目录

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

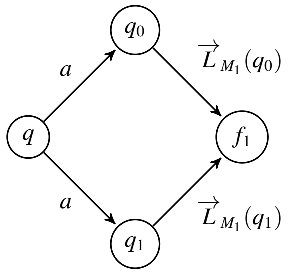
$$\begin{aligned}
&q_0, q_1 \in Q_1, Q_2 \subseteq \mathbb{P}(Q_1), \forall p \in Q_2, p = (q_0, q_1)\\
&\overrightarrow{L}_{M_2}(p) = \overrightarrow{L}_{M_1}(q_0) \cup \overrightarrow{L}_{M_1}(q_1)\\
&\Rightarrow\\
&\overrightarrow{L}_{M_2}(p) = \bigcup_{q \in p} \overrightarrow{L}_{M_1}(q)\\
&\Rightarrow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

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

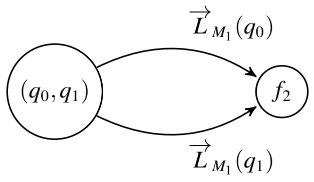
图 1.1: $M_2 = suseful_s \circ subsetopt(M_1)$

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

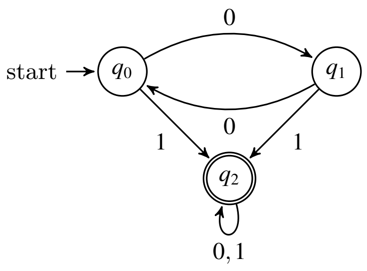

q_0
a
$\overrightarrow{L}_{M_1}(q_0)$
q
f_1
a
$\overrightarrow{L}_{M_1}(q_1)$
q_1

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

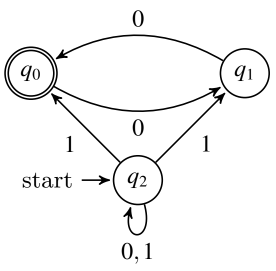

$\overrightarrow{L}_{M_1}(q_0)$
(q_0, q_1)
f_2
$\overrightarrow{L}_{M_1}(q_1)$

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

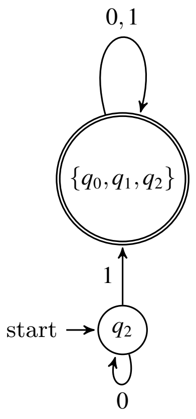
图 1.2: $M_2 = suseful_s \circ subsetopt(M_1)$



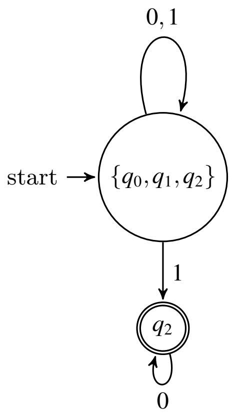
(a) $M_0 = (Q_0, V, T_0, \emptyset, S_0, F_0)$



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



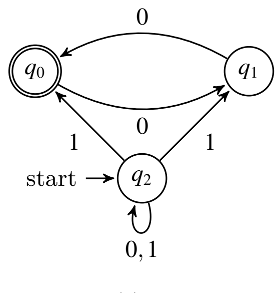
(d) $M_1 = R \circ useful_s \circ subsetopt \circ R(M_0)$

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

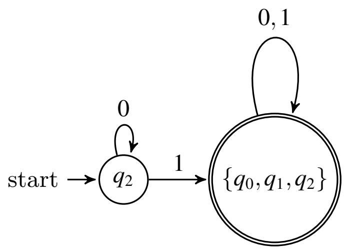
is specified since it is of no importance in what follows. The mapping T is extended to $T \times V^*$ in the usual manner where V^* denotes the set of all finite strings (including the empty string ε) of symbols from V

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 $T(t,x) \in F$.

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



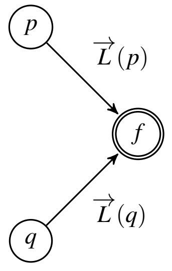
(a) M



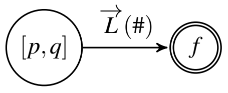
(b) $useful_s \circ subsetopt(M)$

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 $E \subseteq Q \times Q$

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

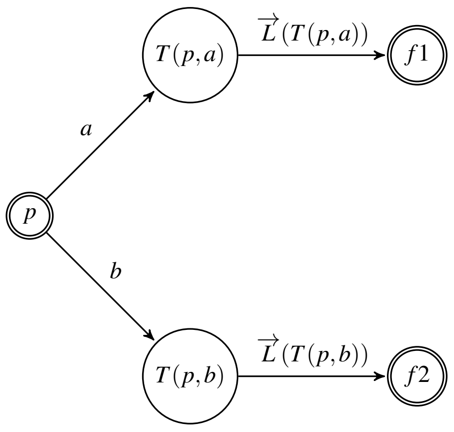


图 1.6: $L(p)$

a: $((T(p,a),T(q,a)) \in E_0$; b: $((T(p,a),T(q,a)) \notin E_0$, c: $((T(p,a),T(q,a)) \in E_1(part_1)$; d: $((T(p,a),T(q,a)) \notin E_1(part_1)$;

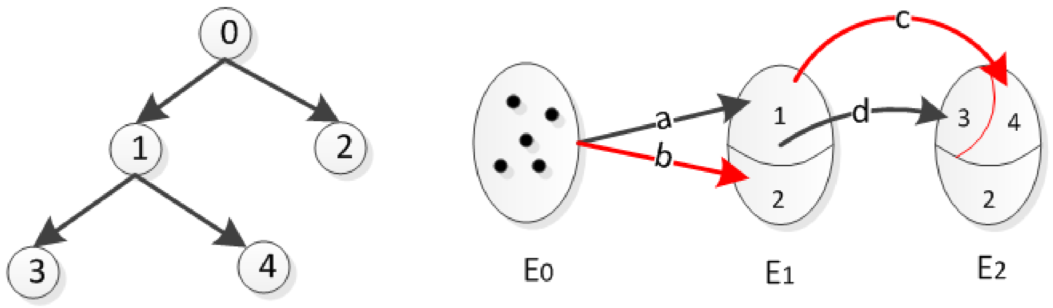


图 1.7: Approximating $E, E_0 = (Q \setminus F)^2 \cup F^2$

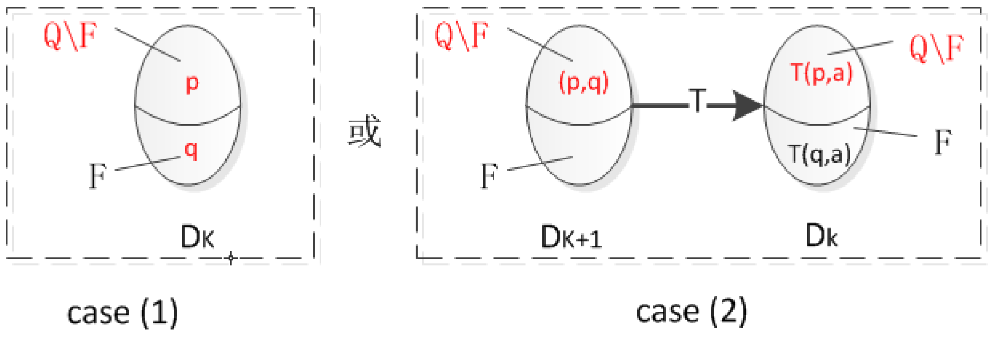


图 1.8: Approximating $D, D_0 = ((Q \setminus F) \times F) \cup (F \times (Q \setminus F))$

initial, $P = [Q]_{E_0} = \{F, Q \setminus F\}, Q_0, Q_1 \in P$,
if $(\exists p,q \in Q_0, T(p,a) \in Q_1$, and $T(q,a) \notin Q_1)$, then
split Q_0 (wrt Q_1) $\rightarrow$ two parts, (1): $Q_0' \in Q_1$ and (2): $(Q_0 \setminus Q_0') \notin Q_1$

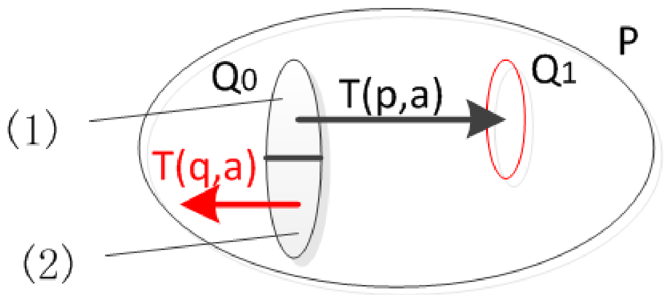


图 1.9: split Q_0 wrt Q_1

1.4 From [Gries73]

1.4.1 Problem Definition

DFA: $A = (S, I, \delta, F)$, No initial state is specified since it is of no importance in what follows.

Definition 1.2 (equivalent states). States s and t are said to be equivalent if for each $x \in I^*, \delta(s,x) \in F$. if and only if $\delta(t,x) \in F$.

We want an algorithm which finds equivalent states of a finite automaton.

Example 1.1. Consider the automaton with $S = \{a,b,c,d,e\}, I = \{0,1\}, F = \{d,e\}$ and δ is given by the arc of diagram of Fig. 1.10.

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

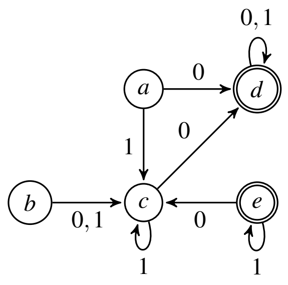


图 1.10: Finite state automaton

1.4.2 The Basic Algorithm

Definition 1.3 (acceptable partition). A partitioning of the states into blocks $B_1, B_2, \ldots, B_p$ is acceptable if (a) no block contains both a final and a nonfinal state, and (b) if s and t are equivalent states then they are in the same block.

