

目录

1	automata abstract	3
1.1	Finite automata	3
1.2	Properties of finite automata	6
1.3	Σ -algebras and regular expressions	10
1.4	Constructing ε -lookahead automata	13
1.5	Towards the Berry-Sethi construction	13
1.6	The Berry-Sethi construction	16
1.7	The McNaughton-Yamada-Clushkov construction	17
1.8	The dual of the Berry-Sethi construction	17
1.9	Algorithm 4.52 (Aho-Sethi-Ullman)	17
1.10	Others	21
1.11	Linear equation	25
	References	28
2	Finite automata minimization algorithms	31
2.1	Introduction	31
2.2	Brzozowski's algorithm	31
2.3	Minimization by equivalence of states	31
2.4	From [Gries73]	35
2.4.1	Problem Definition	35
2.4.2	The Basic Algorithm	35
2.5	From [Hopcroft71]	39
2.6	From [Ratnesh95]	42
2.6.1	Myhill-Nerode Characterization	44
2.6.2	Minimization	47
2.7	From wiki – Partition of a set	49
2.8	From [Kenneth2012]	51
2.9	From [Jean2011]	55
2.9.1	Moor's algorithm	58
2.9.2	Hopcroft's algorithm	59
2.10	From [Knuutila2001]	60

2.10.1	Sets,relations and mappings	60
2.10.2	Preliminaries	62
2.10.3	The classical algorithm	62
2.10.4	Atomic refinements	63
	References.....	67
3	Hopcroft's algorithm	69
3.0.1	algorithm	69
3.0.2	Minimization example 1.....	69
3.1	Minimization example 2	81
3.2	Minimization example 3 ($1 \rightarrow 2 \rightarrow 3 \rightarrow 4 \rightarrow 5 \rightarrow 6$).....	84
3.3	Minimization example 4	100
	References.....	105
4	[蒋宗礼 2013](第 1 章绪论)	107
4.1	集合的基础知识	107
4.1.1	集合之间的关系.....	107
4.2	关系	109
4.2.1	二元关系.....	109
4.2.2	等价关系与等价类.....	110
4.2.3	关系的合成 (composition)	111
4.2.4	递归定义 (recursive definition) 与归纳证明.....	112
4.2.5	关系的闭包.....	114
4.3	语言	115
4.3.1	字母表 (alphabet)	115
4.3.2	句子 (sentence)/字 (word)/字符串 (string)	116
4.3.3	并置/连结 (concatenation)	117
4.3.4	前缀与后缀.....	117
4.3.5	子串 (substring)	118
4.3.6	语言 (language).....	118
4.3.7	语言的乘积 (<i>product</i>)	119
4.4	Exercise and Solution	120
5	[蒋宗礼 2013](第 3 章有穷状态自动机)	127
5.1	语言的识别	127
5.2	有穷状态自动机 (finite automaton,FA).....	129
5.3	<i>NFA</i>	136
5.4	Exercise and Solution	138
6	[蒋宗礼 2013](第 4 章正则表达式)	149
6.1	RE 的形式定义	149
6.2	正则表达式 RE 与 FA 等价	151
6.3	正则语言 RL 可以用正则表达式 RE 表示	154

6.4	正则语言等价模型的总结	159
6.5	Exercise and Solution	160
6.6	正则代换 (regular substitution)	162
7	Supervisor	165

Chapter 1

automata abstract

1.1 Finite automata

Definition 1.1 (Finite automaton). A finite automaton (an FA) is a 6-tuple (Q, V, T, E, S, F) where

- Q is a finite set of states,
- V is an alphabet,
- $T \in \mathbb{P}(Q \times V \times Q)$ is a transition relation,
- $E \in \mathbb{P}(Q \times Q)$ is an ε -transition relation
- $S \subseteq Q$ is a set of start states, and
- $F \subseteq Q$ is a set of final states.

□

```
class FA: virtual public FAabs {
    // Q is a finite set of states
    StatePool Q;
    // S is a set of start states, F is a set of final states
    StateSet S, F;
    // Transitions maps each State to its out-transitions.
    TransRel Transitions;
    // E is the epsilon transition relation.
    StateRel E;
}
```

StatePool: All states in an automaton are allocated from a StatePool. StatePool's can be merged together to form a larger one. (Care must be taken to rename any relations or sets (during merging) that depend on the one StatePool.) State is in $[0, \text{next})$

```
class StatePool {
    int next; // The next one to be allocated.
}
```

StateSet: The StateSet is normally associated (implicitly) with a particular StatePool; whenever a StateSet is to interact with another (from a different StatePool), its States must be renamed (to avoid a name clash). The capacity of a StateSet must be explicitly managed; many set operations are not bounds-checked when `assert()` is turned off.

```
class StateSet :protected BitVec {
    // How many States can this set contain?
    // [O, domain()) can be contained in *this.
    inline int domain() const;

    // set How many States can this set contain.
    // [O, r) can be contained in *this.
    inline void set_domain(const int r);
}
class BitVec {
    // used max number bits in data, denote width(domain), [0,
    bits_in_use) ==> [0, width)
    int bits_in_use;
    // number of words, 1, 2, 3, ...
    int words;
    // save bytes of words, [0, 1, 2, ... width(domain)]
    unsigned int *data;
}
```

transition relation: $T \in Q \rightarrow \mathbb{P}(V \times Q)$, $T(p) = \{(a, q) | (p, a, q) \in T\}$, 表示状态 p 的 out-transitions. see Fig 1.1

```
// V -> Q
struct TransPair {
    CharRange transition_label;
    State transition_destination;
}
class TransImpl { TransPair *data; }
class Trans:protected TransImpl { }

// map: state(r) -> (T=Trans) out-transitions of r
// SteteTo::data[r] = out-transitions of state r
class TransRel:public StateTo<Trans> {}

// map: state(r) -> T
// data[r] = T
template <class T> class SteteTo {
T *data; // 动态数组的index(即状态的index)状态的out-transitions
```

```

}
class FA:virtual public FAabs {
TransRel Transitions; // maps each State to its out-transitions.
}

```

ε -relation: $E \in \mathbb{P}(Q \times Q) \Rightarrow E \in Q \rightarrow \mathbb{P}(Q), E(p) = \{q \mid (p, q) \in E\}$, 表示 ε 连接状态 p 和状态 q .

```

// Implement binary relations on States. This is most often used for
// epsilon transitions.
// map: state(r) -> {StateSet}
// StateTo::data[r] = {StateSet}, 表示状态r与{StateSet}的二元关系
class StateRel :protected StateTo<StateSet> {
}

class FA:virtual public FAabs {
// E is the epsilon transition relation.
StateRel E;
}

```

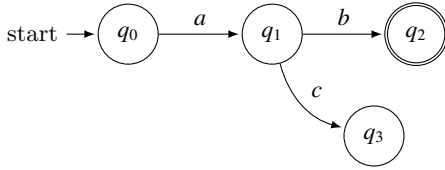


图 1.1: q_1 in-transition: $\{(q_0, a, q_1)\}$; q_1 out-transition: $\{(q_1, b, q_2), (q_1, c, q_3)\}$

[WATSON93a, p6] **the signatures of the transition relations:**

$$T \in \mathbb{P}(Q \times V \times Q)$$

$$T \in V \rightarrow \mathbb{P}(Q \times Q)$$

$$T \in Q \times Q \rightarrow \mathbb{P}(V)$$

$$T \in Q \times V \rightarrow \mathbb{P}(Q)$$

$$T \in Q \rightarrow \mathbb{P}(V \times Q)$$

for example, the function $T \in Q \rightarrow \mathbb{P}(V \times Q)$ is defined as $T(p) = \{(a, q) : (p, a, q) \in T\}$

$$T \in \mathbb{P}(Q \times V \times Q), T = \{(p, a, q)\}$$

$$T \in Q \rightarrow \mathbb{P}(V \times Q), T(p) = \{(a, q) : (p, a, q) \in T\}$$

$$p, q \in Q, a \in V$$

$$T : Q \times V \rightarrow \mathbb{P}(Q)$$

$$T(p, a) = \{q\}$$

According to Convention A.4 (Tuple projection):

$$\begin{aligned}
T &\in \mathbb{P}(Q \times V \times Q), T = \{(p, a, q)\} \\
\pi_2(T) &= \{a \mid (p, a, q) \in T\}, \bar{\pi}_2(T) = \{(p, q) \mid (p, a, q) \in T\} \\
T \in Q &\rightarrow \mathbb{P}(V \times Q), T(p) = \{(a, q) \mid (p, a, q) \in T\} \\
\pi_2(T(p)) &= \{q \mid (p, a, q) \in T\}, \bar{\pi}_2(T(p)) = \{a \mid (p, a, q) \in T\} \\
Q_{map} : Q \times V, T(p) &= \{(a, q) \mid (p, a, q) \in T\} \\
Q_{map}(q) &= \{a\} \\
Q_{map} : Q \times V, T &\in \mathbb{P}(Q \times V \times Q) \\
\pi_1(T) &= \{p \mid (p, a, q) \in T\}, \bar{\pi}_1(T) = \{(a, q) \mid (p, a, q) \in T\} \\
Q_{map} &= (\bar{\pi}_1(T))^R = \{(a, q) \mid (p, a, q) \in T\}^R = \{(q, a) \mid (p, a, q) \in T\}
\end{aligned}$$

1.2 Properties of finite automata

$$M = (Q, V, T, E, S, F), M_0 = (Q_0, V_0, T_0, E_0, S_0, F_0), M_1 = (Q_1, V_1, T_1, E_1, S_1, F_1)$$

Definition 1.2 (Size of an FA). Define the size of an FA as $|M| = |Q|$

Definition 1.3 (Isomorphism 同构 (\cong) of FA's). We define isomorphism (\cong) as an equivalence relation on FA's. M_0 and M_1 are isomorphic (written $M_0 \cong M_1$) if and only if $V_0 = V_1$ and there exists a bijection 双射 $g \in Q_0 \rightarrow Q_1$ such that

- $T_1 = \{(g(p), a, g(q)) \mid (p, a, q) \in T_0\}$
- $E_1 = \{(g(p), g(q)) \mid (p, q) \in E_0\}$
- $S_1 = \{g(s) \mid s \in S_0\}$ and
- $F_1 = \{g(f) \mid f \in F_0\}$

(see Fig 1.2). □

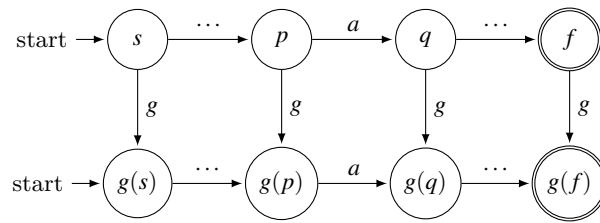


图 1.2: Isomorphism $M_0 \cong M_1$ if and only if $V_0 = V_1$ and there exists a bijection $g \in Q_0 \rightarrow Q_1$

Definition 1.4 (Extending the transition relation T). We extend transition relation $T \in V \rightarrow \mathbb{P}(Q \times Q)$ to $T^* \in V^* \rightarrow \mathbb{P}(Q \times Q)$ as follows:

$$T^*(\epsilon) = E^*$$

and (for $a \in V, w \in V^*$)

$$T^*(aw) = E^* \circ T(a) \circ T^*(w)$$

Operator \circ (composition is defined in Convention 1).

This definition could also have been presented symmetrically. □

Note 1.1. $s_1, s_2, s_3, s_4 \in Q, a \in V, w \in V^*$

$$E = T(\epsilon) = \{(s_1, s_2)\}, T(a) = \{(s_2, s_3)\}, T^*(w) = \{(s_3, s_4)\}$$

$$\begin{aligned} T^*(aw) &= E^* \circ T(a) \circ T^*(w) \\ &= \{(s_1, s_2)\} \circ \{(s_2, s_3)\} \circ \{(s_3, s_4)\} = \{(s_1, s_3)\} \circ \{(s_3, s_4)\} = \{(s_1, s_4)\} \end{aligned}$$

Note 1.2. $T \in Q \times V \rightarrow \mathbb{P}(Q)$, extend to: $T^* \in Q \times V^* \rightarrow \mathbb{P}(Q)$

$$\forall q \in Q, w \in V^*, a \in V,$$

1. $T^*(q, \epsilon) = q$
2. $T^*(q, wa) = T(T^*(q, w), a)$

$$\begin{aligned} T^*(q, a) &= T^*(q, \epsilon a) \\ &= T(T^*(q, \epsilon), a) \\ &= T(q, a) \end{aligned}$$

两值相同，不用区分这两个符号。

Convention 1 (Relation composition) Given sets A, B, C (not necessarily different) and two relations, $E \subseteq A \times B$ and $F \subseteq B \times C$, we define relation composition (infix operator 中缀操作符 \circ) as:

$$E \circ F = \{(a, c) | (\exists b \in B), (a, b) \in E \wedge (b, c) \in F\}$$

□

Note 1.3. if $\exists b \in B, (a, b) \in E, (b, c) \in F$, then

$$E : A \rightarrow B \Rightarrow E(a) = b$$

$$F : B \rightarrow C \Rightarrow F(b) = c$$

$$E \circ F = \{(a, b)\} \circ \{(b, c)\} = \{a, c\}$$

$$\begin{aligned} (E \circ F)(a) &= F(E(a)) \\ &= F(b) = c \end{aligned}$$

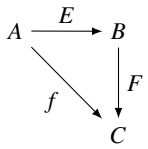


图 1.3: $E \circ F = (F \circ E)(a) = F(E(a)) = c = f(a)$

Remark 1.1. We also sometimes use the signature $T^* \in Q \times Q \rightarrow \mathbb{P}(V^*)$

□

Note 1.4. $T(p, q) = \{w | p, q \in Q, w \in V^*\}$

Remark 1.2. if $E = \emptyset$ then $E^* = \emptyset^* = I_Q$ where I_Q is the identity relation 单位关系 on the states of M .

Definition 1.5 (The language between states). The language between any two states $q_0, q_1 \in Q$ is $T^*(q_0, q_1)$.

□

Definition 1.6 (Left and right languages). The left language of a state (in M) is given by function, $\overleftarrow{L}_M \in Q \rightarrow \mathbb{P}(V^*)$, where

$$\overleftarrow{L}_M(q) = (\cup s : s \in S : T^*(s, q))$$

The right language of a state (in M) is given by function $\overrightarrow{L}_M \in Q \rightarrow \mathbb{P}(V^*)$, where

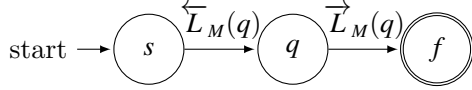
$$\overrightarrow{L}_M(q) = (\cup f : f \in F : T^*(q, f))$$

The subscript M is usually dropped when no ambiguity can arise. □

Example 1.1. $T^* \in Q \times Q \rightarrow \mathbb{P}(V^*)$, $\overleftarrow{L}_M, \overrightarrow{L}_M \in Q \rightarrow \mathbb{P}(V^*)$.

$\overleftarrow{L}_M(q) = \{\text{能引导 } M \text{ 从开始状态到达 } q \text{ 状态的字符串集合}\}$, (从 q 往左看)

$\overrightarrow{L}_M(q) = \{\text{能引导 } M \text{ 从开始状态到达 } q \text{ 状态的字符串集合}\}$, (从 q 往右看)



see Fig 1.4.

$$\begin{aligned}
 \overleftarrow{L}_M(q_2) &= (s \rightarrow q_1 \rightarrow q_2) \cup (s \rightarrow (q_1 \rightarrow q_3)^* \rightarrow q_1 \rightarrow q_2) \cup (s \rightarrow (q_1 \rightarrow q_3)^* \rightarrow q_3 \rightarrow q_2) \\
 &= [(s \rightarrow q_1 \rightarrow q_2) \cup (s \rightarrow (q_1 \rightarrow q_3)^* \rightarrow q_1 \rightarrow q_2)] \cup (s \rightarrow (q_1 \rightarrow q_3)^* \rightarrow q_3 \rightarrow q_2) \\
 &= (s \rightarrow (q_1 \rightarrow q_3)^* \rightarrow q_1 \rightarrow q_2) \cup (s \rightarrow (q_1 \rightarrow q_3)^* \rightarrow q_3 \rightarrow q_2) \\
 &= \{1(10)^*0, 1(10)^*1\} \\
 \overrightarrow{L}_M(q_2) &= \{01^*0, 10^*1(001^*0 + (10)^*1)\}
 \end{aligned}$$

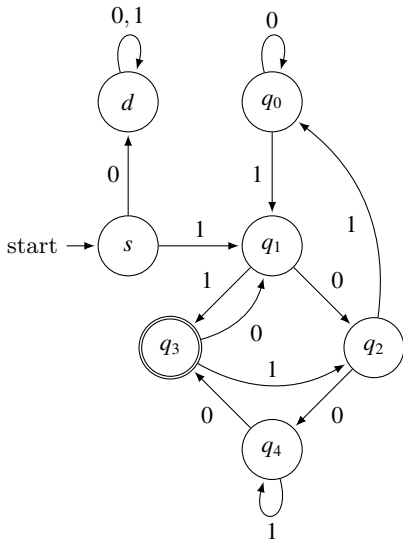


图 1.4: $\{x | x \in \{0,1\}^+ \text{ 且当把 } x \text{ 看成二进制数时, } x \text{ 模 } 5 \text{ 与 } 3 \text{ 同余,}$

要求当 x 为 0 时, $|x| = 1$, 且当 $x \neq 0$ 时, x 的首字符为 1} 语言对应的 DFA

Definition 1.7 (Language of an FA). The language of a finite automaton (with alphabet V) is given by the function $L_{FA} \rightarrow \mathbb{P}(V^*)$ defined as:

$$L_{FA}(M) = (\cup s, f : s \in S \wedge f \in F : T^*(s, f)) \text{ (所有从开始状态到接受状态的字符串集合)}$$

□

Property 1.1 (Language of an FA). From the definition of left and right languages (of a state), we can also write:

$$L_{FA}(M) = (\cup f : f \in F : \overleftarrow{L}(f)) \text{ (所有从 } s \text{ 到 } f \text{ 的字符串集合, 从 } f \text{ 向左看)}$$

and

$$L_{FA}(M) = (\cup s : s \in S : \overrightarrow{L}(s)) \text{ (所有从 } s \text{ 到 } f \text{ 的字符串集合, 从 } s \text{ 向右看)}$$

□

Definition 1.8 (ε -free 无 ε 转移). Automaton M is ε -free if and only if $E = \emptyset$.

□

Remark 1.3. Even if M is ε -free it is still possible that $\varepsilon \in L_{FA}(M) : \text{in this case } S \cap F \neq \emptyset$. (开始状态也是接受状态)

□

Form [WATSON93a, Convention A.4] (Tuple projection).

Convention 2 (Tuple projection) For an n -tuple $t = (x_1, x_2, \dots, x_n)$ we use the notation $\pi_i(t) (1 \leq i \leq n)$ to denote tuple element x_i ; we use the notation $\bar{\pi}_i(t) (1 \leq i \leq n)$ to denote the $(n-1)$ -tuple $(x_1, \dots, x_{i-1}, x_{i+1}, \dots, x_n)$. Both π and $\bar{\pi}$ extend naturally to sets of tuples.

□

Form [WATSON93a, Definition A.20] (Tuple and relation reversal).

Definition 1.9 (Tuple and relation reversal). For an n -tuple (x_1, x_2, \dots, x_n) define reversal as (postfix and superscript) function R :

$$(x_1, x_2, \dots, x_n)^R = (x_n, x_{n-1}, \dots, x_2, x_1)$$

Given a set A of tuples, we define $A^R = \{x^R : x \in A\}$.

□

Definition 1.10 (Reachable states). For M we can define a reachability relation $Reach(M) \subseteq (Q \times Q)$ defined as

$$Reach(M) = (\bar{\pi}_2(T) \cup E)^* \text{ see}^1$$

Functions π and $\bar{\pi}$ are defined in Convention 2. Similarly the set of start-reachable states is defined to be:

$$SReachable(M) = Reach(M)(S) \text{ see}^2$$

and the set of final-reachable states is defined to be:

$$FReachable(M) = (Reach(M))^R(F) \text{ see}^3$$

Reversal of a relation is defined in Definition 1.9. The set of useful states is: $Reachable(M) = SReachable(M) \cap FReachable(M)$

□

Remark 1.4. For $FA M = (Q, V, T, E, S, F)$, function $SReachable$ satisfies the following interesting property:

$$q \in SReachable(M) \equiv \overleftarrow{L}_M(q) \neq \emptyset$$

$FReachable$ satisfies a similar property:

$$q \in FReachable(M) \equiv \overrightarrow{L}_M(q) \neq \emptyset$$

□

Example 1.2. $T \in \mathbb{P}(Q \times V \times Q)$, $T = \{(p, a, q) | p, q \in Q, a \in V\}$,

$$\bar{\pi}_2(T) = \{(p, q) | (p, a, q) \in T\}$$

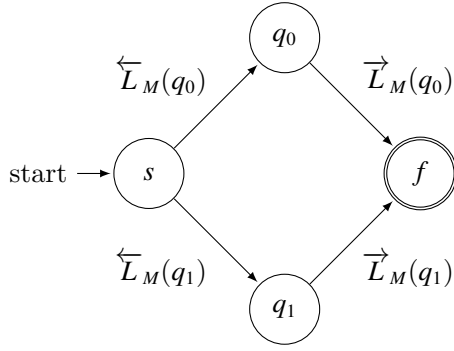
$$Q_{map} = (\bar{\pi}_1(T))^R, Q_{map} = \{(a, q) | (p, a, q) \in T\}^R = \{(q, a) | (p, a, q) \in T\}$$

□

¹ $\{(p_1, q_1), (p_2, q_2), \dots\}$

² 从 start state 可以到达的状态集合

³ 可以到达 final state 的状态集合



e.g. $p = \{1, 2\} \in \mathcal{Q}_1 \subseteq \mathbb{P}(\mathcal{Q}_0)$, $\vec{L}_{M_1}(p) = \vec{L}_{M_0}(1) \cup \vec{L}_{M_0}(2)$

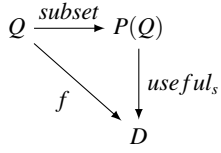


图 1.5: $\text{subset} \circ \text{useful}_s = \text{useful}_s(\text{subset}(Q, V, \emptyset, S, F)) = (D, V, T', \emptyset, S', F')$

1.3 Σ -algebras and regular expressions

Σ -homomorphism

X 集合中的元素与有序集 S 中的元素一一对应, 称 X 是 S -sorted. $S = \{1, 3, 7, 9\}, X = \{d, a, c, f\}, s \in S, X_s \in X$
 S 是有序的, $S_{s_1} = 1, S_{s_2} = 3, S_{s_3} = 7, S_{s_4} = 9$
 X 与 S 中的元素一一对应。 $X_{s_1} = d, X_{s_2} = a, X_{s_3} = c, X_{s_4} = f$

Σ -homomorphism 同态: $(V, F) \Leftrightarrow (W, G)$, 载体 (V, W) 和操作 (F, G) 一一对应。

Σ -homomorphism function: $h \in V \rightarrow W$

$$L(v) = (h \circ f)(v) = h(f(v)) = g(w) = L_{reg} = L_V = L_W$$

$$L(v) = (g \circ h)(v) = g(h(v)) = g(w) = L_{reg} = L_V = L_W$$

$$\Rightarrow h(f(v)) = g(h(v))$$

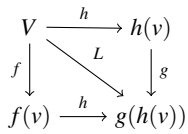


图 1.6: $(h \circ f)(v) = (g \circ h)(v) \Rightarrow h(f(v)) = g(h(v))$

$$\begin{array}{ccc}
(e_1, e_2) & \xrightarrow{h} & (h(e_1), h(e_2)) \\
f \downarrow & \searrow L & \downarrow g \\
f(e_1, e_2) \circ & \xrightarrow{h} & g(h(e_1), h(e_2))
\end{array}$$

图 1.7: $(h \circ f)(e_1, e_2) = (g \circ h)(e_1, e_2) \Rightarrow h(f(e_1, e_2)) = g(h(e_1), h(e_2))$

Example 1.3. $\Sigma = (S, \Gamma)$, sort: $expr$, $\Gamma := \{a, plus\}$, a is a constant. operator $plus : expr \times expr \rightarrow expr$.

Σ -term algebra: $plus[a, a], plus[plus[a, plus[a, a]], a]$

Σ -algebra X , carrier set: natural number, constant 0. operator $f_{plus}(x, y) = (x \max y) + 1$

Σ -homomorphism function("expression tree height"): $h_{expr} : \Sigma\text{-term algebra} \rightarrow X$

$$\begin{aligned}
(h_{expr} \circ plus)(s) &= (f_{plus} \circ h_{expr})(s) \\
h_{expr}(plus(s)) &= f_{plus}(h_{expr}(s)) \\
left : s \leftarrow e, f &\Rightarrow plus[e, f] \\
right : s \leftarrow e, f &\Rightarrow f_{plus}(h_{expr}(e), h_{expr}(f)) \\
h_{expr}(plus(e, f)) &= f_{plus}(h_{expr}(e), h_{expr}(f)) \\
&= (h_{expr}(e) \max h_{expr}(f)) + 1 \\
&\text{and,} \\
h_{expr}(a) &= 0
\end{aligned}$$

$$\begin{array}{ccc}
S & \xrightarrow{h_{expr}} & X \\
plus \downarrow & \searrow L & \downarrow f_{plus} \\
\circ & \xrightarrow{h_{expr}} & \circ
\end{array}$$

图 1.8: $(h_{expr} \circ plus)(s) = (f_{plus} \circ h_{expr})(s) \Rightarrow h_{expr}(plus(s)) = f_{plus}(h_{expr}(s))$

$$\begin{array}{ccc}
(e, f) & \xrightarrow{h_{expr}} & (h_{expr}(e), h_{expr}(f)) \\
plus \downarrow & \searrow L & \downarrow f_{plus} \\
plus[e, f] & \xrightarrow{h_{expr}} & f_{plus}(h_{expr}(e), h_{expr}(f))
\end{array}$$

图 1.9: $(h_{expr} \circ plus)(e, f) = (f_{plus} \circ h_{expr})(e, f) \Rightarrow h_{expr}(plus[e, f]) = f_{plus}(h_{expr}(e), h_{expr}(f))$

Definition 1.11 (Regular expressions). We define regular expressions (over alphabet V) as the Σ -term algebra over signature $\Sigma = (S, O)$ where

- S consists of a single sort Reg (for regular expression), and
- O is a set of several constans: $\varepsilon, \emptyset, a_1, a_2, \dots, a_n$; Reg (where $V = \{a_1, a_2, \dots, a_n\}$) and five operators $. : Reg \times Reg \rightarrow Reg$ (the dot operator), $\cup : Reg \times Reg \rightarrow Reg$, $*$: $Reg \rightarrow Reg$, $+$: $Reg \rightarrow Reg$, and $?$: $Reg \rightarrow Reg$.

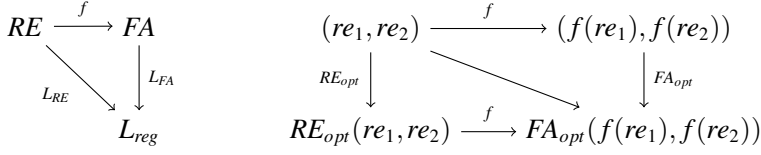
$V := RE$ (正则表达式), $W := FA$ (有限自动机)

Σ -homomorphism function: $f \in RE \rightarrow FA$

$F : RE_{opt}$ 运算, 二元: union(or),concat; 一元: star,plus,question;

常量:epsilon,empty,symbol

$G : FA_{opt}$ 运算, 同上



```

//Sigma.h
template<class T>
class Reg : public T {
// Helper for constructing the homomorphic image of a regular expression
.
// T is carrier set: RE,FA,RFA,
// 各自的操作, 分别在Sig-RE.cpp, Sig-FA.cpp, Sig-RFA.cpp中定义
inline void homomorphic_image(const RE& r);
Reg<T>& epsilon();
Reg<T>& empty();
Reg<T>& symbol(const CharRange r);
Reg<T>& Or(const Reg<T>& r);
Reg<T>& concat(const Reg<T>& r);
Reg<T>& star();
Reg<T>& plus();
Reg<T>& question();
}

```

Definition 1.12 (The nullable Σ -algebra). We define the *nullable* Σ -algebra as follows:

- The carrier set is $\{true, false\}$.
- $a \in V, E_1, E_2 \in RE, \varepsilon \in E_1^*, \varepsilon \in E_1^?, \varepsilon \notin E_1^+$

$$\begin{aligned}
nullable(\varepsilon) &= true \\
nullable(\emptyset) &= nullable(a) = false \\
nullable(E_1 \vee E_2) &= nullable(E_1 \cup E_2) \\
nullable(E_1 \wedge E_2) &= nullable(E_1 \cdot E_2) \\
nullable(E_1^*) &= true \\
nullable(E_1^+) &= nullable(E_1) \\
nullable(E_1^?) &= true \\
nullable(E_1) &= \begin{cases} true & \varepsilon \in E_1 \\ false & \varepsilon \notin E_1 \end{cases}
\end{aligned}$$

1.4 Constructing ε -lookahead automata

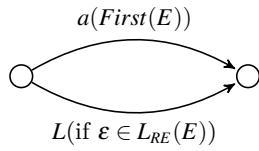


图 1.10: Lookahead function: $look(E, L) = First \cup \text{if } (Null(E)) \text{ then } L \text{ else } \emptyset$ fi

1.5 Towards the Berry-Sethi construction

$$\begin{aligned}
\varepsilon \in L_{FA}(M) &\equiv s \in F \\
\text{start} &\rightarrow \textcircled{s}
\end{aligned}$$

图 1.11: $\varepsilon \in L_{FA}(M) \equiv s \in F$

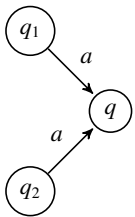


图 1.12: All in-transitions to a state are on the same symbol (in V)

a	b	c

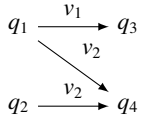
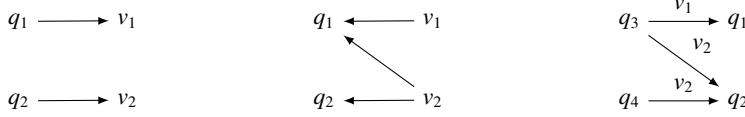


图 1.13: $q_4(\text{in-transition}) = \{(q_1, b), (q_2, b)\}, q_1(\text{out-transition}) = \{(a, q_3), (b, q_4)\}$



(a) $Q \rightarrow V$ 一一对应关系 (b) $V \rightarrow \mathbb{P}(Q)$ 一对多的关系 (c) 物理含义, 进入状态的字母是唯一的。

$$Q_{map}(q_1) = \{v_1\}, Q_{map}^{-1}(v_1) = \{q_1\}, (q_3, v_1, q_1) \in T$$

$$Q_{map}(q_2) = \{v_2\}, Q_{map}^{-1}(v_2) = \{q_2\}, (q_3, v_2, q_2) \in T, (q_4, v_2, q_2) \in T$$

图 1.14: Q_{map}, Q_{map}^{-1}

Definition 1.13 (RFA). A reduced FA (RFA) is a 7-tuple $(Q, V, follow, first, last, null, Q_{map})$ where

- Q is a finite set of states,
- V is an alphabet,
- $follow \in \mathbb{P}(Q \times Q)$ is a follow relation (replace the transition relation: $T \in \mathbb{P}(Q \times V \times Q)$),
- $first \subseteq Q$ is a set of initial states (replacing $T(s)$ in an LBFA),
- $null \in \{true, false\}$ is a Boolean value (encoding $s \in F$ in an LBFA, $\varepsilon \in L_{FA}(M) \equiv s \in F$), and
- $Q_{map} \in \mathbb{P}(Q \times V), Q_{map}(q) = \{v\}, one \rightarrow one$. maps each state to exactly one symbol. i.e. $Q_{map} \in Q \rightarrow V$.
 $Q_{map}(q) = \{a | (p, a, q) \in T\}$ 表示 (q, v) 的一一对应关系。物理含义是进入 q 状态的唯一字母 a
 class RFA 中表示 its inverse: $Q_{map}^{-1} : V \rightarrow \mathbb{P}(Q)$, 部分函数 $Q_{map}^{-1}(a) = \{q | (p, a, q) \in T\}$

□

```

class RFA : virtual public FAabs{
// Q is a finite set of states
StatePool Q;

// first(subset Q) is a set of initial states(replacing T(s) in an LBFA)
,
// last(subset Q) is a set of final states ,
StateSet first , last;

// Qmap (in P( Q x V)) maps each state to exactly one symbol (it is also
viewed as Qmap in Q --> V,
// and its inverse as Qmap^-1 in V --> P(Q)[the set of all partial
functions from V to P(Q)].
// Trans用struct TransPair 表示:T(a) = { q | (p,a,q) in T },
// 因此这里表示Qmap的inverse, V --> P(Q)

```



```

Trans Qmap_inverse;

// follow(in P(Q x Q)) is a follow relation(replacing the transition
    relation),
StateRel follow;

// null (in {true,false}) is a Boolean value (encoding s in F in an LBFA
    )
// if epsilon属于LBFA, true; final set中包含s
// {true, false} ==> {1, 0}
int Nullable;
}

```

```

// V -> Q
struct TransPair {
    CharRange transition_label;
    State transition_destination;
}

class TransImpl { TransPair *data; }
class Trans:protected TransImpl { }

}

```

$$rfa \circ R(E) = R \circ rfa(E)$$

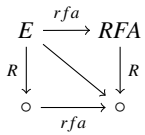


图 1.15: $rfa \circ R(E) = R \circ rfa(E)$

Definition 1.14. (Dual of a function) We assume two sets A and B whose reversal operators are R and R' respectively. Two functions, $f \in A \rightarrow B$ and $f_d \in A \rightarrow B$ are one another's dual if and only if

$$f(a) = (f_d(a^R))^{R'}$$

In some cases we relax the equality to isomorphism (when isomorphism is defined on B). □

$$\begin{aligned}
f_d \circ R(a) &= R' \circ f(a) \Rightarrow f_d(R(a)) = R'(f(a)) \Rightarrow \\
f_d(a^R) &= (f(a))^{R'} \Rightarrow f(a) = (f_d(a^R))^{R'}
\end{aligned}$$

$$\begin{array}{ccc}
A & \xrightarrow{f} & B \\
R \downarrow & \searrow & \downarrow R' \\
\circ & \xrightarrow{f_d} & \circ
\end{array}$$

图 1.16: $f(a) = (f_d(a^R))^{R'}$

$$\begin{aligned}
C_{\cdot, RFA}(rfa(\$), rfa(E)) &= C_{\cdot, RFA}(C_{\$, RFA}, rfa(E)) = L_{RE}(\$E) = \{\$ \} L_{RE}(E) \\
cover & \text{ 简单剔除 } E \text{ 的第一个字符。 } L_{FA} \circ convert \circ rfa(E) = V^{-1} L_{RE}(E) \\
convert(C_{\cdot, RFA}(C_{\$, RFA}, rfa(E))) &= rfa(E)
\end{aligned}$$

$$\begin{array}{ccc}
RE & \xrightarrow{rfa} & RFA \\
\searrow L_{RE} & & \downarrow L_{FA} \\
& & L_{reg}
\end{array}
\quad
\begin{array}{ccc}
(\$E) & \xrightarrow{rfa} & (rfa(\$), rfa(E)) \\
\searrow RE \cdot & & \downarrow FA \cdot \\
\$ \cdot E & \xrightarrow{rfa} & rfa(\$) \cdot rfa(E)
\end{array}$$

图 1.17: $convert(C_{\cdot, RFA}(C_{\$, RFA}, rfa(E))) = rfa(E)$

1.6 The Berry-Sethi construction

Algorithm 2.45(implements $useful_s \circ subset$):

initial: $D = \emptyset, U = S$

$$\begin{aligned}
d &:= \bigcup_{q \in u} T(q, a) \\
\{q_1, q_2\} &\xrightarrow{a} \{T(q_1, a), T(q_2, a)\}
\end{aligned}$$

using Algorithm 2.45 for $decode(RFA \rightarrow LBFA)$

$$\begin{aligned}
d &:= \bigcup \{q | q \in first \wedge Q_{map}(q) = a\} \\
\{s\} &\xrightarrow{a} \{d\}
\end{aligned}$$

note:

$$\begin{aligned}
d &:= \bigcup_{p \in u} \{q | (p, q) \in follow \wedge Q_{map}(q) = a\} \\
u &= \{p_1, p_2\}, d = \{q_1, q_2\}, \\
follow(p_1) &= q_1, follow(p_2) = q_2, Q_{map}(q_1) = Q_{map}(q_2) = a \\
\{p_1, p_2\} &\xrightarrow{a} \{q_1, q_2\}
\end{aligned}$$

$$RE \rightarrow [DFA]_{\cong}$$

$$MYG(E) = \text{useful}_s \circ \text{subset} \circ \text{decode} \circ \text{rfa}(E)$$

$$\begin{array}{c} RE \xrightarrow{\text{rfa}} [RFA]_{\cong} \xrightarrow{\text{decode}} [NFA]_{\cong} \\ \downarrow \text{subset} \\ [DFA]_{\cong} \xleftarrow{\text{Complete}} [DFA]_{\cong} \xleftarrow{\text{useful}_s} P(Q) \end{array}$$

