $(p,q) \in E$

图 1.5: Equivalence relation $E \subseteq Q \times Q$

$\overrightarrow{L}(p) = \bigcup_{a \in V}(\{a\} \cdot \overrightarrow{L}(T(p,a)) \cup \{\varepsilon | p \in F\}$

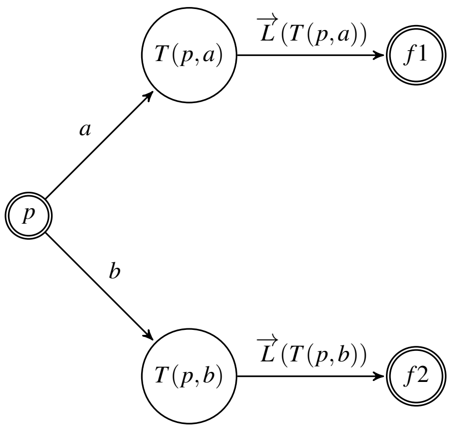


图 1.6: $L(p)$

1.4 From [Hopcroft71]

The algorithm for finding the equivalence classes of Q is described below:

Algorithm 1 The algorithm for finding the equivalence classes of Q

```
Input: $M = (Q, V, T, \_, F)$
Output: The equivalence classes of $Q$
  Step 1. For each $s \in Q$ and each $a \in V$ construct
      $T^{-1}(s,a) = \{t | T(t,a) = s\}$    计算状态 s 的 in-transitionsss
  Step 2. construct $B(1) = F, B(2) = Q - F$ and for each $a \in V$ and $1 \le i \le 2$ construct
  for each $a \in V$ do
      for $i = 1$; $i < n$; $i{+}{+}$ do
          $\hat{B}(B(i), a) = \{s | s \in B(i)$ and $T^{-1}(s,a) \neq \emptyset\}$;
      end for
  end for
  Step 3. Set $k = 3$;
  Step 4. For each $a \in V$ construct $L(a)$
  for each $a \in V$ do
      if $|\hat{B}(B(1),a)| \le |\hat{B}(B(2),a)|$ then
          $L(a) = \hat{B}(B(1),a)$;
      else
          $L(a) = \hat{B}(B(2),a)$;
      end if
  end for
  Step 5. Select $a \in V$ and $i \in L(a)$. The algorithm terminates when $L(a) = \emptyset$ for each $a \in V$.
  Step 6. Delete $i$ from $L(a)$.
  Step 7. For each $j < k$ such that there exists $t \in B(j)$ with $T(t,a) \in \hat{B}(B(i),a)$, perform steps 7a,7b,7c, and 7d.
  Step 7a. partition $B(j)$ into
      $B'(j) = \{t | T(t,a) \in \hat{B}(B(i),a)\}$ and
      $B''(j) = B(j) - B'(j)$
  Step 7b. Replace $B(j)$ by $B'(j)$ and constant $B(k) = B''$. Construct the corresponding $\hat{B}(B(j),a)$ and $\hat{B}(B(k),a)$ for each $a \in V$.
  Step 7c. For each $a \in V$ modify $L(a)$ as follows.
  if $j \notin L(a) \& 0 < |\hat{B}(B(j),a)| \le |\hat{B}(B(k)),a|$ then
      $L(a) = L(a) \cup \{j\}$;
  else
      $L(a) = L(a) \cup \{k\}$;
  end if
  Step 7d. Set $k = k + 1$.
  Step 8. Return to Step 5.
```

Example 1.1. $Q = \{1,2,3,4,5,6\}, V = \{0,1\}, T$ see Fig. 1.7

The algorithm for finding the equivalence classes of Q is described below:

Step 1. For each $s \in Q$ and each $a \in V$ construct $T^{-1}(s,a) = \{t | T(t,a) = s\}$

$T^{-1}(1,0) = \emptyset, T^{-1}(2,0) = \{1\}, T^{-1}(3,0) = \{2\}, \cdots, T^{-1}(6,0) = \{5\}$

$T^{-1}(1,1) = \{1\}, T^{-1}(2,1) = \{2\}, \cdots T^{-1}(6,1) = \{6\},$

Step 2. $B(1) = F = \{6\}, B(2) = Q - F = \{1,2,3,4,5\}$
for each $a \in V$ and $i \in [1,2]$ construct $\hat{B}(B(i),a) = \{s | s \in B(i) \text{ and } T^{-1}(s,a) \neq \emptyset\}$;
$\hat{B}(B(1),0) = \{6\}, \hat{B}(B(2),0) = \{2,3,4,5\}$
$\hat{B}(B(1),1) = \{6\}, \hat{B}(B(1),1) = \{1,2,3,4,5\}$

Step 3. Set $k = 3$
Step 4. For each $a \in V$ construct $L(a)$
$L(0) = \{6\}$, since $|\hat{B}(B(1),0)| = 1 \leq |\hat{B}(B(2),0)| = 4$.
$L(1) = \{6\}$, since $|\hat{B}(B(1),1)| = 1 \leq |\hat{B}(B(2),1)| = 5$.
Step 5. Select $a \in V$ and $i \in L(a)$. The algorithm terminates when $L(a) = \emptyset$ for each $a \in V$.
$a = 0$
$i = 1, \hat{B}(B(i),0) = \{6\}$
Step 6. Delete i From $L(a)$.
$L(0) = L(0) - B(i) = \emptyset$
Step 7. For each $j < k$ such that there exists $t \in B(j)$ with $T(t,a) \in \hat{B}(B(i),a)$, perform steps 7a,7b,7c, and 7d.