Lemma 1.1. *The partitioning $B_1 = F, B_2 = S - F$ is acceptable.*

Lemma 1.2. *The partitioning $B_1, B_2, \ldots, B_p$ gives the blocks of equivalent states if and only if (a) the partitioning is acceptable and (b) for each pair of blocks B_i, B_j and symbol $a \in I$*

$$s,t \in B_i, \delta(s,a) \in B_j \textit{ implies } \delta(t,a) \in B_j \tag{1.1}$$

Lemma 1.3. *Let $B_1, B_2, \ldots, B_p$ be an acceptable partitioning. Suppose there are two blocks B_i, B_j and symbol $a \in I$ such that*

$$s,t \in B_i, \delta(s,a) \in B_j \textit{ but } \delta(t,a) \notin B_j \tag{1.2}$$

Then s and t are not equivalent states, and we get a new acceptable partitioning by replacing B_i by the two blocks

$$\{s \in B_i | \delta(s,a) \in B_j\} \textit{ and } \{s \in B_i | \delta(s,a) \notin B_j\} \tag{1.3}$$

This splitting of B_i as described is called splitting B_i with respect to the pair (B_i, a) or simply splitting B_i wrt (B_j, a). We now write the following algorithm:

Algorithm 1 splitting B_i wrt (B_j, a)

```
$B_1 \leftarrow F; B_2 \leftarrow S - F$; [initially there are two blocks]
while $\exists a, B_i, B_j$ such that Eq. (1.2) holds do
    SPLIT: split $B_i$ wrt $(B_j, a)$
end while
```

B,P are relations, S is a sequence of statements.

$$P \wedge B\{S\}P \text{ implies } P \ \{\text{While } B \text{ do } S \text{ end}\} \ P \wedge \neg B \tag{1.4}$$

For algorithm (1), let P be the relation "the partitioning is acceptable". By Lemma 1.1, P is true just before execution of the while loop, while by Lemma 1.3 execution of S always yields an acceptable partitioning. The relation B is "$\exists a, B_i, B_j such that Eq.(1.2) holds$". Thus, P and $\neg B$ hold after execution of the loop, but these are the sufficient requirements described in Lemma 1.2 for the final partitioning to be the described one.

Our refinement (2) determines all splittings wrt a pair (B_i, a) and then performs all these splittings at the same time.

Algorithm 2 splitting B_i wrt (B_j, a)

```
$B_1 \leftarrow F; B_2 \leftarrow S - F$; [initially there are two blocks]
while $\exists a, B_i, B_j$ such that Eq. (1.2) holds do
    Determine the splittings of all blocks wrt $(B_j, a)$
    Split each block as just determined.
end while
```

The algorithm is not very efficient, it's $|I||S|^2$.

The result of splitting all blocks wrt (B_j, a) is that any future block B (including the final blocks) satisfies one of the following:

$$\begin{aligned} &\text{(a) for all } s \in B \quad \delta(s,a) \in B_j, \text{ or} \\ &\text{(b) for all } s \in B \quad \delta(s,a) \notin B_j \end{aligned} \tag{1.5}$$

Lemma 1.4. *Suppose all blocks have been split wrt (B_j, a). Then there is no need to split any future block wrt (B_j, a).*

Lemma 1.5. *Suppose a block B_j is split into blocks $\bar{B}_j$ and $\tilde{B}_j$. Consider a symbol a. Splitting all blocks wrt to any two of the three pairs $(B_j, a), (\bar(B)_j, a)$, and $(\tilde{B}_j, a)$ performs the same function as splitting all blocks wrt all three pairs.*

证明. Suppose we split all blocks wrt (B_j, a) and $(\bar{B}_j, a)$. This implies that each future block B satisfies one of the following:

$$s \in B \text{ implies } \delta(s,a) \in B_j \text{ and } \delta(s,a) \in \bar{B}_j, \text{ or}$$
$$s \in B \text{ implies } \delta(s,a) \in B_j \text{ and } \delta(s,a) \notin \bar{B}_j, \text{ or}$$
$$s \in B \text{ implies } \delta(s,a) \notin B_j \text{ and } \delta(s,a) \notin \bar{B}_j, \text{ or}$$
$$s \in B \text{ implies } \delta(s,a) \notin B_j \text{ and } \delta(s,a) \in \bar{B}_j$$

Since $\bar{B}_j \cup \tilde{B}_j = B_j$ and $\bar{B}_j \cap \tilde{B}_j = \emptyset$, we infer that one of the following holds:

$$s \in B \text{ implies } \delta(s,a) \in \tilde{B}_j, \text{ or}$$
$$s \in B \text{ implies } \delta(s,a) \notin \tilde{B}_j$$

This is precisely what splitting all blocks wrt $(\tilde{B}_j,a)$ accomplishes (see (1.5)). We leave to the reader to prove the rest of the theorem (in the same fashion)–that splitting wrt $(\bar{B}_j,a)$ and $(\tilde{B}_j,a)$ accomplishes the task of splitting wrt (B_j,a); and that splitting wrt (B_j,a) and $(\tilde{B},a)$ accomplishes the task of splitting wrt $(\bar{B},a)$. □

Lemma 1.6. *Let the two initial blocks be $B_1 = F$ and $B_2 = S - F$. For a given symbol a, it is necessary to split all blocks wrt only one of the pairs (B_1,a) and (B_2,a).*

证明. Consider Lemma 1.5, with $B_j = S, \bar{B}_j = F, and \tilde{B}_j = S - F$. We already know that $(s,a) \in B_j$ for any symbol a, so it is not necessary to split wrt (B_j,a). Hence we need only split wrt either $(\bar{B}_j,a)$ or $(\tilde{B}_j,a)$, but not both. □

Let us consider the possibility of maintaining a list L of all pairs (B_j,a) wrt which some blocks may have to be split. Another way to put it is that if we know it is not necessary to split any B (including B_j itself) wrt a pair (B_j,a) we won't put that pair on the list. We can then keep splitting until the list L becomes empty.

see Algorithm 3, It remains to show that execution time is no worse than proportional to $|I||S|\log(|S|)$.

Meaning of List L. L is a list of pairs (B_j,a) wrt which we must attempt to split all blocks so that either 1.5(a) or (b) will hold for each block. If B_j is a block and $(B_j,a) \notin L$ for some a, then either 1.5(a) or (b) already holds, or we are assured by other means that either 1.5(a) or (b) will hold when the algorithm terminates.

Let us now look closer at splitting. Splitting a block B_i wrt (B_j,a) replaces B_i by two blocks $\bar{B}_i$ and $\tilde{B}_i$ which satisfy:

$$s \in \bar{B}_i \text{ implies } \delta(s,a) \in B_j$$
$$s \in \tilde{B}_i \text{ implies } \delta(s,a) \notin B_j$$

Given block B_i let us split it by removing from it those states s such that $\delta(s,a) \in B_j$, and putting these states in a new block B_k, called B_i's twin. Thus B_i is split into B_i and B_k.

In order to determine the splitting of all blocks wrt (B_j,a), we need to make a list D (say) of all states which must be removed from blocks–which satisfy the property $\delta(s,a) \in B_j$. Statement c thus looks like (algorithm 4):

Statement e, which actually splits blocks, could be written as (algorithm 5):

While correct, statement e is too inefficient since each time it is executed it must manipulate each block, and this would lead to a $|I||S|^2$ algorithm.

Hence we must refine e further to look only at blocks which have a chance of being partitioned – which contain states in D. We can also recognize a case where removing states is unnecessary. If for all $s \in B, \delta(s,a) \in B_j$ then $\{s \in B_i | \delta(s,a) \notin B_j\}$ is empty. We end up with the following algorithm e:

Algorithm 3 splitting B_i wrt (B_j, a)

```
$B_1 \leftarrow F; B_2 \leftarrow S - F; L = \emptyset$ [initially there are two blocks]
for all $c \in I$ do
    if $B_1$ is smaller than $B_2$ then
        $add(B_1, c)$ to $L$
    else
        $add(B_2, c)$ to $L$
    end if
end for
while $L \neq \emptyset$ do
    b: Pick one pair $(B_j, a) \in L$;
    c: Determine splittings of all blocks wrt $(B_j, a)$;
    d: $L \leftarrow L - (B_j, a)$;
    e: Split each block as determined in $c$;
    f: /* Fix $L$ according to the splits that occurred in step e */
    for all block $B$ just split into $\bar{B}$ and $\tilde{B}$(say) do
        for all $c \in I$ do
            if $(B, c) \in L$ then
                $L \leftarrow L + (\bar{B}, c) + (\tilde{B}, c) - (B, c)$
            else
                if $\bar{B}$ is smaller than $\tilde{B}$ then
                    $add(\bar{B}, c)$ to $L$
                else
                    $add(\tilde{B}, c)$ to $L$
                end if
            end if
        end for
    end for
end while
```

Algorithm 4 c: Determine the splittings of all blocks wrt (B_i, a)

```
$D \leftarrow \emptyset$
for each $s \in B_j$ do
    if $\delta^{-1}(s, a) \neq \emptyset$ then
        $D \leftarrow D \cup \delta^{-}(s, a)$
    end if
end for
```

Algorithm 5 e:Split each block as determined in statement c

```
for each block $B_i$ in the partition do
    $B_k \leftarrow B_i \cap D$; ($B_k$ is a newly generated block – $B_i$'s twin)
    $B_i \leftarrow B_i - B_k$;
end for
```

Algorithm 6 e: Split each block as just determined

```
$BI \leftarrow$ block number in which $s$ appears;
**if** all $s \in BI$ have $\delta(s,a) \in B_j$ **then**
    [no need to split block–do nothing]
**else**
    **if** $BI$ has no twin $BK$ yet **then**
        generate $BI$'s twin $BK$ and set $BK \leftarrow \emptyset$
    **else**
        Move $s$ from $BI$ to its twin $BK$
    **end if**
**end if**
```

Two important ideas helped us in reducing the running time to $mn\log(n)$. The first was that if a block B is split into $\bar{B}$ and $\tilde{B}$ we need only split wrt two of the three pairs (B,a), $(\bar{B},a)$, and $(\tilde{B},a)$. The second was that in case we need put only one of $(\bar{B},a)$ and $(\tilde{B},a)$ in L, we should put the one whose block ($\bar{B}$ or $\tilde{B}$) contains the fewest number of states.