图 1.18: $MYG(E) = \text{useful}_s \circ \text{subset} \circ \text{decode} \circ \text{rfa}(E)$

1.7 The McNaughton-Yamada-Clushkov construction

1.8 The dual of the Berry-Sethi construction

$$\begin{array}{ccc} \text{Dual construction: } R \circ f \circ R & & f : FA \text{ construction} \\ RE \xrightarrow{R} RE \xrightarrow{f} FA \xrightarrow{R} FA & & \begin{array}{ccc} RE & \xrightarrow{f} & FA \\ & \searrow L_{RE} & \downarrow L_{FA} \\ & & L_{reg} \end{array} \end{array}$$

图 1.19: Dual construction: $R \circ f \circ R$

$$R \circ R \text{ is the identity} \Rightarrow R \circ R(A) = A$$

1.9 Algorithm 4.52 (Aho-Sethi-Ullman)

note:

$$d := \bigcup_{q \in u} \{ \text{follow}(q) \mid Q_{\text{map}}(q) = a \}$$

$$q \xrightarrow{a} \text{follow}(q)$$

note:

$$T_0(b) = (p, p'), T_1(b) = (q, q')$$

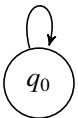
$$\pi_2(T_0(b)) = p', \pi_2(T_1(b)) = q'$$

$$T_0(s_0, a) = p, T_1(s_1, a) = q$$

$$Q' = \{q_0\} \cup (\bigcup_{b \in V} \{ \pi_2(T_0(b)) \times \pi_2(T_1(b)) \})$$

$$M_0 : s_0 \xrightarrow{a} p \xrightarrow{b} p'$$

$$M_1 : s_1 \xrightarrow{a} q \xrightarrow{b} q'$$



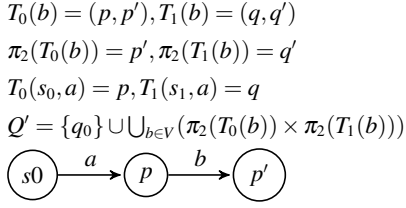


图 1.20: Intersection of LBFA's

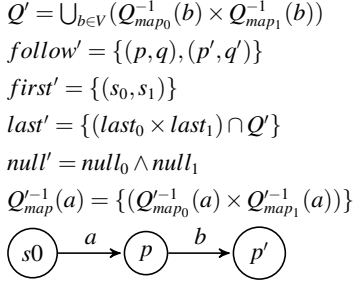


图 1.21: Intersection of RFA's

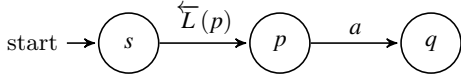
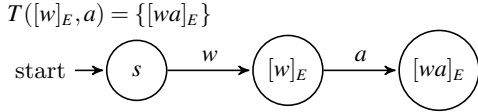
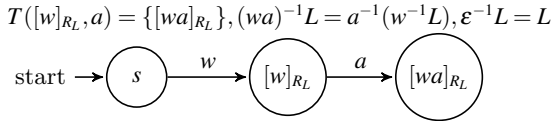
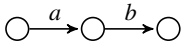
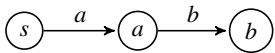


图 1.22: text111

图 1.23: $T([w]_E, a) = \{[wa]_E\}$ 图 1.24: $T([w]_{R_L}, a) = \{[wa]_{R_L}\}$ 图 1.25: $a, b \in V, (a, b) \in Follow(E)$ 图 1.26: $BSenc(E)$

$$\circ \xrightarrow{a} \circ q_0$$

图 1.27: text

notes:

$[V^*]_{R_L} = V^*/R_L$ 表示右不变的等价关系, 每个等价关系对应一个状态。

$[\epsilon]_{R_L}$ 表示 ϵ 所在的等价类对应的状态, 就是开始状态

$[V^*]_E = V^*/R_E$ 表示右不变的等价关系, 每个等价关系对应一个状态。

$[\epsilon]_E$ 表示 ϵ 所在的等价类对应的状态, 就是开始状态

$$\begin{aligned}
 First((a \cup \epsilon)b^*) &= (Defn. \quad First(E \cdot F), \epsilon \in (a \cup \epsilon), Null(E) = true) \\
 &= (\Rightarrow First(E) \cup First(F)) \\
 &= First(a \cup \epsilon) \cup First(b^*) \\
 &= (Defn. \quad First(E \cup F) = First(E) \cup First(F), First(E^*) = First(E)) \\
 &= (First(a) \cup First(\epsilon)) \cup First(b) \\
 &= \{a \cup \emptyset\} \cup \{b\} = \{a, b\}
 \end{aligned}$$

$$\begin{aligned}
 First((a \cup \epsilon)b^*) &= First(ab^* \cup \epsilon b^*) \\
 &= (Defn. \quad First(E \cup F) = First(E) \cup First(F)) \\
 &= First(ab^*) \cup First(b^*) \\
 &= (Defn. \quad First(E \cdot F), \epsilon \notin \{a\}, Null(E) = false \Rightarrow First(ab^*) = First(a) \cup \emptyset = \{a\}) \\
 &= First(a) \cup First(b) = \{a, b\}
 \end{aligned}$$

$$\begin{aligned}
 Last((a \cup \epsilon)b^*) &= (Defn. \quad Last(E \cdot F), \epsilon \in (a \cup \epsilon), Null(E) = true) \\
 &= (\Rightarrow Last(E) \cup Last(F)) \\
 &= Last(a \cup \epsilon) \cup Last(b^*) \\
 &= (Defn. \quad Last(E \cup F) = Last(E) \cup Last(F), Last(E^*) = Last(E)) \\
 &= (Last(a) \cup Last(\epsilon)) \cup Last(b) \\
 &= \{a \cup \emptyset\} \cup \{b\} = \{a, b\}
 \end{aligned}$$

$$\begin{aligned}
Null((a \cup \varepsilon)b^*) &= (Defn. \quad Null(E \cdot F) = Null(E \wedge F), \varepsilon \in L_{RE} \equiv Null(E)) \\
&= Null(a \cup \varepsilon) \wedge Null(b^*) \\
&= (Defn. \quad Null(E \cup F) = Null(E \vee F), Null(E^*) = true) \\
&= (Null(a) \vee Null(\varepsilon)) \wedge true \\
&= true \wedge true = true
\end{aligned}$$

$$\begin{aligned}
Last(a \cup \varepsilon) &= Last(a) \cup Last(\varepsilon) = \{a\} \cup \emptyset = \{a\} \\
First(b^*) &= First(b) = \{b\} \\
Follow(a \cup \varepsilon) &= Follow(a) \cup Follow(\varepsilon) = \emptyset \cup \emptyset = \emptyset \\
Follow(b^*) &= Follow(b) \cup (Last(b) \times First(b)) = \emptyset \cup \{(b, b)\} = \{(b, b)\}
\end{aligned}$$

$$\begin{aligned}
Follow((a \cup \varepsilon)b^*) &= Follow(a \cup \varepsilon) \cup Follow(b^*) \cup (Last(a \cup \varepsilon) \times First(b^*)) \\
&= \emptyset \cup \{(b, b)\} \cup \{(\{a\} \times \{b\})\} \\
&= \{(b, b)\} \cup \{(a, b)\} \\
&= \{(a, b), (b, b)\}
\end{aligned}$$

$$L_0 = L, L_1 = w^{-1}L, L_2 = a^{-1}(w^{-1}L) = (aw)^{-1}L$$

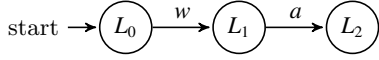


图 1.28: Construction 5.19(MNmin)

$$E_0 = E, E_1 = [v^{-1}E]_{\sim}, E_2 = \{a^{-1}[v^{-1}E]_{\sim}\} = \{[va]_{\sim}^{-1}E\}$$

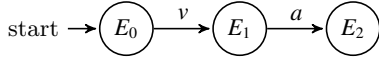


图 1.29: Construction 5.34 (Brzozowski)

$$(\forall u, a, u \in V^* \wedge a \in V), (\exists v \in V^*, [u]_E \cdot a \subseteq [v]_E)$$

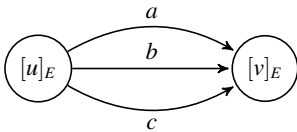
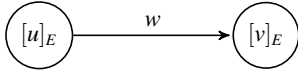


图 1.30: Definition 5.2 (Right invariance of an equivalence relation)

$$u, w \in V^*, (\forall u, w, (\exists v \in V^*, [u]_E \cdot \{w\} \subseteq [v]_E))$$

图 1.31: Right invariance of an equivalence relation $[u], [v]$

$M = (Q, V, E, s, F)$, M 所确定的 V^* 上的关系 R_M 定义为: 对于 $\forall x, y \in V^*$,

$$xR_M y \Leftrightarrow T^*(s, x) = T^*(s, y)$$

\Rightarrow

$$xR_M y \Leftrightarrow \exists q \in Q, x, y \in \overleftarrow{L}(q)$$

按照这个定义所得的关系 R_M , 实际上是 V^* 上的等价关系, 利用这个关系, 可以将 V^* 划分成不多于 $|Q|$ 个等价类。

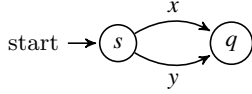


图 1.32: Equivalence classes of an equivalence relation

$\forall x, y \in V^*$, 如果 $xR_M y$, 则在 x 和 y 后无论接 V^* 中的任何字符串 z , xz 和 yz 要么都属于 L , 要么都不属于 L 。

$$xR_M y \Leftrightarrow (\forall z \in V^*, xz \in L \Leftrightarrow yz \in L)$$

q 是自动机的一个状态, 从开始状态到达该状态的字符串 ($\overleftarrow{L}(q)$) 是一个等价关系, 用 $[q]_E$ 表示。

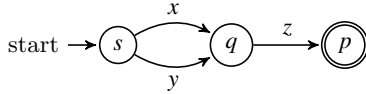


图 1.33: Equivalence classes of an equivalence relation

Def:

$$M = (Q, V, T, E, S, F), T^* \in (Q \times Q) \rightarrow \mathbb{P}(V^*)$$

$$\overleftarrow{L}_M(q), \overrightarrow{L}_M(q) \in Q \rightarrow \mathbb{P}(V^*)$$

$$\overleftarrow{L}_M(q) = \{x | x \in V^*, T^*(s, q) = x, s \in S\}$$

$$\overrightarrow{L}_M(q) = \{x | x \in V^*, T^*(q, f) = x, f \in F\}$$

$$L_{FA}(Q, V, T, E, S, F) = \bigcup_{f \in F} (\overleftarrow{L}(f))$$

1.10 Others

Definition 1.15 (Prefix-closure[Chrison2007]). Let $L \subseteq V^*$, then

$$\overline{L} := \{s \in V^* : (\exists t \in V^*) [st \in L]\}$$

In words, the prefix closure of L is the language denoted by \overline{L} and consisting of all the prefixes in L . In general, $L \subseteq \overline{L}$.

L is said to be prefix-closed if $L = \overline{L}$. Thus language L is prefix-closed if any prefix of any string in L is also an element of L .

$$L_1 = \{\varepsilon, a, aa\}, L_1 = \overline{L}_1, L_1 \text{ is prefix-closed.}$$

$$L_2 = \{a, b, ab\}, \overline{L}_2 = \{\varepsilon, a, b, ab\}, L_2 \subset \overline{L}_2, L_2 \text{ is not prefix closed.}$$

$u \in V^*, [u]_E$ 表示与字符串 u 等价的一系列字符串。如果用自动机表示: $[u]_E = \overleftarrow{L}(q)$, 表示引导自动机由开始状态到达 q 状态的所有字符串, 构成对应于 q 状态的一个等价的字符串, 等价类的指数 (个数) = 自动机的状态数: $\#E = |Q|$ 。如果用完全自动机表示, 每个状态下, 所有字母均可发生, 所以对应每个字母, 都有后续状态。

$M \in DFA, Complete(M), \forall p \in Q, a \in V, T(p, a) \neq \emptyset \Rightarrow$

$$\overleftarrow{L}(p) \cdot \{a\} \subseteq \overleftarrow{L}(T(p, a))$$

$$\text{Let } [u]_E = \overleftarrow{L}(p), [v]_E = \overleftarrow{L}(T(p, a)), \Rightarrow$$

$$(\forall u, a | u \in V^*, a \in V) (\exists v | v \in V^*, [u]_E \cdot \{a\} \subseteq [v]_E)$$

因此, 这些等价关系都是右不变的等价关系。任意一个字符串连接一个字符 (或字符串) 后, 还是存在路径可以到达自动机的某个状态。

$V = \{0, 1\}$, 三个状态, 三个等价类, 均属于右不变的等价关系: $m, n \geq 0, [u]_E = \overleftarrow{L}(q_0) = \{(00)^m\}, [v]_E = \overleftarrow{L}(q_1) = \{0(00)^m\}, [v']_E = \overleftarrow{L}(q_2) = \{(00)^m 1, 0(00)^n 1\}$

$$[(00)^m, (00)^n] \in [u]_E, [0(00)^m, 0(00)^n] \in [v]_E, [0(00)^m 1, 0(00)^n 1] \in [v']_E$$

$$[u]_E \cdot \{0\} = \overleftarrow{L}(q_1) \subseteq [v]_E, [u]_E \cdot \{1\} = \overleftarrow{L}(q_2) \subseteq [v']_E, \text{ 因此 } [u]_E \text{ 是右不变的等价关系。}$$

考察 R_L 右不变等价关系:

$[u]_E \subseteq [v]_{R_L}, [V^*]_E \subseteq [V^*]_{R_L}, E \subseteq R_L, \#E \geq \#R_L$ 表示 E 关系是 R_L 关系的 refinement (“加细”) 划分。因此, $(x, y) \in E \Rightarrow (x, y) \in R_L$, but $(x, y) \in R_L \not\Rightarrow (x, y) \in E$ 。

例如, $x = \overleftarrow{L}(q_0), y = \overleftarrow{L}(q_1)$

$(x, y) \in R_L$, but $(x, y) \notin E; (x, x) \in E \Rightarrow (x, x) \in R_L$

$$\text{Let } [u]_E = \overleftarrow{L}(q_0), [v]_E = \overleftarrow{L}(q_1),$$

验证: $([u]_E, [v]_E) \in R_L$

$$[u]_E \cdot \{0\} = \overleftarrow{L}(q_1) \notin L, [u]_E \cdot \{1\} = \overleftarrow{L}(q_2) \in L$$

$$[v]_E \cdot \{0\} = \overleftarrow{L}(q_0) \notin L, [v]_E \cdot \{1\} = \overleftarrow{L}(q_2) \in L$$

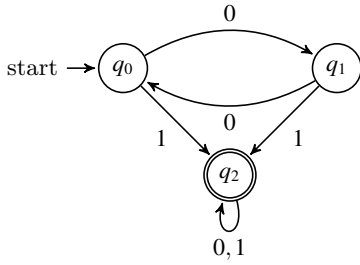


图 1.34: Equivalence classes of an equivalence relation

Definition 1.16 (Post-closure[Chrison2007]). Let $L \subseteq V^*$ and $s \in L$. Then the post-language of L after s , denoted by L/s , is the language

$$L/s := \{t \in V^* : st \in L\}$$

By definition, $L/s = \emptyset$ if $s \notin \bar{L}$.

Definition 1.17 (Left derivatives[WATSON93a]). Given language $A \subseteq V^*$ and $w \in V^*$ we define the left derivative of A with respect to w as:

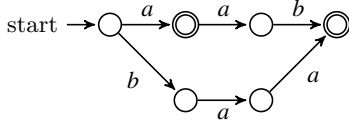
$$w^{-1}A = \{x \in V^* : wx \in A\}$$

A 关于 w 的左导数, 就是 A 中: $\{w$ 的后缀组成的字符串集合}。

Sometimes derivatives are written as $D_w A$ or as $\frac{dA}{dw}$. Right derivatives are analogously defined. Derivatives can also be extended to $B^{-1}A$ where B is also a language.

Example 1.4. $A = \{a, aab, baa\}, a^{-1}A = D_a A = \frac{dA}{da} = \{\varepsilon, ab, \emptyset\} = \{\varepsilon, ab\}$ □

$L = \{a, aab, baa\}$



$a^{-1}L = \{\varepsilon, ab, \emptyset\} = \{\varepsilon, ab\}$

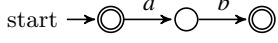
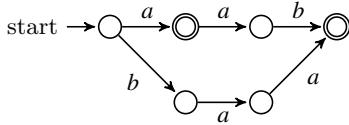


图 1.35: $a^{-1}L$

$L = \{a, aab, baa\}$



$V^{-1}L = \{\varepsilon, aa, ab\}, V \in \{a, b\}$

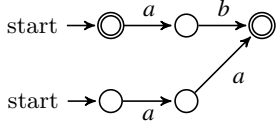


图 1.36: $V^{-1}L$

Example 1.5. $L = \{ba, baa, baab, ca\}, w = \{ba\},$

$$w^{-1}L = \{\varepsilon, a, ab, \emptyset\} = \{\varepsilon, a, ab\}$$

$$(wa)^{-1}L = (baa)^{-1}L = \{\emptyset, \varepsilon, b, \emptyset\} = \{\varepsilon, b\}$$

$$a^{-1}(w^{-1}L) = a^{-1}\{\varepsilon, a, ab\} = \{\emptyset, \varepsilon, b\} = \{\varepsilon, b\}$$

$$w \in L \equiv \varepsilon \in w^{-1}L, \text{ and } (wa)^{-1}L = a^{-1}(w^{-1}L)$$

□

Example 1.6. $a^{-1}\{a\} = \{\varepsilon\}; \quad a^{-1}\{b\} = \emptyset, \quad \Leftarrow \text{if } (a \neq b)$ □

Example 1.7. $L_0 = \{ab\}, L_1 = \{ac\}, L_0L_1 = \{abac\}$

$$a^{-1}(L_0L_1) = \{bac\}$$

$$a^{-1}(L_0L_1) = (a^{-1}L_0)L_1 \cup \emptyset \quad \Leftarrow (\varepsilon \notin L_0)$$

$$= \{b\}L_1 = \{bac\}$$

□

Example 1.8. $L_0 = \{\varepsilon, ab\}, L_1 = \{ac\}, L_0L_1 = \{ac, abac\}$

$$a^{-1}(L_0L_1) = \{c, bac\}$$

$$a^{-1}(L_0L_1) = (a^{-1}L_0)L_1 \cup a^{-1}L_1 \quad \Leftarrow (\varepsilon \in L_0)$$

$$= \{\emptyset, b\}L_1 \cup \{c\} = \{c, bac\}$$

□

证明. $a^{-1}(L_0L_1)$

$$1. \text{if } (\varepsilon \in L_0) \Rightarrow a^{-1}(L_0L_1) = (a^{-1}L_0)L_1 \cup a^{-1}L_1$$

$$\begin{aligned}
L_0 &= (L_0 \setminus \{\varepsilon\}) \cup \{\varepsilon\} \\
a^{-1}(L_0 L_1) &= a^{-1}(((L_0 \setminus \{\varepsilon\}) \cup \{\varepsilon\}) L_1) \\
&= a^{-1}(L_0 L_1 \cup L_1) \\
a^{-1}L_0 &= a^{-1}((L_0 \setminus \{\varepsilon\}) \cup \{\varepsilon\}) \\
&= a^{-1}(L_0 \setminus \{\varepsilon\}) \cup a^{-1}\{\varepsilon\} \\
&= a^{-1}L_0 \cup \emptyset = a^{-1}L_0
\end{aligned}$$

From [Hopcroft2008, p99]

(1) 如果 L 是一个语言, a 是一个符号, 则 L/a (称作 L 和 a 的商) 是所有满足如下条件的串 w 的集合: wa 属于 L 。例如, 如果 $L = \{a, aab, baa\}$, 则 $L/a = \{\epsilon, ba\}$, 证明: 如果 L 是正则的, 那么 L/a 也是。提示: 从 L 的 DFA 出发, 考虑接受状态的集合。

(2) 如果 L 是一个语言, a 是一个符号, 则 $a \setminus L$ 是所有满足如下条件的串 w 的集合: aw 属于 L 。例如, 如果 $L = \{a, aab, baa\}$, 则 $a \setminus L = \{\epsilon, ab\}$, 证明: 如果 L 是正则的, 那么 $a \setminus L$ 也是。提示: 记得正则语言在反转运算下是封闭的, 又由 (1) 知, 正则语言的商运算下是封闭的。

Definition 1.18 (Kleene-closure[Chrison2007]). Let $L \subseteq V^*$, then

$$L^* := \{\epsilon\} \cup L \cup LL \cup LLL \cup \dots$$

This is the same operation that we defined above for the set V , except that now it is applied to set L whose elements may be strings of length greater than one. An element of L^* is formed by the concatenation of a finite (but possibly arbitrarily large) number of elements of L ; this includes the concatenation of "zero" elements, that is the empty string ϵ . Note that $*$ operation is idempotent: $(L^*)^* = L^*$.

$$\begin{aligned} L^* &= \{\epsilon\} + L^+ \\ &= \{\epsilon\} \cup (L \setminus \{\epsilon\})L^* \\ &= \{\epsilon\} + L + LL + LLL + \dots \end{aligned}$$

1.11 Linear equation

see [Jean2018, 5.3,p64].

We give an algorithm to covert an automaton to a rational(regular) expression. The algorithm amounts to solving a system of linear equations on languages. We first consider an equation of the form

$$X = KX + L \tag{1.1}$$

Proposition 1.1 (Arden's Lemma). *if K does not contain the empty word, then $X = K^*L$ is the unique solution of the equation $X = KX + L$.*

where K and L are languages and X is the unknown. When K does not contain the empty word, the equation admits a unique solution.

证明. Replacing X by K^*L in the expression $KX + L$, one gets

$$K(K^*)L + L = K^+L + L = (K^+L + L) = K^*L,$$

and hence $X = K^*L$ is a solution of (1.1). see¹

1

$$\begin{aligned} K^* &= \{\epsilon\} + K^+ \\ &= \{\epsilon\} + (K \setminus \{\epsilon\})K^* \\ &= \{\epsilon\} + K + KK + KKK + \dots \end{aligned}$$

To Prove uniqueness, consider two solutions X_1 and X_2 of (1.1). By symmetry, it suffices to show that each word u of X_1 also belongs to X_2 . Let us prove this result by induction on the length of u .

If $|u| = 0$, u is the empty word² and if $u \in X_1 = KX_1 + L$, then necessarily $u \in L$ since $\varepsilon \notin K$. But in this case, $u \in KX_2 + L = X_2$. see³

For the induction step, consider a word u of X_1 of length $n + 1$. Since $X_1 = KX_1 + L$, u belongs either to L or to KX_1 . If $u \in L$, then $u \in KX_2 + L = X_2$. If $u \in KX_1$ then $u = kx$ for some $k \in K$ and $x \in X_1$. Since k is not the empty word, one has necessarily $|x| \leq n$ and hence by induction $x \in X_2$. [see⁴] It follows that $u \in KX_2$ and finally $u \in X_2$. This conclude the induction and the proof of the proposition. \square

From [Wonham2018, p74] The *length* $|s|$ of a string $s \in \Sigma^*$ is defined according to

$$|\varepsilon| = 0; |s| = k, \text{ if } s = \sigma_1 \sigma_2 \cdots \sigma_k \in \Sigma^+$$

Thus $|cat(s, t)| = |s| + |t|$.

A *language* over Σ is any subset of Σ^* , i.e. an element of the power set $Pwr(\Sigma^*)$; thus the definition includes both the empty language \emptyset , and Σ^* itself.

Note the distinction between \emptyset (the language with no strings) and ε (the string with no symbols). For instance the language $\{\varepsilon\}$ is nonempty, but contains only the empty string.

From [Wonham2018, p78]

Proposition 1.2 ([Wonham2018]).

1. If $L = M^*N$ then $L = ML + N$
2. If $\varepsilon \notin M$ then $L = ML + N$ implies $L = M^*N$ \square

Part(2) is Known as Arden's rule. Taken with Part(1) it says that if $\varepsilon \notin M$ then $L = M^*N$ is the unique solution of $L = ML + N$; in particular if $L = ML$ (with $\varepsilon \notin M$) then $L = \emptyset$

Exercise 1.1. Show by counterexample that the restriction $\varepsilon \notin M$ in Arden's rule cannot be dropped.

Solution 1.1. Examples text goes here.

Exercise 1.2. Prove Arden's rule. Hint: If $L = ML + N$ then for every $k \geq 0$

$$L = M^{k+1}L + (M^k + M^{k-1} + \cdots + M + \varepsilon)N$$

Solution 1.2.

² The empty word $= \varepsilon$, $|\varepsilon| = 0$; if a language $M = \{\varepsilon\}$, $|M| = 1$, The empty language $M = \emptyset$, $|M| = 0$. 文献 [Jean2018] 用 1 表示 ε , 因为 $\varepsilon K = K\varepsilon = K$, 因此, ε 是连接运算的单位元, 正是 1 表示的用意。0 表示 \emptyset , 它是并运算的单位元, $K \cup \emptyset = \emptyset \cup K = K$.

³ In this case, $|u| = 0$, $X = \{\varepsilon\}$, $|X| = 1$. i.e. $\varepsilon = K\varepsilon + L$, $\varepsilon = K + L$

⁴ $u = kx$, $|u| = |kx| = n + 1$, $\varepsilon \notin K$, $|k| \geq 1$, $|x| \leq n$, 由假设知, u 属于 X_1 , 归纳 $|x| = 0$, $|x| = 1, \dots, n$, $x \in X_2$.

Preliminaries :

$$M^* = M^k + M^{k-1} + \cdots + M^1 + M^0 \quad (k \geq 0)$$

$$= M^k + M^{k-1} + \cdots + M^1 + \varepsilon$$

$$= M^+ + \varepsilon$$

$$= MM^* + \varepsilon$$

$$= (M \setminus \{\varepsilon\})M^* + \varepsilon$$

$$M^+ = M^k + M^{k-1} + \cdots + M^1 \quad (k > 0)$$

$$= M(M^k + M^{k-1} + \cdots + M^1 + M^0)$$

$$= MM^*$$

$$M^0 = \{\varepsilon\} = 1$$

$$M\varepsilon = \varepsilon M = M$$

$$\varepsilon + \varepsilon = \varepsilon$$

$$M + M = M$$

证明.

$$L = ML + N \Rightarrow$$

$$M^0 L = M^1 L + M^0 N \quad (1.2)$$

$$M^1 L = M^2 L + M^1 N \quad (1.3)$$

$$M^2 L = M^3 L + M^2 N \quad (1.4)$$

$$\dots \quad (1.5)$$

\Rightarrow

$$(M^0 + M^1 + M^2 + \cdots)L = (M^1 + M^2 + M^3 + \cdots)L + (M^0 + M^1 + M^2 + \cdots)N$$

\Rightarrow

so, if $L = ML + N$, then for every $k \geq 0$

$$L = M^{k+1}L + (M^k + M^{k-1} + \cdots + M + M^0)N$$

\Rightarrow

$$L = M^{k+1}L + (M^k + M^{k-1} + \cdots + M + \varepsilon)N \quad (1.6)$$

(1) $k = 0$

$$L = ML + (\varepsilon)N = ML + N$$

$$\Rightarrow (1 - M)L = N$$

$$(\varepsilon - M)L = N$$

由于 $\varepsilon \notin M$, 左端不会消去 $\{\varepsilon\}$. 因此, 只能在 N 中找 L , 仅有唯一解: $L = \{\varepsilon\} = \{\text{empty word}\} \subseteq N$.

From [R.Su and Wonham2004, definition 2.3]

Definition 1.19. Let

$$G_A = (X_A, \Sigma, \xi_A, x_{A,0}, X_{A,m})$$

$$G_B = (X_B, \Sigma, \xi_B, x_{B,0}, X_{B,m})$$

G_B is a DES-epimorphic image(满射像) of G_A under DES-epimorphism $\theta : X_A \rightarrow X_B$ if

1. $\theta : X_A \rightarrow X_B$ is surjective(满射)
2. $\theta(x_{A,0}) = x_{B,0}$ and $\theta(X_{A,m}) = X_{B,m}$
3. $(\forall x \in X_A)(\forall \sigma \in \Sigma) \xi_A(x, \sigma)! \Rightarrow [\xi_B(\theta(x), \sigma)! \& \xi_B(\theta(x), \sigma) = \theta(\xi_A(x, \sigma))]$
4. $(\forall x \in X_B)(\forall \sigma \in \Sigma) \xi_B(x, \sigma)! \Rightarrow [(\exists x' \in X_A) \xi_A(x', \sigma)! \& \theta(x') = x]$

In particular, G_B is DES-isomorphic(同构) to G_A if $\theta : X_A \rightarrow X_B$ is bijective(双射).

see figure 1.37.

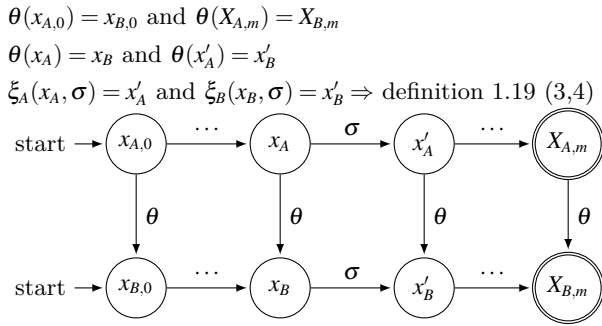


图 1.37: definition 1.19, G_B is a DES-epimorphic image(满射像) of G_A under DES-epimorphism $\theta : X_A \rightarrow X_B$

References

- Hopcroft2008. John E. Hopcroft, Rajeev Motwani, Jeffrey D. Ullman 著, 孙家骅等译, 自动机理论、语言和计算机导论, Third Edition, 机械工业出版社, 2008.7
- WATSON93a. WATSON, B. W. *A taxonomy of finite automata construction algorithms*, Computing Science Note 93/43, Eindhoven University of Technology, The Netherlands, 1993. Available by ftp from ftp.win.tue.nl in pub/techreports/pi.
- WATSON93b. WATSON, B. W. *A taxonomy of finite automata minimization algorithms*, Computing Science Note 93/44, Eindhoven University of Technology, The Netherlands, 1993. Available by ftp from ftp.win.tue.nl in pub/techreports/pi.
- WATSON94a. WATSON, B. W. *An introduction to the FIRE engine: A C++ toolkit for FInite automata and Regular Expressions*, Computing Science Note 94/21, Eindhoven University of Technology, The Netherlands, 1994. Available by ftp from ftp.win.tue.nl in pub/techreports/pi
- WATSON94b. WATSON, B.W. *The design. and implementation of the FIRE engine: A C++ toolkit for FInite automata and Regular Expressions*, Computing Science Note 94/22, Eindhoven University of Technology, The Netherlands, 1994. Available by ftp from ftp.win.tue.nl in pub/techreports/pi.
- Chrison2007. Christos G. Cassandras and Stéphane Lafortune, *Introduction to Discrete Event Systems*, Second Edition, New York, Springer, 2007
- Wonham2018. W. M. Wonham and Kai Cai, *Supervisory Control of Discrete-Event Systems*, Revised 2018.01.01
- Jean2018. Jean-Éric Pin, *Mathematical Foundations of Automata Theory*, Version of June 15, 2018

- 蒋宗礼 2013. 蒋宗礼, 姜守旭, 形式语言与自动机理论 (第 3 版), 清华大学出版社, 2013.05
- Lipschutz2007. S. Lipschutz and M. L. Lipson, *Schaum's Outline of Theory and Problems of Discrete Mathematics*, Third Edition, New York: McGraw-Hill, 2007.
- Rosen2007. K. H. Rosen, *Discrete Mathematics and Its Applications*, Seventh Edition, New York: McGraw-Hill, 2007.
- R.Su and Wonham2004. R. Su and W. M. Wonham, *Supervisor reduction for discrete-event systems*, Discrete Event Dyn. Syst., vol. 14, no. 1, pp. 31-53, Jan. 2004.
- 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*, Université Paris-Est Marne-la-Vallée 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.