Step 7a. Partition $B(j)$ into
$B'(j) = \{t | T(t,a) \in \hat{B}(B(i),a)\} = \{5\}$ and
$B''(j) = B(j) - B'(j)$

Step 7b.

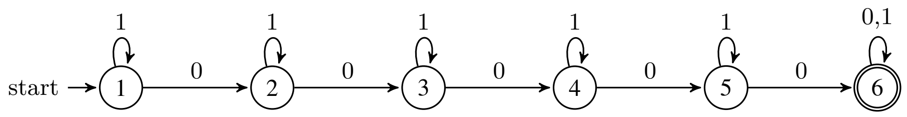


图 1.7: Minimizing example

Example 1.2. Consider the automaton with $Q = \{a,b,c,d,e\}, V = 0,1, F = \{d,e\}$, and T is given by the arcs of diagram of Fig. (1.8).

{a,b} is not equivalent, since $T(a,0) \in F$ but $T(b,0) \notin F$.
{d,e} is not equivalent, since $T(d,0) \in F$ but $T(e,0) \notin F$.
Sets of equivalent states: {a,c},{b},{d},{e}
另外一种描述：

1. $(a,b) \notin E$, since $T(a,0) \in F$ but $T(b,0) \notin F$.
2. $(d,e) \notin E$, since $T(d,0) \in F$ but $T(e,0) \notin F$.
3. $(a,c) \in E$, since $(a,c) \in E \equiv (a \in F \equiv c \in F) \wedge (\forall v \in V, (T(a,v), T(c,v) \in E)$
   $(a \notin F, c \notin F) \Rightarrow (a \in F \equiv c \in F)$
   $T(a,0) = T(c,0) = \{d\} \Rightarrow (T(a,0), T(c,0)) \in E$
   $T(a,1) = T(c,1) = \{c\} \Rightarrow (T(a,1), T(c,1)) \in E$

□.

Algorithm:

1. $B_1 \leftarrow F; B_2 \leftarrow (Q-F)$
   $B_1 = \{d,e\}; B_2 = \{a,b,c\}$
2. $|B_1| = 2, |B_2| = 3. \Rightarrow L \leftarrow (B_1, c)$
   $T(d,0) = \{d\} \in F; T(e,0) = \{c\} \notin F$
   $\Rightarrow (d,e)$ is not equivalent states.
   $T(d,1) = \{d\} \in F; T(e,1) = \{e\} \in F. \Rightarrow$ 无法判断。
   $L = (B_1, 0);$
3. split (d,c)
   $T(d,0) = \{d\} \in F; T(c,0) = \{d\} \notin F$ 无法判断
   $T(d,1) = \{d\} \in F; T(c,1) = \{c\} \notin F$
   $\Rightarrow (d,c)$ is not equivalent states.

{a,b},{d,e}is not equivalent states.
Sets of equivalent states: {a,c},{b},{d},{e}

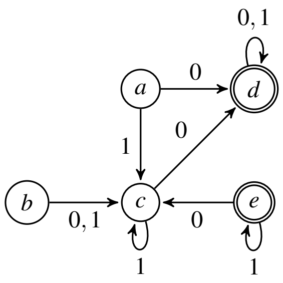


图 1.8: Finite state automaton

1.5 From [Ratnesh95]

FSM: Finte State Machine
NFSM: Non-deterministic Finite State Machine without ε−moves
DFSM: Deterministic Finite State Machine

Definition 1.2 (prefix closure of K). The prefix closure of K, denoted $pr(K) \subseteq \Sigma^*$, is the language

$$pr(K) := \{s \in \Sigma^* | \exists t \in K : s \leq t\}$$

Example 1.3 (Language). Consider for example a buffer of capacity one; it has two different states: empty and full. When an *arrival* event occurs in the empty state, then the buffer becomes full; and when a *departure* event occurs in the full state, then the buffer becomes empty. No other state transition can occur in the buffer. Suppose initially the buffer is empty. Then the language of the buffer consists of all possible sequences of the type:

$$arrival \cdot departure \cdot arrival \cdot departure \ldots,$$

where "·" denotes the operation of concatenation. □

Example 1.4 (Generated language). Consider the buffer of Example 1.3 Let a, d denote the arrival, departure events respectively. Then the generated language of the buffer is $pr((a\cdot d)^*)$. Suppose a trace $s \in pr((a\cdot d)^*)$ corresponds to completion of a task if and only if its execution results in the empty state of the buffer. Then the marked language of the buffer equals $(a\cdot d)^*$. □

Example 1.5 (language model). Consider the buffer of Examples 1.3 and 1.4 with language model $[(ad)^*, pr((ad)^*)]$. The directed graph shown in Figure 1.9 represents a DSM $G := (X,\Sigma,\alpha,x_0,X_m)$ for the buffer, where $X = \{empty, full\}; \Sigma = \{a,d\}; x_0 = empty; X_m = \{empty\}$; and $\alpha(empty,a) = full, \alpha(full,d) = empty$. Note that $\alpha(empty,d)$ and $\alpha(full,a)$ are not defined; hence the transition function is a partial map. (A node in the graph represents a state; a label on a node represents the name of the corresponding state; a directed edge represents a state transition; a label on a directed edge represents the name of the corresponding event; an arrow entering a node represents an initial state; and a circled node represents a marked state.) □

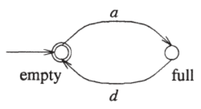


图 1.9: Graph representing a DSM

Example 1.6 (Synchronous). Consider for example a manufacturing production line consisting of a machine (M) and a buffer (B) of capacity one operating in synchrony as shown in Figure 1.10. The event set Σ_1 of M consists of events a_1 representing arrival into the machine, and d_1 representing departure from the machine; whereas the event set Σ_2 of B consists of events d_1 representing departure from the machine, and d_2 representing departure from buffer. The synchronous composition of two systems is also shown if Figure 1.10.