1.5 From [Hopcroft71]

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

Example 1.2. $Q = \{1,2,3,4,5,6\}, V = \{0,1\}, T$ see Fig. 1.11

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

Algorithm 7 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.
```

$$B''(j) = B(j) - B'(j)$$

Step 7b.

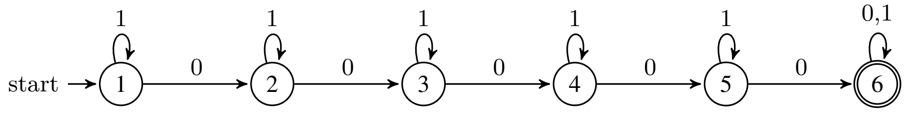


图 1.11: Minimizing example

Example 1.3. 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. (1.12).

{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}

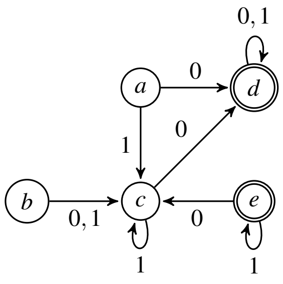


图 1.12: Finite state automaton

1.6 From [Ratnesh95]

FSM: Finte State Machine

NFSM: Non-deterministic Finite State Machine without $\varepsilon-$moves

DFSM: Deterministic Finite State Machine

Definition 1.4 (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 1.4 (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 "$\cdot$" denotes the operation of concatenation. □

Example 1.5 (Generated language). Consider the buffer of Example 1.4 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 1.6 (language model). Consider the buffer of Examples 1.4 and 1.5 with language model $[(ad)^*, pr((ad)^*)]$. The directed graph shown in Figure 1.13 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.) □

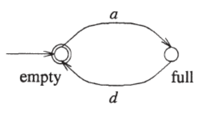


图 1.13: Graph representing a DSM

Example 1.7 (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 1.14. 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 1.14.

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。

□

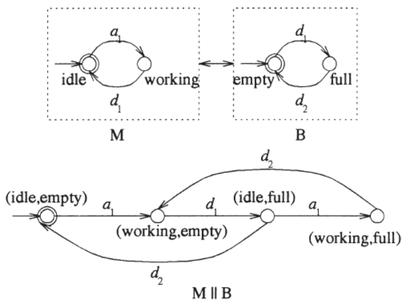


图 1.14: Diagram illustrating synchronous composition of DFSMs

Example 1.8 (ε-NSM). Consider the ε-NSM of Fig. 1.15 Then $\varepsilon_G^*(1) = \{1,2,3\}, \varepsilon_G^*(2) = \{2,3\}, \varepsilon_G^*(3) = \{3\}$.

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. □

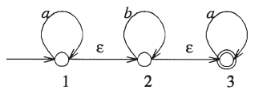


图 1.15: Diagram illustrating an $\varepsilon - NSM$

Example 1.9 (Completion and reverse). Completion and reverse of the DSM of Figure 1.17(b) are shown in Figure 1.16 (a) and 1.16 (b) respectively. The state labels have been changed.

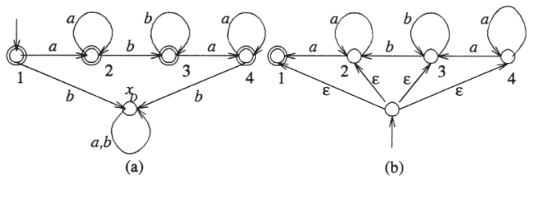


图 1.16: Diagram illustrating completion and reverse operations

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 1.10. (NFSM to DFSM) Consider the NFSM $G := (X, \Sigma, \alpha, x_0, X_m)$ shwown in Figure 1.17(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 1.17(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 $\mathscr{X}$;

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

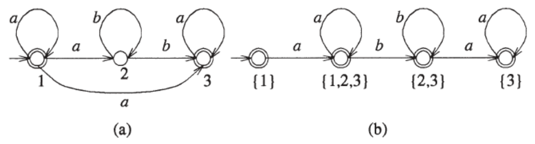


图 1.17: Diagram illustrating NFSM to DFSM conversion

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 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 1.18(a); corresponding DFSM obtained using the construction outlined in Theorem 1.1 is shown in Figure 1.18(b).

1.6.1 Myhill-Nerode Characterization

Definition 1.5 (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.

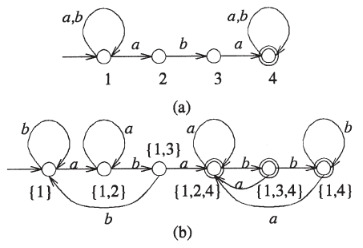


图 1.18: NFSM accepting strings with aba as a substring, and corresponding DFSM

Definition 1.6 (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 $K \subseteq \Sigma^*$ 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) ⇒ (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 1.11 (equivalence classes). Consider for example the marked language $K_m = (ad)^*$ of the buffer of capacity one of Example 1.5. 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}

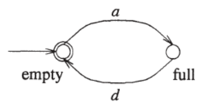
Hence Myhill-Nerode construction yields the DFSM $G := (\hat{X}, \Sigma, \hat{\alpha}, \hat{x}_0, \hat{X}_m)$, while
$\hat{X} = \{[\varepsilon](R_{K_m}), [a](R_{K_m}), [d](R_{K_m})\}$; $\hat{x}_0 = [\varepsilon](R_{K_m}); X_m = \{[\varepsilon](R_{K_m})\}$;
and

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

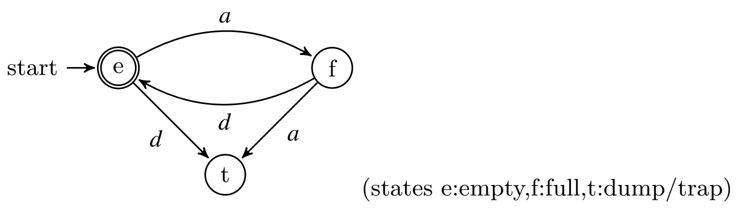
See figure 1.19, State e: empty,$[\varepsilon](R_{K_m})$; State f: full,$[a](R_{K_m})$; State t: dump/trap,$[d](R_{K_m})$.

□

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



(a) buffer model



(b) equivalence classes

图 1.19: equivalence classes

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.

1.6.2 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.

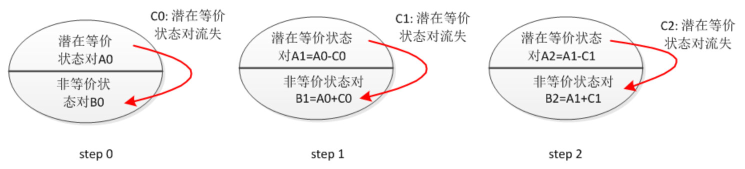


图 1.20: Diagram illustrating algorithm minimize

Algorithm 8 The algorithm for a minimal DFSM, see Figure 1.20