Chapter 2

Finite automata minimization algorithms

2.1 Introduction

2.2 Brzozowski's algorithm

ε -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: $M_1 = (Q_1, V, T_1, \emptyset, S_1, F_1)$

$$\text{NFA: } M_1 \rightarrow \text{DFA: } M_2, M_2 = \text{suseful}_s \circ \text{subseopt}(M_1)$$

$$q_0, q_1 \in Q_1, Q_2 \subseteq \mathbb{P}(Q_1), \forall p \in Q_2, p = (q_0, q_1)$$

$$\vec{L}_{M_2}(p) = \vec{L}_{M_1}(q_0) \cup \vec{L}_{M_1}(q_1)$$

$$\Rightarrow$$

$$\vec{L}_{M_2}(p) = \bigcup_{q \in p} \vec{L}_{M_1}(q)$$

$$\Rightarrow$$

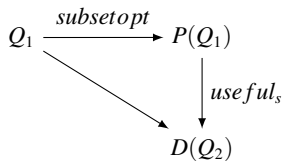
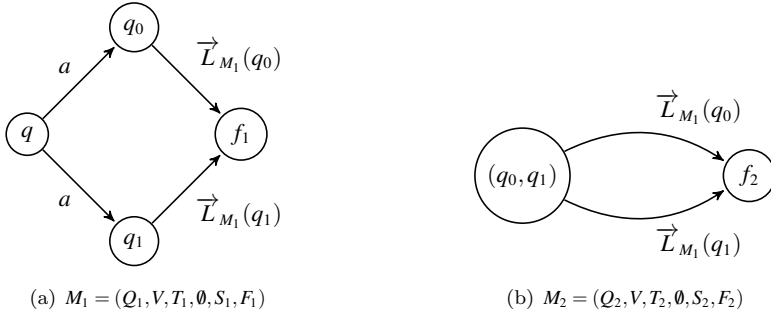
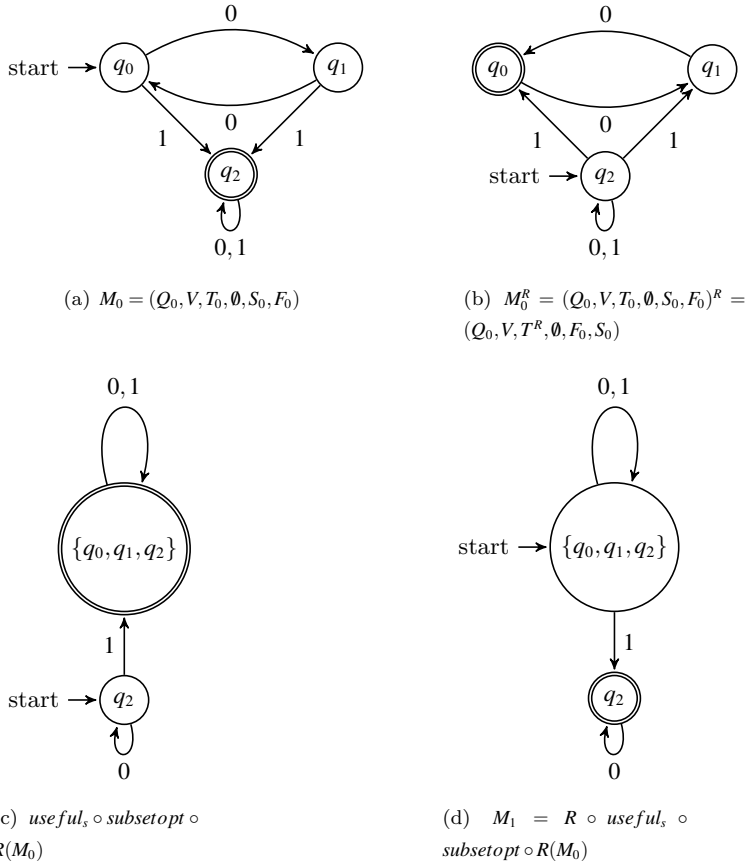


图 2.1: $M_2 = \text{suseful}_s \circ \text{subseopt}(M_1)$

2.3 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

图 2.2: $M_2 = \text{suseful}_s \circ \text{subseopt}(M_1)$ 图 2.3: $M_1 = R \circ \text{useful}_s \circ \text{subseopt} \circ R(M_0)$

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

start: $U = \{q_2\}, D = \emptyset$

$u = q_2 : T(q_2, 0) = \{q_2\}, T(q_2, 1) = \{q_0, q_1, q_2\}$

add new state to D , $D = \{q_2, \{q_0, q_1, q_2\}\}$

$u = \{q_0, q_1, q_2\} : T(\{q_0, q_1, q_2\}, 0) = T(q_0, 0) \cup T(q_1, 0) \cup T(q_2, 0) = \{q_1\} \cup \{q_0\} \cup \{q_2\} = \{q_0, q_1, q_2\}$

$T(\{q_0, q_1, q_2\}, 1) = T(q_0, 1) \cup T(q_1, 1) \cup T(q_2, 1) = \emptyset \cup \emptyset \cup \{q_0, q_1, q_2\} = \{q_0, q_1, q_2\}$

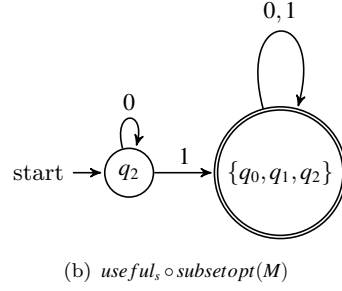
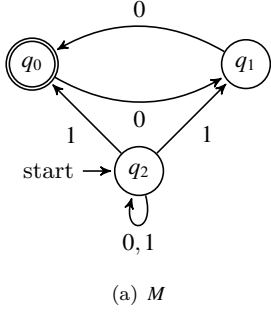


图 2.4: $usefals \circ subsetopt(M)$

Equivalence relation $E \subseteq Q \times Q$

$(p, q) \in E \equiv (\vec{L}(p) = \vec{L}(q))$

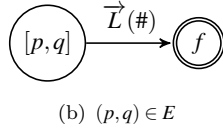
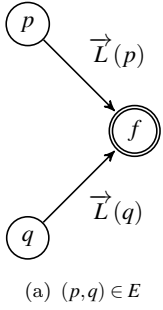


图 2.5: Equivalence relation $E \subseteq Q \times Q$

$\vec{L}(p) = \bigcup_{a \in V} (\{a\} \cdot \vec{L}(T(p, a)) \cup \{\epsilon \mid p \in F\})$

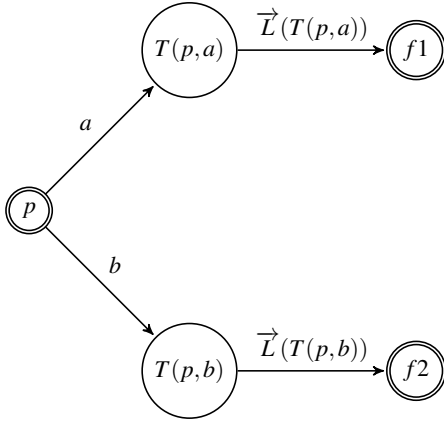


图 2.6: $L(p)$

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: $((T(p,a), T(q,a)) \notin E_1(part_1)$;

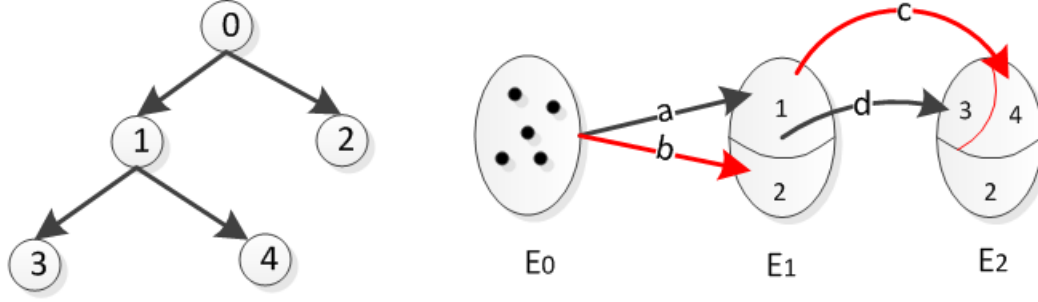


图 2.7: Approximating $E, E_0 = (Q \setminus F)^2 \cup F^2$

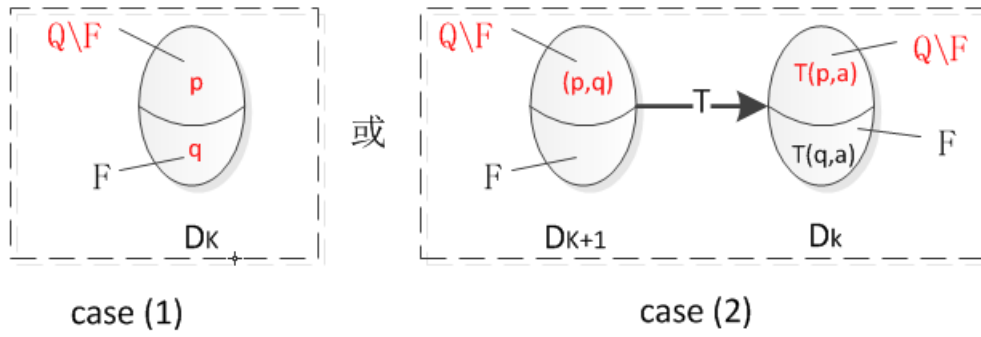


图 2.8: Approximating $D, D_0 = ((Q \setminus F) \times F) \cup (F \times (Q \setminus F))$

initial partition, $P = [Q]_{E_0} = \{F, Q \setminus F\}, Q_0, Q_1 \in P$,

if $(\exists p, q \in Q_0, T(p,a) \in Q_1, \text{ and } T(q,a) \notin Q_1)$, then

split Q_0 wrt (Q_1, a) into two parts, (1): $Q'_0, T(p,a) \in Q_1$ and (2): $(Q_0 \setminus Q'_0), T(p,a) \notin Q_1$

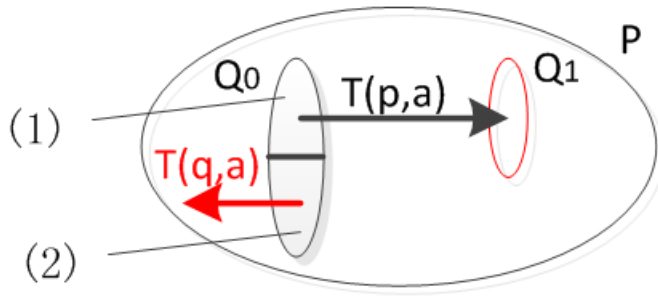


图 2.9: split Q_0 wrt Q_1

2.4 From [Gries73]

2.4.1 Problem Definition

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

Definition 2.2 (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 2.1. 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. 2.10.

$\{a, b\}, \{d, e\}$ is not equivalent states.

Sets of equivalent states: $\{a, c\}, \{b\}, \{d\}, \{e\}$

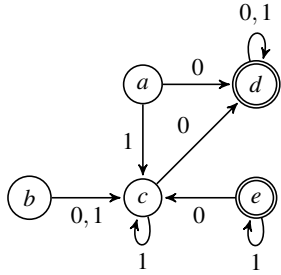


图 2.10: Finite state automaton

2.4.2 The Basic Algorithm

Definition 2.3 (acceptable partition). A partitioning of the states into blocks B_1, B_2, \dots, 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 2.1. The partitioning $B_1 = F, B_2 = S - F$ is acceptable.

Lemma 2.2. The partitioning B_1, B_2, \dots, 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 $a \in I$

$$s, t \in B_i, \delta(s, a) \in B_j \text{ implies } \delta(t, a) \in B_j \quad (2.1)$$

Lemma 2.3. Let B_1, B_2, \dots, B_p be an acceptable partitioning. Suppose there are two blocks B_i, B_j and symbol $a \in I$ such that

$$s, t \in B_i, \delta(s, a) \in B_j \text{ but } \delta(t, a) \notin B_j \quad (2.2)$$

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

$$\{s \in B_i \mid \delta(s, a) \in B_j\} \text{ and } \{s \in B_i \mid \delta(s, a) \notin B_j\} \quad (2.3)$$

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. (2.2) holds do
    SPLIT: split  $B_i$  wrt  $(B_j, a)$ 
end while

```

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

$$P \wedge B\{S\}P \text{ implies } P \{ \text{While } B \text{ do } S \text{ end} \} P \wedge \neg B \quad (2.4)$$

For algorithm (1), let P be the relation “the partitioning is acceptable”. By Lemma 2.1, P is true just before execution of the while loop, while by Lemma 2.3 execution of S always yields an acceptable partitioning. The relation B is “ $\exists a, B_i, B_j$ such that Eq. (2.2) holds”. Thus, P and $\neg B$ hold after execution of the loop, but these are the sufficient requirements described in Lemma 2.2 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. (2.2) 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:

$$\begin{aligned}
 & \text{(a) for all } s \in B \quad \delta(s, a) \in B_j, \text{ or} \\
 & \text{(b) for all } s \in B \quad \delta(s, a) \notin B_j
 \end{aligned} \quad (2.5)$$

Lemma 2.4. *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 2.5. *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:

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

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:

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

This is precisely what splitting all blocks wrt (\tilde{B}_j, a) accomplishes (see (2.5)). 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}_j, a) accomplishes the task of splitting wrt (\bar{B}_j, a) . \square

Lemma 2.6. *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 2.5, with $B_j = S, \bar{B}_j = F, \text{ and } \tilde{B}_j = S - F$. 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. \square

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 2.5(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 2.5(a) or (b) already holds, or we are assured by other means that either 2.5(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:

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

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
     $D \leftarrow D \cup \delta^{-1}(s, a)$ 
  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.

2.5 From [Hopcroft71]

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

Example 2.2. $Q = \{1, 2, 3, 4, 5, 6\}, V = \{0, 1\}, T$ see Fig. 3.3

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 $T^{-1}(s, a) = \{t | T(t, a) = s\}$
 $T^{-1}(1, 0) = \emptyset, T^{-1}(2, 0) = \{1\}, T^{-1}(3, 0) = \{2\}, \dots, T^{-1}(6, 0) = \{5\}$
 $T^{-1}(1, 1) = \{1\}, T^{-1}(2, 1) = \{2\}, \dots, T^{-1}(6, 1) = \{6\},$
- 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 $L(a)$
 $L(0) = \{6\}, \quad \text{since } |\hat{B}(B(1), 0)| = 1 \leq |\hat{B}(B(2), 0)| = 4.$
 $L(1) = \{6\}, \quad \text{since } |\hat{B}(B(1), 1)| = 1 \leq |\hat{B}(B(2), 1)| = 5.$
- 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

$$T^{-1}(s, a) = \{t \mid T(t, a) = s\} \quad \text{计算状态 } s \text{ 的 in-transitionss}$$

 Step 2. construct $B(1) = F, B(2) = Q - F$ and for each $a \in V$ and $1 \leq i \leq 2$ construct

for each $a \in V$ **do**

 for $i = 1; i < n; i++$ **do**

$$\hat{B}(B(i), a) = \{s \mid s \in B(i) \text{ and } T^{-1}(s, a) \neq \emptyset\};$$

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 $|\hat{B}(B(1), a)| \leq |\hat{B}(B(2), a)|$ **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

$$B'(j) = \{t \mid T(t, a) \in \hat{B}(B(i), a)\} \text{ and}$$

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

 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 $j \notin L(a) \& 0 < |\hat{B}(B(j), a)| \leq |\hat{B}(B(k), a)|$ **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.

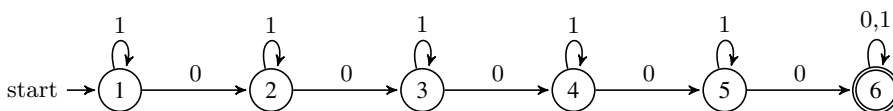


图 2.11: Minimizing example

Example 2.3. 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. (3.5).

$\{a, b\}$ 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: $\{a, c\}, \{b\}, \{d\}, \{e\}$

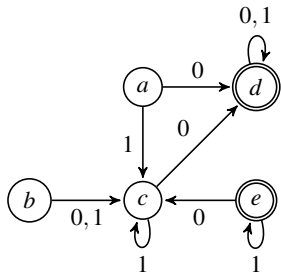


图 2.12: Finite state automaton

2.6 From [Ratnesh95]

FSM: Finite State Machine

NFSM: Non-deterministic Finite State Machine without ε -moves

DFSM: Deterministic Finite State Machine

Definition 2.4 (prefix closure of K). The prefix closure of K , denoted $pr(K) \subseteq \Sigma^*$, is the language

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

Example 2.4 (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:

$$arrival \cdot departure \cdot arrival \cdot departure \dots,$$

where “ \cdot ” denotes the operation of concatenation. □

Example 2.5 (Generated language). Consider the buffer of Example 2.4. 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 2.6 (language model). Consider the buffer of Examples 2.4 and 2.5 with language model $[(ad)^*, pr((ad)^*)]$. The directed graph shown in Figure 2.13 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.) □

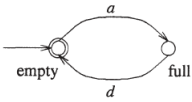


图 2.13: Graph representing a DSM

Example 2.7 (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 2.14. 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 in Figure 2.14.

Note 2.1. 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。

□

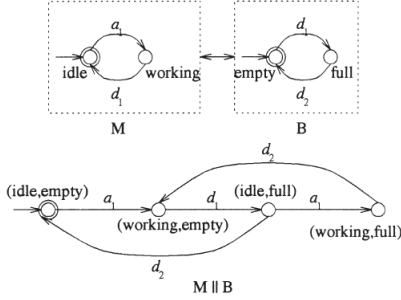
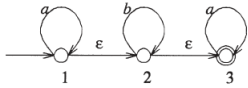


图 2.14: Diagram illustrating synchronous composition of DFMSs

Example 2.8 (ϵ -NSM). Consider the ϵ -NSM of Fig. 2.15 Then $\epsilon_G^*(1) = \{1, 2, 3\}$, $\epsilon_G^*(2) = \{2, 3\}$, $\epsilon_G^*(3) = \{3\}$.

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

图 2.15: Diagram illustrating an ϵ -NSM

Example 2.9 (Completion and reverse). Completion and reverse of the DSM of Figure 2.17(b) are shown in Figure 2.16 (a) and 2.16 (b) respectively. The state labels have been changed.

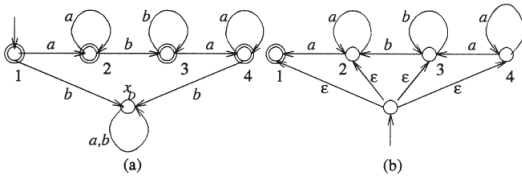


图 2.16: Diagram illustrating completion and reverse operations

Theorem 2.1 (power set construction). Let $G := (X, \Sigma, \alpha, x_0, X_m)$ be a NFSM. Then there exists a language model equivalent DFSM $\mathcal{G} := (\mathcal{X}, \Sigma, \hat{\alpha}, \{x_0\}, \mathcal{X}_m)$ and $L(\mathcal{G}) = L(G)$

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

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

Then it is easily shown that $(L_m(\mathcal{G}), L(\mathcal{G})) = (L_m(G), L(G))$

Example 2.10. (NFSM to DFSM) Consider the NFSM $G := (X, \Sigma, \alpha, x_0, X_m)$ shown in Figure 2.17(a). The language equivalent DFSM $\mathcal{G} := (\mathcal{X}, \Sigma, \hat{\alpha}, \{x_0\}, \mathcal{X}_m)$ obtained using the power set construction is shown in Figure 2.17(b).

Note that $\mathcal{X} = \mathcal{X}_m = \{\{1\}, \{1, 2, 3\}, \{2, 3\}, \{3\}\}$, as $X_m = \{1, 3\}$ which has a nonempty intersection with each state in \mathcal{X} ;

$$\begin{aligned} \hat{\alpha}(\{1\}, a) &= \alpha(1, a) = \{1, 2, 3\}; \\ \hat{\alpha}(\{1, 2, 3\}, a) &= \alpha(1, a) \cup \alpha(2, a) \cup \alpha(3, a) = \{1, 2, 3\}; \\ \hat{\alpha}(\{1, 2, 3\}, b) &= \alpha(1, b) \cup \alpha(2, b) \cup \alpha(3, b) = \{2, 3\}; \\ &\text{etc.} \end{aligned}$$

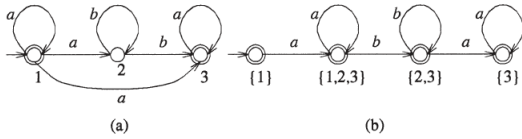


图 2.17: Diagram illustrating NFSM to DFSM conversion

Remark 2.1. It follows from Theorems 2.1 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 2.18(a); corresponding DFSM obtained using the construction outlined in Theorem 2.1 is shown in Figure 2.18(b).

2.6.1 Myhill-Nerode Characterization

Definition 2.5 (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 .

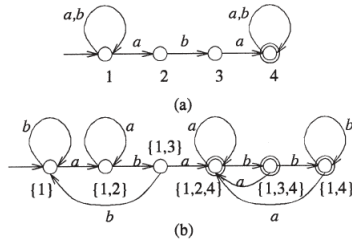


图 2.18: NFSM accepting strings with aba as a substring, and corresponding DFSM

Definition 2.6 (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 2.2. 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 2.2 (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) \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: $\hat{X} := \{[s](R_K) | s \in \Sigma^*\}$; $\hat{x}_0 := [\varepsilon](R_K)$; $\hat{X}_m = \{[s](R_K) | s \in K\}$; 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 $\hat{\alpha}(\hat{x}_0, s) = [s](R_K) \in \hat{X}_m$, 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. \square

The construction of the DFSM G in the proof of Theorem 2.2 is known as the Myhill-Nerode construction. The following example illustrates such a construction.

Example 2.11 (equivalence classes). Consider for example the marked language $K_m = (ad)^*$ of the buffer of capacity one of Example 2.5. 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 2.2 that R_{K_m} is of finite index. It can be easily verified that

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

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 $\hat{X} = \{[\varepsilon](R_{K_m}), [a](R_{K_m}), [d](R_{K_m})\}$; $\hat{x}_0 = [\varepsilon](R_{K_m})$; $\hat{X}_m = \{[\varepsilon](R_{K_m})\}$; and

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

See figure 2.19, State e: empty, $[\varepsilon](R_{K_m})$; State f: full, $[a](R_{K_m})$; State t: dump/trap, $[d](R_{K_m})$. \square

Remark 2.2. 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 $[s](R_K)$ from \hat{G} (and all transitions leading into/out of it), where \hat{G} is the DFSM in the

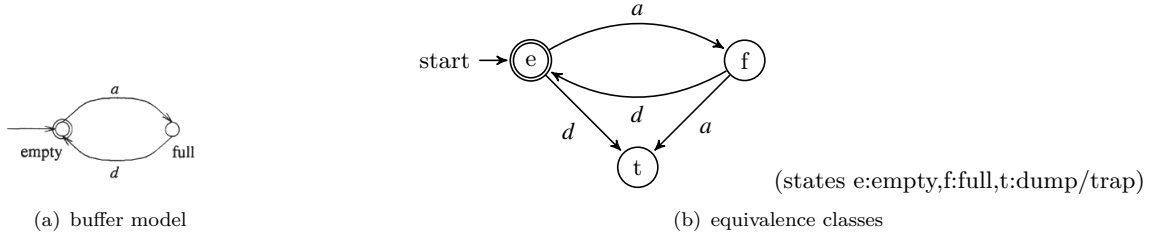


图 2.19: equivalence classes

proof of Theorem 2.2 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 .

2.6.2 Minimization

Theorem 2.3 (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 minimal 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 DFSM 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

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

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 2.3 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 2.3. Note that if $K = \Sigma^*$, then it can be accepted by a minimal DFSM having a single state.

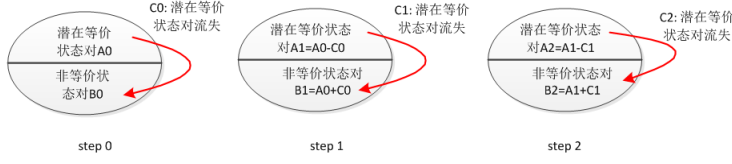


图 2.20: Diagram illustrating algorithm minimize

Algorithm 8 The algorithm for a minimal DFSM, see Figure 2.20

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:

$A_0 := [X_m \times X_m] \cup [(\bar{X} - X_m) \times (\bar{X} - X_m)] = \{(\text{final 状态对}) \cup (\text{非 final 状态对})\} = \{\text{潜在的等价状态对}\};$

$B_0 := [\bar{X} \times \bar{X} - A_0] = \{(\text{全体状态对}) - A_0\} = \{(\text{final 状态}, \text{非 final 状态})\} = \{\text{非等价状态对}\} = \{\text{非 } A_0 \text{ 状态对}\};$

$k := 0.$

2: Iteration step:

$$\begin{aligned} C_k &:= \{(x, x') \in A_k \mid \exists \sigma \in \Sigma \text{ s.t. } (\alpha(x, \sigma), \alpha(x', \sigma)) \in B_k\} \\ &= \{A_0 \text{ 中两大类中的 (状态对) 存在字母 } \sigma \text{ 进入 } B_0 \text{ 类}\} \\ &= \{\text{潜在状态对} \rightarrow \text{非等价状态对}\} \end{aligned}$$

$$A_{k+1} := A_k - C_k$$

$$\begin{aligned} B_{k+1} &:= B_k \cup C_k = [\bar{X} \times \bar{X}] - A_{k+1} \\ &= \{\text{非 } A_{k+1} \text{ 状态对}\} \end{aligned}$$

3: Termination step:

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

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 2.12. Consider the complete DFSM of Figure 2.21. 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:

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

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 2.21, which does not contain the dump state.

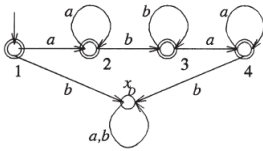


图 2.21: the minimal DFSM is not contain the dump state

2.7 From wiki – Partition of a set

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

Definition 2.7 (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 2.13 (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 2.8 (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 2.9 (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 2.22.

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.

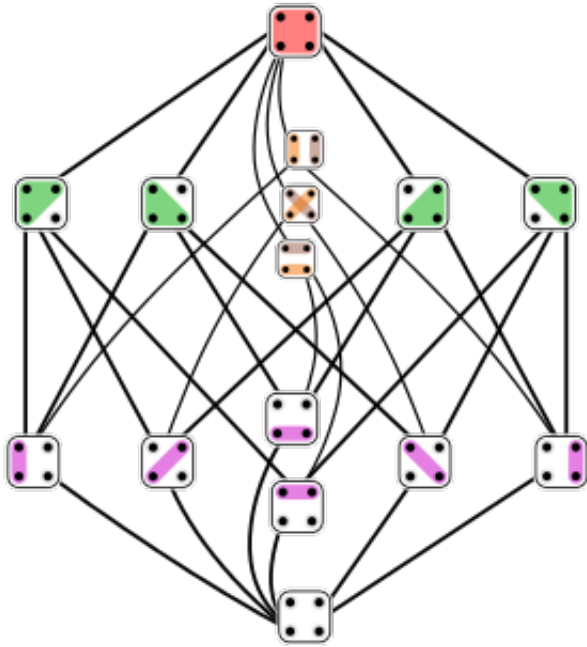


图 2.22: Partitions of a 4-set ordered by refinement

2.8 From [Kenneth2012]

Definition 2.10 (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 2.11 (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 2.4. 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 2.14. 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 2.15 (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 2.16. 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 2.17. 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 2.12 (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 2.18. 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。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 2.19. 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

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

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

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

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

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

例如,

$$[0]_4 = \{\dots, -8, -4, 0, 4, 8, \dots\}$$

$$[1]_4 = \{\dots, -7, -3, 1, 5, 9, \dots\}$$

Example 2.20. 对于例2.17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:

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

Theorem 2.4. 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2.4, 这些等价类或者是相等的或者是不相交的, 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, 一族子集 $A_i, i \in I$, (其中 I 是下标的集合) 构成 S 的划分, 当且仅当

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

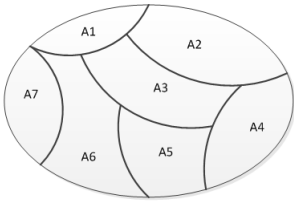


图 2.23: Partition of set

Example 2.21. 假设 $S = \{1, 2, 3, 4, 5, 6\}$, 一族集合 $A_1 = \{1, 2, 3\}, A_2 = \{4, 5\}, A_3 = \{6\}$ 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 2.5. 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 2.22. 假设 $S = \{1, 2, 3, 4, 5, 6\}$, 一族集合 $A_1 = \{1, 2, 3\}, A_2 = \{4, 5\}, A_3 = \{6\}$ 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 2.1. 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 2.23. 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

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

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:

$$[0]_4 = \{\dots, -8, -4, 0, 4, 8, \dots\}$$

$$[1]_4 = \{\dots, -7, -3, 1, 5, 9, \dots\}$$

$$[2]_4 = \{\dots, -6, -2, 2, 6, 10, \dots\}$$

$$[3]_4 = \{\dots, -5, -1, 3, 7, 11, \dots\}$$

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 2.24. 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

$$[011]_{R_3} = \{011, 0110, 0111, 01100, 01101, 011111, \dots\}$$

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 $[\lambda]_{R_3} = \{\lambda\}$

$$[0]_{R_3} = \{0\}, [1]_{R_3} = \{1\},$$

$$[00]_{R_3} = \{00\}, [01]_{R_3} = \{01\}, [10]_{R_3} = \{10\}, [11]_{R_3} = \{11\},$$

(2) 每个长度大于 3 的位串必须和以下 8 个位串之一等价:

$$[000]_{R_3} = \{000, 0000, 0001, 00000, 00001, 00010, 00011, \dots\}$$

$$[001]_{R_3} = \{001, 0010, 0011, 00100, 00101, 00110, 00111, \dots\}$$

$$[010]_{R_3} = \{010, 0100, 0101, 01000, 01001, 01010, 01011, \dots\}$$

$$[011]_{R_3} = \{011, 0110, 0111, 01100, 01101, 01110, 01111, \dots\}$$

$$[100]_{R_3} = \{100, 1000, 1001, 10000, 10001, 10010, 10011, \dots\}$$

$$[101]_{R_3} = \{101, 1010, 1011, 10100, 10101, 10110, 10111, \dots\}$$

$$[110]_{R_3} = \{110, 1100, 1101, 11000, 11001, 11010, 11011, \dots\}$$

$$[111]_{R_3} = \{111, 1110, 1111, 11100, 11101, 11110, 11111, \dots\}$$

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。

2.9 From [Jean2011]

Definition 2.13 (Partitions and equivalence relations). A partition of a set E is a family P of nonempty, pairwise disjoint subsets of E such that $E = \bigcup_{\mathcal{P} \in P} \mathcal{P}$. 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 $\mathcal{P} \cap \mathcal{Q}$, for $\mathcal{P} \in P$ and $\mathcal{Q} \in Q$. The notation is extended to a set of partitions in the usual way: we write $P = P_1 \wedge \cdots \wedge P_n$ for the common refinement of P_1, \dots, 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 $\mathcal{P} \cap F$, for $\mathcal{P} \in P$. 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

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

Definition 2.14 (Minimal automaton). We consider a deterministic automaton $\mathcal{A} = (Q, i, F)$ 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 \mathcal{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 $L_q(\mathcal{A})$ which is the set of words recognized by the subautomaton rooted at q , that is

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

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 $\{w \in A^* \mid i \cdot w = q\}$. The automaton \mathcal{A} is *minimal* if $L_p(\mathcal{A}) \neq L_q(\mathcal{A})$ for each pair of distinct states p, q . The equivalence relation \equiv defined by

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

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 \mathcal{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 2.15 (Partitions and automata). Again, we fix a deterministic automaton $\mathcal{A} = (Q, i, F)$ 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

$$(P, a) \mid R$$

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

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

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

$$(P, a)|R = (P^c, a)|R \quad (2.6)$$

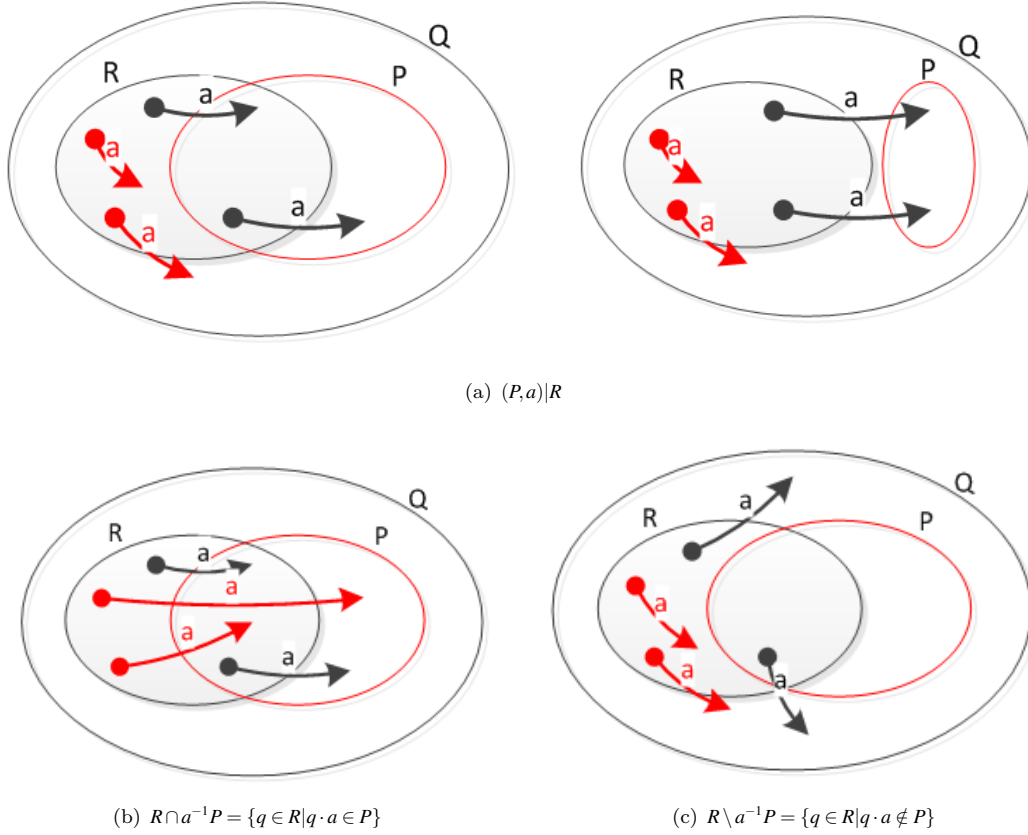


图 2.24: $(P, a)|R = (R \cap a^{-1}P) \cup (R \setminus a^{-1}P)$

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

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

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

Lemma 2.7. *Let P be a set of states, and let $\mathcal{P} = \{P_1, P_2\}$ 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 consequently

$$(P, a)|R \geq (P_1, a)|R \wedge (P_2, a)|R, \quad (2.7)$$

$$(P_1, a)|R \geq (P, a)|R \wedge (P_2, a)|R \quad (2.8)$$

Example 2.25. We consider the automata given in Figure 2.25 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.

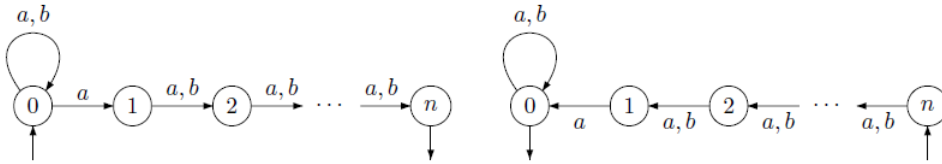


图 2.25: The automaton on the left recognizing the language A^*aA^n . It has $n+1$ states and the minimal deterministic automaton for this language has 2^n states. The automaton on the right is its reversal. It is minimal and recognizes $A^n a A^*$.

2.9.1 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 2.16 (Moore equivalence of order h). Let $\mathcal{A} = (Q, i, F)$ be an automaton over an alphabet A . Define, for $q \in Q$ and $h \geq 0$, the set

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

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

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

Algorithm 9 Moore's minimization algorithm**Input:** $\mathcal{A} = (Q, i, F)$ **Output:** The equivalence classes of Q $\hat{P} \leftarrow \{F, F^c\}$ \triangleright The initial partition**repeat** $\hat{P}' \leftarrow \hat{P}$ $\triangleright \hat{P}'$ 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 with n elements can be done in time $O(kn^2)$ by brute force. A radix sort improves the running time to $O(kn)$. With $k = \text{Card}(A)$, 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 $l = n^2$. 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.

2.9.2 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.

2.9.2.1 Outline

The algorithm is outlined in the function HOPCROFT given in algorithm 15. 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 \mathcal{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 $\hat{P} = P_1, \dots, P_n$ 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:** $\mathcal{A} = (Q, i, F)$ **Output:** The equivalence classes of Q

```

1:  $\hat{P} \leftarrow \{F, F^c\}$   $\triangleright$  The initial partition
2:  $W \leftarrow \emptyset$   $\triangleright$  The waiting set
3: for all  $a \in A$  do
4:    $ADD((\min(F, F^c), a), W)$   $\triangleright$  initialization of the waiting set
5: end for
6: while  $W \neq \emptyset$  do
7:    $(W, a) \leftarrow TakeSome(W)$   $\triangleright$  Take and remove some splitter
8:   for all  $P \in \hat{P}$  do which is split by  $(W, a)$ 
9:      $P', P'' \leftarrow (W, a)|P$   $\triangleright$  Compute the split
10:    Replace  $P$  by  $P'$  and  $P''$  in  $\hat{P}$   $\triangleright$  Refine the partition
11:   end for
12:   for all  $b \in A$  do  $\triangleright$  Update the waiting set
13:     if  $(P, b) \in W$  then
14:       Replace  $(P, b)$  by  $(P', b)$  and  $(P'', b)$  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 $O(|A|n \log n)$.

2.10 From [Knuutila2001]

2.10.1 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 apb . 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. $a\rho = \{b \in B | apb\}$. The converse of ρ is the relation $\rho^{-1} = \{(b, a) | apb\}$. Obviously $b\rho^{-1} = \{a | apb\}$.

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

$$\begin{aligned}\rho^0 &= \omega_A \\ \rho^{n+1} &= \rho^n \circ \rho, \quad n \geq 0.\end{aligned}$$

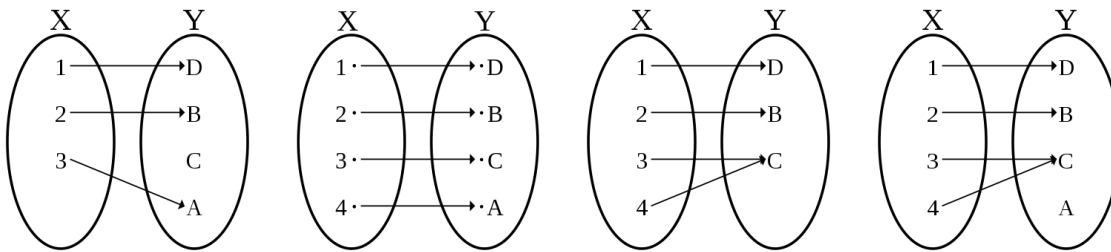
The relation ρ is called reflexive if $\omega_A \subseteq \rho$; symmetric if $\rho^{-1} \subseteq \rho$; and transitive if $\rho^2 \subseteq \rho$.

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 ρ 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* $\phi : A \rightarrow B$ 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 $\phi : A \rightarrow B$ to a set $C \subseteq A$ is the mapping $\phi|C : C \rightarrow B$ where $\phi|C = \phi \cap (C \times B)$.

The *composition* of two mappings $\phi : A \rightarrow B$ and $\psi : B \rightarrow C$ is the mapping $\phi\psi : A \rightarrow C$, where $\phi\psi$ is the product of ϕ and ψ as relations. The *kernel* $\phi\phi^{-1}$ of a mapping $\phi : A \rightarrow B$, 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 $\phi : A \rightarrow B$ 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-surjective (b) An injective surjective (c) A non-injective surjective (d) A non-injective non-function (injection, not a bijection) (bijection) function (surjection, not a bijection) surjective function (also not a bijection)

图 2.26: 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

2.10.2 Preliminaries

Definition 2.17 ($a\rho_G b$). 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 2.18 ($\omega_Q, \iota_Q, Eq(Q)$). 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 2.19 (the reduced G). DFA G is reduced, if $a\rho_G b$ implies $a = b$, i.e. $\rho_G = \omega_Q$

Definition 2.20 ($Con(G)$). 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 2.21 (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 2.22 (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 $w = x_1 x_2 \dots x_n$ is accepted by G if and only if $x_n(\dots(x_2(x_1(q_0)))\dots) \in F$.

2.10.3 The classical algorithm

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

Proposition 2.2. 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:

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

Then the following hold.

- (1) $\rho_0 \supseteq \rho_1 \supseteq \dots$
- (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 2.2 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 2.26. We construct (the worst kind of) a DFA Fig. 2.27.

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

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.

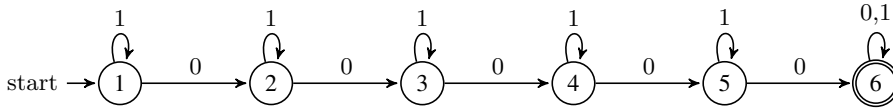


图 2.27: Minimizing example

2.10.4 Atomic refinements

Note that the construction of Proposition 2.2 reaches the situation $\rho_i = \rho_{i+1}$ when ρ_i becomes a congruence, i.e.

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

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

$$\begin{aligned}
\rho_i \neq \rho_{i+1} &\Leftrightarrow (\exists a, b \in Q, x \in \Sigma) & (a, b) \in \rho_i \text{ and } (\delta(a, x), \delta(b, x)) \notin \rho_i \\
&\Leftrightarrow (\exists B \in Q/\rho_i, x \in \Sigma) & (a, b) \in B \text{ and } (\delta(a, x), \delta(b, x)) \notin \rho_i \\
&\Leftrightarrow (\exists B, C \in Q/\rho_i, x \in \Sigma) & (a, b) \in B \text{ and } \delta(a, x) \in C, \text{ and } \delta(b, x) \notin C \\
&\Leftrightarrow (\exists B, C \in Q/\rho_i, x \in \Sigma) & (a, b) \in B \text{ and } \delta(B, x) \cap C \neq \emptyset, \text{ and } \delta(B, x) \not\subseteq C
\end{aligned}$$

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

$$\delta(B, x) \cap C \neq \emptyset \text{ and } \delta(B, x) \not\subseteq C \quad (2.9)$$

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 2.2 in the terms of these atomic refinements.