Note 1.1. d_1 是共享事件，因此，(idle,empty) 状态下，d_1 不能发生, 仅发生 a_1 事件; (working,empty) 状态下，d_1 在 M 和 B 中的转移函数均有定义，因此该共享事件可以发生该共享事件。非共享事件 (a_1,d_2) 在 M 或 B 中的转移函数有一个有定义，即可发生。

□

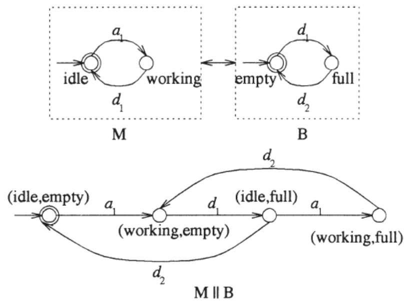


图 1.10: Diagram illustrating synchronous composition of DFSMs

Example 1.7 (ε-NSM). Consider the ε-NSM of Fig. 1.11 Then $\varepsilon^*_G(1) = \{1,2,3\}, \varepsilon^*_G(2) = \{2,3\}, \varepsilon^*_G(3) = \{3\}$.

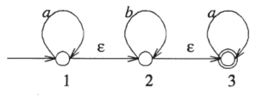


图 1.11: Diagram illustrating an $\varepsilon - NSM$

so $T(1,\varepsilon) = \varepsilon_G^*(1) = \{1,2,3\}; T(1,a) = \varepsilon_G^*(T(T(1,\varepsilon),a)) = \varepsilon_G^*(T(\{1,2,3\},a) = \varepsilon_G^*(\{1,3\}) = \{1,2,3\}; T(1,ab) = \varepsilon_G^*(T(\{1,2,3\},b) = \varepsilon_G^*(\{2\}) = \{2,3\}$, etc. □

Example 1.8 (Completion and reverse). Completion and reverse of the DSM of Figure 1.13(b) are shown in Figure 1.12 (a) and 1.12 (b) respectively. The state labels have been changed.

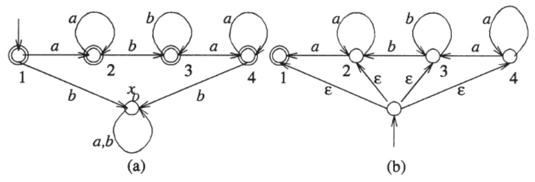


图 1.12: Diagram illustrating completion and reverse operations

Theorem 1.1 (power set construction). *Let $G := (X,\Sigma,\alpha,x_0,X_m)$ be a NFSM. Then there exists a language model equivalent DFSM $\mathscr{G} := (\mathscr{X},\Sigma,\hat{\alpha},\{x_0\},\mathscr{X}_m)$ and $L(\mathscr{G}) = L(G)$*

证明. Define $\mathscr{X} := 2^X$, $\mathscr{X}_m := \{\hat{X} \in \mathscr{X} | \hat{X} \cap X_m \neq \emptyset\}$, and

$$\forall \hat{X} \in \mathscr{X}, \sigma \in \Sigma : \hat{\alpha}(\hat{X},\sigma) := \bigcup_{x \in \hat{X}} \alpha(x,\sigma)$$

Then is is easily show that $(L_m(\mathscr{G}), L(\mathscr{G})) = (L_m(G), L(G))$

Example 1.9. (NFSM to DFSM) Consider the NFSM $G := (X,\Sigma,\alpha,x_0,X_m)$ shwown in Figure 1.13(a). The language equivalent DFSM $\mathscr{G} := (\mathscr{X},\Sigma,\hat{\alpha},\{x_0\},\mathscr{X}_m)$ obtained using the power set construction is shown in Figure 1.13(b).

Note that $\mathscr{X} = \mathscr{X}_m = \{\{1\},\{1,2,3\},\{2,3\},\{3\}\}$, as $X_m = \{1,3\}$

which has a nonempty intersection with each state in $\mathscr{X}$;

$$\hat{\alpha}(\{1\},a) = \alpha(1,a) = \{1,2,3\};$$
$$\hat{\alpha}(\{1,2,3\},a) = \alpha(1,a) \cup \alpha(2,a) \cup \alpha(3,a) = \{1,2,3\};$$
$$\hat{\alpha}(\{1,2,3\},b) = \alpha(1,b) \cup \alpha(2,b) \cup \alpha(3,b) = \{2,3\};$$
$$etc.$$

Remark 1.1. It follows from Theorems 1.1 that if a language model (K_m,K) can be represented as a finite state machine G, then there also exists a DFSM G' such that $(L_m(G'),L(G')) = (K_m,K)$. Thus if we are only concerned with DESs that have finitely many states, then we can assume without loss of generality that they

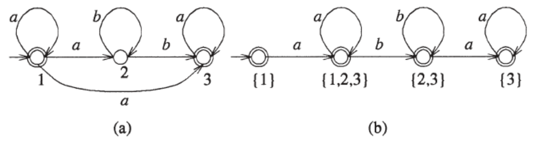


图 1.13: Diagram illustrating NFSM to DFSM conversion

can be represented as DFSMs. We will see below that although the finiteness of states is not needed for most of the analysis, it is needed for developing all the decision algorithms.