```
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:
    $C_k := \{(x, x') \in A_k | \exists \sigma \in \Sigma \text{ s.t. } (\alpha(x, \sigma), \alpha(x', \sigma)) \in B_k\}$
         $= \{A_0 \text{ 中两大类中的 (状态对) 存在字母 } \sigma \text{ 进入 } B_0 \text{ 类)}\}$
         $= \{\text{潜在状态对} \rightarrow \text{非等价状态对}\}$
    $A_{k+1} := A_k - C_k$
    $B_{k+1} := B_k \cup C_k = [\bar{X} \times \bar{X}] - A_{k+1}$
         $= \{\text{非 } A_{k+1} \text{ 状态对}\}$
3: Termination step:
    if $A_{k+1} = A_k$, then stop; else, $k := k + 1$, and goto step 2.
```

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

Example 1.12. Consider the complete DFSM of Figure 1.21. Then $X_m = \{1, 2, 3, 4\}$ and $X = \{1, 2, 3, 4, x_D\}$. Algorithm 8 can be applied to minimize the DFSM as follows:

$$\begin{aligned}
A_0 &= [X_m \times X_m] \cup \{(x_D \times x_D)\}; \\
&= \{(1,1),(2,2),(3,3),(4,4),(1,2),(2,1),(1,3),(3,1),(1,4),(4,1),(2,3),(3,2),(2,4),(4,2),(3,4),(4,3)\} \cup \{(x_D,x_D)\} \\
&= \{(1,1),(2,2),(3,3),(4,4)\} \cup \{(1,2),(1,3),(1,4),(2,3),(2,4),(3,4)\} \cup \{(2,1),(3,1),(4,1),(3,2),(4,2),(4,3)\} \cup \{x_D,x_D\} \\
&= \{\text{潜在的等价状态对}\} \\
B_0 &= [X \times X] - A_0 \\
&= \{(1,x_D),(x_D,1),(2,x_D),(x_D,2),(3,x_D),(x_D,3),(4,x_D),(x_D,4)\} \\
&= \{(1,x_D),(2,x_D),(3,x_D),(4,x_D)\} \cup \{(x_D,1),(x_D,2),(x_D,3),(x_D,4)\} \\
&= \{\text{非等价状态对}\} = \{\text{非 } A_0 \text{ 状态对}\} \\
C_0 &= \{(1,2),(2,1),(1,3),(3,1),(2,4),(4,2),(3,4),(4,3)\} \\
&= \{(1,2),(1,3),(2,4),(3,4)\} \cup \{(2,1),(3,1),(4,2),(4,3)\} \\
&= \{\text{潜在状态对 } (A_0) \to (B_0) \text{ 非等价状态对}\} \\
A_1 &= A_0 - C_0 \\
&= \{(1,1),(2,2),(3,3),(4,4)\} \cup \{(1,4),(2,3)\} \cup \{(4,1),(3,2)\} \cup \{x_D,x_D\} \\
B_1 &= [X \times X] - A_1 \\
&= \{\text{非 } A_1 \text{ 状态对}\} \\
C_1 &= \{\text{潜在状态对 } (A_1) \to (B_1) \text{ 非等价状态对}\} \\
&= \{(1,4),(4,1),(2,3),(3,2)\} \\
A_2 &= A_1 - C_1 \\
&= \{(1,1),(2,2),(3,3),(4,4),(x_D,x_D)\} \\
B_2 &= [X \times X] - A_2
\end{aligned}$$

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 1.21, which does not contain the dump state.

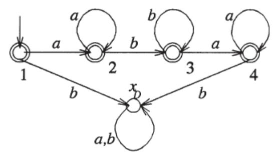


图 1.21: the minimal DFSM is not contain the dump state

1.7 From wiki – Partition of a set

From `https://en.wikipedia.org/wiki/Partition_of_a_set`.

Definition 1.7 (a partition of a set). A partition of a set X is a set of nonempty subsets of X such that every element x in X is in exactly one of these subsets (i.e., X is a disjoint union of the subsets). Equivalently, a family of sets P is a partition of X if and only if all of the following conditions hold:

- The family P does not contain the empty set (that is $\emptyset \notin P$).
- The union of the sets in P is equal to X (that is $\bigcup_{A \in P} A = X$) . The sets in P are said to **cover** X.
- The intersection of any two distinct sets in P is empty (that is $(\forall A, B \in P)\ A \neq B \implies A \cap B = \emptyset$). The elements of P are said to be pairwise disjoint.

The sets in P are called the *blocks, parts* or *cells* of the partition.

The rank of P is $|X||P|$, if X is finite.

Example 1.13 (a partition of a set). (From `https://en.wikipedia.org/wiki/Partition_of_a_set`)

- The empty set $\emptyset$ has exactly one partition,namely $\emptyset$.
- For any nonempty set $X, P = \{X\}$ is a partition of X, called the trivial partition.
  Particularly, every singleton set $\{x\}$ has exactly one partition, namely $\{\{x\}\}$.
- For any non-empty proper subset A of a set U, the set A together with its complement form a partition of U, namely, $\{A, U \setminus A\}$.
- The set $\{1,2,3\}$ has these five partitions (one partition per item):
  - $\{\{1\},\{2\},\{3\}\}$, sometimes written 1|2|3.
  - $\{\{1,2\},\{3\}\}$, or 12|3.
  - $\{\{1,3\},\{2\}\}$, or 13|2.
  - $\{\{1\},\{2,3\}\}$, or 1|23.
  - $\{\{1,2,3\}\}$, or 123 (in contexts where there will be no confusion with the number).
- The following are not partitions of $\{1,2,3\}$:
  - { {}, {1, 3}, {2} } is not a partition (of any set) because one of its elements is the empty set.
  - $\{\{1,2\},\{2,3\}\}$ is not a partition (of any set) because the element 2 is contained in more than one block.
  - $\{\{1\},\{2\}\}$ is not a partition of $\{1,2,3\}$ because none of its blocks contains 3; however, it is a partition of $\{1,2\}$.

Definition 1.8 (Partitions and equivalence relations). For any equivalence relation on a set X, the set of its equivalence classes is a partition of X. Conversely, from any partition P of X, we can define an equivalence relation on X by setting $x \equiv y$ precisely when x and y are in the same part in P. Thus the notions of equivalence relation and partition are essentially equivalent.

The axiom of choice guarantees for any partition of a set X the existence of a subset of X containing exactly one element from each part of the partition. This implies that given an equivalence relation on a set one can select a canonical representative element from every equivalence class.

Definition 1.9 (Refinement of partitions). A partition α of a set X is a refinement of a partition ρ of X —and we say that α is finer than ρ and that ρ is coarser than α—if every element of α is a subset of some element of ρ. Informally, this means that α is a further fragmentation of ρ. In that case, it is written that $\alpha \leq \rho$.

This finer-than relation on the set of partitions of X is a partial order (so the notation " $\leq$ " is appropriate). Each set of elements has a least upper bound and a greatest lower bound, so that it forms a lattice, and more specifically (for partitions of a finite set) it is a geometric lattice. The partition lattice of a 4-element set has 15 elements and is depicted in the Hasse diagram on the Fig 1.22.

Based on the cryptomorphism between geometric lattices and matroids, this lattice of partitions of a finite set corresponds to a matroid in which the base set of the matroid consists of the atoms of the lattice, namely, the partitions with $n-2$ singleton sets and one two-element set. These atomic partitions correspond one-for-one with the edges of a complete graph. The matroid closure of a set of atomic partitions is the finest common coarsening of them all; in graph-theoretic terms, it is the partition of the vertices of the complete graph into the connected components of the subgraph formed by the given set of edges. In this way, the lattice of partitions corresponds to the lattice of flats of the graphic matroid of the complete graph.

Another example illustrates the refining of partitions from the perspective of equivalence relations. If D is the set of cards in a standard 52-card deck, the same-color-as relation on D -which can be denoted C -has two equivalence classes: the sets red cards and black cards. The 2-part partition corresponding to C has a refinement that yields the same-suit-as relation S, which has the four equivalence classes spades, diamonds, hearts, and clubs.

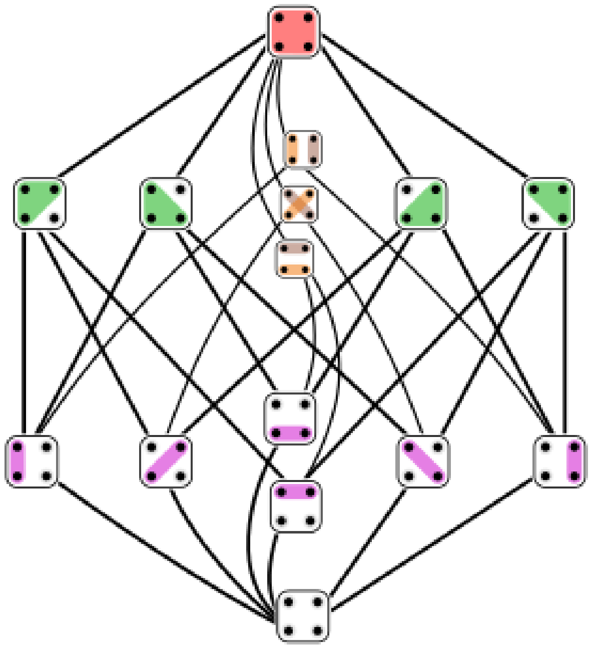


图 1.22: Partitions of a 4-set ordered by refinement

1.8 From [Kenneth2012]

Definition 1.10 (等价关系). 定义在集合 A 上的关系叫做等价关系，如果它是自反的、对称的和传递的。

Definition 1.11 (等价元素). 如果两个元素 a 和 b 由于等价关系而相关联，则称它们是等价的。记法 $a \sim b$ 通常用来表示对于某个特定的等价关系来说，a 和 b 是等价的元素。

Note 1.4. 在等价关系中，若两个元素有关系，就可以说它们是等价的。为了使等价元素的概念有意义，每个元素都应该等价于它自身，因为对于等价关系来说，自反性是一定成立的。在等价关系中，说 a 和 b 是相互关联也是正确的 (而不仅是 a 关联于 b)，因为如果 a 关联于 b, 右对称性, b 也关联于 a。

此外，因为等价关系是传递的，所以如果 a 和 b 等价且 b 和 c 等价，则可得出 a 和 c 也是等价的。

Example 1.14. 设 R 是定义在整数集上的关系，满足 aRb 当且仅当 $a=b$ 或 $a=-b$。可以证明 R 是自反的，对称的和传递的。因此 R 是等价关系。

Example 1.15 (模 m 同余). 设 m 是大于 1 的整数。证明以下关系是定义在整数集上的等价关系。

$$R=\{(a,b)|a\equiv b \pmod m\}$$

证明. $a\equiv b \pmod m$, 当且仅当 m 整除 $a-b$。注意 $a-a=0$ 能被 m 整除，因为 $0=0\cdot m$。因此 $a\equiv a \pmod m$, 从而模 m 同余关系是自反的。

假设 $a\equiv b \pmod m$, 那么 $a-b$ 能被 m 整除，即 $a-b=km$, 其中 k 是整数。从而 $b-a=(-k)m$, 即 $b\equiv a \pmod m$, 因此模 m 同余关系是对称的。

下面假设 $a\equiv b \pmod m$ 和 $b\equiv c \pmod m$, 那么 m 整除 $a-b$ 和 $b-c$。因此存在整数 k 和 l, 使得 $a-b=km$ 和 $b-c=lm$, $\implies a-c=(a-b)+(b-c)=km+lm=(k+l)m$。于是 $a\equiv c \pmod m$, 从而模 m 同余关系是传递的。