Proposition 2.3. *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:*

$$\begin{aligned}
Q/\theta_0 &= \{F, Q - F\}, \\
Q/\theta_{i+1} &= \begin{cases} (Q/\theta_i - \{B\}) \cup \{B_{C,x}, B^{C,x}\} & \text{if 2.9 holds for some } B, C \in Q/\theta_i, x \in \Sigma, \\ Q/\theta_i & \text{otherwise.} \end{cases}
\end{aligned}$$

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. (2.9) 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 $(a, b) \notin \theta_{i+1}$, 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+1} only when they are shown to be inequivalent in ρ_G , formally $(a, b) \notin \theta_{i+1} \implies (a, b) \notin \rho_G$.

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

Note that the construction given in Proposition 2.3 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. (2.9) holds.

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

$$ref(B, \theta) = \{(C, x) \in (Q/\theta) \times \Sigma \mid x(B) \cap C \neq \emptyset \text{ and } x(B) \not\subseteq C\}$$

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 Computing ρ_G using atomic refinements

```

1: function EQUIVALENCE( $G$ )
2:    $Q/\theta \leftarrow \{F, Q - F\}$ 
3:   while  $(\exists B, C \in Q/\theta, x \in \Sigma \text{ s.t. Eq. 2.9 holds})$  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  $(\exists \text{some}(C, x) \in (Q/\theta) \times \Sigma \text{ with } \text{obj}(C, x, \theta) \neq \emptyset)$  do
4:     for  $B \in \text{obj}(C, x, \theta)$  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

```

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

Lemma 2.8. *Let $G = (Q, \Sigma, \delta, q_0, F)$, $\theta \in \text{Eq}(Q)$ 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 consider 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 2.8.
- 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 2.3, 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  $B \in \text{obj}(C, x, \theta)$  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 2.9. *Let $G = (Q, \Sigma, \delta, q_0, F)$, $\theta \in \text{Eq}(Q)$ 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'', y) 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. 2.28, 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.

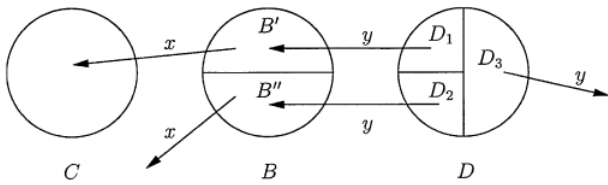


图 2.28: Illustration of Lemma 2.9

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*, Université Paris-Est Marne-la-Vallée 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.

Chapter 3

Hopcroft's algorithm

An automaton is *accessible* when for any state $p \in Q$, there exists a word $w \in V^*$ such that $q_0 \cdot w = p$. It is *co-accessible* when for any state $p \in Q$, there exists $w \in V^*$ such that $p \cdot w \in F$.

3.0.1 algorithm

Member function `min_Hopcroft` implements Hopcroft's $n \log n$ minimization algorithm, as presented in [[WATSON94b], Algorithm 4.8].

The combination of the out-transitions of all of the States is stored in a **CRSet** C .

Set L from the abstract algorithm is implemented as a mapping from States to int (an array of int is used).

Array L should be interpreted as follows: if State q a representative, then the following pairs still require processing (are still in abstract set L):

$$([q], C_0), \dots, ([q], C_{L(q)-1})$$

The remaining pairs do not require processing:

$$([q], C_{L(q)}), \dots, ([q], C_{|C|-1})$$

This implementation facilitates quick scanning of L for the next valid State-CharRange pair.

3.0.2 Minimization example 1

CRSet C; // the out labels of State's: 'a' 'b'

int L[9]; // the index of L = q: 对应等价类 [q]; L[q] 表示正在处理等价类 [q] 的字符在 C 中的 index。

$L = \{0, 0, 0, 0, 0, 0, 0, 0, 0\}$

DFA

$Q = [0, 9)$

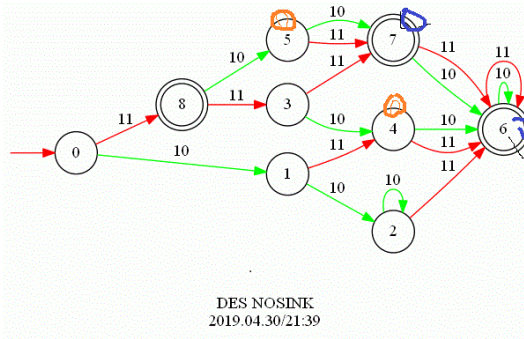
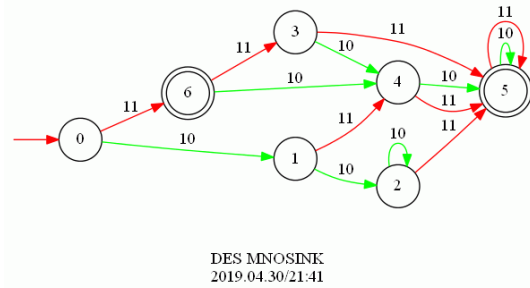
$S = \{ 0 \}$

Algorithm 14 Hopcroft's minimization algorithm**Input:** $G = (Q, V, T, q_0, F)$ **Output:** The equivalence classes of Q

```

1:  $P \leftarrow [Q]_{E_0} = \{F, Q \setminus F\}$   $\triangleright$  The initial partitions is  $[Q]_{E_0}$ , it's the total euivalence relation.
2:  $L \leftarrow \emptyset$   $\triangleright$  The waiting set
3: for all  $a \in V$  do
4:    $ADD((\min(F, Q \setminus F), a), L)$   $\triangleright$  initialization of the waiting set
5: end for
6: while  $L \neq \emptyset$  do
7:    $P_{old} = P$ ;
8:    $(Q_1, a) \leftarrow TakeSome(L)$   $\triangleright$  Take and remove some splitter
9:    $L = L \setminus \{(Q_1, a)\}$ ;
10:  for all  $Q_0 \in P_{old}$  do
11:     $Q_0$  is split by  $(Q_1, a)$   $\triangleright$  Compute the split,  $Q_0$  is splitted into  $Q'_0$  and  $Q''_0$ 
12:     $Q'_0 = \{p | p \in Q_0 \wedge T(p, a) \in Q_1\}$ 
13:     $Q''_0 = Q_0 \setminus Q'_0$ 
14:     $P = P \setminus \{Q_0\} \cup \{Q'_0, Q''_0\}$   $\triangleright$  Refine the partition, Replace  $Q_0$  by  $Q'_0$  and  $Q''_0$  in  $P$ .
15:    for all  $b \in V$  do  $\triangleright$  Update the waiting set
16:      if  $(Q_0, b) \in L$  then
17:         $L = L \setminus \{(Q_0, b)\} \cup \{(Q'_0, b), (Q''_0, b)\}$   $\triangleright$  Replace  $(Q_0, b)$  by  $(Q'_0, b)$  and  $(Q''_0, b)$  in  $L$ 
18:      else
19:         $ADD((\min(Q', Q''), b), L)$ 
20:      end if
21:    end for
22:  end for
23: end while

```

(a) DFA, Eq class= $\{\{4, 5\}, \{6, 7\}\}$ 

(b) mini-DFA

图 3.1: Minimization example 1

Algorithm 15 Hopcroft's minimization algorithm**Input:** $G = (Q, V, T, q_0, F)$ **Output:** The equivalence classes of Q

```

1:  $P \leftarrow [Q]_{E_0} = \{F, Q \setminus F\}$   $\triangleright$  The initial partitions is  $[Q]_{E_0}$ , it's the total euivalence relation.
2:  $L \leftarrow 0$   $\triangleright$  The waiting set
3:  $C = V$   $\triangleright$  C is all symbols set
4: if  $|F| \leq |Q \setminus F|$  then  $\triangleright$  initialization of the waiting set
5:    $L[q] = C.Size(), [q]$  is the representative of the  $F$ 
6: else
7:    $L[q] = C.Size(), [q]$  is the representative of the  $Q \setminus F$ 
8: end if
9: while (1) do
10:   if all  $L[q]=0$  then
11:     break;
12:   end if
13:   Find the first pair in L that still needs processing.  $(Q_1, a) = [q], L[q] \neq 0$   $\triangleright$  Take and remove some splitter
14:    $P_{old} = P$   $\triangleright$  current partitions
15:    $L[q] - -;$   $\triangleright$  Mark this element of L as processed.
16:   for all  $Q_0 \in P_{old}$  do
17:      $Q_0$  is split by  $(Q_1, a)$   $\triangleright$  Compute the split,  $Q_0$  is splitted into  $Q'_0$  and  $Q''_0$ 
18:      $Q'_0 = \{p | p \in Q_0 \wedge T(p, a) \in Q_1\}$ 
19:      $Q''_0 = \{Q_0 \setminus Q'_0\}$ 
20:      $P = P \setminus \{Q_0\} \cup \{Q'_0, Q''_0\}$   $\triangleright$  Refine the partition, Replace  $Q_0$  by  $Q'_0$  and  $Q''_0$  in  $P$ .
21:      $p = Q_0$ 
22:      $r = Q'_0$ 
23:     if  $[r] \neq Invalid$  then  $\triangleright$  Update the waiting set
24:       if  $(|p| \leq |r|)$  then
25:          $L[r] = L[p]$   $\triangleright$  [r] 待处理 L[p] 剩下的字符
26:          $L[p] = C.size()$   $\triangleright$  新的 [p], 待处理 C[0]...C[C.size()-1]
27:       else
28:          $L[r] = C.size()$   $\triangleright$  // 新的 [r], 待处理 C[0]...C[C.size()-1]
29:       end if
30:     end if
31:   end for
32: end while

```

$F = \{ 6 \ 7 \ 8 \}$

Transitions =

$0 \rightarrow \{ 'a' \rightarrow 1 \quad 'b' \rightarrow 8 \}$

$1 \rightarrow \{ 'a' \rightarrow 2 \quad 'b' \rightarrow 4 \}$

$2 \rightarrow \{ 'a' \rightarrow 2 \quad 'b' \rightarrow 6 \}$

$3 \rightarrow \{ 'a' \rightarrow 4 \quad 'b' \rightarrow 7 \}$

$4 \rightarrow \{ ['a', 'b'] \rightarrow 6 \}$

$5 \rightarrow \{ ['a', 'b'] \rightarrow 7 \}$

$6 \rightarrow \{ ['a', 'b'] \rightarrow 6 \}$

$7 \rightarrow \{ ['a', 'b'] \rightarrow 6 \}$

```
8->{ 'a'->5 'b'->3 }
```

```
current = -1
```

```
is the DFA Useful?: 1
```

The combination for all the out labels of State's: $C = \{ 'a' 'b' \}$

L:

```
0 1 2 3 4 5 6 7 8
```

```
0 0 0 0 0 0 0 0 0
```

```
Initialize partitions, E0:
```

```
StateEqRel
```

```
{ 0 1 2 3 4 5 }
```

```
{ 6 7 8 }
```

```
Initialize Lrepr={F}:
```

```
{ 6 }
```

L:

```
0 1 2 3 4 5 6 7 8
```

```
0 0 0 0 0 0 2 0 0
```

```
===== Iterate: k=1
```

L:

```
0 1 2 3 4 5 6 7 8
```

```
0 0 0 0 0 0 1 0 0
```

```
Partitions:
```

```
StateEqRel
```

```
{ 0 1 2 3 4 5 }
```

```
{ 6 7 8 }
```

```
pick [q] in L: ([q], a) = ([6], 'b')
```

```
split [p] w.r.t ([6], 'b')
```

```
==split [0] w.r.t ([6], 'b')
```

```
new split of [0] is [1]
```

```
[p] = { 0 2 3 4 5 }
```

```
[r] = { 1 }
```

```
p and r are the new representatives. Now update L with the smallest of
[0] and [1]
```

```
using [r] = [1], L[r] = C.size();
```

```
after update L:
```

L:

```
0 1 2 3 4 5 6 7 8
```

```

0 2 0 0 0 0 1 0 0
==split [6] w.r.t ([6], 'b')
new split of [6] is [8]
[p] = {6 7}
[r] = {8}
p and r are the new representatives. Now update L with the smallest of
[6] and [8]
using [r] = [8], L[r] = C.size();
after update L:
L:
0 1 2 3 4 5 6 7 8
0 2 0 0 0 0 1 0 2
===== Iterate: k = 2

L:
0 1 2 3 4 5 6 7 8
0 1 0 0 0 0 1 0 2
Partitions:
StateEqRel
{0 2 3 4 5}
{1}
{6 7}
{8}

pick [q] in L: ([q], a) = ([1], 'b')
split [p] w.r.t ([1], 'b')
==split [0] w.r.t ([1], 'b')
new split of [0] is [-1]
==split [1] w.r.t ([1], 'b')
new split of [1] is [-1]
==split [6] w.r.t ([1], 'b')
new split of [6] is [-1]
==split [8] w.r.t ([1], 'b')
new split of [8] is [-1]
===== Iterate: k = 3

L:
0 1 2 3 4 5 6 7 8
0 0 0 0 0 0 1 0 2
Partitions:
StateEqRel
{0 2 3 4 5}
{1}

```

$\{ _6 _7 \}$
 $\{ _8 \}$

$\text{pick_}[q] _ \text{in_} L : ([q], a) = ([1], 'a')$

$\text{split_}[p] _ \text{w.r.t_} ([1], 'a')$

$\equiv \text{split } [0] _ \text{w.r.t_} ([1], 'a')$

$\text{new_split_of_} [0] _ \text{is_} [2]$

$[p] = \{ _0 \}$

$[r] = \{ _2 _3 _4 _5 \}$

p and r are the new representatives. Now update L with the smallest of $[0]$ and $[2]$

using $[p] _ = [0]$, $L[r] = L[p]$; $_L[p] = C.size()$;

after update L :

L :

$0 _ 1 _ 2 _ 3 _ 4 _ 5 _ 6 _ 7 _ 8$

$2 _ 0 _ 0 _ 0 _ 0 _ 0 _ 1 _ 0 _ 2$

$\equiv \text{split } [1] _ \text{w.r.t_} ([1], 'a')$

$\text{new_split_of_} [1] _ \text{is_} [-1]$

$\equiv \text{split } [6] _ \text{w.r.t_} ([1], 'a')$

$\text{new_split_of_} [6] _ \text{is_} [-1]$

$\equiv \text{split } [8] _ \text{w.r.t_} ([1], 'a')$

$\text{new_split_of_} [8] _ \text{is_} [-1]$

$\equiv \text{Iterate : } _k = 4$

L :

$0 _ 1 _ 2 _ 3 _ 4 _ 5 _ 6 _ 7 _ 8$

$1 _ 0 _ 0 _ 0 _ 0 _ 0 _ 1 _ 0 _ 2$

Partitions:

StateEqRel

$\{ _0 \}$

$\{ _1 \}$

$\{ _2 _3 _4 _5 \}$

$\{ _6 _7 \}$

$\{ _8 \}$

$\text{pick_}[q] _ \text{in_} L : ([q], a) = ([0], 'b')$

$\text{split_}[p] _ \text{w.r.t_} ([0], 'b')$

$\equiv \text{split } [0] _ \text{w.r.t_} ([0], 'b')$

$\text{new_split_of_} [0] _ \text{is_} [-1]$

$\equiv \text{split } [1] _ \text{w.r.t_} ([0], 'b')$

$\text{new_split_of_} [1] _ \text{is_} [-1]$

$\equiv \text{split } [2] _ \text{w.r.t_} ([0], 'b')$

```

new_split_of [2] is [-1]
==split [6] w.r.t ([0], 'b')
new_split_of [6] is [-1]
==split [8] w.r.t ([0], 'b')
new_split_of [8] is [-1]
=====Iterate: k=5

L:
0 1 2 3 4 5 6 7 8
0 0 0 0 0 0 1 0 2
Partitions:
StateEqRel
{ 0 }
{ 1 }
{ 2 3 4 5 }
{ 6 7 }
{ 8 }

pick [q] in L: ([q], a) = ([0], 'a')
split [p] w.r.t ([0], 'a')
==split [0] w.r.t ([0], 'a')
new_split_of [0] is [-1]
==split [1] w.r.t ([0], 'a')
new_split_of [1] is [-1]
==split [2] w.r.t ([0], 'a')
new_split_of [2] is [-1]
==split [6] w.r.t ([0], 'a')
new_split_of [6] is [-1]
==split [8] w.r.t ([0], 'a')
new_split_of [8] is [-1]
=====Iterate: k=6

L:
0 1 2 3 4 5 6 7 8
0 0 0 0 0 0 0 0 2
Partitions:
StateEqRel
{ 0 }
{ 1 }
{ 2 3 4 5 }
{ 6 7 }
{ 8 }

```

```

pick [q] in L: ([q], a) = ([6], 'a')
split [p] w.r.t ([6], 'a')
==split [0] w.r.t ([6], 'a')
new split of [0] is [-1]
==split [1] w.r.t ([6], 'a')
new split of [1] is [-1]
==split [2] w.r.t ([6], 'a')
new split of [2] is [4]
[p] = {2, 3}
[r] = {4, 5}
p and r are the new representatives. Now update L with the smallest of
[2] and [4]
using [p] = [2], L[r] = L[p]; L[p] = C.size();
after update L:
L:
0 1 2 3 4 5 6 7 8
0 0 0 2 0 0 0 0 0 2
==split [6] w.r.t ([6], 'a')
new split of [6] is [-1]
==split [8] w.r.t ([6], 'a')
new split of [8] is [-1]
===== Iterate: k = 7
L:
0 1 2 3 4 5 6 7 8
0 0 0 1 0 0 0 0 0 2
Partitions:
StateEqRel
{0}
{1}
{2, 3}
{4, 5}
{6, 7}
{8}

pick [q] in L: ([q], a) = ([2], 'b')
split [p] w.r.t ([2], 'b')
==split [0] w.r.t ([2], 'b')
new split of [0] is [-1]
==split [1] w.r.t ([2], 'b')
new split of [1] is [-1]
==split [2] w.r.t ([2], 'b')

```

```

new_split_of [2] is [-1]
==split [4] w.r.t ([2], 'b')
new_split_of [4] is [-1]
==split [6] w.r.t ([2], 'b')
new_split_of [6] is [-1]
==split [8] w.r.t ([2], 'b')
new_split_of [8] is [-1]
=====Iterate: k=8

L:
0 1 2 3 4 5 6 7 8
0 0 0 0 0 0 0 0 2
Partitions:
StateEqRel
{0}
{1}
{2 3}
{4 5}
{6 7}
{8}

pick [q] in L: ([q], a) = ([2], 'a')
split [p] w.r.t ([2], 'a')
==split [0] w.r.t ([2], 'a')
new_split_of [0] is [-1]
==split [1] w.r.t ([2], 'a')
new_split_of [1] is [-1]
==split [2] w.r.t ([2], 'a')
new_split_of [2] is [3]
[p] = {2}
[r] = {3}
p and r are the new representatives. Now update L with the smallest of
[2] and [3]
using [p] = [2], L[r] = L[p]; L[p] = C.size();
after update L:
L:
0 1 2 3 4 5 6 7 8
0 0 2 0 0 0 0 0 2
==split [4] w.r.t ([2], 'a')
new_split_of [4] is [-1]
==split [6] w.r.t ([2], 'a')
new_split_of [6] is [-1]

```

```

====split [8] w.r.t ([2], 'a')
new_split_of [8] is [-1]
=====Iterate: k=9
L:
0 1 2 3 4 5 6 7 8
0 0 0 1 0 0 0 0 0 2
Partitions:
StateEqRel
{ 0 }
{ 1 }
{ 2 }
{ 3 }
{ 4 5 }
{ 6 7 }
{ 8 }

pick [q] in L: ([q], a) = ([2], 'b')
split [p] w.r.t ([2], 'b')
====split [0] w.r.t ([2], 'b')
new_split_of [0] is [-1]
====split [1] w.r.t ([2], 'b')
new_split_of [1] is [-1]
====split [2] w.r.t ([2], 'b')
new_split_of [2] is [-1]
====split [3] w.r.t ([2], 'b')
new_split_of [3] is [-1]
====split [4] w.r.t ([2], 'b')
new_split_of [4] is [-1]
====split [6] w.r.t ([2], 'b')
new_split_of [6] is [-1]
====split [8] w.r.t ([2], 'b')
new_split_of [8] is [-1]
=====Iterate: k=10
L:
0 1 2 3 4 5 6 7 8
0 0 0 0 0 0 0 0 0 2
Partitions:
StateEqRel
{ 0 }
{ 1 }
{ 2 }

```


$\{ \sqcup 3 \sqcup \}$

$\{ \sqcup 4 \sqcup \sqcup 5 \sqcup \}$

$\{ \sqcup 6 \sqcup \sqcup 7 \sqcup \}$

$\{ \sqcup 8 \sqcup \}$

$\text{pick}_{\sqcup} [q]_{\sqcup} \text{in}_{\sqcup} L : ([q], a) = ([2], 'a')$

$\text{split}_{\sqcup} [p]_{\sqcup} \text{w.r.t}_{\sqcup} ([2], 'a')$

$\equiv \text{split} [0]_{\sqcup} \text{w.r.t}_{\sqcup} ([2], 'a')$

$\text{new}_{\sqcup} \text{split}_{\sqcup} \text{of}_{\sqcup} [0]_{\sqcup} \text{is}_{\sqcup} [-1]$

$\equiv \text{split} [1]_{\sqcup} \text{w.r.t}_{\sqcup} ([2], 'a')$

$\text{new}_{\sqcup} \text{split}_{\sqcup} \text{of}_{\sqcup} [1]_{\sqcup} \text{is}_{\sqcup} [-1]$

$\equiv \text{split} [2]_{\sqcup} \text{w.r.t}_{\sqcup} ([2], 'a')$

$\text{new}_{\sqcup} \text{split}_{\sqcup} \text{of}_{\sqcup} [2]_{\sqcup} \text{is}_{\sqcup} [-1]$

$\equiv \text{split} [3]_{\sqcup} \text{w.r.t}_{\sqcup} ([2], 'a')$

$\text{new}_{\sqcup} \text{split}_{\sqcup} \text{of}_{\sqcup} [3]_{\sqcup} \text{is}_{\sqcup} [-1]$

$\equiv \text{split} [4]_{\sqcup} \text{w.r.t}_{\sqcup} ([2], 'a')$

$\text{new}_{\sqcup} \text{split}_{\sqcup} \text{of}_{\sqcup} [4]_{\sqcup} \text{is}_{\sqcup} [-1]$

$\equiv \text{split} [6]_{\sqcup} \text{w.r.t}_{\sqcup} ([2], 'a')$

$\text{new}_{\sqcup} \text{split}_{\sqcup} \text{of}_{\sqcup} [6]_{\sqcup} \text{is}_{\sqcup} [-1]$

$\equiv \text{split} [8]_{\sqcup} \text{w.r.t}_{\sqcup} ([2], 'a')$

$\text{new}_{\sqcup} \text{split}_{\sqcup} \text{of}_{\sqcup} [8]_{\sqcup} \text{is}_{\sqcup} [-1]$

$\equiv \text{Iterate} :_{\sqcup} k_{\sqcup} =_{\sqcup} 11$

L:

$0_{\sqcup} 1_{\sqcup} 2_{\sqcup} 3_{\sqcup} 4_{\sqcup} 5_{\sqcup} 6_{\sqcup} 7_{\sqcup} 8$

$0_{\sqcup} 0_{\sqcup} 0_{\sqcup} 0_{\sqcup} 0_{\sqcup} 0_{\sqcup} 0_{\sqcup} 0_{\sqcup} 1$

Partitions:

StateEqRel

$\{ \sqcup 0 \sqcup \}$

$\{ \sqcup 1 \sqcup \}$

$\{ \sqcup 2 \sqcup \}$

$\{ \sqcup 3 \sqcup \}$

$\{ \sqcup 4 \sqcup \sqcup 5 \sqcup \}$

$\{ \sqcup 6 \sqcup \sqcup 7 \sqcup \}$

$\{ \sqcup 8 \sqcup \}$

$\text{pick}_{\sqcup} [q]_{\sqcup} \text{in}_{\sqcup} L : ([q], a) = ([8], 'b')$

$\text{split}_{\sqcup} [p]_{\sqcup} \text{w.r.t}_{\sqcup} ([8], 'b')$

$\equiv \text{split} [0]_{\sqcup} \text{w.r.t}_{\sqcup} ([8], 'b')$

$\text{new}_{\sqcup} \text{split}_{\sqcup} \text{of}_{\sqcup} [0]_{\sqcup} \text{is}_{\sqcup} [-1]$

$\equiv \text{split} [1]_{\sqcup} \text{w.r.t}_{\sqcup} ([8], 'b')$

$\text{new}_{\sqcup} \text{split}_{\sqcup} \text{of}_{\sqcup} [1]_{\sqcup} \text{is}_{\sqcup} [-1]$

```

==split [2] w.r.t ([8] , 'b')
new_split_of [2] is [-1]
==split [3] w.r.t ([8] , 'b')
new_split_of [3] is [-1]
==split [4] w.r.t ([8] , 'b')
new_split_of [4] is [-1]
==split [6] w.r.t ([8] , 'b')
new_split_of [6] is [-1]
==split [8] w.r.t ([8] , 'b')
new_split_of [8] is [-1]

=====Iterate : k=12

L:
0 1 2 3 4 5 6 7 8
0 0 0 0 0 0 0 0 0
Partitions :
StateEqRel
{ 0 }
{ 1 }
{ 2 }
{ 3 }
{ 4 5 }
{ 6 7 }
{ 8 }

pick [q] in L : ([q] , a) = ([8] , 'a')
split [p] w.r.t ([8] , 'a')
==split [0] w.r.t ([8] , 'a')
new_split_of [0] is [-1]
==split [1] w.r.t ([8] , 'a')
new_split_of [1] is [-1]
==split [2] w.r.t ([8] , 'a')
new_split_of [2] is [-1]
==split [3] w.r.t ([8] , 'a')
new_split_of [3] is [-1]
==split [4] w.r.t ([8] , 'a')
new_split_of [4] is [-1]
==split [6] w.r.t ([8] , 'a')
new_split_of [6] is [-1]
==split [8] w.r.t ([8] , 'a')
new_split_of [8] is [-1]

```

*****_minDFA

DFA

$Q = [0, 7)$

$S = \{0\}$

$F = \{5, 6\}$

Transitions =

$0 \rightarrow \{ 'a' \rightarrow 1, 'b' \rightarrow 6 \}$

$1 \rightarrow \{ 'a' \rightarrow 2, 'b' \rightarrow 4 \}$

$2 \rightarrow \{ 'a' \rightarrow 2, 'b' \rightarrow 5 \}$

$3 \rightarrow \{ 'a' \rightarrow 4, 'b' \rightarrow 5 \}$

$4 \rightarrow \{ 'a', 'b' \rightarrow 5 \}$

$5 \rightarrow \{ 'a', 'b' \rightarrow 5 \}$

$6 \rightarrow \{ 'a' \rightarrow 4, 'b' \rightarrow 3 \}$

current = 1

3.1 Minimization example 2

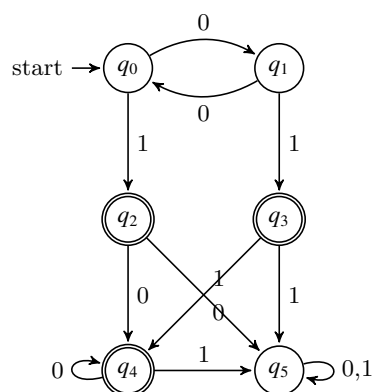


图 3.2: Minimization example 2

DFA

$Q = [0, 6)$

$S = \{0\}$

$F = \{2, 3, 4\}$

Transitions =

$0 \rightarrow \{ '0' \rightarrow 1, '1' \rightarrow 2 \}$

$1 \rightarrow \{ '0' \rightarrow 0, '1' \rightarrow 3 \}$

```

2->{ '0'->4 '1'->5 }
3->{ '0'->4 '1'->5 }
4->{ '0'->4 '1'->5 }
5->{ ['0','1']->5 }

```

```
current = -1
```

```
is the DFA Useful?: 0
```

The combination for all the out labels of State's: $C = \{ '0', '1' \}$

```

L:
0 1 2 3 4 5
0 0 0 0 0 0
Initialize partitions, E0:
StateEqRel
{ 0 1 5 }
{ 2 3 4 }

Initialize Lrepr = {F}:
{ 2 }

L:
0 1 2 3 4 5
0 0 2 0 0 0
===== Iterate: k=1
L:
0 1 2 3 4 5
0 0 1 0 0 0
Partitions:
StateEqRel
{ 0 1 5 }
{ 2 3 4 }

pick [q] in L: ([q], a) = ([2], '1')
split [p] w.r.t ([2], '1')
== split [0] w.r.t ([2], '1')
new split of [0] is [5]
[p] = { 0 1 }
[r] = { 5 }
p and r are the new representatives. Now update L with the
smallest of [0] and [5]

```

```

using [r] = [5], L[r] = C.size();
after update L:
L:
0 1 2 3 4 5
0 0 1 0 0 2
==split [2] w.r.t ([2], '1')
new split of [2] is [-1]
===== Iterate: k = 2
L:
0 1 2 3 4 5
0 0 0 0 0 2
Partitions:
StateEqRel
{ 0 1 }
{ 2 3 4 }
{ 5 }

pick [q] in L: ([q], a) = ([2], '0')
split [p] w.r.t ([2], '0')
==split [0] w.r.t ([2], '0')
new split of [0] is [-1]
==split [2] w.r.t ([2], '0')
new split of [2] is [-1]
==split [5] w.r.t ([2], '0')
new split of [5] is [-1]
===== Iterate: k = 3
L:
0 1 2 3 4 5
0 0 0 0 0 1
Partitions:
StateEqRel
{ 0 1 }
{ 2 3 4 }
{ 5 }

pick [q] in L: ([q], a) = ([5], '1')
split [p] w.r.t ([5], '1')
==split [0] w.r.t ([5], '1')
new split of [0] is [-1]
==split [2] w.r.t ([5], '1')
new split of [2] is [-1]

```

```

=====split [5] w.r.t ([5], '1')
new_split_of [5] is [-1]
=====Iterate: k=4
L:
0 1 2 3 4 5
0 0 0 0 0 0
Partitions:
StateEqRel
{0 1}
{2 3 4}
{5}

pick [q] in L: ([q], a) = ([5], '0')
split [p] w.r.t ([5], '0')
=====split [0] w.r.t ([5], '0')
new_split_of [0] is [-1]
=====split [2] w.r.t ([5], '0')
new_split_of [2] is [-1]
=====split [5] w.r.t ([5], '0')
new_split_of [5] is [-1]

*****minDFA

DFA
Q=[0,3)
S={0}
F={1}
Transitions=
0->{'0'>0, '1'>1}
1->{'0'>1, '1'>2}
2->{'0', '1'>2}

current=-1
\end{list}

```

3.2 Minimization example 3 ($1 \rightarrow 2 \rightarrow 3 \rightarrow 4 \rightarrow 5 \rightarrow 6$)

split Q_0 w.r.t. $(Q_1, 0)$, see Fig. 3.4.

Ex 1: $Q_0 = \{0, 1, 2, 3, 4\}, Q_1 = \{5\}$

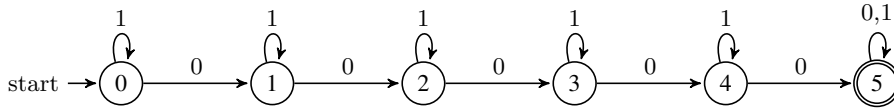


图 3.3: Minimizing example

Since, $Q'_0 = \{4\} \subseteq Q_0$, and $T(Q'_0, 0) = T(\{4\}, 0) = 5 \in Q_1$

$Q''_0 = Q_0 \setminus Q'_0 = \{0, 1, 2, 3\} \subseteq Q_0$, and $T(Q''_0, 0) = T(\{0, 1, 2, 3\}, 0) = T(\{0\}, 0) \cup T(\{1\}, 0) \cup T(\{2\}, 0) \cup T(\{3\}, 0) = \{1\} \cup \{2\} \cup \{3\} = \{1, 2, 3\} \notin Q_1$

$\therefore Q_0$ w.r.t. $(Q_1, 0)$ is splitted into two parts. part(1) $Q'_0 = \{4\}$, part(2) $Q''_0 = Q_0 \setminus Q'_0 = \{0, 1, 2, 3\}$

Ex 2: $Q_0 = \{5\}, Q_1 = \{0, 1, 2, 3, 4\}$

Since, $\forall q \in Q_0, \nexists T(q, a) \in Q_1$

$\therefore Q_0$ w.r.t. $(Q_1, 0)$ 是一个无效的 split, 同样 $Q_0 = Q_1 = \{5\}$ 也是一个无效的 split。

Ex 3: $Q_0 = \{0, 1, 2, 3, 4\}, Q_1 = \{0, 1, 2, 3, 4\}$

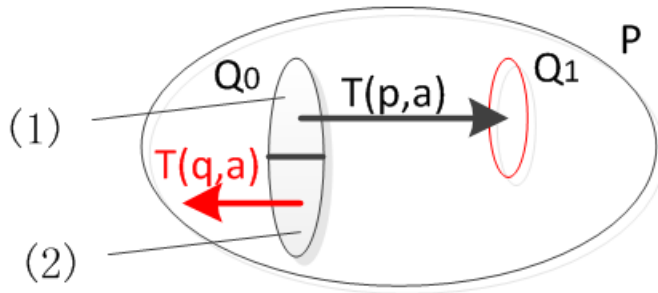
Since, $Q'_0 = \{0, 1, 2, 3\} \subseteq Q_0$, and $T(Q'_0, 0) = T(\{0, 1, 2, 3\}, 0) = T(\{0\}, 0) \cup T(\{1\}, 0) \cup T(\{2\}, 0) \cup T(\{3\}, 0) = \{1\} \cup \{2\} \cup \{3\} = \{1, 2, 3\} \in Q_1$

$Q''_0 = Q_0 \setminus Q'_0 = \{4\} \subseteq Q_0$, and $T(Q''_0, 0) = T(\{4\}, 0) = 5 \notin Q_1$

$\therefore Q_0$ w.r.t. $(Q_1, 0)$ is splitted into two parts. part(1) $Q'_0 = \{0, 1, 2, 3\}$, part(2) $Q''_0 = Q_0 \setminus Q'_0 = \{4\}$

if $(\exists p, q \in Q_0, T(p, a) \in Q_1, \text{ and } T(q, a) \notin Q_1)$, then

split Q_0 w.r.t. $(Q_1, a) \rightarrow$ two parts, (1): $Q'_0 \in Q_1$ and (2): $(Q_0 \setminus Q'_0) \notin Q_1$

图 3.4: split Q_0 w.r.t. (Q_1, a)

开始于: $[q] = \{Q \setminus F\} = \{0, 1, 2, 3, 4\}$, split $[p]$ w.r.t. $([q], a)$

***** DFA

DFA

$Q = [0, 6)$

$S = \{ 0 \}$

```
F = { 5 }
```

```
Transitions =
```

```
0->{ '0'->1 '1'->0 }
```

```
1->{ '0'->2 '1'->1 }
```

```
2->{ '0'->3 '1'->2 }
```

```
3->{ '0'->4 '1'->3 }
```

```
4->{ '0'->5 '1'->4 }
```

```
5->{ [ '0', '1' ]->5 }
```

```
current = -1
```

```
is the DFA Useful?: 1
```

The combination **for** all the out labels of State's: $C = \{ '0', '1' \}$

```
L:
```

```
0 1 2 3 4 5
```

```
0 0 0 0 0 0
```

```
Initialize partitions, E0:
```

```
StateEqRel
```

```
{ 0 1 2 3 4 }
```

```
{ 5 }
```

```
L:
```

```
0 1 2 3 4 5
```

```
2 0 0 0 0 0
```

```
===== Iterate: k=1
```

```
L:
```

```
0 1 2 3 4 5
```

```
1 0 0 0 0 0
```

```
Partitions:
```

```
StateEqRel
```

```
{ 0 1 2 3 4 }
```

```
{ 5 }
```

```
pick [q] in L: ([q], a) = ([0], '1')
```

```
split [p] w.r.t ([0], '1')
```

```
== split [0] w.r.t ([0], '1')
```

```
new split of [0] is [-1]
```

```
== split [5] w.r.t ([0], '1')
```

```
new split of [5] is [-1]
```

```
===== Iterate: k=2
```



```

L:
0_1_2_3_4_5
0_0_0_0_0_0
Partitions:
StateEqRel
{0_1_2_3_4}
{5}

pick[q] in L: ([q], a) = ([0], '0')
split[p] w.r.t ([0], '0')
== split [0] w.r.t ([0], '0')
new_split_of [0] is [4]
[p] = {0_1_2_3}
[r] = {4}
p and r are the new representatives. Now update L with the smallest of
[0] and [4]
using [r] = [4], L[r] = C.size();
after update L:
L:
0_1_2_3_4_5
0_0_0_0_2_0
== split [5] w.r.t ([0], '0')
new_split_of [5] is [-1]
===== Iterate: k = 3

L:
0_1_2_3_4_5
0_0_0_0_1_0
Partitions:
StateEqRel
{0_1_2_3}
{4}
{5}

pick[q] in L: ([q], a) = ([4], '1')
split[p] w.r.t ([4], '1')
== split [0] w.r.t ([4], '1')
new_split_of [0] is [-1]
== split [4] w.r.t ([4], '1')
new_split_of [4] is [-1]
== split [5] w.r.t ([4], '1')
new_split_of [5] is [-1]