However, it should be noted that although DFSMs are useful in developing decision algorithms, it is conceptually easier to obtain a NFSM from the given description of a language. For example, suppose $\Sigma = \{a,b\}$, and suppose we wish to represent the language with the property that every string in it must contain *aba* as a substring. A NFSM for the same is shown in Figure 1.14(a); corresponding DFSM obtained using the construction outlined in Theorem 1.1 is shown in Figure 1.14(b).

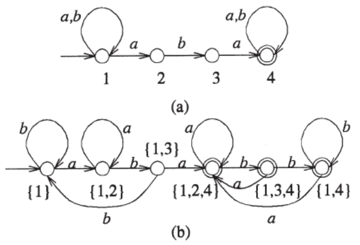


图 1.14: NFSM accepting strings with aba as a substring, and corresponding DFSM

1.5.1 Myhill-Nerode Characterization

Definition 1.3 (equivalence relation(R_K)). Given a language $K \subseteq \Sigma^*$, it induces an equivalence relation, denoted R_K, on Σ^*:

$$\forall s,t \in \Sigma^* : s \cong t(R_K) \Leftrightarrow [K \setminus \{s\} = K \setminus \{t\}]$$

For each $s \in \Sigma^*, [s]R_K \subseteq \Sigma^*$ is used to denote the equivalence class containing the string s.

Definition 1.4 (equivalence relation(R_G)). Given a DFSM $G := (X, \Sigma, \alpha, x_0, X_m)$, it induces an equivalence relation, denoted R_G, on Σ^*:

$$\forall s,t \in \Sigma^* : s \cong t(R_G) \Leftrightarrow [\alpha(x_0,s) = \alpha(x_0,t)] \vee [\alpha(x_0,s) = \alpha(x_0,t) \text{ undefined}]$$

For each $s \in \Sigma^*, [s]R_G \subseteq \Sigma^*$ is used to denote the equivalence class containing the string s.

Note 1.2. Note the R_G, the *index* of R_G, i.e., the number of equivalence classes of R_G, is one more than the number of states in G, $|R_G| = |G| + 1$. (The set of all strings that do not belong to $L(G)$ belong to a single equivalence class of R_G.)

It can be easily seen that the equivalence relation R_G refines the equivalence relations $R_{L_m(G)}$ and $R_{L(G)}$. In other words,

$$\forall s,t \in \Sigma^* : s \cong t(R_G) \Rightarrow [s \cong t(R_{L_m(G)})] \wedge [s \cong t(R_{L(G)})]$$

An equivalence relation R on Σ^* is said to be *right invariant (with respect to concatenation)* if

$$\forall s,t \in \Sigma^* : s \cong t(R) \Rightarrow su \cong tu(R)$$

It is easy to verify that R_K as well as R_G defined above are right invariant.The following proposition is due to Myhill and Nerode:

Theorem 1.2 (Myhill and Nerode). *Let $K \subseteq \Sigma^a st$ be a language. Then the following are equivalent:*

1. *K is regular.*
2. *K can be written as union of some of the equivalence classes of a right invariant equivalence relation of finite index.*
3. *R_K is of finite index.*

证明. (1) $\Rightarrow$ (2): Suppose K is regular. Then there exists a DFSM G such that $L_m(G) = K$. Then clearly K can be written as union of the following equivalence classes of R_G:

$$\{[s](R_G) | s \in K\}$$

This proves the first assertion implies the second assertion, as R_G is right invariant.

(2) $\Rightarrow$ (3): Let R be a right invariant equivalence relation of finite index such that K can be written as union of some of the equivalence classes of R. In order to show that R_K is of finite index it suffices to show that R refines R_K. Pick $s,t \in \Sigma^*$ such that $s \cong t(R)$. Since R is right invariant, for any $u \in \Sigma^*, su \cong tu(R)$. Since K equals union of some of the equivalence classes of R, this implies $su \in K$ if and only if $tu \in K$. In other words, $s \cong t(R_K)$, which proves that the second assertion implies the third assertion.

(3) $\Rightarrow$ (1): Finally, suppose R_K is of finite index. Define a DFSM $G := (\hat{X}, \Sigma, \hat{\alpha}, \hat{x}_0, \hat{X}_m)$ as follows: $\hat{X} := \{[s](R_K) | s \in \Sigma^*\}; x_0 := [\varepsilon](R_K); \hat{X}_m = \{[s](R_K) | s \in K\}$; and

$$\forall [s](R_k) \in \hat{X}, \sigma \in \Sigma : \hat{\alpha}([s](R_K), \sigma) := [s\sigma](R_K)$$

Then it is readily verified that for each $s \in \Sigma^*, \hat{\alpha}(\hat{x}_0, s) = [s](R_K)$. Hence from definition of marked language we obtain that $s \in L_m(G)$ if and only if $\hat{\alpha}(\hat{x}_0, s) = [s](R_K) \in X_m$, i.e., if and only if $s \in K$. Thus $L_m(G) = K$. Since $\hat{G}$ is a DFSM (as R_K is of finite index), this implies that K is regular; so the third assertion implies the first assertion. □

The construction of the DFSM G in the proof of Theorem 1.2 is known as the Myhill-Nerode construction. The following example illustrates such a construction.