综上所述，模 m 同余关系是等价的。

Example 1.16. 设 R 是定义在英文字母组成的字符串的集合上的关系，满足 aRb 当且仅当 $l(a)=l(b)$, 其中 $l(x)$ 是字符串 x 的长度。证明 R 是等价关系。

证明. 因为 $l(a)=l(a)$, 所以只要 a 是一个字符串，就有 aRa, 故 R 是自反的。其次，假设 aRb, 即 $l(a)=l(b)$, 那么有 bRa, 因为 $l(b)=l(a)$, 所以 R 是对称的。最后，假设 aRb 且 bRc, 那么有 $l(a)=l(b), l(b)=l(c) \implies l(a)=l(c)$, 即 aRc, 从而 R 是传递的。由于 R 是自反的，对称的和传递的，所以 R 是等价关系。

Example 1.17. 设 n 是正整数，S 是字符串集合。假定 R_n 是 S 上的关系，sR_nt 当且仅当 $s=t$ 或者 s 和 t 都至少含有 n 个字符，且 s 和 t 的前 n 个字符相同。就是说，少于 n 个字符的字符串只于它自身的 R_n 相关；一个至少含有 n 个字符的字符串 s 与字符串 t 相关当且仅当 t 也含有至少 n 个字符且 t 以 s 最前面的 n 个字符开始。例如，设 $n=3$, S 是所有位串的集合，sR_3t 当 $s=t$ 或者 s 和 t 均为长度至少为 3 的位串，且前 3 位相同。例如，$01R_301$, $00111R_3001101$, 但是 $(01,010)\notin R_3, (0101,01110)\notin R_3$

证明：对所有的字符串集 S 和所有的正整数 n, R_n 是定义在 S 上的等价关系。

证明. 设 s 是 S 中的一个字符串，由于 $s=s$, 可得 sR_ns, 所以 R_n 关系是自反的。

如果 sR_nt, 那么或者 $s=t$ 或者 s 和 t 都至少含有 n 个字符，且以相同的 n 个字符开始。这意味着 tR_ns 成立。所以 R_n 是对称的。

现在假设 sR_nt 且 tR_nu。则有 $s=t$ 或者 s 和 t 都至少含有 n 个字符，且以相同的 n 个字符开始。还有 $t=u$ 或者 t 和 u 都至少含有 n 个字符，且以相同的 n 个字符开始。由此可以推出 $s=u$ 或者 s 和 u 都至少含有 n 个字符，且以相同的 n 个字符开始 (因为在这种情形下，我们知道 s,t 和 u 都至少含有 n 个字符，且 s 和 u 都与 t 一样以相同的 n 个字符开始)。所以 R_n 是传递的。

综上所述，模 R_n 是一个等价关系。

Definition 1.12 (等价类). 设 R 是定义在集合 A 上的等价关系。与 A 中的一个元素 a 有关系的所有元素的集合叫做 a 的等价类。A 的关于 R 的等价类记作 $[a]_R$。当只考虑一个关系时，我们将省去下标 R 并把这个等价类写作 $[a]$。

换句话说，如果 R 是定义在集合 A 上的等价关系，则元素 a 的等价类是

$$[a]_R = \{s|(a,s) \in R\}$$

如果 $b \in [a]_R, b$ 叫做这个等价类的**代表元**。一个等价类的任何元素都可以作为这个类的代表元。也就是说，选择特定元素作为一个类的代表元没有特殊要求。

Example 1.18. 设 R 是定义在整数集上的关系，满足 aRb 当且仅当 $a=b$ 或 $a=-b$。可以证明 R 是自反的，对称的和传递的。因此 R 是等价关系。

在这个等价关系中，一个整数对应于它自身和它的相反数数。从而一个整数的等价类是：$[a] = \{a, -a\}$。这个集合包含两个不同的整数，除非 $a=0$。例如, $[7] = \{7, -7\}, [-5] = \{-5, 5\}, [0] = \{0\}$。

Example 1.19. 对于模 4 同余关系，0 和 1 的等价类是什么？

0 的等价类包含使得 $a \equiv 0 \pmod 4$ 的所有整数 a。这个类中的整数是能被 4 整除的那些整数。因此，对于这个关系，0 的等价类是

$$[0] = \{\cdots, -8, -4, 0, 4, 8, \cdots\}$$

1 的等价类包含使得 $a \equiv 1 \pmod 4$ 的所有整数 a。这个类中的整数是当被 4 除时余数为 1 的那些整数。因此，对于这个关系，1 的等价类是

$$[1] = \{\cdots, -7, -3, 1, 5, 9, \cdots\}$$

用正整数 m 代替 4, 很容易推广到模 m 同余关系的等价类–模 m 的同余类。整数 a 模 m 的同余类记作 $[a]_m$, 满足

$$[a]_m = \{\cdots, a-2m, a-m, a, a+m, a+2m, \cdots\}$$

例如，

$$[0]_4 = \{\cdots, -8, -4, 0, 4, 8, \cdots\}$$
$$[1]_4 = \{\cdots, -7, -3, 1, 5, 9, \cdots\}$$

Example 1.20. 对于例1.17中所有位串集合上的等价关系 R_3, 串 0111 的等价类是什么？等价于 0111 的是以 011 开始，至少含有 3 位的位串：

$$[011]_{R_3} = \{011, 0110, 0111, 01100, 01101, 0111111, \cdots\}$$

Theorem 1.4. *设 R 是定义在集合 A 上的等价关系，下面的关于集合 A 中 a,b 两个元素的命题是等价的。*

(i) aRb
(ii) $[a] = [b]$
(iii) $[a] \cap [b] \neq \emptyset$

证明. 首先证明 (i) 推出 (ii)。

假设 aRb, 我们将通过 $[a] \subseteq [b]$ 和 $[b] \supseteq [a]$ 来证明 $[a] = [b]$。

假设 $c \in [a]$, 那么 aRc。因为 aRb 是对称的，所以 bRa。又由于 R 是传递的以及 bRa 和 aRc, 就得到 bRc, 所以 $c \in [b]$。这就证明了 $[a] \subseteq [b]$。类似地，可证明 $[a] \supseteq [b]$。

其次我们将证明 (ii) 推出 (iii)。假设 $[a] = [b]$, 这就证明了 $[a] \cap [b] \neq \emptyset$, 因为 $[a]$ 是非空的 (由 R 的自反性 $a \in [a]$)。

下面证明 (iii) 推出 (i)。假设 $[a] \cap [b] \neq \emptyset$, 那么存在元素 c 满足 $c \in [a]$ 且 $c \in [b]$。换句话说, aRc 且 bRc。由对称性, 有 cRb。再根据传递性，由 aRc 和 cRb，就有 aRb。

因为 $(i) \implies (ii), (ii) \implies (iii), (iii) \implies (i)$, 所以三个命题是等价的。

现在我们将说明一个等价关系怎样划分一个集合。设 R 是定义在集合 A 上的的等价关系，R 的所有等价类的并集就是集合 A, 因为 A 的每个元素 a 都在它自己的等价类，即 $[a]_R$ 中。换句话说，

$$\bigcup_{a\in A}[a]_R = A$$

此外，有定理1.4, 这些等价类或者是相等的或者是不相交的的，因此当 $[a]_R \neq [b]_R$ 时，

$$[a]_R \cap [b]_R = \emptyset$$

这两个结论证明了等价类构成 A 的划分, 因为它们将 A 分成不相交的子集。更确切的说，集合 S 的**划分**是 S 的不相交的非空的子集构成的集合，且它们的并集就是 S。换句话说, 一簇子集 $A_i, i\in I$, (其中 I 是下标的集合) 构成 S 的划分，当且仅当

$$\begin{aligned} A_i &\neq \emptyset && i\in I \\ A_i \cap A_j &= \emptyset && i\neq j \\ \bigcup_{i\in I} A_i &= S \end{aligned}$$

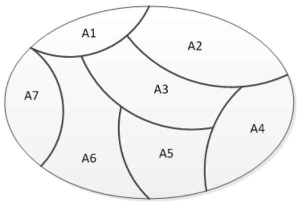


图 1.23: Partition of set

Example 1.21. 假设 $S=\{1,2,3,4,5,6\}$, 一簇集合 $A_1=\{1,2,3\}, A_2=\{4,5\}, A_3=\{6\}$ 构成 S 的一个划分，因为这些集合是不相交的，且它们的并集是 S。

Theorem 1.5. 设 R 是定义在集合 S 上的等价关系。那么 R 的等价类构成 S 的划分。反过来，给定集合 S 的划分 $\{A_i | i\in I\}$, 则存在一个等价关系 R，它以集合 $A_i (i\in I)$ 作为它的等价类。

Example 1.22. 假设 $S=\{1,2,3,4,5,6\}$, 一簇集合 $A_1=\{1,2,3\}, A_2=\{4,5\}, A_3=\{6\}$ 构成 S 的一个划分，因为这些集合是不相交的，且它们的并集是 S。列出这个划分所产生的等价关系 R 中的有序对。

Solution 1.1. 划分中的子集是 R 的等价类。有序对 $(a,b)\in R$, 当且仅当 a 和 b 在划分的同一个子集中。

由于 $A_1=\{1,2,3\}$ 是一个等价类，因此有序对 $(1,1),(1,2),(1,3),(2,1),(2,2),(2,3),(3,1),(3,2),(3,3)$ 属于 R。

由于 $A_2=\{4,5\}$ 是一个等价类, 因此有序对 $(4,4),(4,5),(5,4),(5,5)$ 也属于 R。

由于 $A_3=\{6\}$ 是一个等价类, 因此有序对 $(6,6)$ 也属于 R。

Example 1.23. 模 m 同余关系的等价类–模 m 的同余类。整数 a 模 m 的同余类记作 $[a]_m$, 满足

$$[a]_m = \{\cdots, a-2m, a-m, a, a+m, a+2m, \cdots\}$$

例如，模 4 同余存在 4 个等价类，对应于 $[0]_4,[1]_4,[2]_4,[3]_4$, 它们的集合是:

$$
\begin{aligned}
[0]_4 &= \{\cdots,-8,-4,0,4,8,\cdots\}\\
[1]_4 &= \{\cdots,-7,-3,1,5,9,\cdots\}\\
[2]_4 &= \{\cdots,-6,-2,2,6,10,\cdots\}\\
[3]_4 &= \{\cdots,-5,-1,3,7,11,\cdots\}
\end{aligned}
$$

这些同余类是不相交的，并且每个整数恰好在它们中的一个，换句话说，这些同余类构成了一个划分。

Example 1.24. 设 R_3 是一个等价关系，s,t 是位串，sR_3t，如果 $s=t$ 或者 s 和 t 均为长度至少为 3 的位串，且前 3 位相同。例如，$01R_301$, $00111R_3001101$, 但是 $(01,010)\notin R_3,(0101,01110)\notin R_3$

等价于 0111 的是以 011 开始，至少含有 3 位的位串，其对应的等价类是

$$[011]_{R_3}=\{011,0110,0111,01100,01101,0111111,\cdots\}$$

由 R_3 等价关系，在所有字符串集合上产生的一个划分是：

(1) 每个长度小于 3 的位串只和它自身等价。因此 $[\lambda]_{R_3}=\{\lambda\}$

$$
\begin{aligned}
&[0]_{R_3}=\{0\},[1_{R_3}=\{1\}],\\
&[00]_{R_3}=\{00\},[01]_{R_3}=\{01\},[10]_{R_3}=\{10\},[11]_{R_3}=\{11\},
\end{aligned}
$$

(2) 每个长度大于 3 的位串必须和以下 8 个位串之一等价：

$$
\begin{aligned}
[000]_{R_3}&=\{000,0000,0001,00000,00001,00010,00011,\cdots\}\\
[001]_{R_3}&=\{001,0010,0011,00100,00101,00110,00111,\cdots\}\\
[010]_{R_3}&=\{010,0100,0101,01000,01001,01010,01011,\cdots\}\\
[011]_{R_3}&=\{011,0110,0111,01100,01101,01110,01111,\cdots\}\\
[100]_{R_3}&=\{100,1000,1001,10000,10001,10010,10011,\cdots\}\\
[101]_{R_3}&=\{101,1010,1011,10100,10101,10110,10111,\cdots\}\\
[110]_{R_3}&=\{110,1100,1101,11000,11001,11010,11011,\cdots\}\\
[111]_{R_3}&=\{111,1110,1111,11100,11101,11110,11111,\cdots\}
\end{aligned}
$$

这 15 个等价类是不相交的，并且每个位串都恰好属于它们之一，这些等价类是所有位串构成的集合的一个划分。

1.9 From [Jean2011]

Definition 1.13 (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 1.14 (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 1.15 (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

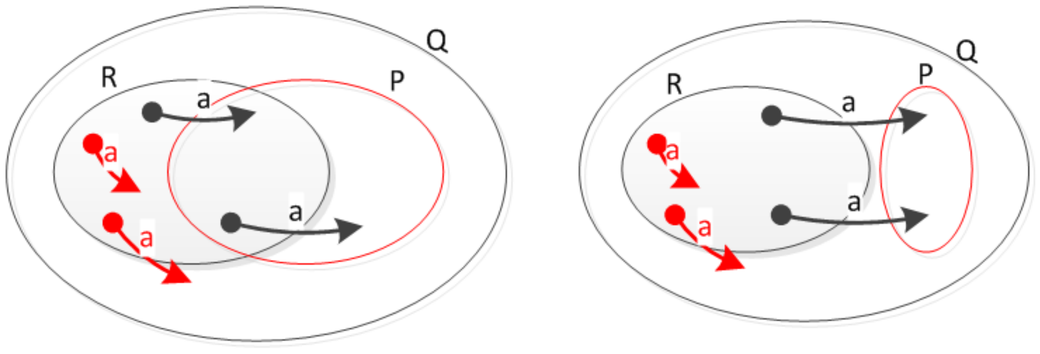
$$(P, a)|R$$

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

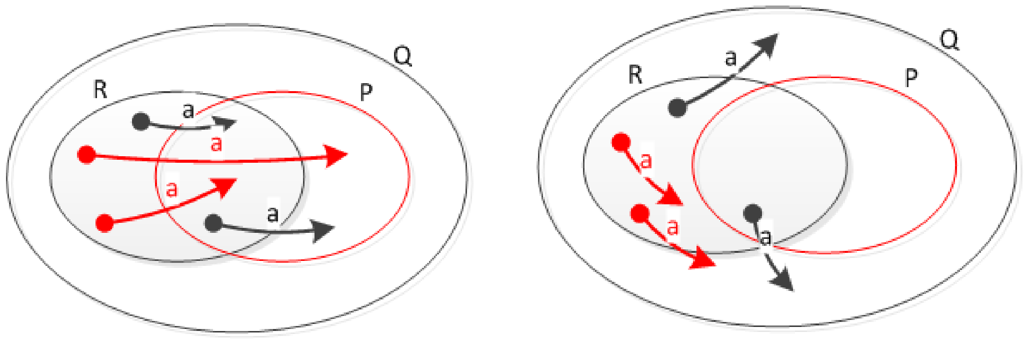
$$R \cap a^{-1}P = \{q \in R | q \cdot a \in P\} \text{ and } R \setminus a^{-1}P = \{q \in R | q \cdot a \notin P\}$$

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

$$(P,a)|R = (P^c,a)|R \tag{1.6}$$



(a) $(P,a)|R$



(b) $R \cap a^{-1}P = \{q \in R | q \cdot a \in P\}$ (c) $R \setminus a^{-1}P = \{q \in R | q \cdot a \notin P\}$

图 1.24: $(P,a)|R = (R \cap a^{-1}P) \cup (R \setminus a^{-1}P)$

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 1.7. *Let P be a set of states, and let $\mathscr{P} = \{P_1, P_2\}$ 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

$$(P,a)|R \geq (P_1,a)|R \wedge (P_2,a)|R, \tag{1.7}$$

$$(P_1,a)|R \geq (P,a)|R \wedge (P_2,a)|R \tag{1.8}$$

Example 1.25. We consider the automata given in Figure 1.25 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.

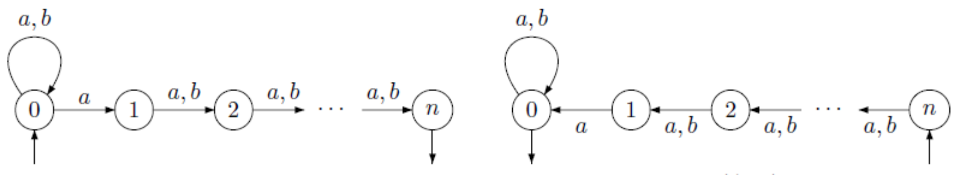


图 1.25: The automaton on the left recognizing the language A^*aA^n. It has $n+1$ states and the minimal deterministic automaton for this language has 2^n states. The automaton on the right is its reversal. It is minimal and recognizes A^naA^*.

1.9.1 Moor's algorithm

The minimization algorithm given by Moore computes the Nerode equivalence by a stepwise refinement of some initial equivalence. All automata are assumed to be deterministic.

Definition 1.16 (Moore equivalence of order h). Let $\mathscr{A} = (Q,i,F)$ be an automaton over an alphabet A. Define, for $q \in Q$ and $h \geq 0$, the set

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

The Moore equivalence of order h is the equivalence $\equiv_h$ on Q defined by

$$p \equiv_h q \Leftrightarrow L_p^{(h)}(\mathscr{A}) = L_q^{(h)}(\mathscr{A})$$

Algorithm 9 Moore's minimization algorithm