```

```

=====Iterate : k=4

L:
0 1 2 3 4 5
0 0 0 0 0 0
Partitions:
StateEqRel
{ 0 1 2 3 }
{ 4 }
{ 5 }

pick [q] in L : ([q], a) = ([4], '0')
split [p] w.r.t ([4], '0')
==split [0] w.r.t ([4], '0')
new split of [0] is [3]
[p] = { 0 1 2 }
[r] = { 3 }
p and r are the new representatives. Now update L with the smallest of
    [0] and [3]
using [r] = [3], L[r] = C.size();
after update L:
L:
0 1 2 3 4 5
0 0 0 2 0 0
==split [4] w.r.t ([4], '0')
new split of [4] is [-1]
==split [5] w.r.t ([4], '0')
new split of [5] is [-1]
=====Iterate : k=5

L:
0 1 2 3 4 5
0 0 0 1 0 0
Partitions:
StateEqRel
{ 0 1 2 }
{ 3 }
{ 4 }
{ 5 }

pick [q] in L : ([q], a) = ([3], '1')
split [p] w.r.t ([3], '1')
==split [0] w.r.t ([3], '1')

```

```

new_split_of [0] is [-1]
==split [3] w.r.t ([3], '1')
new_split_of [3] is [-1]
==split [4] w.r.t ([3], '1')
new_split_of [4] is [-1]
==split [5] w.r.t ([3], '1')
new_split_of [5] is [-1]
=====Iterate: k=6

L:
0 1 2 3 4 5
0 0 0 0 0 0
Partitions:
StateEqRel
{0 1 2}
{3}
{4}
{5}

pick [q] in L: ([q], a) = ([3], '0')
split [p] w.r.t ([3], '0')
==split [0] w.r.t ([3], '0')
new_split_of [0] is [2]
[p] = {0 1}
[r] = {2}
p and r are the new representatives. Now update L with the smallest of
[0] and [2]
using [r] = [2], L[r] = C.size();
after update L:
L:
0 1 2 3 4 5
0 0 2 0 0 0
==split [3] w.r.t ([3], '0')
new_split_of [3] is [-1]
==split [4] w.r.t ([3], '0')
new_split_of [4] is [-1]
==split [5] w.r.t ([3], '0')
new_split_of [5] is [-1]
=====Iterate: k=7

L:
0 1 2 3 4 5
0 0 1 0 0 0

```



```

after_update_L:
L:
0_1_2_3_4_5
2_0_0_0_0_0
==split [2]_w.r.t_([2], '0')
new_split_of [2]_is [-1]
==split [3]_w.r.t_([2], '0')
new_split_of [3]_is [-1]
==split [4]_w.r.t_([2], '0')
new_split_of [4]_is [-1]
==split [5]_w.r.t_([2], '0')
new_split_of [5]_is [-1]
=====Iterate: k=9

L:
0_1_2_3_4_5
1_0_0_0_0_0
Partitions:
StateEqRel
{0_}
{1_}
{2_}
{3_}
{4_}
{5_}

pick [q]_in_L: ([q], a) = ([0], '1')
split [p]_w.r.t_([0], '1')
==split [0]_w.r.t_([0], '1')
new_split_of [0]_is [-1]
==split [1]_w.r.t_([0], '1')
new_split_of [1]_is [-1]
==split [2]_w.r.t_([0], '1')
new_split_of [2]_is [-1]
==split [3]_w.r.t_([0], '1')
new_split_of [3]_is [-1]
==split [4]_w.r.t_([0], '1')
new_split_of [4]_is [-1]
==split [5]_w.r.t_([0], '1')
new_split_of [5]_is [-1]
=====Iterate: k=10

L:

```

```
0_1_2_3_4_5
```

```
0_0_0_0_0_0
```

```
Partitions:
```

```
StateEqRel
```

```
{_0_}
```

```
{_1_}
```

```
{_2_}
```

```
{_3_}
```

```
{_4_}
```

```
{_5_}
```

```
pick_[q]_in_L:([q],a)=([0], '0')
```

```
split_[p]_w.r.t_([0], '0')
```

```
==split[0]_w.r.t_([0], '0')
```

```
new_split_of_[0]_is_[-1]
```

```
==split[1]_w.r.t_([0], '0')
```

```
new_split_of_[1]_is_[-1]
```

```
==split[2]_w.r.t_([0], '0')
```

```
new_split_of_[2]_is_[-1]
```

```
==split[3]_w.r.t_([0], '0')
```

```
new_split_of_[3]_is_[-1]
```

```
==split[4]_w.r.t_([0], '0')
```

```
new_split_of_[4]_is_[-1]
```

```
==split[5]_w.r.t_([0], '0')
```

```
new_split_of_[5]_is_[-1]
```

```
*****_minDFA
```

```
DFA
```

```
Q_=[0,6)
```

```
S_={_0_}
```

```
F_={_5_}
```

```
Transitions_
```

```
0->{_0'>1_ '1'>0_}
```

```
1->{_0'>2_ '1'>1_}
```

```
2->{_0'>3_ '1'>2_}
```

```
3->{_0'>4_ '1'>3_}
```

```
4->{_0'>5_ '1'>4_}
```

```
5->{[_0', '1']>5_}
```

```
current_=-1
```

开始于: $[q] = \{F\} = \{5\}$, split $[p]$ w.r.t. $([q], a)$

***** DFA

DFA

$Q = [0, 6)$

$S = \{ 0 \}$

$F = \{ 5 \}$

Transitions =

$0 \rightarrow \{ '0' \rightarrow 1 \quad '1' \rightarrow 0 \}$

$1 \rightarrow \{ '0' \rightarrow 2 \quad '1' \rightarrow 1 \}$

$2 \rightarrow \{ '0' \rightarrow 3 \quad '1' \rightarrow 2 \}$

$3 \rightarrow \{ '0' \rightarrow 4 \quad '1' \rightarrow 3 \}$

$4 \rightarrow \{ '0' \rightarrow 5 \quad '1' \rightarrow 4 \}$

$5 \rightarrow \{ ['0', '1'] \rightarrow 5 \}$

current = -1

is the DFA Useful?: 1

The combination for all the out labels of State's: $C = \{ '0'_{00} '1'_{00} \}$

L:

$0_{01} 1_{02} 2_{03} 3_{04} 4_{05}$

$0_{00} 0_{00} 0_{00} 0_{00}$

Initialize partitions, E0:

StateEqRel

$\{ 0_{00} 1_{00} 2_{00} 3_{00} 4_{00} \}$

$\{ 5_{00} \}$

Initialize Lrepr = {F}:

$\{ 5_{00} \}$

L:

$0_{01} 1_{02} 2_{03} 3_{04} 4_{05}$

$0_{00} 0_{00} 0_{00} 0_{02}$

===== Iterate: k = 1

L:

$0_{01} 1_{02} 2_{03} 3_{04} 4_{05}$

$0_{00} 0_{00} 0_{00} 0_{01}$

Partitions:

StateEqRel

$\{ 0_{00} 1_{00} 2_{00} 3_{00} 4_{00} \}$

$\{ _5 \}$

$\text{pick_}[q] _ \text{in_} L : ([q], a) = ([5], '1')$

$\text{split_}[p] _ \text{w.r.t_} ([5], '1')$

$\equiv \text{split } [0] _ \text{w.r.t_} ([5], '1')$

$\text{new_split_of_} [0] _ \text{is_} [-1]$

$\equiv \text{split } [5] _ \text{w.r.t_} ([5], '1')$

$\text{new_split_of_} [5] _ \text{is_} [-1]$

$\equiv \text{Iterate :_} k = 2$

L:

$0_1_2_3_4_5$

$0_0_0_0_0_0$

Partitions:

StateEqRel

$\{ _0_1_2_3_4 \}$

$\{ _5 \}$

$\text{pick_}[q] _ \text{in_} L : ([q], a) = ([5], '0')$

$\text{split_}[p] _ \text{w.r.t_} ([5], '0')$

$\equiv \text{split } [0] _ \text{w.r.t_} ([5], '0')$

$\text{new_split_of_} [0] _ \text{is_} [4]$

$[p] = \{ _0_1_2_3 \}$

$[r] = \{ _4 \}$

p and r are the new representatives. Now update L with the smallest of $[0]$ and $[4]$

using $[r] = [4]$, $L[r] = C.size()$;

after update L :

L:

$0_1_2_3_4_5$

$0_0_0_0_2_0$

$\equiv \text{split } [5] _ \text{w.r.t_} ([5], '0')$

$\text{new_split_of_} [5] _ \text{is_} [-1]$

$\equiv \text{Iterate :_} k = 3$

L:

$0_1_2_3_4_5$

$0_0_0_0_1_0$

Partitions:

StateEqRel

$\{ _0_1_2_3 \}$

$\{ _4 \}$

$\{ _5 \}$


```

pick [q] in L: ([q], a) = ([4], '1')
split [p] w.r.t ([4], '1')
==split [0] w.r.t ([4], '1')
new split of [0] is [-1]
==split [4] w.r.t ([4], '1')
new split of [4] is [-1]
==split [5] w.r.t ([4], '1')
new split of [5] is [-1]
===== Iterate: k=4

L:
0 1 2 3 4 5
0 0 0 0 0 0
Partitions:
StateEqRel
{ 0 1 2 3 }
{ 4 }
{ 5 }

pick [q] in L: ([q], a) = ([4], '0')
split [p] w.r.t ([4], '0')
==split [0] w.r.t ([4], '0')
new split of [0] is [3]
[p] = { 0 1 2 }
[r] = { 3 }
p and r are the new representatives. Now update L with the smallest of
[0] and [3]
using [r] = [3], L[r] = C.size();
after update L:
L:
0 1 2 3 4 5
0 0 0 2 0 0
==split [4] w.r.t ([4], '0')
new split of [4] is [-1]
==split [5] w.r.t ([4], '0')
new split of [5] is [-1]
===== Iterate: k=5

L:
0 1 2 3 4 5
0 0 0 1 0 0
Partitions:

```

StateEqRel

{012}

{3}

{4}

{5}

pick [q] in L: ([q], a) = ([3], '1')

split [p] w.r.t ([3], '1')

== split [0] w.r.t ([3], '1')

new split of [0] is [-1]

== split [3] w.r.t ([3], '1')

new split of [3] is [-1]

== split [4] w.r.t ([3], '1')

new split of [4] is [-1]

== split [5] w.r.t ([3], '1')

new split of [5] is [-1]

==== Iterate: k=6

L:

012345

000000

Partitions:

StateEqRel

{012}

{3}

{4}

{5}

pick [q] in L: ([q], a) = ([3], '0')

split [p] w.r.t ([3], '0')

== split [0] w.r.t ([3], '0')

new split of [0] is [2]

[p] = {01}

[r] = {2}

p and r are the new representatives. Now update L with the smallest of [0] and [2]

using [r] = [2], L[r] = C.size();

after update L:

L:

012345

002000

== split [3] w.r.t ([3], '0')

```

new_split_of [3] is [-1]
==split [4] w.r.t ([3], '0')
new_split_of [4] is [-1]
==split [5] w.r.t ([3], '0')
new_split_of [5] is [-1]
=====Iterate: k=7

```

L:

0 1 2 3 4 5

0 0 1 0 0 0

Partitions:

StateEqRel

{ 0 1 }

{ 2 }

{ 3 }

{ 4 }

{ 5 }

pick [q] in L: ([q], a) = ([2], '1')

split [p] w.r.t ([2], '1')

==split [0] w.r.t ([2], '1')

new_split_of [0] is [-1]

==split [2] w.r.t ([2], '1')

new_split_of [2] is [-1]

==split [3] w.r.t ([2], '1')

new_split_of [3] is [-1]

==split [4] w.r.t ([2], '1')

new_split_of [4] is [-1]

==split [5] w.r.t ([2], '1')

new_split_of [5] is [-1]

=====Iterate: k=8

L:

0 1 2 3 4 5

0 0 0 0 0 0

Partitions:

StateEqRel

{ 0 1 }

{ 2 }

{ 3 }

{ 4 }

{ 5 }

```

pick [q] in L: ([q], a) = ([2], '0')
split [p] w.r.t ([2], '0')
==split [0] w.r.t ([2], '0')
new split of [0] is [1]
[p] = {0}
[r] = {1}
p and r are the new representatives. Now update L with the smallest of
[0] and [1]
using [p] = [0], L[r] = L[p]; L[p] = C.size();
after update L:
L:
0 1 2 3 4 5
2 0 0 0 0 0
==split [2] w.r.t ([2], '0')
new split of [2] is [-1]
==split [3] w.r.t ([2], '0')
new split of [3] is [-1]
==split [4] w.r.t ([2], '0')
new split of [4] is [-1]
==split [5] w.r.t ([2], '0')
new split of [5] is [-1]
===== Iterate: k = 9
L:
0 1 2 3 4 5
1 0 0 0 0 0
Partitions:
StateEqRel
{0}
{1}
{2}
{3}
{4}
{5}

pick [q] in L: ([q], a) = ([0], '1')
split [p] w.r.t ([0], '1')
==split [0] w.r.t ([0], '1')
new split of [0] is [-1]
==split [1] w.r.t ([0], '1')
new split of [1] is [-1]
==split [2] w.r.t ([0], '1')

```

```

new_split_of [2] is [-1]
==split [3] w.r.t ([0], '1')
new_split_of [3] is [-1]
==split [4] w.r.t ([0], '1')
new_split_of [4] is [-1]
==split [5] w.r.t ([0], '1')
new_split_of [5] is [-1]
=====Iterate:k=10

```

L:

0 1 2 3 4 5

0 0 0 0 0 0

Partitions:

StateEqRel

{0}

{1}

{2}

{3}

{4}

{5}

pick[q] in L: ([q], a) = ([0], '0')

split[p] w.r.t ([0], '0')

==split [0] w.r.t ([0], '0')

new_split_of [0] is [-1]

==split [1] w.r.t ([0], '0')

new_split_of [1] is [-1]

==split [2] w.r.t ([0], '0')

new_split_of [2] is [-1]

==split [3] w.r.t ([0], '0')

new_split_of [3] is [-1]

==split [4] w.r.t ([0], '0')

new_split_of [4] is [-1]

==split [5] w.r.t ([0], '0')

new_split_of [5] is [-1]

*****minDFA

DFA

Q=[0,6)

S={0}

F={5}

```

Transitions□=
0->{□'0'->1□□'1'->0□}
1->{□'0'->2□□'1'->1□}
2->{□'0'->3□□'1'->2□}
3->{□'0'->4□□'1'->3□}
4->{□'0'->5□□'1'->4□}
5->{□['0','1']->5□}

current□=□-1

```

开始于: $[q] = \{Q \setminus F\} = \{0, 1, 2, 3, 4\}$, split $[p]$ w.r.t. $([q], a)$
或者开始于 $[q] = \{F\} = \{5\}$, split $[p]$ w.r.t. $([q], a)$ 处理结果是一致的。

3.3 Minimization example 4

$\{0, 1\}, \{3, 4\}$ is not equivalent states.

Sets of equivalent states: $\{0, 2\}, \{1\}, \{3\}, \{4\}$

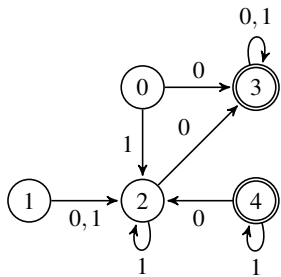


图 3.5: Finite state automaton

Partitions: $\{ \{0, 1, 2\} \{3, 4\} \}$ $Q_0 = Q_1 = \{3, 4\}$ split Q_0 w.r.t. $(Q_1, '1')$ is invalid.
Since $T(\{3, 4\}, '1') = \{3, 4\} = Q' = Q_1$, but $Q'' = Q_0 \setminus Q'_0 = \emptyset$
so, split $[3]$ w.r.t. $([3], '1')$ is invalid.

***** DFA

DFA

$Q = [0, 5)$

$S = \{ 0 \}$

$F = \{ 3, 4 \}$

Transitions =

$0 \rightarrow \{ '0' \rightarrow 3 \quad '1' \rightarrow 2 \}$

```

1->{ ['0','1']->2 }
2->{ '0'->3  '1'->2 }
3->{ ['0','1']->3 }
4->{ '0'->2  '1'->4 }

```

```
current = -1
```

```
is the DFA Useful?: 1
```

The combination `for` all the out labels of State'súC₀={₀'0'₀₀'1'₀}

```

00000000L:
000000000_1_2_3_4
000000000_0_0_0_0_0
00000000Initialize_partitions,_E0:
00000000StateEqRel
00000000{_0_1_2_}
00000000{_3_4_}

00000000
00000000Initialize_L_repr={F}:
00000000{_3_}

00000000L:
000000000_1_2_3_4
000000000_0_0_0_2_0

00000000=====Iterate:_k_=1

00000000L:
000000000_1_2_3_4
000000000_0_0_0_1_0
00000000Partitions:
00000000StateEqRel
00000000{_0_1_2_}
00000000{_3_4_}

00000000
00000000pick_q_in_L:([q],a)=([3], '1')
00000000split_p_w.r.t.([3], '1')
00000000==split[0]_w.r.t.([3], '1')
00000000new_split_of[0]_is_[-1]
00000000==split[3]_w.r.t.([3], '1')
00000000new_split_of[3]_is_[-1]

00000000=====Iterate:_k_=2

00000000L:

```

```

0 1 2 3 4
0 0 0 0 0
Partitions:
StateEqRel
{ 0 1 2 }
{ 3 4 }

pick [q] in L: ([q], a) = ([3], '0')
split [p] w.r.t ([3], '0')
==split [0] w.r.t ([3], '0')
new split of [0] is [1]
[p] = { 0 2 }
[r] = { 1 }
p and r are the new representatives. Now update L with the
smallest of [0] and [1]
using [r] = [1], L[r] = C.size();
after update L:
L:
0 1 2 3 4
0 2 0 0 0
==split [3] w.r.t ([3], '0')
new split of [3] is [4]
[p] = { 3 }
[r] = { 4 }
p and r are the new representatives. Now update L with the
smallest of [3] and [4]
using [p] = [3], L[r] = L[p]; L[p] = C.size();
after update L:
L:
0 1 2 3 4
0 2 0 2 0
===== Iterate: k = 3
L:
0 1 2 3 4
0 1 0 2 0
Partitions:
StateEqRel
{ 0 2 }
{ 1 }
{ 3 }
{ 4 }

```



```

#####
#####pick [q] in L: ([q], a) = ([1], '1')
#####split [p] w.r.t ([1], '1')
#####==split [0] w.r.t ([1], '1')
#####new split of [0] is [-1]
#####==split [1] w.r.t ([1], '1')
#####new split of [1] is [-1]
#####==split [3] w.r.t ([1], '1')
#####new split of [3] is [-1]
#####==split [4] w.r.t ([1], '1')
#####new split of [4] is [-1]
#####=====Iterate: k=4
#####L:
#####0 1 2 3 4
#####0 0 0 2 0
#####Partitions:
#####StateEqRel
#####{ 0 2 }
#####{ 1 }
#####{ 3 }
#####{ 4 }
#####
#####pick [q] in L: ([q], a) = ([1], '0')
#####split [p] w.r.t ([1], '0')
#####==split [0] w.r.t ([1], '0')
#####new split of [0] is [-1]
#####==split [1] w.r.t ([1], '0')
#####new split of [1] is [-1]
#####==split [3] w.r.t ([1], '0')
#####new split of [3] is [-1]
#####==split [4] w.r.t ([1], '0')
#####new split of [4] is [-1]
#####=====Iterate: k=5
#####L:
#####0 1 2 3 4
#####0 0 0 1 0
#####Partitions:
#####StateEqRel
#####{ 0 2 }
#####{ 1 }
#####{ 3 }

```

```

{}4
{}
pick[q] in L: ([q], a) = ([3], '1')
split[p] w.r.t ([3], '1')
==split[0] w.r.t ([3], '1')
new split of [0] is [-1]
==split[1] w.r.t ([3], '1')
new split of [1] is [-1]
==split[3] w.r.t ([3], '1')
new split of [3] is [-1]
==split[4] w.r.t ([3], '1')
new split of [4] is [-1]
===== Iterate: k=6
L:
0 1 2 3 4
0 0 0 0 0
Partitions:
StateEqRel
{}0 2
{}1
{}3
{}4
{}
pick[q] in L: ([q], a) = ([3], '0')
split[p] w.r.t ([3], '0')
==split[0] w.r.t ([3], '0')
new split of [0] is [-1]
==split[1] w.r.t ([3], '0')
new split of [1] is [-1]
==split[3] w.r.t ([3], '0')
new split of [3] is [-1]
==split[4] w.r.t ([3], '0')
new split of [4] is [-1]
{}
***** minDFA
{}
DFA
Q=[0,4)
S={0}
F={2,3}
Transitions=

```

```

0->{ [ ' 0 ' -> 2 [ ' 1 ' -> 0 [
1->{ [ ' 0 ' , ' 1 ' ] -> 0 [
2->{ [ ' 0 ' , ' 1 ' ] -> 2 [
3->{ [ ' 0 ' -> 0 [ ' 1 ' -> 3 [

current [ = [ -1

```

References

- Hopcroft2008. John E. Hopcroft, Rajeev Motwani, Jeffrey D. Ullman 著, 孙家骥等译, 自动机理论、语言和计算机导论, Third Edition, 机械工业出版社, 2008.7
- WATSON93a. WATSON, B. W. *A taxonomy of finite automata construction algorithms*, Computing Science Note 93/43, Eindhoven University of Technology, The Netherlands, 1993. Available by ftp from ftp.win.tue.nl in pub/techreports/pi.
- WATSON93b. WATSON, B. W. *A taxonomy of finite automata minimization algorithms*, Computing Science Note 93/44, Eindhoven University of Technology, The Netherlands, 1993. Available by ftp from ftp.win.tue.nl in pub/techreports/pi.
- WATSON94a. WATSON, B. W. *An introduction to the FIRE engine: A C++ toolkit for FInite automata and Regular Expressions*, Computing Science Note 94/21, Eindhoven University of Technology, The Netherlands, 1994. Available by ftp from ftp.win.tue.nl in pub/techreports/pi
- WATSON94b. WATSON, B.W. *The design. and implementation of the FIRE engine: A C++ toolkit for FInite automata and Regular Expressions*, Computing Science Note 94/22, Eindhoven University of Technology, The Netherlands, 1994. Available by ftp from ftp.win.tue.nl in pub/techreports/pi.
- Chrison2007. Christos G. Cassandras and Stéphane Lafortune, *Introduction to Discrete Event Systems*, Second Edition, New York, Springer, 2007
- Wonham2018. W. M. Wonham and Kai Cai, *Supervisory Control of Discrete-Event Systems*, Revised 2018.01.01
- Jean2018. Jean-Éric Pin, *Mathematical Foundations of Automata Theory*, Version of June 15, 2018
- 蒋宗礼 2013. 蒋宗礼, 姜守旭, 形式语言与自动机理论 (第 3 版), 清华大学出版社, 2013.05
- Lipschutz2007. S. Lipschutz and M. L. Lipson, *Schaum's Outline of Theory and Problems of Discrete Mathematics*, Third Edition, New York: McGraw-Hill, 2007.
- Rosen2007. K. H. Rosen, *Discrete Mathematics and Its Applications*, Seventh Edition, New York: McGraw-Hill, 2007.
- R.Su and Wonham2004. R. Su and W. M. Wonham, *Supervisor reduction for discrete-event systems*, Discrete Event Dyn. Syst., vol. 14, no. 1, pp. 31-53, Jan. 2004.
- 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*, Université Paris-Est Marne-la-Vallée 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.

Chapter 4

[蒋宗礼 2013](第 1 章绪论)

- 集合：集合的表示、集合之间的关系、集合的基本运算。
- 关系：主要介绍了二元关系相关的内容。包括等价关系、等价分类、关系合成、关系闭包。
- 递归定义与归纳证明。
- 图：无向图、有向图、树的基本概念。
- 语言与形式语言：自然语言的描述，形式语言和自动机理论的出现，形式语言和自动机理论对计算机科学与技术学科人才能力培养的作用
- 基本概念：字母表、字母、句子、字母表上的语言、语言的基本运算

4.1 集合的基础知识

4.1.1 集合之间的关系

Definition 4.1. 设 A, B 是两个集合，如果集合 A 中的元素都是集合 B 的元素，则称集合 A 是集合 B 的子集 (subset)，集合 B 是集合 A 的包集 (container)。记作 $A \subseteq B$ ，也可以记作 $B \supseteq A$ 。

由定义可知， $A \subseteq B$ 的充要条件是：对于 A 中的每一个元素 a ，均有 $a \in B$ 。为了简洁起见， P_1 是 P_2 的充要条件记为

$$P_1 \iff P_2$$

或者

$$P_2 \text{ iff } P_1$$

经常使用全称量词和存在量词： $\forall x$ 表示对所有的 x ； $\exists x$ 表示存在一个 x 。按照此约定，有

$$A \subseteq B \iff \forall x \in A, x \in B \text{ 成立,}$$

也就是

$$A \subseteq B \text{ iff } \forall x \in A, x \in B \text{ 成立。}$$

Definition 4.2. 设 A, B 是两个集合，如果 $A \subseteq B$ ，且 $\exists x \in B$ ，但 $x \notin A$ ，则称 A 是 B 的真子集 (proper subset)，记作 $A \subset B$ 。

Definition 4.3. 如果集合 A, B 含有的元素完全相同，则称集合 A 与集合 B 相等 (equivalence)，记作 $A = B$ 。

对任意集合 A, B, C ，不难得出下列结论：

1. $A = B$ iff $A \subseteq B$ 且 $B \subseteq A$ 。
2. 如果 $A \subseteq B$, 则 $|A| \leq |B|$ 。
3. 如果 $A \subset B$, 则 $|A| < |B|$ 。
4. 如果 A 是有穷集, 且 $A \supset B$, 则 $|B| > |A|$ 。
5. 如果 $A \subseteq B$, 则 $\forall x \in A$, 有 $x \in B$ 。
6. 如果 $A \subset B$, 则 $\forall x \in A$, 有 $x \in B$ 并且 $\exists x \in B$, 但 $x \notin A$ 。
7. 如果 $A \subseteq B$ 且 $B \subseteq C$, 则 $A \subseteq C$ 。
8. 如果 $A \subset B$ 且 $B \subset C$, 或者 $A \subseteq B$ 且 $B \subset C$, 或者 $A \subset B$ 且 $B \subseteq C$, 则 $A \subset C$ 。
9. 如果 $A = B$, 则 $|A| = |B|$ 。

Definition 4.4. 设 A, B 是两个集合, A 与 B 的对称差 (*symmetric difference*) 是一个集合, 该集合由属于 A 但不属于 B , 以及属于 B 但不属于 A 的所有元素组成, 记作 $A \otimes B$ 。

$$A \otimes B = \{a | a \in A \text{ 且 } a \notin B \text{ 或者 } a \notin A \text{ 且 } a \in B\}$$

显然, 对集合 A, B , 有

$$A \otimes B = (A \cup B) - (A \cap B) = (A - B) \cup (B - A)$$

\otimes 为对称差运算符, $A \otimes B$ 读作 A 对称减 B (A 与 B 的对称差)。

Definition 4.5. 设 A, B 是两个集合, A 与 B 的笛卡儿积 (*Cartesian product*) 是一个集合, 该集合该集合是由所有这样的有序对 (ab) 组成的: 其中 $a \in A, b \in B$, 记作 $A \times B$ 。

$$A \times B = \{(a, b) | a \in A \text{ 且 } b \in B\}$$

\times 为笛卡儿乘运算符。 $A \times B$ 读作 A 叉乘 B 。

对任意集合 A, B, C , 不难得出下列结论:

1. $A \times B \neq B \times A$ 。
2. $(A \times B) \times C \neq A \times (B \times C)$ 。
3. $A \times A \neq A$ 。
4. $A \times \emptyset = \emptyset$ 。
5. $A \times (B \cup C) = (A \times B) \cup (A \times C)$ 。
6. $(B \cup C) \times A = (B \times A) \cup (C \times A)$ 。
7. $A \times (B \cap C) = (A \times B) \cap (A \times C)$ 。
8. $(B \cap C) \times A = (B \times A) \cap (C \times A)$ 。
9. $A \times (B - C) = (A \times B) - (A \times C)$ 。
10. $(B - C) \times A = (B \times A) - (C \times A)$ 。
11. 当 A, B 为有穷集时, $|A \times B| = |A| |B|$ 。

Definition 4.6. 设 A 是一个集合, A 的幂集 (*power set*) 是一个集合, 该集合由 A 的所有子集组成, 记作 2^A 。

$$2^A = \{B | B \subseteq A\}$$

2^A 读作 A 的幂集。

对于任意集合 A, B , 则有下列结论:

1. $\emptyset \in 2^A$ 。

2. $\emptyset \subseteq 2^A$ 。
3. $\emptyset \subset 2^2$ 。
4. $2^\emptyset = \{\emptyset\}$ 。
5. $A \in 2^A$ 。
6. 如果 A 是有穷集合, 则 $|2^A| = 2^{|A|}$ 。
7. $2^{A \cap B} = 2^A \cap 2^B$ 。
8. if $A \subseteq B$, then $2^A \subseteq 2^B$ 。

Definition 4.7. 设 A 是论域 U 上的一个集合, A 的补集 (*complementary set*) 是一个集合, 该集合由在 U 中, 但不在 A 中的所有元素组成, 记作 \bar{A} 。

$$\bar{A} = U - A$$

\bar{A} 读作 A (关于论域 U) 的补集 (U 中子集 A 的补集)。

在实际工作和生活中, 人们都会在一定程度上讨论问题, 讨论问题的这个范围叫作论域。如果集合 A 是论域 U 上的一个集合, 则 $A \subseteq U$ 。

设 U 是论域, A, B 是 U 上的集合, 则有下列结论:

1. $\bar{\emptyset} = U$ 。
2. $\bar{U} = \emptyset$ 。
3. if $A \subseteq B$, then $\bar{B} \subseteq \bar{A}$ 。
4. $A \cup \bar{A} = U$ 。
5. $A \cap \bar{A} = \emptyset$ 。
6. $B = \bar{A} \iff A \cup B = U \& A \cap B = \emptyset$ 。
7. De Morgan 定律

$$\overline{A \cap B} = \bar{A} \cup \bar{B}。$$

$$\overline{A \cup B} = \bar{A} \cap \bar{B}。$$

4.2 关系

- 二元关系
- 递归定义与归纳证明
- 关系的闭包

4.2.1 二元关系

Definition 4.8. 设 A, B 是两个集合, 任意的 $R \subset A \times B$, R 是 A 到 B 的二元关系 (*binary relation*)。

$(a, b) \in R$, 表示 a 与 b 满足关系 R , 按照中缀形式, 也可以表示为 aRb 。

A 称为定义域 (*domain*), B 称为值域 (*range*)。

当 $A = B$ 时, 则称 R 是 A 上的二元关系。

Definition 4.9. R 是 A 上的二元关系,

1. 如果对任意一个 $a \in A$, 有 $(a, a) \in R$, 则称 R 是自反的 (*reflexive*)。
2. 如果对任意一个 $a \in A$, 有 $(a, a) \notin R$, 则称 R 是反自反的 (*irreflexive*)。
3. 如果对任意的 $a, b \in A$, 当 $(b, a) \in R$ 时, 必有 $(a, b) \in R$, 则称 R 是对称的 (*symmetric*)。
4. 如果对任意的 $a, b \in A$, 当 $(b, a) \in R$ 和 $(a, b) \in R$ 同时成立, 必有 $a = b$, 则称 R 是反对称的 (*asymmetric*)。
5. 如果对任意的 $a, b, c \in A$, 当 $(a, b) \in R$ 和 $(b, c) \in R$ 同时成立, 必有 $(a, c) \in R$, 则称 R 是传递的 (*transition*)。

条件 (1),(3),(5) 合并在一起, 叫作关系的三歧性: 自反性, 对称性, 传递性。

Example 4.1. 关系的性质。

1. $=$ 关系是自反的, 对称的, 传递的。
2. $>, <$ 关系是反自反的, 传递的。
3. \geq, \leq 关系是自反的, 反对称的, 传递的。
4. 集合之间的包含关系是自反的, 反对称的, 传递的。
5. 整数集上的模 n 同余关系是自反的, 对称的, 传递的。
6. 通常意义下的父子关系是反自反的, 非传递的。
7. 通常意义下的兄弟关系是反自反的, 传递的。
8. 通常意义下的祖先关系是反自反的, 传递的。

4.2.2 等价关系与等价类

Definition 4.10. 如果集合 A 上的二元关系 R 是自反的, 对称的, 传递的, 则称 R 是等价关系 (*equivalence relation*)。

Example 4.2. 等价关系示例。

1. 实数集上的 “ $=$ ” 关系。
2. 整数集上的模 n 同余关系。
3. 通常意义下的 “在同一个学校工作” 的关系。
4. “户口在同一省、市、自治区” 的关系。

由此, 可以考虑利用集合 S 上的等价关系 R 将 S 划分成若干个等价类。

Definition 4.11. 设 R 是集合 S 上的等价关系, 则满足如下要求的 S 的划分 $S_1, S_2, S_3, \dots, S_n, \dots$ 称为 S 关于 R 的等价划分, S_i 称为等价类 (*equivalence class*)。它们满足以下各条:

1. $S = S_1 \cup S_2 \cup S_3 \cup \dots \cup S_n \cup \dots$ 。
2. if $i \neq j$, then $S_i \cap S_j = \emptyset$ 。
3. 对任意的 i, S_i 中的任意两个元素 a, b, aRb 恒成立。
4. 对任意的 $i, j, i \neq j, S_i$ 中的任意元素 a 和 S_j 中的任意元素 b, aRb 恒不成立。

R 将 S 分成的等价类的个数称为 R 在 S 上的指数 (*index*)。有时候, R 将 S 分成有穷多个等价类, 此时称 R 具有有穷指数, 反之称为无穷指数。

Example 4.3. 等价类。

1. “ $=$ ” 关系将自然数 \mathbb{N} 分成无穷多个等价类: $\{1\}, \{2\}, \{3\}, \dots$ 。

2. 非负整数集上的模 5 同余关系将 $\{0, 1, 2, 3, \dots\}$ 分成 5 个等价类:

$\{0, 5, 10, 15, 20, \dots\}$

$\{1, 6, 11, 16, 21, \dots\}$

$\{2, 7, 12, 17, 22, \dots\}$

$\{3, 8, 13, 18, 23, \dots\}$

$\{4, 9, 14, 19, 24, \dots\}$

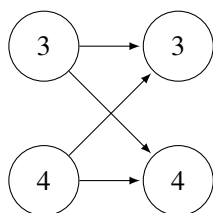
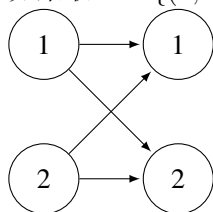
3. 某计算机学院 2001 年招收本科生 420 名, 分成 12 个班, 按同班同学关系划分, 这 420 名学生分成 12 个等价类, 每个等价类对应一个班。

Note 4.1. 值得注意的是, 给定集合 S 上的一个等价关系 R , R 就确定了 S 的一个等价划分。当给定另一个不同的等价关系时, 它会确定 S 的一个新的等价划分。

Example 4.4. 令 $S = \{1, 2, 3, 4\}$.