Example 1.10 (equivalence classes). Consider for example the marked language $K_m = (ad)^*$ of the buffer of capacity one of Example 1.4. Then the generated language of the buffer is $pr((a \cdot d)^*) = pr(K_m)$. Suppose a trace $s \in pr((a \cdot d)^*)$ corresponds to completion of a task if and only if its execution results in the empty state of the buffer. Then the marked language of the buffer equals $(a \cdot d)^*$.

Clearly, K_m is a regular language. Hence it follows from Theorem 1.2 that R_{K_m} is of finite index. It can be easily verified that

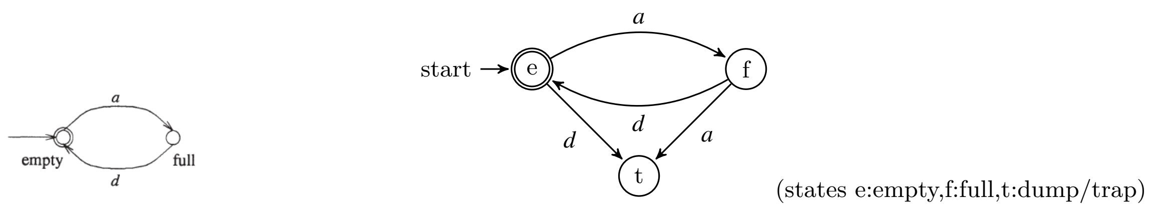
$$[\varepsilon](R_{K_m}) = (ad)^* = K_m$$
$$[a](R_{K_m}) = (ad)^* a = pr(K_m) - K_m$$
$$[d](R_{K_m}) = \{a,d\}^* - pr(K_m)$$

and these are the only equivalence classes of R_{K_m}

Hence Myhill-Nerode construction yields the DFSM $G := (\hat{X}, \Sigma, \hat{\alpha}, \hat{x}_0, \hat{X}_m)$, while $\hat{X} = \{[\varepsilon](R_{K_m}), [a](R_{K_m}), [d](R_{K_m})\}$; $\hat{x}_0 = [\varepsilon](R_{K_m}); X_m = \{[\varepsilon](R_{K_m})\}$;
and

$$\hat{\alpha}([\varepsilon](R_{K_m}), a) = [a](R_{K_m});$$
$$\hat{\alpha}([\varepsilon](R_{K_m}), d) = [d](R_{K_m});$$
$$\hat{\alpha}([a](R_{K_m}), a) = [aa](R_{K_m}) = [d](R_{K_m});$$
$$\hat{\alpha}([a](R_{K_m}), d) = [ad](R_{K_m}) = [\varepsilon](R_{K_m});$$
$$\hat{\alpha}([d](R_{K_m}), a) = [da](R_{K_m}) = [d](R_{K_m});$$
$$\hat{\alpha}([d](R_{K_m}), d) = [dd](R_{K_m}) = [d](R_{K_m});$$

See figure 1.15, State e: empty,$[\varepsilon](R_{K_m})$; State f: full,$[a](R_{K_m})$; State t: dump/trap,$[d](R_{K_m})$.



(a) buffer model (b) equivalence classes

图 1.15: equivalence classes

□

Remark 1.2. Given a regular language K, there always exists a DFSM G such that $L_m(G) = K$. Hence there exists a minimal such DFSM (one with a minimal number of states). Let G' be the DFSM obtained by removing the state $[s](R_K)$ form $\hat{G}$ (and all transitions leading into/out of it), where $\hat{G}$ is the DFSM in the proof of Theorem 1.2 and $s \in \Sigma^*$ is such that $s \notin pr(K)$.Then it is easy to see that $L_m(G') = K$. In fact G' is a minimal DFSM with marked language K. In order to see this first note that the number of states in $\hat{G}$ is $|R_K|$, the index of R_K. Since G' is obtained by removing a single state from $\hat{G}$, the number of states in G' equals $|R_K| - 1$. Let G be any DFSM with $L_m(G) = K$. Then as noted above number of states in G equals $|R_G| - 1$. Also,as noted above, R_G refines $R_{L_m(G)} = R_K$, which implies that $|R_K| \leq |R_G|$. Thus $|R_K| - 1 \leq |R_G| - 1$, which proves that G' is a minimal DFSM with marked language K.

1.5.2 Minimization

Theorem 1.3 (Minimal K). *Suppose $K \subseteq \Sigma^*$ is a regular language. Then a trim DFSM $G := (X, \Sigma, \alpha, x_0, X_m)$ with marked language K is mininmal if and only if*

$$\forall x, x' \in X, s \in \Sigma^* : (\alpha(x,s), \alpha(x',s)) \in X_m \times X_m \Rightarrow x = x'.$$

Suppose $K \subseteq \Sigma^*$ is regular. Let $G := (X, \Sigma, \alpha, x_0, X_m)$ be a trim DSFSM such that $L_m(G) = K$. We are interested in obtaining a minimal DFSM with marked language K by combining some of the "language equivalent" states of G. Note that if $K = \Sigma^*$, then it can be accepted by a minimal DFSM having a single state. Hence we assume without loss of generality that $K \neq \Sigma^*$.