```
Input: $\mathscr{A} = (Q,i,F)$
Output: The equivalence classes of $Q$
  $\hat{P} \leftarrow \{F, F^c\}$                ▷ The initial partition
  repeat
      $\hat{P}' \leftarrow \hat{P}$              ▷ $\hat{P}'$ is the old partition, $\hat{P}$ is the new one
      for all $a \in A$ do
          $\hat{P}_a \leftarrow \bigwedge_{P \in \hat{P}} (P,a)|Q$
      end for
      $\hat{P} \leftarrow \hat{P} \wedge \bigwedge_{a \in A} \hat{P}_a$
  until $P = P'$
```

The computation is described in algorithm 9. It is realized by a loop that refines the current partition. The computation of the refinement of k partitions of a set swith n elements can be done in time $O(kn^2)$ by brute force. A radix sort improves the running time to $O(kn)$. With $k = Card(A)$, each tour in the loop is realized in time $O(kn)$, so the total time is $O(lkn)$, where l is the number of refinement steps in the computation of the Nerode equivalence $\equiv$, that is the depth of the automaton.

The worst case behavior is obtained for $l = n2$. We say that automata having maximal depth are slow and more precisely are slow for Moore automata.

Radix sort(see, `https://en.wikipedia.org/wiki/Radix_sort`) In computer science, radix sort is a non-comparative integer sorting algorithm that sorts data with integer keys by grouping keys by the individual digits which share the same significant position and value.

1.9.2 Hopcroft's algorithm

Hopcroft has given an algorithm that computes the minimal automaton of a given deterministic automaton. The running time of the algorithm is $O(kn\log n)$ where k the cardinality of the alphabet and n is the number of states of the given automaton.

1.9.2.1 Outline

The algorithm is outlined in the function HOPCROFT given in algorithm 10. We denote by $min(P,P')$ the set of smaller size of the two sets P and P', and any one of them if they have the same size.

Given a deterministic automaton $\mathscr{A}$, Hopcroft's algorithm computes the coarsest congruence which saturates the set F of final states. It starts from the partition F, F^c which obviously saturates F and refines it until it gets a congruence. These refinements of the partition are always obtained by splitting some class into two classes.

The algorithm proceeds as follows. It maintains a current partition $\hat{P} = P_1, \ldots, P_n$ and a current set W of splitters, that is of pairs (W,a) that remain to be processed, where W is a class of $\hat{P}$ and a is a letter. The set W is called the waiting set. The algorithm stops when the *waiting* set W becomes empty. When it stops, the

Algorithm 10 Hopcroft's minimization algorithm