1. 通常意义的“=”关系将 S 分成 4 个等价类: $\{1\}, \{2\}, \{3\}, \{4\}$ 。

2. 如果取 $R = \{(1, 1), (2, 1), (1, 2), (2, 2), (3, 3), (3, 4), (4, 3), (4, 4)\}$, 则将 S 分成两个等价类: $\{1, 2\}, \{3, 4\}$ 。



4.2.3 关系的合成 (composition)

Definition 4.12. 设 $R_1 \subseteq A \times B$ 是 A 到 B 的关系, $R_2 \subseteq B \times C$ 是 B 到 C 的关系, 则 R_1 与 R_2 的合成 (composition) $R_1 \circ R_2$ 是 A 到 C 的关系。

$$R_1 \circ R_2 = \{(a, c) | \exists (a, b) \in R_1 \& (b, c) \in R_2\}$$

Note 4.2. if $\exists b \in B, (a, b) \in R_1, (b, c) \in R_2$, then

$$R_1 : A \rightarrow B \Rightarrow R_1(a) = b$$

$$R_2 : B \rightarrow C \Rightarrow R_2(b) = c$$

$$R_1 \circ R_2 = \{a, c\}$$

$$(R_1 \circ R_2)(a) = R_2(R_1(a))$$

$$= R_2(b) = c$$

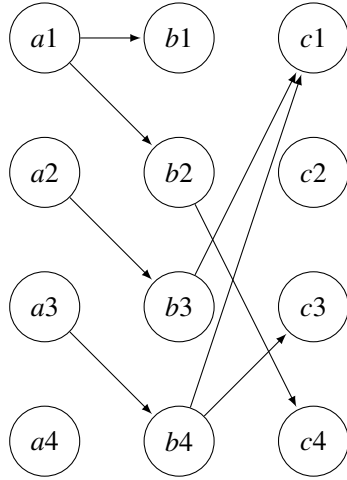
Example 4.5. 设 R_1, R_2 是集合 $\{1, 2, 3, 4\}$ 上的关系, 其中

$$R_1 = (1, 1), (1, 2), (2, 3), (3, 4)$$

$$R_2 = (2, 4), (4, 1), (4, 3), (3, 1)$$

则

$$R_1 \circ R_2 = \{(1, 4), (2, 1), (3, 1), (3, 3)\}$$



设 R_1, R_2, R_3 分别是 S 上的二元关系, 可以证明以下结论:

1. $R_1 \circ R_2 \neq R_2 \circ R_1$ 。
2. $(R_1 \circ R_2) \circ R_3 = R_1 \circ (R_2 \circ R_3)$ 。(结合律)
3. $(R_1 \cup R_2) \circ R_3 = (R_1 \circ R_3) \cup (R_2 \circ R_3)$ 。(合成对 \cup 的右分配律)
4. $R_3 \circ (R_1 \cup R_2) = (R_3 \circ R_1) \cup (R_3 \circ R_2)$ 。(合成对 \cup 的左分配律)
5. $(R_1 \cap R_2) \circ R_3 \subseteq (R_1 \circ R_3) \cap (R_2 \circ R_3)$ 。(合成对 \cap 的右分配律)
6. $R_3 \circ (R_1 \cap R_2) \subseteq (R_3 \circ R_1) \cap (R_3 \circ R_2)$ 。(合成对 \cap 的左分配律)

4.2.4 递归定义 (recursive definition) 与归纳证明

• 递归定义 (recursive definition)

- 又称为归纳定义 (inductive definition), 它来定义一个集合。
- 集合的递归定义由三个部分组成:
 - 基础 (basis): 用来定义该集合的最基本的元素。
 - 归纳 (induction): 指出用集合中的元素来构造集合的新元素的规则。
 - 极小性限定: 指出一个对象是所定义集合中的元素的充要条件是它可以通过有限次的使用基础和归纳条款中所给的规定构造出来。

• 归纳证明

- 与递归定义相对应
- 归纳证明方法包括三大步:
 - 基础 (basis): 证明最基本元素具有相应性质。
 - 归纳 (induction): 证明如果某些元素具有相应性质, 则根据这些元素用所规定的方法得到新元素也有相应的性质。

- 根据归纳法原理, 所有的元素具有相应的性质。

Example 4.6. 算术表达式的递归定义:

1. 基础 (basis): 常数是算术表达式, 变量是算术表达式;
2. 归纳 (induction): 如果 E_1, E_2 是算术表达式, 则 $+E_1, -E_1, E_1 + E_2, E_1 - E_2, E_1 * E_2, E_1 / E_2, E_1 \wedge E_2, Fun(E_1)$ 是算术表达式。其中 Fun 为函数名。
3. 只有满足 (1) 和 (2) 的式子才是算术表达式。

Definition 4.13. 设 R 是 S 上的关系, 我们递归地定义 R^n 的幂:

1. $R^0 = \{(a, a) | a \in S\}$
2. $R^i = R^{i-1}R \quad (i = 1, 2, 3, \dots)$

Example 4.7. 对有穷集合 A , 证明 $|2^A| = 2^{|A|}$ 。

证明. 设 A 为一个有穷集合, 施归纳于 $|A|$:

1. 基础 (basis): 当 $|A| = 0$ 时, 由幂集定义, $|2^A| = |\{\emptyset\}| = 1$ 。而 $2^{|A|} = 2^0 = 1$ 。所以有 $|2^A| = 2^{|A|}$ 对 $|A| = 0$ 成立。
2. 归纳 (induction): 假设 $|A| = n$ 时结论成立, 这里 $n \geq 0$, 往证当 $|A| = n + 1$ 时结论成立。

为此, 不妨设 $A = B \cup \{a\}, a \notin B$, 即

$$|2^A| = |B \cup \{a\}| = |B| + |\{a\}| = |B| + 1$$

由幂集的定义知

$$2^A = 2^B \cup \{C \cup \{a\} | C \in 2^B\}$$

由于 $a \notin B$, 所以

$$2^B \cap \{C \cup \{a\} | C \in 2^B\} = \emptyset$$

由 $\{C \cup \{a\} | C \in 2^B\}$ 的构造方法知道, 可以按如下方法构造一个一一对应的映射 $f: \{C \cup \{a\} | C \in 2^B\} \rightarrow 2^B$, 使

$$f(C \cup \{a\}) = C$$

所以

$$|\{C \cup \{a\} | C \in 2^B\}| = |2^B|$$

故

$$\begin{aligned} |2^A| &= |2^B \cup \{C \cup \{a\} | C \in 2^B\}| \\ &= |2^B| + |\{C \cup \{a\} | C \in 2^B\}| \\ &= |2^B| + |2^B| \\ &= 2|2^B| \end{aligned}$$

显然, $|B| = n$, 由归纳假设知

$$|2^B| = 2^{|B|}$$

从而有

$$|2^A| = 2|2^B| = 2 \times |2^B| = 2^{|B|+1} = 2^{|A|}$$

这就是说, 结论对 $|A| = n + 1$ 成立。

3. 由归纳法原理, 结论对任意有穷集合成立。

Example 4.8. 表达式的前缀形式是指将运算符写在前面, 后跟相应的运算对象。如: $+E_1$ 的前缀形式为 $+E_1$, $E_1 + E_2$ 的前缀形式为 $+E_1E_2$, $E_1 * E_2$ 的前缀形式为 $*E_1E_2$, $E_1 \wedge E_2$ 的前缀形式为 $\wedge E_1E_2$, $Fun(E_1)$ 的前缀形式为 $FunE_1$ 。证明 *Example 4.6* 所定义的表达式可以用这里定义的前缀形式表示。

证明. 设 E 为 *Example 4.6* 所定义的算术表达式, 现对 E 中所含的运算符 (包括函数引用) 的个数实施归纳。设 E 中含 n 个运算符。

1. 基础 (basis): 当 $n = 0$ 时, 表达式为一个常数或者变量, 结论显然成立。
2. 归纳 (induction): 假设 $n \leq k$ 时结论成立, 这里 $k \geq 0$, 往证当 $n = k + 1$ 时结论成立。

由于 E 中含有 $k + 1$ 个运算符, 所以必须是如下情况中的一种:

- a. 当 $E = +E_1$ 时, 我们知道 E_1 中的运算符个数为 k , 由归纳假设, E_1 有对应的前缀形式 F_1 , 从而 E 的前缀形式为 $+F_1$ 。
- b. 当 $E = -E_1$ 时, 类似地, E 的前缀形式为 $-F_1$ 。
- c. 当 $E = E_1 + E_2$ 时, E_1, E_2 中的运算符个数分别小于等于 k , 由归纳假设, E_1, E_2 有对应的前缀形式 F_1, F_2 , 从而 E 的前缀形式为 $+F_1F_2$ 。
- d. 对 $E = E_1 - E_2, E = E_1 * E_2, E = E_1 / E_2, E = E_1 \wedge E_2$ 的情况进行类似的讨论, 它们的前缀形式分别为: $-F_1F_2, *E_1E_2, /E_1E_2, \wedge E_1E_2$ 。
- e. 当 $E = Fun(E_1)$ 时, 我们知道 E_1 中的运算符个数为 k , 由归纳假设, E_1 有对应的前缀形式 F_1 , 从而 E 的前缀形式为 $FunF_1$ 。

综上所述, 结论对 $n = k + 1$ 成立。

3. 由归纳法原理, 结论对 *Example 4.6* 所定义的所有表达式成立。

4.2.5 关系的闭包

Definition 4.14. 设 P 是关于关系的性质的集合, 关系 R 的 P 闭包 (closure) 是包含 R 并且具有 P 中所有性质的最小关系。

Definition 4.15. 设 R 是 S 上二元关系, R 的正闭包 (positive closure) R^+ 的定义为:

- (1) $R \subseteq R^+$ 。
- (2) 如果 $(a, b), (b, c) \in R^+$, 则 $(a, c) \in R^+$ 。
- (3) 除 (1)、(2) 外, R^+ 不包含有其他任何元素。

可以证明, R^+ 具有传递性, 因此又称其为传递闭包 (transitive closure)。还可以证明, 对于任意的二元关系 R , 有

$$R^+ = R \cup R^2 \cup R^3 \cup R^4 \cup \dots$$

且当 S 为有穷集时, 有

$$R^+ = R \cup R^2 \cup R^3 \cup \dots \cup R^{|S|}$$

Definition 4.16. 设 R 是 S 上二元关系, R 的克林闭包 (Kleene closure) R^* 的定义为:

- (1) $R^0 \subseteq R^*, R \subseteq R^*$ 。
- (2) 如果 $(a, b), (b, c) \in R^*$ 则 $(a, c) \in R^*$ 。
- (3) 除 (1)、(2) 外, R^* 不再含有其他任何元素。

可以证明, R^* 具有自反性和传递性, 因此又称其为自反传递闭包 (reflexive and transitive closure)。

由定义4.15和4.16可知, 对于任意二元关系 R , 有

$$\begin{aligned} R^* &= R^0 \cup R^+ \\ &= R^0 \cup R \cup R^2 \cup R^3 \cup \dots \end{aligned}$$

而且当 S 为有穷集时:

$$R^* = R^0 \cup R \cup R^2 \cup R^3 \cup \dots \cup R^{|S|}$$

设 R_1, R_2 是 S 上的两个二元关系, 则

- (1) $\emptyset^+ = \emptyset$
- (2) $(R_1^+)^+ = R_1^+$
- (3) $(R_1^*)^* = R_1^*$
- (4) $R_1^+ \cup R_2^+ \subseteq (R_1 \cup R_2)^+$
- (5) $R_1^* \cup R_2^* \subseteq (R_1 \cup R_2)^*$

4.3 语言

4.3.1 字母表 (alphabet)

- 字母表是一个非空有穷集合, 字母表中的元素称为该字母表的一个字母 (*letter*)。又叫做符号 (*symbol*)、或者字符 (*character*)。
- 非空性
- 有穷性
- 字符的两个特性
 - 整体性 (monolith), 也叫不可分性
 - 可辨认性 (distinguishable), 也叫可去区分性
- 字母表的乘积 (product)

$$\Sigma_1 \Sigma_2 = \{ab | a \in \Sigma_1, b \in \Sigma_2\}$$

- 字母表 Σ 的 n 次幂

$$\Sigma^0 = \{\epsilon\}$$

$$\Sigma^n = \Sigma^{n-1} \Sigma$$

ϵ 是由 Σ 中的 0 个字符组成的。

- Σ 的正闭包

$$\Sigma^+ = \Sigma \cup \Sigma^2 \cup \Sigma^3 \cup \dots$$

$$\Sigma^+ = \{x | x \text{ 是 } \Sigma \text{ 中的至少一个字符连接而成的字符串}\}$$

- Σ 的克林闭包

$$\Sigma^* = \Sigma^0 \cup \Sigma^+ = \Sigma^0 \cup \Sigma \cup \Sigma^2 \cup \Sigma^3 \cup \dots$$

$$\Sigma^* = \{x | x \text{ 是 } \Sigma \text{ 中的若干个, 包括 0 个字符, 连接而成的字符串}\}$$

Example 4.9. {alphabet}

$\{a,b,c,d\}$

$\{a,b,c,\dots,z\}$

$\{0,1\}$

$\{a,a',b,b'\}$

$\{aa,ab,bb\}$

$\{\infty, \wedge, \vee, \geq, \leq\}$

Example 4.10. product

$\{0,1\}\{0,1\} = \{00,01,10,00\}$

$\{0,1\}\{a,b,c,d\} = \{0a,0b,0c,0d,1a,1b,1c,1d\}$

$\{a,b,c,d\}\{0,1\} = \{a0,a1,b0,b1,c0,c1,d0,d1\}$

$\{aa,ab,bb\}\{0,1\} = \{aa0,aa1,ab0,ab1,bb0,bb1\}$

Example 4.11. Σ^0, Σ^*

$\{0,1\}^+ = \{0,1,00,01,11,000,001,010,011,100,\dots\}$

$\{0,1\}^* = \{\varepsilon, 0,1,00,01,11,000,001,010,011,100,\dots\}$

$\{a,b,c,d\}^+ = \{a,b,c,d,aa,ab,ac,ad,ba,bb,bc,bd,\dots,aaa,aab,aac,aad,aba,abb,abc,\dots\}$

$\{a,b,c,d\}^* = \{\varepsilon, a,b,c,d,aa,ab,ac,ad,ba,bb,bc,bd,\dots,aaa,aab,aac,aad,aba,abb,abc,\dots\}$

4.3.2 句子 (sentence)/字 (word)/字符串 (string)

- 别称

句子 (sentence), (字符、符号) 行 (line), (字符、符号) 串 (string).

- 句子 (sentence)

Σ 是一个字母表, $\forall x \in \Sigma^*, x$ 叫做 Σ 上的一个句子。

- 句子相等

两个句子被认为相等的, 如果它们对应位置上的字符都对应相等。

- 句子的长度 (length)

- $\forall x \in \Sigma^*$, 句子 x 中字符出现的总个数叫做该句子的长度, 记作 $|x|$ 。

- 长度为 0 的字符串叫空句子, 记作 ε

- 串 x 的 n 次幂

$$x^0 = \varepsilon$$

$$x^n = x^{n-1}x$$

Note 4.3. 注意事项

- ε 是一个句子

• $\{\varepsilon\} \neq \emptyset$ 。这是因为 $\{\varepsilon\}$ 不是一个空集, 它是含有一个空句子的 ε 的集合。 $|\{\varepsilon\}| = 1, |\emptyset| = 0$

Example 4.12.

$$|abaabb| = 6$$

$$|bbaa| = 4$$

$$|\varepsilon| = 0$$

Example 4.13. $x=001, y=1101$

$$x^0 = y^0 = \varepsilon$$

$$x^4 = 001001001001$$

$$y^4 = 1101110111011101$$

4.3.3 并置/连结 (concatenation)

- 并置/连结 (concatenation)

- $x, y \in \Sigma^*$, x, y 的并置是由串 x 直接相接串 y 组成的。记作 xy .

- Σ^* 上的并置运算性质

1. 结合律: $(xy)z = x(yz)$
2. 左消去律: if $xy = xz$, then $y = z$
3. 右消去律: if $yx = zx$, then $y = z$
4. 惟一分解性: 存在惟一确定的 $a_1, a_2, \dots, a_n \in \Sigma$, 使得 $x = a_1 a_2 \cdots a_n$.
5. 单位元素: $\varepsilon x = x\varepsilon = x$

4.3.4 前缀与后缀

设 $x, y, z, w, v \in \Sigma^*$, 且 $x = yz, w = yv$

1. y 是 x 的前缀 (prefix)
2. 如果 $z \neq \varepsilon$, 则 y 是 x 的真前缀 (proper prefix).
3. z 是 x 的后缀 (suffix)
4. 如果 $y \neq \varepsilon$, 则 z 是 x 的真后缀 (proper suffix)
5. y 是 x 和 w 的公共前缀 (common prefix)
6. 如果 x 和 w 的任何公共前缀都是 y 的前缀, 则 y 是 x 和 w 的最大公共前缀。
7. 如果 $x = zy$ 和 $w = vy$, 则 y 是 x 和 w 的公共后缀 (common suffix)。
8. 如果 x 和 w 的任何公共后缀都是 y 的后缀, 则 y 是 x 和 w 的最大公共后缀。

Example 4.14. $\Sigma = \{a, b\}$ 上的句子 $abaabb$:

前缀: $\varepsilon, a, ab, aba, abaa, abaab, abaabb$

真前缀: $\varepsilon, a, ab, aba, abaa, abaab$

后缀: $\varepsilon, b, bb, abb, aabb, baabb, abaabb$

真后缀: $\varepsilon, b, bb, abb, aabb, baabb$

结论

1. x 的任意前缀 y 有惟一的一个后缀 z 与之对应, 使得 $x = yz$; 反之亦然。

2. x 的任意真前缀 y 有惟一的一个真后缀 z 与之对应, 使得 $x = yz$; 反之亦然。
3. $|\{w|w \text{ 是 } x \text{ 的后缀}\}| = |\{w|w \text{ 是 } x \text{ 的前缀}\}|$
4. $|\{w|w \text{ 是 } x \text{ 的真后缀}\}| = |\{w|w \text{ 是 } x \text{ 的真前缀}\}|$
5. $|\{w|w \text{ 是 } x \text{ 的前缀}\}| = |\{w|w \text{ 是 } x \text{ 的真前缀} \cup \{x\}\}|$
6. $|\{w|w \text{ 是 } x \text{ 的前缀}\}| = |\{w|w \text{ 是 } x \text{ 的真前缀} + 1\}|$
7. $|\{w|w \text{ 是 } x \text{ 的后缀}\}| = |\{w|w \text{ 是 } x \text{ 的真后缀} \cup \{x\}\}|$
8. $|\{w|w \text{ 是 } x \text{ 的后缀}\}| = |\{w|w \text{ 是 } x \text{ 的真后缀} + 1\}|$
9. 对于任意字符串 w , w 是自身的前缀, 但不是自身的真前缀; w 是自身的后缀, 但不是自身的真后缀。
10. 对于任意字符串 w , ε 是 w 的前缀, 且是 w 的真前缀; ε 是 w 的后缀, 且是 w 的真后缀。

约定

- 用小写字母表中较为靠前的字母 a, b, c, \dots 表示字母表中的字母
- 用小写字母表中较为靠后的字母 x, y, z, \dots 表示字母表中的句子 (字)
- 用 x^T 表示 x 的倒序。例如, 如果 $x = abc$, 则 $x^T = cba$

4.3.5 子串 (substring)

- 子串 (substring)
 - $w, x, y, z \in \Sigma^*$, 且 $w = xyz$, 则称 y 是 w 的子串。
- 公共子串 (common substring)
 - $t, u, v, w, x, y, z \in \Sigma^*$, 且 $t = uyv, w = xyz$, 则称 y 是 t 和 w 的公共子串 (common substring)。如果 y_1, y_2, \dots, y_n 是 t 和 w 的公共子串, 且 $\max\{|y_1|, |y_2|, \dots, |y_n|\} = |y_j|$, 则称 y_j 是 t 和 w 的最大公共子串。
 - 两个串的最大公共子串并不一定是惟一的。

4.3.6 语言 (language)

$\forall \in \Sigma^*, L$ 称为字母表 Σ 上的一个语言 (language), $\forall x \in L, x$ 叫做 L 的一个句子 (sentence)/字 (word)/字符串 (string)。

Example 4.15. $\Sigma = \{0, 1\}$ 上的不同语言 $\{00, 11\}$

$\{0, 1\}$

$\{0, 1, 00, 11\}$

$\{0, 1, 00, 11, 01, 10\}$

$\{00, 11\}^*$

$\{01, 10\}^*$

$\{00, 01, 10, 11\}^*$

$\{0\}\{0, 1\}^*\{1\}$

$\{0, 1\}^*\{111\}\{0, 1\}^*$

4.3.7 语言的乘积 (product)

$L_1 \subseteq \Sigma_1^*, L_2 \subseteq \Sigma_2^*$, 语言 L_1 与 L_2 的乘积 (product) 是一个语言, 该语言定义为:

$$L_1 L_2 = \{xy | x \in L_1, y \in L_2\}$$

是字母表 $\Sigma_1 \cup \Sigma_2$ 上的语言。

Example 4.16. $\Sigma = \{0, 1\}$

$$L_1 = \{0, 1\}$$

$$L_2 = \{00, 01, 10, 11\}$$

$$L_3 = \{0, 1, 00, 01, 10, 11, 000, \dots\} = \Sigma^+$$

$$L_4 = \{\epsilon, 0, 1, 00, 01, 10, 11, 000, \dots\} = \Sigma^*$$

$$L_5 = \{0^n | n \geq 1\}$$

$$L_6 = \{0^n 1^n | n \geq 1\}$$

$$L_7 = \{1^n | n \geq 1\}$$

$$L_8 = \{0^n 1^m | n, m \geq 1\}$$

$$L_9 = \{0^n 1^n 0^n | n \geq 1\}$$

$$L_{10} = \{0^n 1^m 0^k | n, m, k \geq 1\}$$

$$L_{11} = \{x | x \in \Sigma^+ \text{ 且 } x \text{ 中 } 0 \text{ 和 } 1 \text{ 的个数相同}\}$$

- 上述所有语言都是 L_4 的子集 (子语言);
- L_1, L_2 是有穷语言; 其他为无穷语言; 其中 L_1 是 Σ 上的所有长度为 1 的字组成的语言, L_2 是 Σ 上的所有长度为 2 的字组成的语言;
- L_3, L_4 分别是 Σ 的正闭包和克林闭包;
- $L_5 L_7 \neq L_6$, 但 $L_5 L_7 = L_8$; 同样 $L_9 \neq L_{10}$, 但我们有 $L_6 \subset L_5 L_7, L_9 \subset L_{10}$.
- L_6 中的 word 中的 0 和 1 的个数是相同的, 并且所有的 0 在所有的 1 的前面; L_{11} 中的 word 中虽然保持着 0 和 1 的个数相同, 但它并没有要求所有的 0 在所有的 1 的前面。例如, $0101, 1100 \in L_{11}$, 但是 $0101 \notin L_6$ 。而对 $\forall x \in L_6$, 有 $x \in L_{11}$ 。所以 $L_6 \subset L_{11}$ 。

Example 4.17. x^T example

$$1. \{x | x = x^T, x \in \Sigma\}$$

$$2. \{xx^T | x \in \Sigma^+\}$$

$$3. \{xx^T | x \in \Sigma^*\}$$

$$4. \{xwx^T | x, w \in \Sigma^+\}$$

$$5. \{xx^T w | x, w \in \Sigma^+\}$$

- 幂 $\forall L \in \Sigma^*, L$ 的 n 次幂是一个语言, 该语言定义为

$$1. \text{ 当 } n = 0 \text{ 时, } L^n = \{\epsilon\}$$

$$2. \text{ 当 } n \geq 1 \text{ 时, } L^n = L^{n-1}L$$

- 正闭包

$$L^+ = L \cup L^2 \cup L^3 \cup L^4 \cup \dots$$

- 克林闭包

$$L^* = L^0 \cup L \cup L^2 \cup L^3 \cup L^4 \cup \dots$$

4.4 Exercise and Solution

Exercise 4.1. 设 L 是 Σ 上的一个语言, Σ^* 上的二元关系 R_L 定义为: 对任给的 $x, y \in \Sigma^*$, 如果对于 $\forall z \in \Sigma^*$, 均有 $xz \in L$ 与 $yz \in L$ 同时成立或者同时不成立, 则 $xR_L y$ 。请证明 R_L 是 Σ^* 上的一个等价关系。将 R_L 称为由语言 L 所确定的等价关系。即

$$xR_L y \Leftrightarrow (\forall z \in \Sigma^*, xz \in L \Leftrightarrow yz \in L)$$

实际上, R_L 还有另外一个性质: 如果对任给的 $x, y \in \Sigma^*$, 当 $xR_L y$ 成立时必有 $xzR_L yz$, 对 $\forall z \in \Sigma^*$ 都成立。这将被称为 R_L 的“右不变”性, 你能证明此性质成立吗?

Solution 4.1. 分两步证明 R_L 是“右不变”的等价关系。

证明.

1. R_L 是等价关系。

- 自反性: $\forall x \in \Sigma^*$, 显然对于 $\forall z \in \Sigma^*$, xz 要么是 L 的字符串, 要么不是 L 的字符串。由 R_L 的定义知, $xR_L x$
- 对称性: 不难看出, $xR_L y \Leftrightarrow (\forall z \in \Sigma^*, xz \in L \Leftrightarrow yz \in L) \Leftrightarrow yR_L x$
- 传递性: 设 $xR_L y, yR_L z$ 。

\therefore (由 R_L 的定义知)

$$xR_L y \Leftrightarrow (\forall w \in \Sigma^*, xw \in L \Leftrightarrow yw \in L)$$

$$yR_L z \Leftrightarrow (\forall w \in \Sigma^*, yw \in L \Leftrightarrow zw \in L)$$

\therefore

$$\forall w \in \Sigma^*, xw \in L \Leftrightarrow yw \in L \text{ and } yw \in L \Leftrightarrow zw \in L$$

\Rightarrow

$$\forall w \in \Sigma^*, xw \in L \Leftrightarrow zw \in L$$

故

$xR_L z$, 即 R_L 是等价关系。

2. R_L 是右不变的。

设 $xR_L y$ 。由 R_L 的定义, 对 $\forall w, v \in \Sigma^*$, $xwv \in L \Leftrightarrow ywv \in L$ 。注意到 v 的任意性, 知 $xwR_L yw$ 所以, R_L 是右不变的等价关系。

Exercise 4.2. 设 $\{0,1\}^*$ 上的语言 $L = \{0^n 1^n | n \geq 0\}$, 请给出 $\{0,1\}^*$ 的关于 L 所确定的等价关系 R_L 的等价分类。

Solution 4.2. 根据 **Exercise 4.1** 对 R_L 的定义及其证明, 考虑 R_L 对 $\{0,1\}^*$ 的等价分类时, 主要需根据语言 L 的结构, 分析哪些串按照 L 的要求具有相同的特征。

1. 取 $0^n 1^n, 0^m 1^m \in L, 0^n 1^n \varepsilon \in L, 0^m 1^m \varepsilon \in L$ 同时成立, 但是, 对所有 $x \in \{0,1\}^+, 0^n 1^n x \in L, 0^m 1^m x \in L$ 同时不成立, 符合 R_L 的定义。所以, L 中的元素是属于同一类的。即 $0^n 1^n R_L 0^m 1^m \Leftrightarrow (0^n 1^n \in L \Leftrightarrow 0^m 1^m \in L), n, m \geq 0$ 。

2. 分析是否还有其他的元素与 L 中的元素属于同一类。根据 L 的结构, 一种类型的串不含子串 10, 这种串可以表示成 $0^k 1^h, k \neq h$; 另一种是含子串 10 (L 的字符串不含这种子串), 这种串可以表示成 $x10y$ 。显然, $01\epsilon \in L$, 但是 $0^k 1^h \notin L (k \neq h), x10y \notin L (x, y \in \{0, 1\}^*)$ 。根据等价分类的性质, $\{0, 1\}^*$ 中的不在 L 中的串与在 L 的串不在同一个等价类中。

3. 考察不含子串 10 的串。这些串有如下几种形式:

- a. $0^n, n \geq 1$ 。
- b. $1^n, n \geq 1$ 。
- c. $0^m 1^n, m, n \geq 1, \text{且} m > n$ 。
- d. $0^m 1^n, m, n \geq 1, \text{且} m < n$ 。

对于 $0^n, n \geq 1$, 1^n 接在它后面时, 构成串 $0^n 1^n$ 。显然, 当 $m \neq n$ 时, $0^n 1^n \in L$, 但 $0^m 1^n \notin L$ 。所以, 0^n 和 0^m 一定不在同一个等价类中。

类似的讨论可知, 对于 $0^n, n \geq 1$:

0^n 不可能与形如 $0^m 1^n (m, n \geq 1, \text{且} m < n)$ 的串在同一等价类中;

0^n 不可能与含有子串 10 的串在同一等价类中。

下面再考虑形如 0^k 的串和形如 $0^m 1^n (m, n \geq 1, \text{且} m > n)$ 是否可能在同一等价类中。

注意到当 $m - n = h$ 时,

$$0^h 1^h \in L, 0^m 1^n 1^h \in L$$

同时成立, 但是当 $n \geq 1, x = 01^{h+1}$ 时 ($x \neq 1^h$),

$$0^h x \in L, 0^m 1^n x \notin L$$

成立。所以, 对应 $h > 0$, 令

$$[h] = \{0^m 1^n | m - n = h \text{ 且 } n \geq 1\}$$

$[h]$ 中的元素在同一个等价类中, 而且所有其他的元素都不在这个等价类中。实际上, 当 $h = 0$ 时, 有

$$[0] = L$$

4. 形如 1^m 的串和形如 $0^m 1^n (m, n \geq 1 \text{ 且 } m < n)$ 的串应该在同一等价类中。事实上, 对于 $\{0, 1\}^*$ 中的任意字符串 x ,

$$1^m x \notin L, 0^m 1^n x \notin L (m, n \geq 1 \text{ 且 } m < n)$$

恒成立。所以, 这些字符串在同一个等价类中。

5. 所有含子串 10 的串在同一等价类中。事实上, 设 y, z 是含子串 10 的串, 对于 $\{0, 1\}^*$ 中的任意字符串 x ,

$$yx \notin L, zx \notin L (m, n \geq 1 \text{ 且 } m < n)$$

恒成立。所以, 这些字符串在同一个等价类中。

6. 形如 1^m 的串和含子串 10 的串应该在同一等价类中。事实上, 设 y 是含有子串 10 的串, 对于 $\{0, 1\}^*$ 中的任意字符串 x ,

$$1^m x \notin L, yx \notin L (m \geq 1)$$

恒成立。所以, 这些字符串在同一个等价类中。

综上所述, R_L 确定的 $\{0,1\}^*$ 的等价分类为

$$[10] = \{x10y | x, y \in \{0,1\}^*\} \cup \{0^m 1^n | n - m \geq 1\}$$

$$[0] = \{0^m 1^n | n - m = 0\} = \{0^n 1^n | n \geq 0\}$$

$$[1] = \{0^m 1^n | n - m = 1\}$$

$$[2] = \{0^m 1^n | n - m = 2\}$$

\vdots

$$[h] = \{0^m 1^n | n - m = h\}$$

\vdots

$$\{0\}$$

$$\{00\}$$

\vdots

$$\{0^n\}$$

\vdots

其中, n, m 均为非负整数。

Exercise 4.3. 使用归纳法证明:

对字母表 Σ 的任意字符串 x , x 的前缀有 $|x| + 1$ 个。

Solution 4.3. 证明. 设 $x \in \Sigma^*$, 现对 x 的长度施归纳。为了叙述方便用 $prefix(x)$ 表示字符串 x 的所有前缀组成的集合。

当 $|x| = 0$ 时, 有 $x = \varepsilon$, 由字符串的前缀定义知, $prefix(x) = \{x\}$ 。

ε 就是 x 的唯一前缀, 而

$$\begin{aligned} |prefix(x)| &= |\{\varepsilon\}| \\ &= 1 \\ &= 0 + 1 \\ &= |x| + 1 \end{aligned}$$

所以, 结论对 $|x| = 0$ 成立。

设 $|x| = n$ 时结论成立, $n \geq 0$ 。即

$$|prefix(x)| = |x| + 1$$

现在考察 $|x| = n + 1$ 的情况。为了叙述方便, 不妨设 $x = ya$, 其中 $|y| = n, a \in \Sigma$ 。由归纳假设,

$$|prefix(y)| = |y| + 1$$

首先证明 y 的任何前缀都是 x 的前缀。事实上, 设

$$|prefix(y)| = \{u_1, u_2, \dots, u_n\}$$

对于 $\forall u \in prefix(y)$, 根据前缀的定义, 存在 $v \in \Sigma^*$, 使得 $uv = y$, 注意到 $uva = x$, 所以, u 也是 x 的前缀, 它对应 x 的后缀为 va 。

再注意到 $x = ya$, 所以, 一方面, 对于 $\forall u \in prefix(x)$, 均有

$$u \neq x$$

从而

$$x \notin prefix(y),$$

然而, 由

$$x\varepsilon = x$$

可知, x 是 x 的一个前缀。另一方面, 由 $x = ya$ 知道, 如果 u 是 x 的一个前缀, v 是 u 对应的 x 的后缀, 则有如下两种情况:

(1) $v \geq 1$, 此时必有 $u \in \text{prefix}(y)$ 。

(2) $|v| = 0$, 此时必有 $v = \varepsilon$ 并且 $u = u\varepsilon = ya = x$ 。

由此可见,

$$\begin{aligned}\text{prefix}(x) &= \text{prefix}(y) \cup \{x\} \\ &= \{u_1, u_2, \dots, u_n, x\}\end{aligned}$$

由 $x \notin \text{prefix}(y)$ 可知,

$$\begin{aligned}|\text{prefix}(x)| &= |\text{prefix}(y) \cup \{x\}| \\ &= |\text{prefix}(y)| + |\{x\}| \\ &= |\text{prefix}(y)| + 1\end{aligned}$$

再由归纳假设,

$$\begin{aligned}|\text{prefix}(x)| &= |\text{prefix}(y)| + 1 \\ &= |y| + 1 + 1 \\ &= |x| + 1\end{aligned}$$

表明结论对 $|x| = n + 1$ 成立。

由归纳法原理, 结论对于任意 $x \in \Sigma^*$ 成立。 □

另一种往证 $|x|=n+1$ 时结论成立方法:

设 $|x| = n$ 时结论成立, $n \geq 0$ 。即 $|\text{prefix}(x)| = |x| + 1$

现在考察 $|x| = n + 1$ 的情况。为了叙述方便, 不妨设 $x = ya$, 其中 $|y| = n, a \in \Sigma \text{ 且 } a \neq y$ 。

$x = ya \Rightarrow$

$$\begin{aligned}\text{prefix}(x) &= \text{prefix}(ya) \\ &= \{\varepsilon\} \cup \text{prefix}(y) \cup \text{prefix}(\{ya\}) && \text{(前缀定义)} \\ &= \{\varepsilon\} \cup \text{prefix}(y) \cup \text{prefix}(a) && (\text{prefix}(\{ya\}) = \{\varepsilon\} \cup \text{prefix}(y) \cup \text{prefix}(a)) \\ &= \{\varepsilon\} \cup \text{prefix}(y) \cup \{a\} && (\text{prefix}(a) = \{\varepsilon, a\})\end{aligned}$$

$\because \varepsilon \in \text{prefix}(y), a \neq y,$

$\therefore |\text{prefix}(x)| = |\text{prefix}(y)| + |\{a\}| = |\text{prefix}(y)| + 1$

由归纳假设有, $|\text{prefix}(y)| = |y| + 1$

所以, $|\text{prefix}(x)| = |\text{prefix}(y)| + 1 = (|y| + 1) + 1 = |x| + 1$

表明结论对 $|x| = n + 1$ 成立。

Exercise 4.4. 设 $\Sigma = \{a, b\}$, 求字符串 $aaaaabbbbba$ 的所有前缀的集合, 后缀的集合, 真前缀的集合, 真后缀的集合。

Solution 4.4. 见表 (4.1)。

从表中可以看出, 按照前缀与后缀的对应关系构成原来的字符串的对应关系, 前缀和后缀是一一对应的, 但是, 按照这种对应关系, 真前缀和真后缀不是一一对应的。不过, 若将真后缀和真前缀中的空串 ε 去掉, 则这种一一对应关系是仍然存在的。

另外, 从对这一问题的讨论知, 并不是所有的字符串都有真前缀和真后缀。

表 4.1: $\Sigma = \{a, b\}$, $aaaaabbbba$ 的前缀, 后缀, 真前缀, 真后缀的集合

前缀长度	前缀	是真前缀	对应后缀	是真后缀
0	ε	✓	$aaaaabbbba$	
1	a	✓	$aaaabbbba$	✓
2	aa	✓	$aaabbbba$	✓
3	aaa	✓	$aabbbba$	✓
4	$aaaa$	✓	$abbbba$	✓
5	$aaaaa$	✓	$bbba$	✓
6	$aaaaab$	✓	$bbba$	✓
7	$aaaaabb$	✓	bba	✓
8	$aaaaabbb$	✓	ba	✓
9	$aaaaabbbb$	✓	a	✓
10	$aaaaabbbba$		ε	✓

Exercise 4.5. 设 L_1, L_2, L_3, L_4 分别是 $\Sigma_1, \Sigma_2, \Sigma_3, \Sigma_4$ 上的语言, 能否说 L_1, L_2, L_3, L_4 是某个字母表 Σ 上的语言? 如果能, 请问这个字母表 Σ 是什么样的?