DFSM $G := (X, \Sigma, \alpha, x_0, X_m)$ induces an equivalence relation on X defined as:

$$\forall x, x' \in X : x \cong x' \Leftrightarrow [\forall s \in \Sigma^* : \alpha(x,s) \in X_m \Leftrightarrow \alpha(x',s) \in X_m]$$

Using this equivalence relation we define a language equivalent DFSM $\hat{G} := (\hat{X}, \Sigma, \hat{\alpha}, \hat{x}_0, \hat{X}_m)$, where $\hat{X} := \{[x] | x \in X\}, \hat{x}_0 := [x_0], \hat{X}_m := \{[x] | x \in X_m\}$, and

$$\forall [x] \in \hat{X}, \sigma \in \Sigma : \hat{\alpha}([x], \sigma) := \begin{cases} [\alpha(x,\sigma)] \text{ if } \alpha(x,\sigma) \text{ defined} \\ \text{undefined otherwise} \end{cases}$$

Using the fact that G is trim, it is readily verified that $\hat{G}$ is well defined, and $(L_m(\hat{G}), L(\hat{G})) = (L_m(G), L(G)) = (K, pr(K))$. Moreover, it follows from Theorem 1.3 that $\hat{G}$ is a minimal state machine with marked language K.

An algorithm for efficiently identifying the equivalence classes $\{[x] | x \in X\}$ is presented next. Note that each state pair $(x, x') \in (X_m \times X_m) \cup [(X - X_m) \times (X - X_m)]$ is a possible pair of equivalent states.

First consider $\bar{G} := (\bar{X}, \Sigma, \bar{\alpha}, x_0, X_m)$, the completion of G. Then since $K \neq \Sigma^*, X_m \neq \bar{X} = X \cup \{x_D\}$, while X_D is the "dump" state. This implies $\bar{X} - X_m \neq \emptyset$.

Note 1.3. Note that if $K = \Sigma^*$, then it can be accepted by a minimal DFSM having a single state.

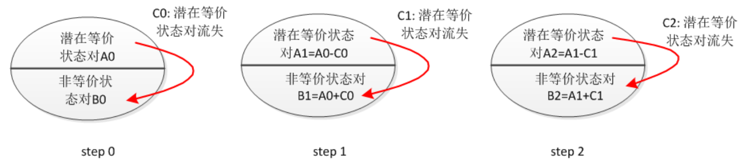


图 1.16: Diagram illustrating algorithm minimize

It can be verified that after termination, each state pair in A_k is an equivalent pair of states. Finally, the minimal state machine G is obtained by combining each state pair in A_k as described above, and removing the equivalence class of the dump state.

Algorithm 2 terminates in $O(m^2)$ steps, where m is the number of states in G.

Algorithm 2 The algorithm for a minimal DFSM, see Figure 1.16

```
Input: $G = (\bar{X}, \Sigma, \alpha, \_, X_m)$
Output: The equivalence classes of $X$
    A: {潜在等价状态对}; B: {非等价状态对}; C: {本次迭代流失的潜等价状态对}
1: Initiation step:
    $A_0 := [X_m \times X_m] \cup [(\bar{X} - X_m) \times (\bar{X} - X_m)] = \{(\text{final 状态对}) \cup (\text{非 final 状态对})\} = \{\text{潜在的等价状态对}\}$;
    $B_0 := [\bar{X} \times \bar{X} - A_0] = \{(\text{全体状态对}) - A_0\} = \{(\text{final 状态, 非 final 状态})\} = \{\text{非等价状态对}\} = \{\text{非 } A_0 \text{ 状态对}\}$;
    $k := 0$.
2: Iteration step:
```

$$\begin{aligned}
C_k &:= \{(x, x') \in A_k | \exists \sigma \in \Sigma \text{ s.t. } (\alpha(x, \sigma), \alpha(x', \sigma)) \in B_k\} \\
&= \{A_0 \text{ 中两大类中的 (状态对) 存在字母 } \sigma \text{ 进入 } B_0 \text{ 类)}\} \\
&= \{\text{潜在状态对} \rightarrow \text{非等价状态对}\} \\
A_{k+1} &:= A_k - C_k \\
B_{k+1} &:= B_k \cup C_k = [\bar{X} \times \bar{X}] - A_{k+1} \\
&= \{\text{非 } A_{k+1} \text{ 状态对}\}
\end{aligned}$$

```
3: Termination step:
    if $A_{k+1} = A_k$, then stop; else, $k := k + 1$, and goto step 2.
```

Example 1.11. Consider the complete DFSM of Figure 1.17. Then $X_m = \{1, 2, 3, 4\}$ and $X = \{1, 2, 3, 4, x_D\}$. Algorithm 2 can be applied to minimize the DFSM as follows:

$A_0 = [X_m \times X_m] \cup \{(x_D \times x_D)\};$

$= \{(1,1),(2,2),(3,3),(4,4),(1,2),(2,1),(1,3),(3,1),(1,4),(4,1),(2,3),(3,2),(2,4),(4,2),(3,4),(4,3)\} \cup \{(x_D,x_D)\}$

$= \{(1,1),(2,2),(3,3),(4,4)\} \cup \{(1,2),(1,3),(1,4),(2,3),(2,4),(3,4)\} \cup \{(2,1),(3,1),(4,1),(3,2),(4,2),(4,3)\} \cup \{x_D,x_D\}$