```
Input: $\mathscr{A} = (Q, i, F)$
Output: The equivalence classes of $Q$
 1: $\hat{P} \leftarrow \{F, F^c\}$                ▷ The initial partition
 2: $W \leftarrow \emptyset$               ▷ The waiting set
 3: for all $a \in A$ do
 4:     $ADD((min(F, F^c), a), W)$             ▷ initialization of the waiting set
 5: end for
 6: while $W \neq \emptyset$ do
 7:     $(W, a) \leftarrow TakeSome(W)$             ▷ Take and remove some splitter
 8:     for all $P \in \hat{P}$ do which is split by $(W, a)$
 9:         $P', P'' \leftarrow (W, a)|P$             ▷ Compute the split
10:         Replace $P$ by $P'$ and $P''$ in $\hat{P}$             ▷ Refine the partition
11:     end for
12:     for all $b \in A$ do             ▷ Update the waiting set
13:         if $(P, b) \in W$ then
14:             Replace $(P, b)$ by $(P', b)$ and $(P'')$ in $W$
15:         else
16:             $ADD((min(P', P''), b), W)$
17:         end if
18:     end for
19: end while
```

partition $\hat{P}$ is the coarsest congruence that saturates F. The starting partition is the partition F, F^c and the starting set W contains all pairs $(min(F, F^c), a)$ for $a \in A$.

The main loop of the algorithm removes one splitter (W, a) from the waiting set W and performs the following actions. Each class P of the current partition (including the class W) is checked as to whether it is split by the pair (W, a). If (W, a) does not split P, then nothing is done. On the other hand, if (W, a) splits P into say P' and P'', the class P is replaced in the partition $\hat{P}$ by P' and P''. Next, for each letter b, if the pair (P, b) is in W, it is replaced in W by the two pairs (P', b) and (P'', b), otherwise only the pair $(min(P', P''), b)$ is added to W.

It should be noted that the algorithm is not really deterministic because it has not been specified which pair (W, a) is taken from W to be processed at each iteration of the main loop. This means that for a given automaton, there are many executions of the algorithm. It turns out that all of them produce the right partition of the states. However, different executions may give rise to different sequences of splitting and also to different running time. Hopcroft has proved that the running time of any execution is bounded by $O(|A| n \log n)$.

1.10 From [Knuutila2001]

1.10.1 Sets,relations and mappings

The cardinality of a set A is denoted by $|A|$.

Let A and B be sets and $\rho \subseteq A \times B$ a (binary) relation from A to B. The fact that $(a,b) \in \rho (a \in A, b \in B)$ is also expressed by writing $a\rho b$. For any $a \in A$, we denote by $a\rho$ the set of elements of B that are in relation ρ with a, i.e. $a\rho = \{b \in B | a\rho b\}$. The converse of is the relation $\rho^{-1} = \{(b,a) | a\rho b\}$.Obviously $b\rho^{-1} = \{a | a\rho b\}$.

Next we consider relations on a set A, i.e. subsets of $A \times A$. Theses include the diagonal relation $\omega_A = \{(a,a) | a \in A\}$, and the universal relation $\iota_A = A \times A$. The powers $\rho^n (n \geq 0)$ of a relation ρ are defined as follows:

$$\rho^0 = \omega_A$$
$$\rho^{n+1} = \rho^n \circ \rho, \qquad n \geq 0.$$

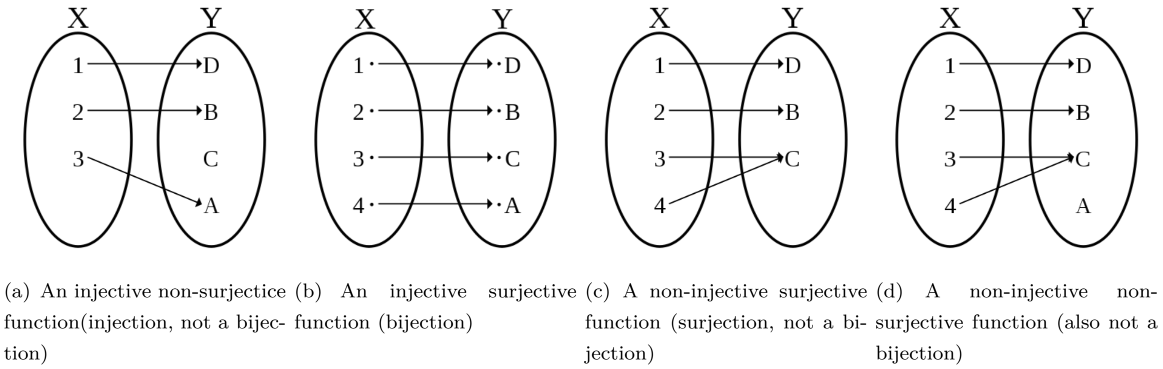
The relation ρ is called reflexive if $\omega_A \subseteq \rho$; symmetric if $\rho^{-1} \subseteq \rho$; and transitive if $\rho^2 \subseteq \rho$.

A relation on A is called an *equivalence relation* on A, if it is reflexive, symmetric and transitive. The set of all equivalence relations on a set A is denoted by $Eq(A)$. It is obvious that both $\omega_A \in Eq(A)$ and $\iota_A \in Eq(A)$. Let $\rho \in Eq(A)$. The ρ-class $a\rho$ of an element $a (\in A)$ is also denoted by a/ρ. The *quotient set* or the *partition of* A with respect to ρ, is $A/\rho = \{a\rho | a \in A\}$. If $\pi \in Eq(A)$ and $\pi \subset \rho$, then the partition A/π is a *refinement* of A/ρ (each ρ-class is a union of some π-classes); this is also expressed by saying that π is *finer* than ρ or that ρ is *coarser* than π. We often define an equivalence relation ρ on A via the set A/ρ, i.e. in the form $A/\rho = \{C_1, \dots, C_m\}$, where the sets C_i are the classes of ρ.

The cardinality of A/ρ is called the *index* of ρ; especially, if A/ρ is finite, then is said to have a *finite index.* For any subset H of A, H/ρ denotes $\{a\rho | a \in H\}$. The equivalence ρ saturates the subset H, if H is the union of some ρ-classes. Hence, ρ saturates H iff $a\rho \in H/\rho$ implies $a\rho \subseteq H$.

A *mapping* or a *function* $\phi : A \to B$ is a relation $\phi \subseteq A \times B$ such that $|a\phi| = 1$ for all $a \in A$. If $a\phi b$, then b is the *image* of a and a is a *preimage* of b. That b is an image of a is expressed by writing $b = \phi(a)$ and that a is a preimage of b is written as $a = \phi^{-1}(b)$. The notation $a = \phi^{-1}(b)$ is justified 合理的, when ϕ is bijective (see below). The *restriction* of a mapping $\phi : A \to B$ to a set $C \subseteq A$ is the mapping $\phi | C : C \to B$ where $\phi | C = \phi \cap (C \times B)$.

The *composition* of two mappings $\phi : A \to B$ and $\psi : B \to C$ is the mapping $\phi\psi : A \to C$, where $\phi\psi$ is the product of ϕ and ψ as relations. The *kernel* $\phi\phi^{-1}$ of a mapping $\phi : A \to B$, also denoted by $\ker\phi$, is an equivalence relation on A and $a\phi\phi^{-1}b$ iff $a\phi = b\phi (a, b \in A)$. A mapping $\phi : A \to B$ is called *injective* if $\ker\phi = \omega_A$; *surjective* (or onto) if $A\phi = B$; and *bijective* if it is both injective and surjective.



(a) An injective non-surjectice function(injection, not a bijection)

(b) An injective surjective function (bijection)

(c) A non-injective surjective function (surjection, not a bijection)

(d) A non-injective non-surjective function (also not a bijection)

图 1.26: The term one-to-one function must not be confused with one-to-one correspondence (a.k.a. bijective function), which uniquely maps all elements in both domain and codomain to each other

1.10.2 Preliminaries

Definition 1.17 ($a\rho_G b$). DFA $G=(Q,\Sigma,\delta,q_0,F)$, Two states a and b of G are equivalent, which we express by writing $a\rho_G b$, if

$$(\forall w \in \Sigma^*)(\delta(a,w) \in F) \Leftrightarrow \delta(b,w) \in F)$$

Definition 1.18 ($\omega_Q, \iota_Q, Eq(Q)$). DFA $G=(Q,\Sigma,\delta,q_0,F)$, the diagonal relation $\omega_Q = \{(a,a)|a \in Q\}$, the universal relation $\iota_Q = Q \times Q$. The set of all equivalence relations on a set Q is denoted by $Eq(Q)$. It is obvious that both $\omega_Q \in Eq(Q)$ and $\iota_Q \in Eq(Q)$.

Definition 1.19 (the reduced G). DFA G is reduced, if $a\rho_G b$ implies $a=b$, i.e. $\rho_G = \omega_Q$

Definition 1.20 ($Con(G)$). A relation $\rho \in Eq(Q)$ is a congruence of G, if

(1) $a\rho b$ implies $\delta(a,x)\rho\delta(b,x)$ for all $a,b \in Q$ and all $x \in \Sigma$, and
(2) ρ saturates F

We denote by $Con(G)$ the set of all congruences of G.

Definition 1.21 (quotient DFA G/ρ). It is well-known that the relation ρ_G is the greatest (coarsest) congruence of G. For any $\rho \in Con(G)$ the quotient DFA G/ρ is defined as $G/\rho = (Q/\rho, \Sigma, \delta/\rho, q_0/\rho, F/\rho)$ where $\delta/\rho(a/\rho, x) = \delta(a,x)/\rho$.

Definition 1.22 (DFA as unary algebras). DFA $G=(Q,\Sigma,\delta,q_0,F)$, Σ is viewed as a set of unary operation symbols and the transition function δ of G is replaced by the Σ-indexed family $(x^G : x \in \Sigma)$ of unary operations which are defined so that for any $x \in \Sigma$ and $a \in Q, x^G(a) = \delta(a,x)$. We shall omit G from the superscript and write simply $x(a)$ for $\delta(a,x)$.

Clearly, a string $w = x_1x_2\ldots x_n$ is accepted by G if and only if $x_n(\ldots(x_2(x_1(q_0))\ldots) \in F$.

1.10.3 The classical algorithm

The classical minimization algorithm is based on the following "layer-wise' definition for the equivalence relation ρ_G.

Proposition 1.2. *Let $G=(Q,\Sigma,\delta,q_0,F)$ and a series $\rho_i (i \geq 0)$ of equivalence relations on Q be defined as follows:*

$$\rho_0 = \{(a,b)|a,b \in F\} \cup \{(a,b)|a,b \in Q-F\},$$
$$\rho_{i+1} = \{(a,b) \in \rho_i | (\forall x \in \Sigma)(\delta(a,x),\delta(b,x)) \in \rho_i\}$$

Then the following hold.

(1) $\rho_0 \supseteq \rho_1 \supseteq \cdots$
(2) If $\rho_i = \rho_{i+1}$ then $\rho_i = \rho_{i+j}$ for all $j > 0$ and furthermore, $\rho_i = \rho_G$.
(3) There exists $0 \leq k \leq |Q|$ such that $\rho_k = \rho_{k+1}$.

The refinements on individual classes

When the construction of Proposition 1.2 is implemented, each refinement step leading from ρ_i to ρ_{i+1} consists of a series of refinements on individual classes of ρ_i. The refinement of a single class B can be implemented with suitably chosen data structures for equivalence relations to run in time $O(|\Sigma||B|)$, and each refinement from ρ_i to ρ_{i+1} takes thus time $O(|\Sigma||Q|)$.

Example 1.26. We construct (the worst kind of) a DFA Fig. 1.27.

$$\begin{aligned}
G/\rho_0 &= \{\{6\},\{1,2,3,4,5\}\},\\
&\quad B=\{1,2,3,4,5\},\delta(5,0)\notin B,\delta(i,0)=i+1\in B,1\le i\le 4,\\
G/\rho_1 &= \{\{6\},\{5\},\{1,2,3,4\}\}\\
&\quad B=\{1,2,3,4\},\delta(4,0)\notin B,\delta(i,0)=i+1\in B,1\le i\le 3,\\
G/\rho_2 &= \{\{6\},\{5\},\{4\},\{1,2,3\}\}\\
&\quad B=\{1,2,3\},\delta(3,0)\notin B,\delta(i,0)=i+1\in B,1\le i\le 2,\\
G/\rho_3 &= \{\{6\},\{5\},\{4\},\{3\},\{1,2\}\}\\
&\quad B=\{1,2\},\delta(2,0)\notin B,\delta(i,0)=i+1\in B,1\le i\le 1,\\
G/\rho_4 &= \{\{6\},\{5\},\{4\},\{3\},\{2\},\{1\}\}
\end{aligned}$$

The classical minimization algorithm is ineffective, since in the worst kind of a DFA, each refinement step from ρ_i to ρ_{i+1} removes always one state from the only nonsingleton class. The work performed by the algorithm (even if we optimize it to avoid considering singleton classes) will then be proportional to $|\Sigma|\sum_{i=1}^{|Q|-2}(|Q|-i)$, which leads to $O(|\Sigma||Q|^2)$ execution time.

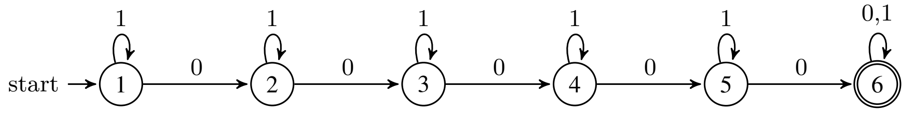


图 1.27: Minimizing example

1.10.4 Atomic refinements

Note that the construction of Proposition 1.2 reaches the situation $\rho_i=\rho_{i+1}$ when ρ_i becomes a congruence, i.e.

$$(\forall a,b\in Q,x\in Q)a\rho_i b \implies \delta(a,x)\rho_i\delta(b,x)$$

.

Now consider the situation where $\rho_i\neq\rho_{i+1}$. Obviously,

$$\begin{aligned}
\rho_i \neq \rho_{i+1} &\Leftrightarrow (\exists a,b \in Q, x \in \Sigma) & (a,b) \in \rho_i \text{ and } (\delta(a,x),\delta(b,x)) \notin \rho_i \\
&\Leftrightarrow (\exists B \in Q/\rho_i, x \in \Sigma) & (a,b) \in B \text{ and } (\delta(a,x),\delta(b,x)) \notin \rho_i \\
&\Leftrightarrow (\exists B,C \in Q/\rho_i, x \in \Sigma) & (a,b) \in B \text{ and } \delta(a,x) \in C, \text{ and } \delta(b,x)) \notin C \\
&\Leftrightarrow (\exists B,C \in Q/\rho_i, x \in \Sigma) & (a,b) \in B \text{ and } \delta(B,x) \cap C \neq \emptyset, \text{ and } \delta(B,x)) \nsubseteq C
\end{aligned}$$

The step from ρ_i to ρ_{i+1} can thus be understood as a series of "atomic' refinements performed for such $x \in \Sigma, B, C \in Q/\rho_i$ that

$$\delta(B,x) \cap C \neq \emptyset \text{ and } \delta(B,x)) \nsubseteq C \tag{1.9}$$

holds, Each such atomic refinement partition then B into $B \cap x^{-1}(C)\}$ and $B - (B \cap x^{-1}C))$. We denote these refinements of B with $B_{C,x}$ and $B^{C,x}$, respectively. We use the notation B' and B'' for these refinements of B when their relation to the pair (C,x) has no importance.

Let us next rewrite Proposition 1.2 in the terms of these atomic refinements.

Proposition 1.3. *Let $G = (Q, \Sigma, \delta, q_0, F)$ be a DFA and series $\theta_i (i \geq 0)$ of equivalence relation on Q be defined as follows:*

$$Q/\theta_0 = \{F, Q - F\},$$

$$Q/\theta_{i+1} = \begin{cases} (Q/\theta_i - \{B\}) \cup \{B_{C,x}, B^{C,x}\} & \textit{if 1.9 holds for some } B,C \in Q/\theta_i, x \in \Sigma, \\ Q/\theta_i & \textit{otherwise.} \end{cases}$$

Then there exists a $k \leq |Q|$ such that $\theta_{i+l} = \theta_i$ for all $l \geq 0$ and $\theta_k = \rho_G$.

证明. The upper bound comes from the observation, that each atomic refinement increases the index of θ_i by one. Naturally this index cannot be increased more than $|Q| - 1$ times.

It is clear that θ_k is both an equivalence relation saturating F (it is a refinement of θ_0) and a congruence of G (since Eq. (1.9) does not hold for θ_k). What remains, is to show that θ_k is also the *greatest* congruence ρ_G of G.

We show first that $\theta_i \supseteq \rho_G$ for all $i \geq 0$. When contraposed, the claim is that if $(a,b) \notin \theta_i$ then $(a,b) \notin \rho_G$ (for all $i \geq 0$). This is clearly true for θ_0, since final and non-final states are not in ρ_G. Suppose then that the claim holds for all $0 \leq l \leq i$ and let $(a,b) \in \theta_i$. If $(a,b) \notin \theta_{i+l}$, it must be the case that for some $x \in \Sigma, (x(a), x(b)) \notin \theta_i$. But this implies (by IA) that $x(a)$ and $x(b)$ are not in ρ_G. Thus, a and b become inequivalent in θ_{i+l} only when they are shown to be inequivalent in ρ_G, formally $(a,b) \notin \theta_{i+l} \implies (a,b) \notin \rho_G$.

It now holds that $\theta_0 \supseteq \theta_1 \supseteq \cdots \supseteq \theta_k$ (by definition), and consequently $\theta_0 \supseteq \theta_1 \supseteq \cdots \supseteq \theta_k \supseteq \rho_G$. Since ρ_G is the greatest congruence of G, and θ_k is a congruence, it must be the case that $\theta_k = \rho_G$. □.

Note that the construction given in Proposition 1.3 does not fix the order in which the triples B, C, x are exploited to refine θ_i. Thus, all the different orderings yield the same result at the end: the unique greatest congruence.

How to efficiently find some triple B, C, x for which Eq. (1.9) holds.

The refiners of B in θ, shortly $ref(B, \theta)$:

$$ref(B,\theta) = \{(C,x) \in (Q/\theta) \times \Sigma | x(B) \cap C \neq \emptyset \text{ and } x(B) \nsubseteq C\}$$

As each B is related to $ref(B,\theta)$, so is each pair (C,x) related to its *objects of refinement* in θ,

Algorithm 11 Comuting ρ_G using atomic refinements