Solution 4.5. 可以说 L_1, L_2, L_3, L_4 是同一个字母表 Σ 上的语言。这里

$$\Sigma = \Sigma_1 \cup \Sigma_2 \cup \Sigma_3 \cup \Sigma_4$$

Exercise 4.6. 设 L_1, L_2, L_3, L_4 分别是 $\Sigma_1, \Sigma_2, \Sigma_3, \Sigma_4$ 上的语言, 证明下列等式成立。

$$(L_1 \cup L_2 \cup L_3 \cup L_4)^* = (L_1^* L_2^* L_3^* L_4^*)^*$$

Solution 4.6. 证明. 考虑到语言就是一系列字符串的集合, 所以, 证明两个语言相等, 实际上就是证明相应的两个集合相等。因此, 为证明

$$(L_1 \cup L_2 \cup L_3 \cup L_4)^* = (L_1^* L_2^* L_3^* L_4^*)^*$$

就是证明下列 (1), (2) 式同时成立。

$$(L_1 \cup L_2 \cup L_3 \cup L_4)^* \subseteq (L_1^* L_2^* L_3^* L_4^*)^* \quad (1)$$

$$(L_1 \cup L_2 \cup L_3 \cup L_4)^* \supseteq (L_1^* L_2^* L_3^* L_4^*)^* \quad (2)$$

首先证明 (1) 式成立。为此, 设

$$x \in (L_1 \cup L_2 \cup L_3 \cup L_4)^*$$

从而存在非负整数 n 和 $x_1, x_2, \dots, x_n, \{x_1, x_2, \dots, x_n\} \subseteq (L_1 \cup L_2 \cup L_3 \cup L_4)$, 使得

$$x = x_1 x_2 \cdots x_n$$

注意到 $\{x_1, x_2, \dots, x_n\} \subseteq (L_1 \cup L_2 \cup L_3 \cup L_4)$, 所以, 对于 $1 \leq j \leq n$

$$x_j \in L_1, x_j \in L_2, x_j \in L_3, x_j \in L_4$$

中至少一个成立, 这表明

$$x_j \in L_1^*, x_j \in L_2^*, x_j \in L_3^*, x_j \in L_4^*$$

中至少一个成立, 再注意到

$$\varepsilon \in L_1^*, \varepsilon \in L_2^*, \varepsilon \in L_3^*, \varepsilon \in L_4^*$$

并且

$$L_1 \subseteq L_1^*, L_2 \subseteq L_2^*, L_3 \subseteq L_3^*, L_4 \subseteq L_4^*$$

使得

$$L_1 \subseteq L_1^* L_2^* L_3^* L_4^*, L_2 \subseteq L_1^* L_2^* L_3^* L_4^*, L_3 \subseteq L_1^* L_2^* L_3^* L_4^*, L_4 \subseteq L_1^* L_2^* L_3^* L_4^*$$

从而

$$x_j \in L_1^* L_2^* L_3^* L_4^*$$

故

$$x = x_1 x_2 \cdots x_n \in (L_1^* L_2^* L_3^* L_4^*)^n$$

亦即

$$x = x_1 x_2 \cdots x_n \in (L_1^* L_2^* L_3^* L_4^*)^*$$

所以 (1) 式成立。

类似易证 (2) 式亦成立。

综上所述 (1), (2) 式同时成立。所以, $(L_1 \cup L_2 \cup L_3 \cup L_4)^* = (L_1^* L_2^* L_3^* L_4^*)^*$ 。

□

Exercise 4.7. 设 L_1, L_2, L_3, L_4 分别是 $\Sigma_1, \Sigma_2, \Sigma_3, \Sigma_4$ 上的语言, 证明下列等式成立否。

$$L_2(L_1 L_2 \cup L_2)^* L_1 = L_1 L_1^* L_2(L_1 L_1^* L_2)^*$$

Solution 4.7. 证明. 此式不成立, 仅举一个反例即可完成证明。

令 $L_1 = \{a\}, L_2 = \{b\}$, 此时,

$$L_2(L_1 L_2 \cup L_2)^* L_1 = \{b\}(\{a\}\{b\} \cup \{b\})^* \{a\} = \{b\}\{ab, b\}^* \{a\}$$

它含的串都是以 $\{b\}\{a\}$ 。

$$L_1 L_1^* L_2(L_1 L_1^* L_2)^* = \{a\}\{a\}^* \{b\}(\{a\}\{a\}^* \{b\})^* = \{a\}^+ \{b\}(\{a\}^+ \{b\})^*$$

它含的串都是以 $\{a\}$ 开头的串。

所以, 此式不成立。

□

Exercise 4.8. 设 $\Sigma = \{0, 1\}$ 请给出 Σ 上的下列语言的形式化表示。

1. 所有长度为偶数的串。
2. 所有含有 3 个连续 0 的串。
3. 所有的倒数第 10 个字符是 0 的串。

Solution 4.8. $\Sigma = \{0, 1\}$

1. 所有长度为偶数的串。可以用以下任一种表示方法 (含 $\varepsilon, |\varepsilon| = 0$, 认为是偶数):

- a. $(\{0, 1\}\{0, 1\})^*$
- b. $(\{00, 01, 10, 11\})^*$
- c. $\{00, 01, 10, 11\} \cup (\{0, 1\}\{0, 1\})^*$

2. 所有含有 3 个连续 0 的串。

$$\{0, 1\}^* 000 \{0, 1\}^*$$

3. 所有的倒数第 10 个字符是 0 的串。

$\{0,1\}^*0\{0,1\}\{0,1\}\{0,1\}\{0,1\}\{0,1\}\{0,1\}\{0,1\}\{0,1\}\{0,1\}$

Chapter 5

[蒋宗礼 2013](第 3 章有穷状态自动机)

主要内容

- 确定的有穷状态自动机 (DFA)
 - 作为对实际问题的抽象、直观物理模型、形式定义, DFA 接受的句子、语言, 状态转移图。
- 不确定的有穷状态自动机 (NFA)
 - 定义;
 - NFA 与 DFA 的等价性;
- 带空移动的有穷状态自动机 ($\epsilon - NFA$)
 - 定义。
 - $\epsilon - NFA$ 与 DFA 的等价性。
- FA 是正则语言的识别器
 - 正则文法 (RG) 与 FA 的等价性。
 - 相互转换方法。
 - 带输出的有穷状态自动机。
 - 双向有穷状态自动机。
- 重点: DFA 的概念, DFA 、 NFA 、 $\epsilon - NFA$ 、 RG 之间的等价转换思路与方法。
- 难点: 对 DFA 概念的理解, DFA 、 RG 的构造方法, RG 与 FA 的等价性证明。

5.1 语言的识别

设文法 G 有如下产生式:

$$S \rightarrow aA|aB$$

$$A \rightarrow aA|c$$

$$B \rightarrow aB|d$$

分析句子 (word) $aaad$ 的过程中可能需要回溯。

$S \Rightarrow aA$ 使用产生式 $S \rightarrow aA$ (5.1)

$\Rightarrow aaA$ 使用产生式 $A \rightarrow aA$ (5.2)

$\Rightarrow aaaA$ 使用产生式 $A \rightarrow aA$ (5.3)

$\Rightarrow aaaaA$ 使用产生式 $A \rightarrow aA$, error! return to (5.3) step. (5.4)

$\Rightarrow aaac$ 使用产生式 $A \rightarrow c$, error! return to (5.3) step. (5.5)

\dots (5.6)

$\Rightarrow aB$ 使用产生式 $S \rightarrow aB$ (5.7)

\dots (5.8)

(5.9)

每一步推导有两种不同的产生式可供选择, 共有 $2^4 = 16$ 种“需要考虑的”不同推导。
推导和归约中的回溯问题将对系统的效率产生极大的影响!

• 识别系统 (模型)

1. 系统具有有穷个状态, 不同的状态代表不同的意义。按照实际的需要, 系统可以在不同的状态下完成规定的任务。
2. 我们可以将输入字符串中出现的字符汇集在一起构成一个字母表。系统处理的所有字符串都是这个字母表上的字符串。
3. 系统在任何一个状态 (当前状态) 下, 从输入字符串中读入一个字符, 根据当前状态和读入的这个字符转到新的状态。当前状态和新的状态可以是同一个状态, 也可以是不同的状态; 当系统从输入字符串中读入一个字符后, 它下一次再读时, 会读入下一个字符。这就是说, 相当于系统维持有一个读写指针, 该指针在系统读入一个字符后指向输入串的下一个字符。
4. 系统中有一个状态, 它是系统的开始状态, 系统在这个状态下开始进行某个给定句子的处理。
5. 系统中还有一些状态表示它到目前为止所读入的字符构成的字符串是语言的一个句子, 把所有将系统从开始状态引导到这种状态的字符串放在一起构成一个语言, 该语言就是系统所能识别的语言。
6. 相应的物理模型
 - a. 一个右端无穷的输入带。
 - b. 一个有穷状态控制器 (finite state control, FSC)。
 - c. 一个读头。
7. 系统的每一个动作由三个节拍构成:
 - a. 读入读头正注视的字符;
 - b. 根据当前状态和读入的字符改变有穷控制器的状态;
 - c. 将读头向右移动一格。

系统识别语言 $\{a^n c | n \geq 1\} \cup \{a^n d | n \geq 1\}$ 的字符串中状态的变化 (figure 5.1)

有穷状态自动机的物理模型

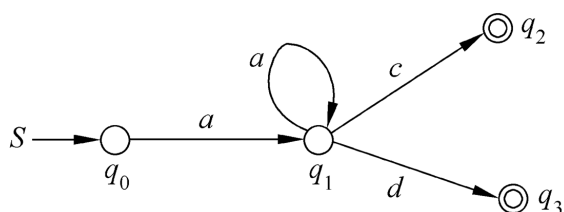
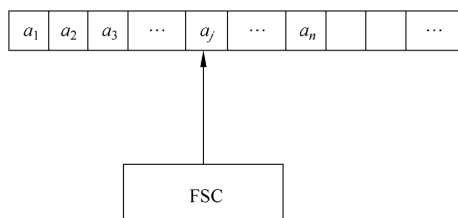
图 5.1: 系统识别语言 $\{a^n c | n \geq 1\} \cup \{a^n d | n \geq 1\}$ 的字符串中状态的变化

图 5.2: 有穷状态自动机的物理模型

5.2 有穷状态自动机 (finite automaton,FA)

Definition 5.1. 有穷状态自动机 (finite automaton,FA) M 是一个五元组

$$M = (Q, \Sigma, \delta, q_0, F)$$

Q : 状态的非空有穷集合。 $\forall q \in Q, q$ 称为 M 的一个状态 (state)。

Σ : 输入字母表 (Input alphabet)。输入字符串都是 Σ 上的字符串。

q_0 : $q_0 \in Q$, 是 M 的开始状态 (initial state), 也可叫做初始状态或者启动状态。

δ : 状态转移函数 (transition function), 有时候又叫做状态转换函数或者移动函数。 $\delta: Q \times \Sigma \rightarrow Q$, 对 $\forall (q, a) \in Q \times \Sigma, \delta(q, a) = p$ 表示: M 在状态 q 读入字符 a , 将状态变成 p , 并将读头向右移动一个带方格而指向输入字符串的下一个字符。

F : $F \subseteq Q$, 是 M 的终止状态 (final state) 集合。 $\forall q \in F, q$ 称为 M 的终止状态, 又称为接受状态 (accept state)。

Note 5.1. 应当指出的是, 虽然将 F 中的状态称为, 终止状态, 并不是说 M 一旦进入这种状态就终止了, 而是说 M 一旦在处理完输入字符串时到达这种状态, M 就接受当前处理的字符串。所以, 有时又称终止状态为接受状态。

将 δ 扩充为

$$\hat{\delta}: Q \times \Sigma^* \rightarrow Q$$

对于任意的 $q \in Q, w \in \Sigma^*, a \in \Sigma$, 定义

1. $\hat{\delta}(q, \epsilon) = q$
2. $\hat{\delta}(q, wa) = \delta(\hat{\delta}(q, w), a)$

$$\begin{aligned} \hat{\delta}(q, a) &= \hat{\delta}(q, \epsilon a) \\ &= \delta(\hat{\delta}(q, \epsilon), a) \\ &= \delta(q, a) \end{aligned}$$

两值相同, 不用区分这两个符号。

确定的有穷状态自动机

由于对于任意的 $q \in Q, a \in \Sigma, \delta(q, a)$ 均有确定的值, 所以, 将这种 FA 称为确定的有穷状态自动机 (*deterministic finite automaton, DFA*)

M 接受 (识别) 的语言

Definition 5.2. 设 $M = (Q, \Sigma, \delta, q_0, F)$ 是一个 FA。对于 $\forall x \in \Sigma^*$ 如果 $\delta(q_0, x) \in F$, 则称 x 被 M 接受, 如果 $\delta(q_0, x) \notin F$, 则称 M 不接受 x 。

$$L(M) = \{x | x \in \Sigma^*, \text{且 } \delta(q_0, x) \in F\}$$

称为由 M 接受 (识别) 的语言

Example 5.1.

$$\begin{aligned} M_1 = & (\{q_0, q_1, q_2\}, \{0\}, \\ & \{\delta(q_0, 0) = q_1, \delta(q_1, 0) = q_2, \delta(q_2, 0) = q_1\}, \\ & q_0, \{q_2\}) \\ M_2 = & (\{q_0, q_1, q_2, q_3\}, \{0, 1, 2\}, \\ & \{\delta(q_0, 0) = q_1, \delta(q_1, 0) = q_2, \delta(q_2, 0) = q_3, \delta(q_3, 0) = q_3 \\ & \delta(q_0, 1) = q_3, \delta(q_1, 1) = q_3, \delta(q_2, 1) = q_3, \delta(q_3, 1) = q_3, \\ & \delta(q_0, 2) = q_3, \delta(q_1, 2) = q_3, \delta(q_2, 2) = q_3, \delta(q_3, 2) = q_3\}, \\ & q_0, \{q_2\}) \quad q_3 \text{ is a trap/dump state.} \end{aligned}$$

不难看出: M_1, M_2 所接受的语言为:

$$L(M_1) = L(M_2) = \{0^{2n} | n \geq 1\}$$

Definition 5.3. 如果 $L(M_1) = L(M_2)$, 则称 M_1 与 M_2 等价。

Example 5.2. 构造一个 DFA, 它接受的语言为

$$\{x000y | x, y \in \{0, 1\}^*\}$$

- q_0 : M 的启动状态;
- q_1 : M 读到了一个 0, 这个 0 可能是子串 “000” 的第 1 个 0;
- q_2 : M 在 q_1 后紧接着又读到了一个 0, 这个 0 可能是子串 “000” 的第 2 个 0;
- q_3 : M 在 q_2 后紧接着又读到了一个 0, 发现输入字符串含有子串 “000”; 因此, 这个状态应该是终止状态。
- $\delta(q_0, 1) = q_0$: M 在 q_0 读到了一个 1, 它需要继续在 q_0 “等待” 可能是子串 “000” 的第 1 个 0 的输入字符 0;
- $\delta(q_1, 1) = q_0$: M 在刚刚读到了一个 0 后, 读到了一个 1, 表明在读入这个 1 之前所读入的 0 并不是子串 “000” 的第 1 个 0, 因此, M 需要重新回到状态 q_0 , 以寻找子串 “000” 的第 1 个 0;
- $\delta(q_2, 1) = q_0$: M 在刚刚发现了 00 后, 读到了一个 1, 表明在读入这个 1 之前所读入的 00 并不是子串 “000” 的前两个 0, 因此, M 需要重新回到状态 q_0 , 以寻找子串 “000” 的第 1 个 0;
- $\delta(q_3, 0) = q_3$: M 找到了子串 “000”, 只用读入该串的剩余部分。

- $\delta(q_3, 1) = q_3$: M 找到了子串 “000”, 只用读入该串的剩余部分。

$$\begin{aligned} M = & (\{q_0, q_1, q_2, q_3\}, \{0, 1\}, \\ & \{\delta(q_0, 0) = q_1, \delta(q_1, 0) = q_2, \delta(q_2, 0) = q_3, \\ & \delta(q_0, 1) = q_0, \delta(q_1, 1) = q_0, \delta(q_2, 1) = q_0, \\ & \delta(q_3, 0) = q_3, \delta(q_3, 1) = q_3\}, \\ & q_0, \{q_3\}) \end{aligned}$$

see: figure(5.3),table(5.1)

表 5.1: 状态转移表

状态说明	状态	输入字符	
		0	1
开始状态	q_0	q_1	q_0
	q_1	q_2	q_0
	q_2	q_3	q_0
终止状态	q_3	q_3	q_3

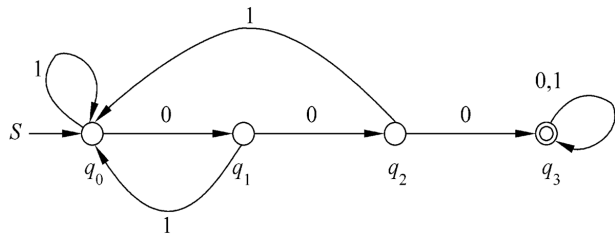


图 5.3: 识别语言 $\{x000y|x, y \in \{0, 1\}^*\}$ 的 DFA

Example 5.3. 构造一个 DFA , 它接受的语言为

$$\{x000|x \in \{0, 1\}^*\}$$

see: figure(5.4)

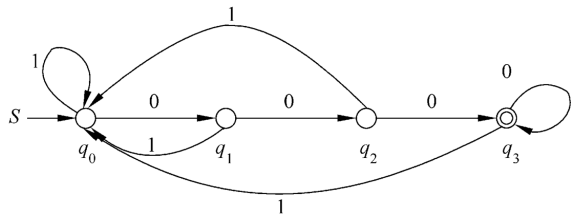


图 5.4: 识别语言 $\{x000|x \in \{0, 1\}^*\}$ 的 DFA

Note 5.2. 几点值得注意:

1. 定义 FA 时, 常常只给出 FA 相应的状态转移图就可以了。
2. 对于 DFA 来说, 并行的弧按其上的标记字符的个数计算, 对于每个顶点来说, 它的出度恰好等于输入字母表中所含的字符的个数。
3. 不难看出, 字符串 x 被 $FA M$ 接受的充分必要条件是, 在 M 的状态转移图中存在一条从开始状态到某一个终止状态的有向路, 该有向路上从第 1 条边到最后一条边的标记依次并置 (连接) 而构成的字符串 x 。简称此路的标记为 x 。
4. 一个 FA 可以有多个终止状态。

Definition 5.4. 即时描述 (instantaneous description, ID)

设 $M = (Q, \Sigma, \delta, q_0, F)$ 是一个 FA ,

- $x, y \in \Sigma^*, \delta(q_0, x) = q, xqy$ 称为 M 的一个即时描述 (instantaneous description, ID), 表示 xy 是 M 正在处理的一个字符串, x 引导 M 从 q_0 启动并到达状态 q , M 当前正注视着 y 的首字符。
- 如果 $xqay$ 是 M 的一个即时描述, 且 $\delta(q, a) = p$, 则 $xqay \vdash_M xapy$ 。
表示 M 在状态 q 时已经处理完 x 并且读头正指向输入字符 a , 此时它读入 a 并转入状态 p , 将读头向右移动一格指向 y 的首字符。
- \vdash_M 看成是 M 的所有即时描述集合上的二元关系, 并用 $\vdash_M^n, \vdash_M^*, \vdash_M^+$ 分别表示 $(\vdash_M)^n, (\vdash_M)^*, (\vdash_M)^+$, 按照二元关系合成的意义, 有:

M 存在即时描述 $\alpha_1, \alpha_2, \alpha_{n-1}$, 使得

$$\alpha \vdash_M \alpha_1, \alpha_1 \vdash_M \alpha_2, \dots, \alpha_{n-1} \vdash_M \beta$$

$\alpha \vdash_M^n \beta$: 表示 M 从即时描述 α 经过 n 次移动到达即时描述 β 。

当 $n=0$ 时, 有 $\alpha = \beta$, 即 $\alpha \vdash_M^0 \alpha$

$\alpha \vdash_M^+ \beta$: 表示 M 从即时描述 α 经过至少 1 次移动到达即时描述 β 。

$\alpha \vdash_M^* \beta$: 表示 M 从即时描述 α 经过若干步移动到达即时描述 β 。

当意义清楚时, 我们将符号 $\vdash_M, \vdash_M^n, \vdash_M^*, \vdash_M^+$ 中的 M 省去, 分别用 $\vdash, \vdash^n, \vdash^*, \vdash^+$ 表示。

Example 5.4. 图 (5.5) 的 DFA 到即时描述 (instantaneous description, ID) 的转换

$$\begin{aligned}
 q_0 101010001 &\vdash 1q_0 010010001 \\
 &\vdash 10q_1 10010001 \\
 &\vdash 101q_0 0010001 \\
 &\vdash 1010q_1 010001 \\
 &\vdash 10100q_2 10001 \\
 &\vdash 101001q_0 0001 \\
 &\vdash 1010010q_1 001 \\
 &\vdash 10100100q_2 01 \\
 &\vdash 101001000q_3 1 \\
 &\vdash 1010010001q_0
 \end{aligned}$$

即

$$q_0 101010001 \vdash_{10} 1010010001 q_0$$

$$q_0 101010001 \vdash_+ 1010010001 q_0$$

$$q_0 101010001 \vdash_* 1010010001 q_0$$

不难证明, 对于 $x \in \Sigma^*$,

$$q_0 x 1 \vdash^+ x 1 q_0$$

$$q_0 x 10 \vdash^+ x 10 q_1$$

$$q_0 x 100 \vdash^+ x 100 q_2$$

$$q_0 x 000 \vdash^+ x 000 q_3$$

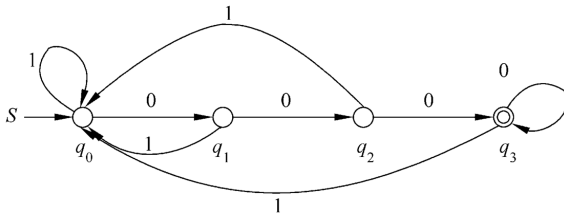


图 5.5: 识别语言 $\{x000|x \in \{0,1\}^*\}$ 的 DFA

Definition 5.5. 设 $M = (Q, \Sigma, \delta, q_0, F)$ 是一个 FA, 对 $\forall q \in Q$, 能引导 FA 从开始状态 q_0 到达 q 的字符串的集合为:

$$\text{set}(q) = \{x | x \in \Sigma^*, \delta(q_0, x) = q\}$$

Note 5.3. 根据定义5.5, DFA M 按照语言的特点给出了 Σ^* 的一个划分, 这种划分相当于 Σ^* 上的一个等价分类, M 的每个状态实际上对应着相应的等价类。这就告诉我们, 用一个状态去表示一个等价类是考虑问题的一个有效思路。

Example 5.5. 对图5.5所给的 DFA 中的所有 q , 求 $\text{set}(q)$ 。

$$\text{set}(q_0) = \{x | x \in \Sigma^*, x = \varepsilon \text{ 或者 } x \text{ 以 } 1 \text{ 结尾}\}$$

$$\text{set}(q_1) = \{x | x \in \Sigma^*, x = 0 \text{ 或者 } x \text{ 以 } 10 \text{ 结尾}\}$$

$$\text{set}(q_2) = \{x | x \in \Sigma^*, x = 00 \text{ 或者 } x \text{ 以 } 100 \text{ 结尾}\}$$

$$\text{set}(q_3) = \{x | x \in \Sigma^*, x \text{ 以 } 000 \text{ 结尾}\}$$

$$\text{set}(q_4) = \{x | x \in \Sigma^*, x \text{ 以 } 001 \text{ 结尾}\}$$

这 5 个集合是两两互不相交。而且这 5 个集合的并正好就是该 DFA 的输入字母表 $\{0,1\}$ 的克林闭包。这就是说, 这 5 个集合是 $\{0,1\}^*$ 的一个划分。依照这个划分, 可以定义一个等价关系, 在同一集合中的字符串满足此等价关系, 不在同一集合中的字符串不满足此等价关系。

一般地, 对于任意一个 DFA, $M = (Q, \Sigma, \delta, q_0, F)$ 我们可以按照如下方式定义关系 R_M :

对 $\forall x, y \in \Sigma^*, x R_M y \Leftrightarrow \exists q \in Q$, 使得 $x \in \text{set}(q)$ 和 $y \in \text{set}(q)$ 同时成立。

按照这个定义所得到的关系实际上是 Σ^* 上的一个等价关系。利用这个关系, 可以将 Σ^* 划分成不多于 $|Q|$ 个等价类。

Example 5.6. 构造一个 *DFA*，它接受的语言为 $\{0^n 1^m 2^k | n, m, k \geq 1\}$ 。 q_0 : M 的启动状态；

q_1 : M 读到至少一个 0，并等待读更多的 0；

q_2 : M 读到至少一个 0 后，读到了至少一个 1，并等待读更多的 1；

q_3 : M 读到至少一个 0 后跟至少一个 1 后，并且接着读到了至少一个 2。

- 先设计“主体框架”，见图5.6
- 再补充细节，见图5.7

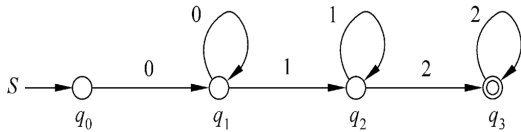


图 5.6: 接受语言 $\{0^n 1^m 2^k | n, m, k \geq 1\}$ 的 *DFA* 的主体框架

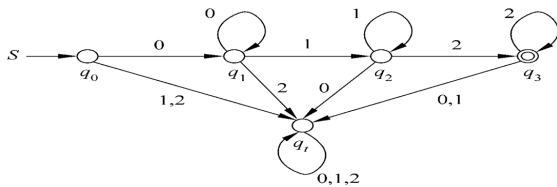


图 5.7: 接受语言 $\{0^n 1^m 2^k | n, m, k \geq 1\}$ 的 *DFA*

本例中有以下几点值得注意：

1. 当 *FA* 一旦进入状态 q_i ，它就无法离开此状态。所以， q_i 相当于一个陷阱状态 (*trap*、*dump*)。一般地，我们将陷阱状态用作在其他状态下发现输入串不可能是该 *FA* 所识别的语言的句子时进入的状态。在此状态下，*FA* 读完输入串中剩余的字符。
2. 在构造一个识别给定语言的 *FA* 时，用画图的方式比较方便、直观。我们可以先根据语言的主要特征画出该 *FA* 的“主体框架”，然后再去考虑画一些细节要求的内容。
3. *FA* 的状态具有一定的记忆功能：不同的状态对应于不同的情况，由于 *FA* 只有有穷个状态，所以，在识别一个语言的过程中，如果有无穷种情况需要记忆，我们肯定是无法构造出相应的 *FA* 的。如对语言 $\{0^n 1^n | n \geq 1\}$ ，当扫描到 n 个 0 时，需要去寻找 n 个 1，由于 n 有无穷多个，所有需要记忆的 0 的个数有无穷多种。因此，无法构造出接受语言 $\{0^n 1^n | n \geq 1\}$ 的 *FA*。也就是说，该语言不说属于 *FA* 可以接受的语言类。

Example 5.7. 构造一个 *DFA*，它接受的语言为 $\{x | x \in \{0, 1\}^*, \text{且当把 } x \text{ 看成二进制数时, } x \text{ 模 } 3 \text{ 与 } 0 \text{ 同余}\}$ 。

根据定义5.5，*DFA* M 按照语言的特点给出了 Σ^* 的一个划分，这种划分相当于 Σ^* 上的一个等价分类， M 的每个状态实际上对应着相应的等价类。这就告诉我们，用一个状态去表示一个等价类是考虑问题的一个有效思路。

题目要求的是 x 模 3 与 0 同余， x 除以 3 的余数只有 3 种：0, 1, 2；任意一个 x 都不例外，因此可以考虑用 3 个状态与这 3 个等价类相联系。

q_0 : 对应除以 3 余数为 0 的 x 组成的等价类，表示十进制值 $(3n + 0 | n \in \mathbb{N})$ ；

q_1 : 对应除以 3 余数为 1 的 x 组成的等价类，表示十进制值 $(3n + 1 | n \in \mathbb{N})$ ；

q_2 : 对应除以 3 余数为 2 的 x 组成的等价类, 表示十进制值 $(3n+2|n \in \mathbb{N})$;

此外, x 作为一个二进制数, x 是非空的字符串, 即 $\varepsilon \notin \{x\}$, 所以还需要一个开始状态。

q_s : M 的开始状态。

q_s : 在此状态下读入 0 时, 有 $x=0$, 所以应该进入状态 q_0 ; 读入 1 时, 有 $x=1$, 所以应该进入状态 q_1 。

即: $\delta(q_s, 0) = q_0$; $\delta(q_s, 1) = q_1$ 。

q_0 : 能引导 M 到达此状态的 x 除以 3 余 0, 所以有: $x = 3n + 0$ 。

读入 0 时, 引导 M 到达下一个状态的字符串为 $x0, x0 = 2(3n+0) = 3 \times 2 \times n + 0$ 。这表明, $x0$ 应该属于 q_0 对应的等价类, 所以, M 在 q_0 状态下读入 0 后, 应该继续保持在状态 q_0 , 即 $\delta(q_0, 0) = q_0$;

读入 1 时, M 到达下一个状态的字符串为 $x1, x1 = 2(3 \times n + 0) + 1 = 3 \times 2n + 1$ 。这表明, $x1$ 应该属于 q_1 对应的等价类, 所以, M 在 q_0 状态下读入 1 后, 应该转到状态 q_1 , 即 $\delta(q_0, 1) = q_1$;

q_1 : 能引导 M 到达此状态的 x 除以 3 余 1, 所以有: $x = 3n + 1$ 。

读入 0 时, 引导 M 到达下一个状态的字符串为 $x0, x0 = 2(3n+1) = 3 \times 2n + 2$ 。所以: $\delta(q_1, 0) = q_2$;

读入 1 时, 引导 M 到达下一个状态的字符串为 $x1, x1 = 2(3n+1) + 1 = 3 \times 2n + 2 + 1 = 3(2n+1)$ 。所以 $\delta(q_1, 1) = q_0$ 。

q_2 : 能引导 M 到达此状态的 x 除以 3 余 2, 所以: $x = 3n + 2$ 。

读入 0 时, 引导 M 到达下一个状态的字符串为 $x0, x0 = 2(3n+2) = 3 \times 2n + 4 = 3(2n+1) + 1$ 。所以 $\delta(q_2, 0) = q_1$;

读入 1 时, 引导 M 到达下一个状态的字符串为 $x1, x1 = 2(3n+2) + 1 = 3 \times 2n + 4 + 1 = 3(2n+1) + 2$ 。所以, $\delta(q_2, 1) = q_2$ 。

接受的语言 $\{x|x \in \{0,1\}^*, \text{且当把 } x \text{ 看成二进制数时, } x \text{ 模 } 3 \text{ 与 } 0 \text{ 同余}\}$ 的 DFA。如图 5.8。

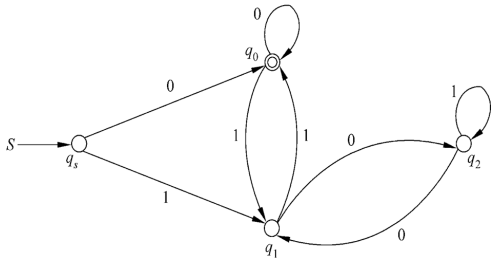


图 5.8: 接受的语言 $\{x|x \in \{0,1\}^*, \text{且当把 } x \text{ 看成二进制数时, } x \text{ 模 } 3 \text{ 与 } 0 \text{ 同余}\}$ 的 DFA

Example 5.8. 构造一个 DFA, 它接受的语言 $L = \{x|x \in \{0,1\}^*,$

且对 x 中任意一个长度不大于 5 的子串 $a_1a_2 \dots a_n, a_1 + a_2 + \dots + a_n \leq 3, n \leq 5\}$ 。

- 输入串 $a_1a_2 \dots a_i \dots a_{i+4}a_{i+5} \dots a_m$
- 当 $i = 1, 2, 3$, 也就是 M 读到输入串的第 1, 2, 3 个字符时, 它需要将这些字符记下来。因为 $a_1 \dots a_i$ 可能需要用来判定输入串的最初 4~5 个字符组成的子串是否满足语言的要求。
- 当 $i = 4, 5$, 也就是 M 读到输入串的第 4, 5 个字符时, 在 $a_1 + a_2 + \dots + a_i \leq 3$ 的情况下, M 需要将 $a_1a_2 \dots a_i$ 记下来; 在 $a_1 + a_2 + \dots + a_i > 3$ 时, M 应该进入陷阱状态 q_t 。
- 当 $i = 6$, 也就是 M 读到输入串的第 6 个字符时, 以前读到的第 1 个字符 a_1 就没用了, 此时它要看 $a_2 + a_3 + \dots + a_6 \leq 3$ 是否成立, M 需要将 $a_2a_3 \dots a_6$ 记下来; 在 $a_1 + a_2 + \dots + a_i > 3$ 时, M 应该进入陷阱状态 q_t 。
- 当 M 完成对子串 $a_1a_2 \dots a_i \dots a_{i+4}$ 的考察, 并发现它满足语言的要求时, M 记下来的是 $a_i \dots a_{i+4}$, 此时它读入输入串的第 $i+5$ 个字符 a_{i+5} , 以前读到的第 i 个字符 a_i 就没有用了, 此时它要看 $a_{i+1} + a_{i+2} + \dots + a_{i+5} \leq 3$

是否成立, 如果成立, M 需要将 $a_{i+1}, a_{i+2}, \dots, a_{i+5}$ 记下来; 在 $a_{i+1} + a_{i+2} + \dots + a_{i+5} > 3$ 时, M 应该进入陷阱状态 q_t 。

- M 需要记忆的内容是有限多种 (共计 $1 + 2 + 4 + 8 + 15 + 26 + 1 = 57$ 个状态), 分别用不同的状态对应要记忆的不同内容:

1. 什么都未读入—— $2^0 = 1$ 种;
2. 记录有 1 个字符—— $2^1 = 2$ 种;
3. 记录有 2 个字符—— $2^2 = 4$ 种;
4. 记录有 3 个字符—— $2^3 = 8$ 种;
5. 记录有 4 个字符—— $2^4 - 1 = 15$ 种;
6. 记录有 5 个字符—— $2^5 - 6 = 26$;
7. 记录当前的输入串不是句子 (word)——1 种。

状态设置, 为了便于理解, 直接要用记忆的内容来区分这些状态:

$q[\varepsilon]$: M 还未读入任何字符;

q_t : 陷阱状态;

$q[a_1 a_2 \dots a_i]$: M 记录有 i 个字符 $1 \leq i \leq 5$; $a_1, a_2, \dots, a_i \in \{0, 1\}$ 。

取 $DFA \ M = (Q, \{0, 1\}, \delta, q[\varepsilon], F)$

$F\{q[\varepsilon] \cup \{q[a_1 a_2 \dots a_i] \mid a_1, a_2, \dots, a_i \in \{0, 1\} \text{ 且 } 1 \leq i \leq 5 \text{ 且 } a_1 + a_2 + \dots + a_i \leq 3\}$

$Q = \{q_t \cup F\}$

$\delta(q[\varepsilon], a_1) = q[a_1]$

$\delta(q[a_1], a_2) = q[a_1 a_2]$

$\delta(q[a_1 a_2], a_3) = q[a_1 a_2 a_3]$

$\delta(q[a_1 a_2 a_3], a) = \begin{cases} q[a_1 a_2 a_3 a] & \text{如果 } a_1 + a_2 + a_3 + a \leq 3 \\ q_t & \text{如果 } a_1 + a_2 + a_3 + a > 3 \end{cases}$

$\delta(q[a_1 a_2 a_3 a_4], a) = \begin{cases} q[a_1 a_2 a_3 a_4 a] & \text{如果 } a_1 + a_2 + a_3 + a_4 + a \leq 3 \\ q_t & \text{如果 } a_1 + a_2 + a_3 + a_4 + a > 3 \end{cases}$

$\delta(q[a_1 a_2 a_3 a_4 a_5], a) = \begin{cases} q[a_1 a_2 a_3 a_4 a_5 a] & \text{如果 } a_2 + a_3 + a_4 + a_5 + a \leq 3 \\ q_t & \text{如果 } a_2 + a_3 + a_4 + a_5 + a > 3 \end{cases}$

$\delta(q[a_t], a_1) = q_t$

以上各式中, $a, a_1, a_2, a_3, a_4, a_5 \in \{0, 1\}$ 。

5.3 NFA

Definition 5.6. 不确定的有穷状态自动机 (non-deterministic finite automaton, NFA) M 是一个五元组

$$M = (Q, \Sigma, \delta, q_0, F)$$

- Q, Σ, q_0, F 的意义同 DFA 。

- $Q \times \Sigma \rightarrow 2^Q$, 对 $\forall (q, a) \in Q \times \Sigma, \delta(q, a) = \{p_1, p_2, \dots, p_m\}$ 表示 M 在状态 q 读入字符 a , 可以选择地将状态变成 p_1 , 或者 p_2, \dots , 或者 p_m , 并将渡头 3 向右移动一个带方格而指向输入字符串的下一个字符。

Note 5.4. FA 的状态转移图, FA 的状态对应的等价类, FA 的即时描述对 NFA 都有效。

将 δ 扩充为 $\hat{\delta}: Q \times \Sigma \rightarrow 2^Q$, 对于任意的 $q \in Q, w \in \Sigma^*, a \in \Sigma$, 定义

1. $\hat{\delta}(q, \varepsilon) = q$
2. $\hat{\delta}(q, wa) = \{p | r \in (q, w), \text{使得 } p \in \delta(r, a)\}$

$$\begin{aligned}\hat{\delta}(q, a) &= \hat{\delta}(q, \varepsilon a) \\ &= \{p | \exists r \in (q, \varepsilon), p \in \delta(r, a)\} \\ &= \{p | \exists r \in \{q\}, p \in \delta(r, a)\} \\ &= \{p | p \in \delta(q, a)\} \\ &= \delta(q, a)\end{aligned}$$

和关于 DFA 的结论一样, 两值相同, 也不用区分这两个符号。

进一步扩充 δ 的定义域: $\delta: 2^Q \times \Sigma^* \rightarrow 2^Q$. 对任意的 $P \subseteq Q, w \in \Sigma^*$

$$\delta(P, w) = \bigcup_{q \in P} \delta(q, w)$$

由于, 对 $\forall (q, w) \in Q \times \Sigma^*$

$$\delta(\{q\}, w) = \bigcup_{q \in \{q\}} \delta(q, w) = \delta(q, w)$$

所以, 不一定严格地区分 δ 的第 1 个分量是一个状态还是一个含有一个元素的集合。

对任意的 $q \in Q, w \in \Sigma^*, a \in \Sigma$:

$$\delta(q, wa) = \delta(\delta(q, w), a)$$

对输入字符串 $a_1 a_2 \dots a_n$

$$\delta(q, a_1 a_2 \dots a_n) = \delta((\dots \delta(\delta(q, a_1), a_2), \dots), a_n)$$

Definition 5.7. 设 $M = (Q, \Sigma, \delta, q_0, F)$ 是一个 NFA , 对于 $\forall x \in \Sigma^*$, 如果 $\delta(q_0, w) \cap F \neq \emptyset$, 则称 x 被 M 接受, 如果 $\delta(q_0, w) \cap F = \emptyset$, 则称 M 不接受 x 。

$$L(M) = \{x | x \in \Sigma^* \text{ 且 } \delta(q_0, w) \cap F \neq \emptyset\}$$

称为由 M 接受 (识别) 的语言。

关于 FA 的等价定义 (5.3) 也适应 NFA

对于一个输入字符, NFA 与 DFA 的差异是前者可以进入若干个状态, 而后者只能进入一个惟一的状态。虽然从 DFA 看待问题的角度来说, NFA 在某一时刻同时进入若干个状态, 但是, 这若干个状态合在一起的“总效果”相当于它处于这些状态对应的一个“综合状态”。因此, 我们考虑让 DFA 用一个状态去对应 NFA 的一组状态。

$NFA \quad M_1 = (Q, \Sigma, \delta_1, q_0, F_1)$ 与 $DFA \quad M_2 = (Q_2, \Sigma, \delta_2, q'_0, F_2)$ 的对应关系:

NFA 从开始状态 q_0 启动, 我们就让相应的 DFA 从状态 $[q_0]$ 启动。所以 $q'_0 = [q_0]$ 。

对于 NFA 的一个状态组 $\{q_1, q_2, \dots, q_n\}$, 如果 NFA 在此状态组时读入字符 a 后可以进入状态组 $\{p_1, p_2, \dots, p_m\}$, 则让相应的 DFA 在状态 $[q_1, q_2, \dots, q_n]$ 读入字符 a 时, 进入状态 $[p_1, p_2, \dots, p_m]$ 。

Theorem 5.1. *NFA 与 DFA 等价。*

证明. 显然只需证明对于任给的 *NFA*, 存在与之等价的 *DFA*. 为此, 设有 *NFA*: $M_1 = (Q, \Sigma, \delta_1, q_0, F_1)$

1. 构造与 M_1 等价的 *DFA* M_2

取 *DFA* $M_2 = (Q_2, \Sigma, \delta_2, [q_0], F_2)$

$Q_2 = 2^Q$

¹ $F_2 = \{[p_1, p_2, \dots, p_m] \mid \{p_1, p_2, \dots, p_m\} \subseteq Q \& \{p_1, p_2, \dots, p_m\} \cap F_1 \neq \emptyset\}$

$\delta_2([q_1, q_2, \dots, q_n], a) = [p_1, p_2, \dots, p_m] \Leftrightarrow \delta_1(\{q_1, q_2, \dots, q_n\}, a) = \{p_1, p_2, \dots, p_m\}$

证明 $\delta_1(q_0, x) = \{p_1, p_2, \dots, p_m\} \Leftrightarrow \delta_2([q_0], x) = [p_1, p_2, \dots, p_m]$.