$= \{$潜在的等价状态对$\}$

$B_0 = [X \times X] - A_0$

$= \{(1,x_D),(x_D,1),(2,x_D),(x_D,2),(3,x_D),(x_D,3),(4,x_D),(x_D,4)\}$

$= \{(1,x_D),(2,x_D),(3,x_D),(4,x_D)\} \cup \{(x_D,1),(x_D,2),(x_D,3),(x_D,4)\}$

$= \{$非等价状态对$\} = \{$非 A_0 状态对$\}$

$C_0 = \{(1,2),(2,1),(1,3),(3,1),(2,4),(4,2),(3,4),(4,3)\}$

$= \{(1,2),(1,3),(2,4),(3,4)\} \cup \{(2,1),(3,1),(4,2),(4,3)\}$

$= \{$潜在状态对 $(A_0) \to (B_0)$ 非等价状态对$\}$

$A_1 = A_0 - C_0$

$= \{(1,1),(2,2),(3,3),(4,4)\} \cup \{(1,4),(2,3)\} \cup \{(4,1),(3,2)\} \cup \{x_D,x_D\}$

$B_1 = [X \times X] - A_1$

$= \{$非 A_1 状态对$\}$

$C_1 = \{$潜在状态对 $(A_1) \to (B_1)$ 非等价状态对$\}$

$= \{(1,4),(4,1),(2,3),(3,2)\}$

$A_2 = A_1 - C_1$

$= \{(1,1),(2,2),(3,3),(4,4),(x_D,x_D)\}$

$B_2 = [X \times X] - A_2$

Clearly, all the state pairs in A_2 are equivalent pairs, i.e. $C_2 = \emptyset$: hence the algorithm terminates. Thus the minimal DFSM is the DFSM of Figure 1.17, which does not contain the dump state.

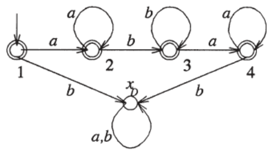


图 1.17: the minimal DFSM is not contain the dump state

1.6 From [Jean2011]

Definition 1.5 (Partitions and equivalence relations). A partition of a set E is a family P of nonempty, pairwise disjoint subsets of E such that $E = \bigcup_{\mathscr{P} \in P} \mathscr{P}$. The *index* of the partition is the number of its elements. A partition defines an equivalence relation $\equiv p$ on E. Conversely, the set of all equivalence classes $[x]$, for $x \in E$, of an equivalence relation on E defines a partition of E. This is the reason why all terms defined for partitions have the same meaning for equivalence relations and vice versa.

A subset F of E is *saturated* 使充满 by P if it is the union of classes of P. Let Q be another partition of E. Then Q is a *refinement* of P, or P is *coarser* than Q, if each class of Q is contained in some class of P. If this holds, we write $Q \leq P$. The index of Q is then larger than the index of P.

Given two partitions P and Q of a set E, we denote by $U = P \wedge Q$ the coarsest partition 粗划分 which refines P and Q. The classes of U are the nonempty sets $\mathscr{P} \cap \mathscr{Q}$, for $\mathscr{P} \in P$ and $\mathscr{Q} \in Q$. The notation is extended to a set of partitions in the usual way: we write $P = P_1 \wedge \cdots P_n$ for the common refinement of $P_1, \ldots, P_n$. If $n = 0$, then P is the universal partition of E composed of the single class E. This partition is the neutral element for the $\wedge$-operation.

Let F be a subset of E. A partition P of E induces a partition P' of F by intersection: P' is composed of the nonempty sets $\mathscr{P} \cap F$, for $\mathscr{P} \in P$ If P and Q are partitions of E and $Q \leq P$, then the restrictions P' and Q' to F still satisfy $Q' \leq P'$.

If P and P' are partitions of disjoint sets E and E', we denote by $P \vee P'$ the partition of $E \cup E'$ whose restriction to E and E' are P and P' respectively. So, one may write

$$P = \bigvee_{\mathscr{P} \in P} \{\mathscr{P}\}$$

Definition 1.6 (Minimal automaton). We consider a deterministic automaton $\mathscr{A} = (Q, i, F)$ over the alphabet A with set of states Q, initial state i, and set of final states F. To each state q corresponds a subautomaton of $\mathscr{A}$ obtained when q is chosen as the initial state. We call it the *subautomaton rooted at q* or simply the automaton at q. Usually, we consider only the trim part of this automaton. To each state q corresponds a language $L_q(\mathscr{A})$ which is the set of words recognized by the subautomaton rooted at q, that is

$$L_q(\mathscr{A}) = \{w \in A^* | q \cdot w \in F\}$$

This language is called the *future* of the state q, or also the *right language* of this state. Similarly one defines the *past* of q, also called the *left language*, as the set $\{w \in A^* | i \cdot w = q\}$. The automaton $\mathscr{A}$ is *minimal* if $L_p(\mathscr{A}) \neq L_q(\mathscr{A})$ for each pair of distinct states p, q. The equivalence relation $\equiv$ defined by

$$p \equiv q \text{ if and only if } L_p(\mathscr{A}) = L_q(\mathscr{A})$$

is a *congruence*, that is $p \equiv q$ implies $p \cdot a \equiv q \cdot a$ for all letters a. It is called the *Nerode congruence*. Note that the Nerode congruence saturates the set of final states. Thus an automaton is minimal if and only if its Nerode equivalence is the identity 恒等式.

Minimizing an automaton is the problem of computing the Nerode equivalence. Indeed, the *quotient* automaton $\mathscr{A}/\equiv$ obtained by taking for set of states the set of equivalence classes of the Nerode equivalence, for the initial state the class of the initial state i, for set of final states the set of equivalence classes of states in F and by defining the transition function by $[p] \cdot a = [p \cdot a]$ accepts the same language, and its Nerode equivalence is the identity. The minimal automaton recognizing a given language is unique.