```
1: function EQUIVALENCE($G$)
2:     $Q/\theta \leftarrow \{F, Q-F\}$
3:     while ($\exists B, C \in Q/\theta, x \in \Sigma$ s.t. Eq. 1.9 holds) do
4:         $Q/\theta \leftarrow (Q/\theta - \{B\} \cup \{B_{C,x}, B^{C,x}\}$
5:     end while
6:     return $\theta$
7: end function
```

Algorithm 12 Refiner-driven implementation

```
1: function EQUIVALENCE($G$)
2:     $Q/\theta \leftarrow \{F, Q-F\}$
3:     while ($\exists some(C,x) \in (Q/\theta) \times \Sigma$ with $obj(C,x,\theta) \neq \emptyset$) do
4:         for $B \in obj(C,x,\theta)$ do
5:             replace $B$ with $B_{C,x}$ and $B^{C,x} \in Q/\theta$
6:         end for
7:     end while
8:     return $\theta$
9: end function
```

$$obj(C,x,\theta) = \{B \in Q/\theta | (C,x) \in ref(B,\theta)\}$$

Lemma 1.8. *Let $G = (Q, \Sigma, \delta, q_0, F), \theta \in Eq(Q)$ and $B, C \in Q/\theta$ and $x \in \Sigma$. Suppose we refine $B \in Q/\theta$ into $B_{C,x}$ and $B^{C,x}$ with respect to (C,x). Let D be a subset of $B_{C,x}$ or $B^{C,x}$, then $D \notin (C,x,\theta)$.*

证明. Let us first condider the case $D \in \{B_{C,x}, B^{C,x}\}$. Then either $x(a) \in C$ for all $a \in D$ or $x(a) \notin C$ for all $a \in D$. The same property holds naturally for all subsets of D.

The bookkeeping needed for telling whether a particular pair (C,x) has been used can be implemented as follows:

- We maintain a set L of candidate refiners, shortly candidates, in $(Q/\theta) \times \Sigma$. Initially $L = \{F, Q-F\} \times X$.
- Pairs (C,x) are now selected from L, not (blindly) from all of $(Q/\theta) \times \Sigma$.
- Every time we select a pair (C,x) from L, we remove it from L. This is justified by Lemma 1.8.
- After refining θ to θ', we must also update L to contain only classes of θ'. We do the following for each $x \in \Sigma$ and each class B refined to B' and B'':
    - if $(B,x) \in L$, we remove (B,x) and add both $(B',x) and (B'',x)$ to L. This is (indirectly) justified by Proposition 1.3, since only the current θ-classes are used in the construction.
    - If $(B,x) \notin L$, we simply add (B') and (B'') to L.

Note that new items are inserted into L only when some class gets refined. Thus, L will eventually become empty, because each iteration removes one element from L, and the total number of elements added into L is bounded by $2|\Sigma||Q|$ (the maximum number of different equivalence classes created in any refinement sequence is $2|Q|-1$. The ideas above are collected into Algorithm 13.

Algorithm 13 Set-driven implementation

```
1: function EQUIVALENCE($G$)
2:     $Q/\theta \leftarrow \{F, Q-F\}$
3:     $L \leftarrow (Q/\theta) \times \Sigma$
4:     while $L \neq \emptyset$ do
5:         remove a pair $(C,x)$ from $L$
6:         for all $B \in obj(C,x,\theta)$ do
7:             replace $B$ with $B_{C,x}$ and $B^{C,x}$ in $Q/\theta$
8:             for all $y \in \Sigma$ do
9:                 if $(B,y) \in L$ then
10:                    replace $(B,y)$ with $(B',y)$ and $(B'',y)$ in $L$
11:                else
12:                    insert $(B',y)$ and $(B'',y)$ to $L$
13:                end if
14:            end for
15:        end for
16:    end while
17:    return $\theta$
18: end function
```

Lemma 1.9. *Let $G = (Q, \Sigma, \delta, q_0, F), \theta \in Eq(Q)$ and $B \in Q/\theta$. Suppose we refine B into B' and B''. Then, for any $y \in \Sigma$, refining all the classes of θ with respect to any two of the pairs $(B,y),(B',y)$ and (B'') gives the same result as refining them with respect to all three of them.*

证明. Consider an arbitrary class $D \in Q/\theta, a \in D$ and $y \in \Sigma$. As seen in Fig. 1.28, the transition $\delta(a,y)$ satisfies exactly one of the following:

(1) $\delta(a,y) \in B'$

(2) $\delta(a,y) \in B''$

(3) $\delta(a,y) \notin B$

Now each possible refinement sequence with respect to any two of the sets B, B' and B'' and letter y partitions D into D_1, D_2, D_3, which is the same as the result yielded by performing all the three refinements.

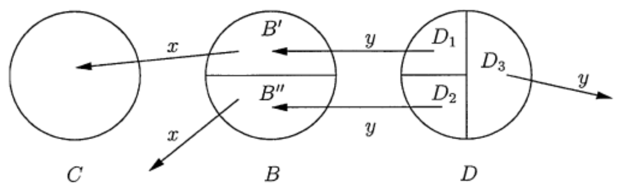


图 1.28: Illustration of Lemma 1.9

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 Algorithm by Hopcroft*, Acta Inf. 2:97 109, 173. © by Springer-Verlag 1973

Knuutila2001. Knuutila, T. *Re-describing an Algorithm by Hopcroft.* Theoret. Computer Science 250 (2001) 333–363.

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.

Kenneth2012. Kenneth H. Rosen 著, 徐六通译, 离散数学及其应用 *Discrete Mathematics and Its Applications*,seventh Edition, 2012, 机械工业出版社, 北京, 2014.