Definition 1.7 (Partitions and automata). Again, we fix a deterministic automaton $\mathscr{A} = (Q, i, F)$ over the alphabet A. It is convenient to use the shorthand P^c for $Q \setminus P$ when P is a subset of the set Q.

Given a set $P \subset Q$ of states and a letter a, we denote by $a^{-1}P$ the set of states q such that $q \cdot a \in P$. Given sets $P, R \subset Q$ and $a \in A$, we denote by

$$(P,a)|R$$

the partition of R composed of the nonempty sets among the two sets

$$R \cap a^{-1}P = \{q \in R | q \cdot a \in P\} \text{ and } R \setminus a^{-1}P = \{q \in R | q \cdot a \notin P\}$$

Note that $R \setminus a^{-1}P = R \cap (a^{-1}P)^c = R \cap a^{-1}(P^c)$ so the definition is symmetric in P and P^c. In particular

$$(P,a)|R = (P^c,a)|R \tag{1.1}$$

The pair (P,a) is called a *splitter*. Observe that $(P,a)|R = \{R\}$ if either $R \cdot a \subset P$ or $R \cdot a \cap P = \emptyset$, and $(P,a)|R$ is composed of two classes if both $R \cdot a \neq \emptyset$ and $R \cdot a \cap P^c \neq \emptyset$ or equivalently if $R \cdot a \not\subseteq P^c$ and $R \cdot a \not\subseteq P$. if $(P,a)|R$ contains two classes, then we say that (P,a) *splits* R. Note that the pair $S = (P,a)$ is called a splitter even if it does not split.

It is useful to extend the notation above to words. Given a word w and sets $P, R \subset Q$ of states, we denote by $w^{-1}P$ the set of states such that $q \cdot w \in P$, and by $(P,w)|R$ the partition of R composed of the nonempty sets among

$$R \cap w^{-1}P = \{q \in R | q \cdot w \in P\} \text{ and } R \setminus w^{-1}P = \{q \in R | q \cdot w \notin P\}$$

As an example, the partition $(F,w)|Q$ is the partition of Q into the set of those states from which w is accepted, and the other ones. A state q in one of the sets and a state q' in the other are sometimes called *separated* by w.

The Nerode equivalence is the coarsest equivalence relation on the set of states that is a (right) congruence saturating F. With the notation of splitters, this can be rephrased as follows.

We use later the following lemma which is already given in Hopcroft' s paper [[Hopcroft71]]. It is the basic observation that ensures that Hopcroft' s algorithm works correctly.

Proposition 1.1. *The partition corresponding to the Nerode equivalence is the coarsest partition P such that no splitter (P,a), with $P \in \mathscr{P}$ and $a \in A$, splits a class in $\mathscr{P}$, that is such that $(P,a)|R = \{R\}$ for all $P, R \in \mathscr{P}$ and $a \in A$.* □

Lemma 1.1. *Let P be a set of states, and let $\mathscr{P} = \{P1, P2\}$ be a partition of P. For any letter a and for any set of states R, one has*

$$(P,a)|R \wedge (P_1,a)|R = (P,a)|R \wedge (P_2,a)|R = (P_1,a)|R \wedge (P_2,a)|R$$

and consequntly

$$(P,a)|R \geq (P_1,a)|R \wedge (P_2,a)|R, \tag{1.2}$$

$$(P_1,a)|R \geq (P,a)|R \wedge (P_2,a)|R \tag{1.3}$$

Example 1.12. We consider the automata given in Figure 1.18 over the alphabet $A = a,b$. Each automaton is the reversal of the other. However, determinization of the automaton on the left requires exponential time and space.

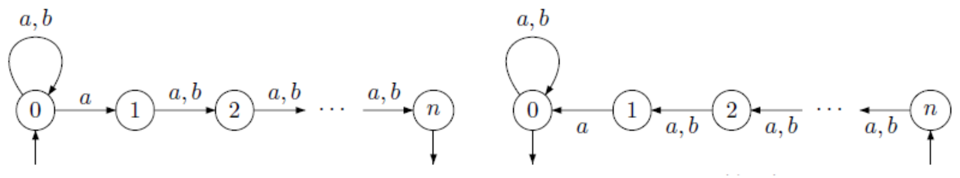


图 1.18: The automaton on the left recognizing the language A^*aA^n. It has $n+1$ states and the minimal deterministic automaton for this language has 2^n states. The automaton on the right is its reversal. It is minimal and recognizes A^naA^*.

References

Hopcroft71. Hopcroft, J.E. *An n log n algorithm for minimizing states in a finite automaton*, in The Theory of Machines and Computations (Z. Kohavi, ed.), pp.180-196, Academic Press, New York, 1971.

Gries73. Gries, D. *Describing an Algorith m by Hopcroft*, Acta Inf. 2:97 109, 173. © by Springer-Verlag 1973

Ratnesh95. Ratnesh Kumar, *Modeling and Control of Logical Discrete Event Systems*, © 1995 by Springer Science+Business Media New York.

Jean2011. Jean Berstel, Luc Boasson, Olivier Carton, Isabelle Fagnot , *Minimization of automata*, Universit*é* Paris-Est Marne-la-Vall*é*e 2010 Mathematics Subject Classification: 68Q45, 2011.