设 $x \in \Sigma^*$, 施归纳于 $|x|$

$x = \varepsilon, \delta_1(q_0, \varepsilon) = \{q_0\}, \delta_2([q_0], \varepsilon) = [q_0]$

设当 $|x| = n$ 时结论成立。下面证明当 $|x| = n + 1$ 时结论也成立。不妨设 $x = wa, |w| = n, a \in \Sigma$

$$\begin{aligned} \delta_1(q_0, wa) &= \delta_1(\delta_1(q_0, w), a) \\ &= \delta_1(\{q_1, q_2, \dots, q_n\}, a) \\ &= \{p_1, p_2, \dots, p_m\} \end{aligned}$$

由归纳假设,

$\delta_1(q_0, w) = \{q_1, q_2, \dots, q_n\} \Leftrightarrow \delta_2([q_0], w) = [q_1, q_2, \dots, q_n]$

根据 δ_2 的定义,

$\delta_2([q_1, q_2, \dots, q_n], a) = [p_1, p_2, \dots, p_m] \Leftrightarrow \delta_1(\{q_1, q_2, \dots, q_n\}, a) = \{p_1, p_2, \dots, p_m\}$

所以,

$$\begin{aligned} \delta_2([q_0], wa) &= \delta_2(\delta_2([q_0], w), a) \\ &= \delta_2(\{q_1, q_2, \dots, q_n\}, a) \\ &= [p_1, p_2, \dots, p_m] \end{aligned}$$

故, 如果 $\delta_1([q_0], wa) = \{p_1, p_2, \dots, p_m\}$ 则必有 $\delta_2([q_0], wa) = [p_1, p_2, \dots, p_m]$.

由上述推导可知, 反向的推导也成立。这就是说, 结论对 $|x| = n + 1$ 也成立。

由归纳法原理, 结论对 $x \in \Sigma^*$ 成立。

2. 证明 $L(M_1) = L(M_2)$

设 $x \in L(M_1)$, 且 $\delta_1(q_0, x) = \{p_1, p_2, \dots, p_m\}$, 从而 $\delta_1(q_0, x) \cap F_1 \neq \emptyset$, 这就是说, $\{p_1, p_2, \dots, p_m\} \cap F_1 \neq \emptyset$, 由 F_2 的定义, $[p_1, p_2, \dots, p_m] \in F_2$.

再由 (1) 知, $\delta_2([q_0], x) = [p_1, p_2, \dots, p_m]$.

所以, $x \in L(M_2)$. 故 $L(M_1) \subseteq L(M_2)$.

反过来推, 可得 $L(M_2) \subseteq L(M_1)$.

从而 $L(M_1) = L(M_2)$ 得证。

综上所述, 定理成立。 □

5.4 Exercise and Solution

Exercise 5.1. 构造识别下列语言的 *DFA*(给出相应的 *DFA* 的形式描述或者画出它们的状态转移图)。

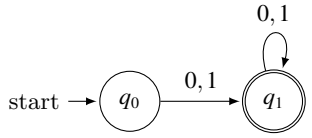
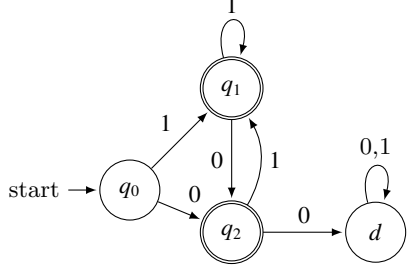
¹ 在表示状态集合时暂时将习惯的“{”和“}”改为用“[”和“]”, 表示把集合的元素汇集成整体。

1. $\{0,1\}^+$ 。
2. $\{x|x \in \{0,1\}^+ \text{ 且 } x \text{ 中不含形如 } 00 \text{ 的子串}\}$ 。
3. $\{x|x \in \{0,1\}^+ \text{ 且当把 } x \text{ 看成二进制数时, } x \text{ 模 } 5 \text{ 与 } 3 \text{ 同余,}$
要求当 x 为 0 时, $|x| = 1$, 且当 $x \neq 0$ 时, x 的首字符为 1}。
4. $\{x|x \in \{0,1\}^+ \text{ 且 } x \text{ 以 } 0 \text{ 开头以 } 1 \text{ 结尾}\}$ 。

Solution 5.1. 构造相应语言的 DFA

1. 构造 $\{0,1\}^+$ 语言的 DFA, 见下表 (5.2)。
2. $\{x|x \in \{0,1\}^+ \text{ 且 } x \text{ 中不含形如 } 00 \text{ 的子串}\}$ 语言的 DFA, 见下表 (5.2)。

表 5.2: 构造相应语言的 DFA

<p>构造 $\{0,1\}^+$ 语言的 DFA。 本题的关键是保证接受的串的长度至少为 1。</p>	
<p>构造 $\{x x \in \{0,1\}^+ \text{ 且 } x \text{ 中不含形如 } 00 \text{ 的子串}\}$ 语言的 DFA。 构造要点是, 自动机启动并读入一个字符后, 就将“精力”集中在考察是否出现 00 子串上, 一旦发现子串 00, 就进入陷阱状态。</p>	

3. $\{x|x \in \{0,1\}^+ \text{ 且当把 } x \text{ 看成二进制数时, } x \text{ 模 } 5 \text{ 与 } 3 \text{ 同余,}$
要求当 x 为 0 时, $|x| = 1$, 且当 $x \neq 0$ 时, x 的首字符为 1}。

DFA 见图 (5.9), 状态转移表见 (5.3,5.4)。构造要点如下:

- a. 以 0 开头的串都不能被 DFA 接受, 包括字符串 0, 所以, 如果 DFA 在启动状态读入的符号为 0, 则直接进入陷阱状态。
- b. 该 DFA 共有 7 种状态: 开始状态、陷阱状态、终止状态各一个。在状态转移图中, 终止状态和其余 4 种状态构成最大的强连通子图。
- c. 模 5 的等价类: $5n, 5n+1, 5n+2, 5n+3, 5n+4$, 其中 $n \geq 0$

4. $\{x|x \in \{0,1\}^+ \text{ 且 } x \text{ 以 } 0 \text{ 开头以 } 1 \text{ 结尾}\}$ 。

DFA 见图 (5.10)。构造要点如下:

- a. 启动时, 只要考虑开始符号是否为 0。

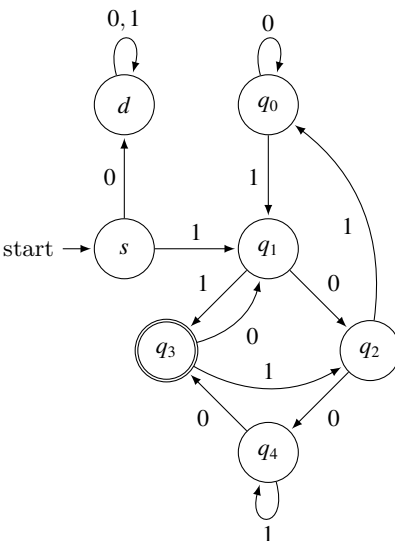


图 5.9: $\{x|x \in \{0,1\}^+ \text{ 且当把 } x \text{ 看成二进制数时, } x \text{ 模 } 5 \text{ 与 } 3 \text{ 同余, 要求当 } x \text{ 为 } 0 \text{ 时, } |x| = 1, \text{ 且当 } x \neq 0 \text{ 时, } x \text{ 的首字符为 } 1\}$ 语言对应的 *DFA*

表 5.3: $\{x|x \in \{0,1\}^+ \text{ 且当把 } x \text{ 看成二进制数时, } x \text{ 模 } 5 \text{ 与 } 3 \text{ 同余, 要求当 } x \text{ 为 } 0 \text{ 时, } |x| = 1, \text{ 且当 } x \neq 0 \text{ 时, } x \text{ 的首字符为 } 1\}$ 语言对应的 *DFA*, 状态 q_0, \dots, q_4 转移表

state	q_0	q_1	q_2	q_3	q_4
意义	$5n$	$5n+1$	$5n+2$	$5n+3(\text{marked})$	$5n+4$
输入字符 0	q_0	q_2	q_4	q_1	q_3
$sx, x=0$	$(5n) \times 2 = 10n$	$(5n+1) \times 2 = 10n+2$	$(5n+2) \times 2 = 10n+4$	$(5n+3) \times 2 = 10n+6$	$(5n+4) \times 2 = 10n+8$
输入字符 1	q_1	q_3	q_0	q_2	q_4
$sx, x=1$	$(5n) \times 2 + 1 = 10n+1$	$(5n+1) \times 2 + 1 = 10n+3$	$(5n+2) \times 2 + 1 = 10n+5$	$(5n+3) \times 2 + 1 = 10n+7$	$(5n+4) \times 2 + 1 = 10n+9$

表 5.4: $\{x|x \in \{0,1\}^+ \text{ 且当把 } x \text{ 看成二进制数时, } x \text{ 模 } 5 \text{ 与 } 3 \text{ 同余, 要求当 } x \text{ 为 } 0 \text{ 时, } |x| = 1, \text{ 且当 } x \neq 0 \text{ 时, } x \text{ 的首字符为 } 1\}$ 语言对应的 *DFA*, 状态 s, d 转移表

state	s	d
意义	开始状态	<i>dump/trap</i> 状态
输入字符 0	d	d
$sx, x=0$	以 0 开头的串都不能被 <i>DFA</i> 接受, 进入陷阱状态	在陷阱状态输入 0,1, 仍然离不开陷阱状态
输入字符 1	q_1	d
$sx, x=1$	余 1, 进入 q_1	在陷阱状态输入 0,1, 仍然离不开陷阱状态

- b. 以 1 开头的字符串都是不可接受的。
- c. 在读入一个字符 0 之后, 当读到 1 时, 可将这个 1 先当成结尾的 1, 如果是, 就停止并接受; 如果不是, 就继续向前扫描。
- d. 被接受串的长度至少为 2。

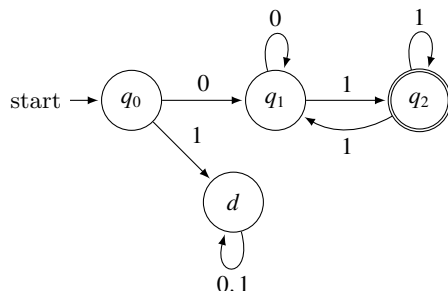


图 5.10: $\{x|x \in \{0,1\}^+ \text{ 且 } x \text{ 以 } 0 \text{ 开头以 } 1 \text{ 结尾}\}$ 语言对应的 DFA

5. $\{x|x \in \{0,1\}^* \text{ 且如果 } x \text{ 以 } 1 \text{ 结尾, 则它的长度为偶数; 如果 } x \text{ 以 } 0 \text{ 结尾, 则长度为奇数}\}$ 。

DFA 见图 (5.11)。构造要点如下:

- a. 语言根据 word 长度的奇偶性对字符串的结尾提出要求, 因此, 它将 $\{0,1\}^+$ 中的字符串分成 4 个等价类, 这 4 个等价类依次为:
- [奇 0]: $\{x|x \in \{0,1\}^+ \text{ 不仅长度为奇数, 而且以 } 0 \text{ 结尾}\}$, 用 $[o0]$ 表示此等价类及其对应的状态, 是终止状态。
 - [奇 1]: $\{x|x \in \{0,1\}^+ \text{ 不仅长度为奇数, 而且以 } 1 \text{ 结尾}\}$, 用 $[o1]$ 表示此等价类及其对应的状态。
 - [偶 0]: $\{x|x \in \{0,1\}^+ \text{ 不仅长度为偶数, 而且以 } 0 \text{ 结尾}\}$, 用 $[e0]$ 表示此等价类及其对应的状态。
 - [偶 1]: $\{x|x \in \{0,1\}^+ \text{ 不仅长度为偶数, 而且以 } 1 \text{ 结尾}\}$, 用 $[e1]$ 表示此等价类及其对应的状态, 是终止状态。
- b. ϵ 不属于上述任何一个等价类, 所以她自己构成独立地构成一个等价类, 而且它是语言的句子 ($\epsilon \in L(M)$), 该等价类对应的状态为终止状态, 用 $[\epsilon]$ 表示此等价类及其对应的状态。
- c. $M = (\{[\epsilon], [o0], [o1], [e0], [e1]\}, \{0,1\}, \delta, \{[\epsilon], [o0], [e1]\})$, 其中 δ 是状态转移函数, 见表 (5.5)。
- 如果将长度 0 看成偶数长, 从此表可以明显地看出, 在读入一个字符时, 已处理的输入串的相应前缀长度的奇偶性“交替变化”: 奇变偶, 偶变奇。而 0,1 区分正好体现了读进来的符号: 是 0 就是 0, 是 1 就是 1。
- d. 如果认为 ϵ 不属于语言 ($\epsilon \notin L(M)$), 则只要将 $[\epsilon]$ 从终止状态集中删除即可。

表 5.5: $\{x|x \in \{0,1\}^* \text{ 且如果 } x \text{ 以 } 1 \text{ 结尾, 则它的长度为偶数; 如果 } x \text{ 以 } 0 \text{ 结尾, 则长度为奇数}\}$ 转移表

state	$[\epsilon]$	$[o0]$	$[o1]$	$[e0]$	$[e1]$
意义	空字符串 (marked)	奇长, 0 尾 (marked)	奇长, 1 尾	偶长, 0 尾	偶长, 1 尾 (marked)
输入字符 0	$[o0]$	$[e0]$	$[e0]$	$[o0]$	$[o0]$
输入字符 1	$[o1]$	$[e1]$	$[e1]$	$[o1]$	$[o1]$

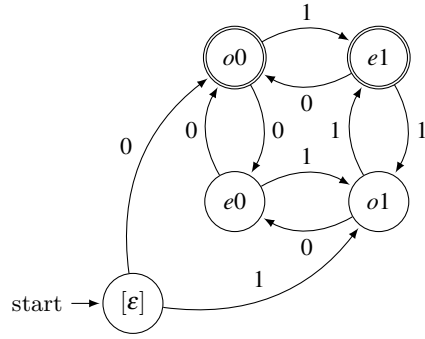


图 5.11: $\{x|x \in \{0,1\}^* \text{ 且如果 } x \text{ 以 } 1 \text{ 结尾, 则它的长度为偶数; 如果 } x \text{ 以 } 0 \text{ 结尾, 则长度为奇数}\}$ 语言对应的
 $DFA([o0] \text{ 奇 } 0, [o1] \text{ 奇 } 1; [e0] \text{ 偶 } 0, [e1] \text{ 偶 } 1)$

Exercise 5.2. 设 $DFA M = (Q, \Sigma, \delta, q_0, F)$, 证明: 对于 $\forall x, y \in \Sigma^*, q \in Q$,

$$\delta(q, xy) = \delta(\delta(q, x), y)$$

Solution 5.2. 证明. 设 $x, y \in \Sigma^*, q \in Q$, 现对 $|y|$ 施归纳。

1. 当 $|y| = 0$ 时, $y = \varepsilon$, 由于对任意的 $q \in Q$, 均有

$$\delta(q, \varepsilon) = q$$

所以

$$\delta(q, x) = \delta(\delta(q, x), \varepsilon) \quad (5.10)$$

另一方面, $x = x\varepsilon$, 使得下式成立

$$\delta(q, x\varepsilon) = \delta(q, x) \quad (5.11)$$

综合这两个等式 (5.10, 5.11) 得

$$\delta(q, x\varepsilon) = \delta(\delta(q, x), \varepsilon)$$

即结论对 $|y| = 0$ 成立。

2. 设结论对 $|y| = n$ 成立, 且 $|ya| = n+1, a \in \Sigma^*$, 此时由 δ 的定义

$$\delta(q, xya) = \delta(\delta(q, xy), a)$$

$$\text{由归纳假设, } \delta(q, xy) = \delta(\delta(q, x), y)$$

从而

$$\delta(q, xya) = \delta(\delta(q, xy), a) = \delta(\delta(\delta(q, x), y), a)$$

注意到 $\delta(q, x)$ 为一个状态, 再由 δ 的定义

$$\delta(\delta(\delta(q, x), y), a) = \delta(\delta(q, x), ya)$$

所以

$$\delta(\delta(q, xy), a) = \delta(\delta(q, x), ya)$$

表明结论对 $|ya| = n+1$ 成立。

由归纳法原理, 结论对任意的 $x, y \in \Sigma^*, q \in Q$ 成立。 \square

Exercise 5.3. 证明: 对于任意的 $DFA M_1 = (Q, \Sigma, \delta, q_0, F_1)$, 存在 $DFA M_2 = (Q, \Sigma, \delta, q_0, F_2)$, 使得 $L(M_2) = \Sigma^* - L(M_1)$ 。

Solution 5.3. 证明. 分两步证明。

1. 构造 M_2

设 $DFA M_1 = (Q, \Sigma, \delta, q_0, F_1)$

取 $DFA M_1 = (Q, \Sigma, \delta, q_0, Q - F_2)$

2. 证明 $L(M_2) = \Sigma^* - L(M_1)$

对任意的 $x \in \Sigma^*$,

$$x \in L(M_2) = \Sigma^* - L(M_1) \Leftrightarrow \delta(q_0, x) \in Q - F_1 \Leftrightarrow \delta(q_0, x) \in Q \text{ 并且 } \delta(q_0, x) \notin F_1 \Leftrightarrow x \in \Sigma^* \text{ 并且 } x \notin L(M_1) \Leftrightarrow x \in \Sigma^* - L(M_1) \quad \square$$

Exercise 5.4. 对于任意的 $DFA M_1 = (Q_1, \Sigma, \delta_1, q_{01}, F_1)$, 请构造 $DFA M_2 = (Q_2, \Sigma, \delta_2, q_{02}, F_2)$, 使得 $L(M_2) = L(M_1)^T$ 其中 $L(M)^T = \{x | x^T \in L(M)\}$ 。

Solution 5.4. 证明. 分三步证明。

1. 构造 $\varepsilon - NFA M$ 使得 $L(M) = L(M_1)$

设 $DFA M_1 = (Q_1, \Sigma, \delta_1, q_{01}, F_1)$

先取 $\varepsilon - NFA M = (Q, \Sigma, \delta, q_0, \{q_{01}\})$, 其中

$$Q = Q_1 \cup \{q_0\}, q_0 \notin Q_1$$

对于任意的 $q, p \in Q, a \in \Sigma$,

$$q \in \delta(p, a) \Leftrightarrow \delta_1(q, a) = p$$

$$\delta(q_0, \varepsilon) = F_1$$

2. 证明 $L(M) = L(M_1)^T$

对任意的 $a_1 a_2 \cdots a_n \in \Sigma^*$,

$$a_1 a_2 \cdots a_n \in L(M) \Leftrightarrow q_0 a_1 a_2 \cdots a_n \vdash q_f a_1 a_2 \cdots a_n \vdash a_1 q_1 a_2 \cdots a_n \vdash a_1 a_2 q_2 \cdots a_n \vdash \cdots \vdash a_1 a_2 \cdots q_{n-1} a_n \vdash a_1 a_2 \cdots q_{01} \text{ 并且 } q_f \in F_1$$

$$\Leftrightarrow q_f \in \delta(q_0, \varepsilon), q_1 \in \delta(q_f, a_1), q_2 \in \delta(q_1, a_2), \cdots, q_{01} \in (q_{n-1}, a_n), \text{ 并且 } q_f \in \delta(F_1)$$

$$\Leftrightarrow \delta(q_1, a_1) = q_f, \delta(q_2, a_2) = q_1, \cdots, \delta(q_{01}, a_n) = q_{n-1}, \text{ 并且 } q_f \in F_1$$

$$\Leftrightarrow q_{01} a_n a_{n-1} \cdots a_1 \vdash a_n q_{n-1} a_{n-1} \cdots a_1 \vdash \cdots \vdash a_n a_{n-1} \cdots q_2 a_2 a_1 \vdash a_n a_{n-1} \cdots a_2 q_1 a_1 \vdash a_n a_{n-1} \cdots a_2 a_1 q_f$$

$$\Leftrightarrow a_n a_{n-1} \cdots a_2 a_1 \Leftrightarrow x \in L(M_1)$$

3. 按照将 $\varepsilon - NFA$ 转换成等价的 NFA , 再将 NFA 转换成等价的 DFA 的方法, 将此 $\varepsilon - NFA$ 转换成满足要求的 $DFA M_2 = (Q_2, \Sigma, \delta_2, q_{02}, F_2)$ 。

Exercise 5.5. 构造识别下列语言的 NFA 。

1. $\{x | x \in \{0, 1\}^+ \text{ 并且 } x \text{ 中含形如 } 10110 \text{ 的子串}\}$ 。

2. $\{x | x \in \{0, 1\}^+ \text{ 并且 } x \text{ 中不含形如 } 10110 \text{ 的子串}\}$ 。

3. $\{x | x \in \{0, 1\}^+ \text{ 以 } 0 \text{ 开头以 } 1 \text{ 结尾}\}$ 。

4. $\{x | x \in \{0, 1\}^* \text{ 且如果 } x \text{ 以 } 1 \text{ 结尾, 则它的长度为偶数; 如果 } x \text{ 以 } 0 \text{ 结尾, 则长度为奇数}\}$ 。 $\{x | x \in \{0, 1\}^+ \text{ 首字符和尾字符}$

Solution 5.5. 构造识别下列语言的 NFA 。

1. 构造 $\{x | x \in \{0, 1\}^+ \text{ 并且 } x \text{ 中含形如 } 10110 \text{ 的子串}\}$ 语言的 NFA 。构造结果如图 (5.12) 所示。

2. 构造 $\{x | x \in \{0, 1\}^+ \text{ 并且 } x \text{ 中不含形如 } 10110 \text{ 的子串}\}$ 语言的 NFA 。

构造要点如下:

- 虽然本小题的语言是上一小题语言的补, 决不能通过将上一小题的所给出的 NFA 的终止状态和非终止状态互换来构造本小题的 NFA 。就是说 **Exercise 5.3** 所给的方法仅适用于 DFA , 不适用于 NFA 。

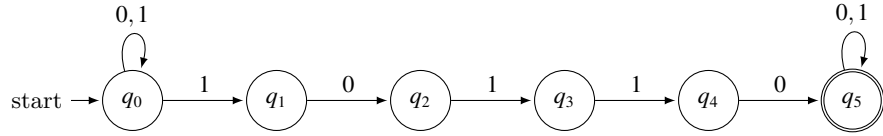


图 5.12: $\{x|x \in \{0,1\}^+$ 并且 x 中含形如 10110 的子串 $\}$ 语言的 NFA

b. 需要构造相应语言的 DFA , 即一种特殊的 NFA 。如图 (5.13) 所示。

c. 根据 **Exercise 5.3** 和上一小题的结果构造主体框架, 然后逐步补充。

3. 构造 $\{x|x \in \{0,1\}^+x \text{ 以 } 0 \text{ 开头以 } 1 \text{ 结尾}\}$, DFA 见图 (5.10)。 NFA 见图 (5.14)。

4. $\{x|x \in \{0,1\}^*$ 且如果 x 以 1 结尾, 则它的长度为偶数; 如果 x 以 0 结尾, 则长度为奇数 $\}$ 。

DFA 见图 (5.11)。 NFA 见图 (5.15)。

5. 构造 $\{x|x \in \{0,1\}^+x \text{ 首字符和尾字符形同}\}$ 语言的 NFA 见图 (5.16)。

Exercise 5.6. 证明: 对于任意的 NFA , 存在与之等价的 NFA , 该 NFA 最多只有一个终止状态。

Solution 5.6. 证明提示:

一般的, NFA 会有多个状态, 当该 NFA 的可达的终止状态数小于等于 1 时, 删除所有的不可达状态即可; 否则, 先删除所有不可达状态, 再构造与之等价的只有一个终止状态的 NFA 。

最简单的办法是对不含不可达状态的 NFA , 将原来的终止状态均改为非终止状态, 然后增加一个终止状态, 并将原来存在的到达原来终止状态弧进行复制, 使之也可以到达新的终止状态。

Exercise 5.7. 构造识别下列语言的 $\varepsilon-NFA$:

$\{x|x \in \{0,1\}^+ \text{ 且 } x \text{ 中不含形如 } 00 \text{ 的子串}\} \cap \{x|x \in \{0,1\}^+ \text{ 且 } x \text{ 中不含形如 } 11 \text{ 的子串}\}$

Solution 5.7. 分步构造, 如图 (5.17)。

Exercise 5.8. 证明: 对于任意的 $FA M_1 = (Q_1, \Sigma_1, \delta_1, q_{01}, F_1)$, $FA M_2 = (Q_2, \Sigma_2, \delta_2, q_{02}, F_2)$, 存在 $FA M$, 使得 $L(M) = L(M_1) \cup L(M_2)$ 。

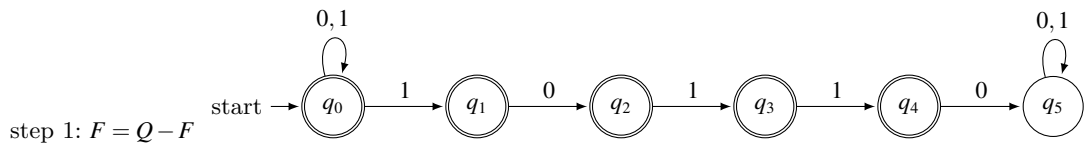
Solution 5.8. 构造新的 $FA M = (Q, \Sigma, \delta, q_0, F)$ 。

1. 新 FA 的输入字母表是原来两个字母表的并。 $\Sigma = \Sigma_1 \cup \Sigma_2$ 。
2. 保留原来两个 FA 的所有状态, 并增加一个新的开始状态 q_0 。 $Q = Q_1 \cup Q_2 \cup \{q_0\}, q_0 \notin Q_1, Q_2$ 。
3. 从新的开始状态 q_0 做 ε 移动, 分别到达原来两个 FA 的启动状态: $\delta(q_0, \varepsilon) = \{q_{01}, q_{02}\}$ 。
4. 保留原来两个 FA 的所有移动。即 $\delta = \delta_1 \cup \delta_2 \cup \{q_0 \times \{q_{01}, q_{02}\}\}$ 。
5. 新的 FA 的终止状态是原来两个 FA 的并。 $F = F_1 \cup F_2$ 。

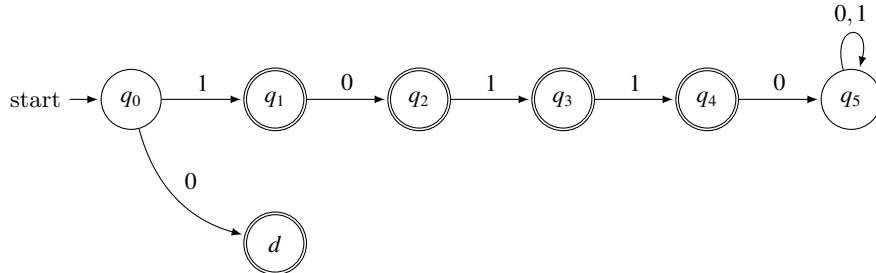
Exercise 5.9. 证明: 对于任意的 $FA M_1 = (Q_1, \Sigma_1, \delta_1, q_{01}, F_1)$, $FA M_2 = (Q_2, \Sigma_2, \delta_2, q_{02}, F_2)$, 存在 $FA M$, 使得 $L(M) = L(M_1)L(M_2)$ 。

Solution 5.9. 构造新的 $FA M = (Q, \Sigma, \delta, q_0, F)$ 。

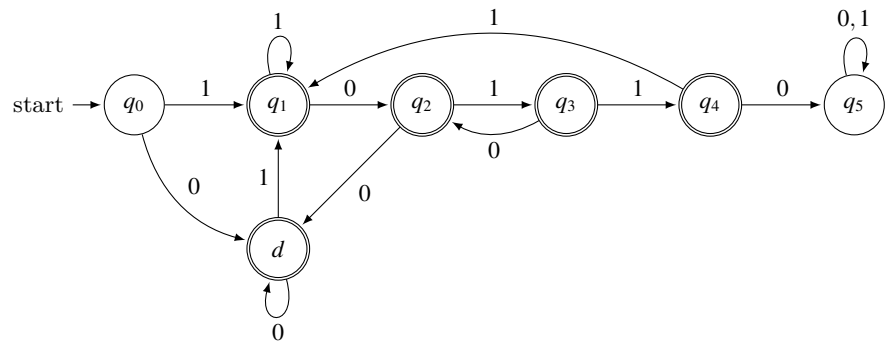
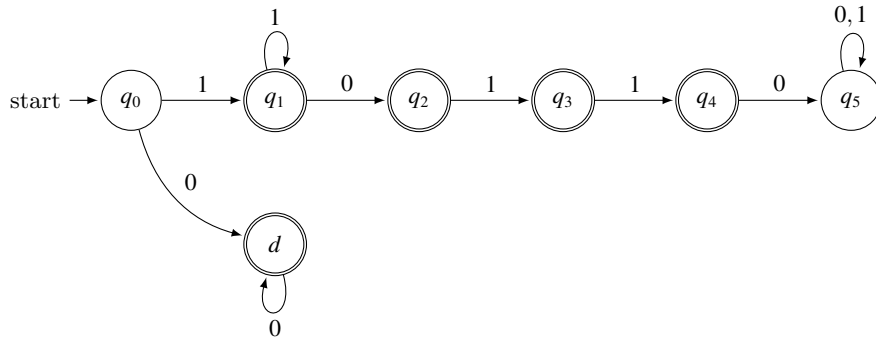
1. 新 FA 的输入字母表是原来两个字母表的并。 $\Sigma = \Sigma_1 \cup \Sigma_2$ 。
2. 保留原来两个 FA 的所有状态。 $Q = Q_1 \cup Q_2 \cup \{q_0\}$ 。
3. M_1 的开始状态为新的 $FA M$ 的启动状态。 $q = q_{01}$ 。
4. 保留原来两个 FA 的所有移动。并对 M_1 的所有终止状态 f , 增加一个到 M_2 的开始状态 q_{02} 的 ε 移动: $\delta = \delta_1 \cup \delta_2 \cup \{f \times \{q_{02}\}\}, \delta(f, \varepsilon) = q_{02}, f \in F_1$ 。



step 2: Add new state d , 拆分 q_0 的自环 $\{0,1\}$



step 3: Add self loop in state q_1 and d . q_1 表示接受 0 个或多个 1 的状态, d 表示接受 0 个或多个 0 的状态,



step 4: Add other edges, 成为最终的 DFA

图 5.13: $\{x|x \in \{0,1\}^+ \text{ 并且 } x \text{ 中不含形如 } 10110 \text{ 的子串}\}$ 语言的 NFA

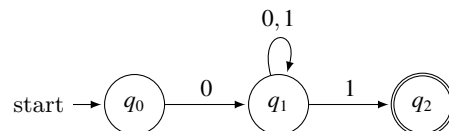
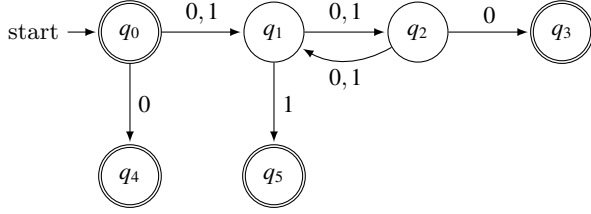
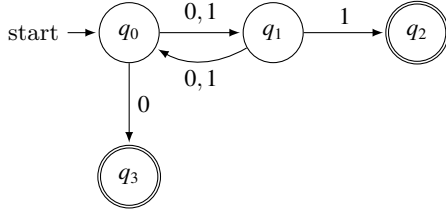
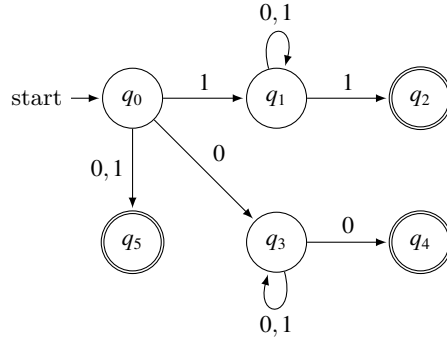


图 5.14: $\{x|x \in \{0,1\}^+ \text{ 以 } 0 \text{ 开头以 } 1 \text{ 结尾}\}$ 语言的 NFA

(1) $\varepsilon \in L(M)$ (2) $\varepsilon \notin L(M)$ 图 5.15: $\{x|x \in \{0,1\}^* \text{ 且如果 } x \text{ 以 } 1 \text{ 结尾, 则它的长度为偶数; 如果 } x \text{ 以 } 0 \text{ 结尾, 则长度为奇数}\}$ 语言的 NFA图 5.16: $\{x|x \in \{0,1\}^+ x\}$ 首字符和尾字符形同} 语言的 NFA

5. 新的 FA 的终止状态是原来两个 FA M_2 的终止状态。 $F = F_1$ 。

Exercise 5.10. 证明: 对于任意的 FA $M_1 = (Q_1, \Sigma_1, \delta_1, q_{01}, F_1)$, FA $M_2 = (Q_2, \Sigma_2, \delta_2, q_{02}, F_2)$, 存在 FA M , 使得 $L(M) = L(M_1) \cap L(M_2)$ 。

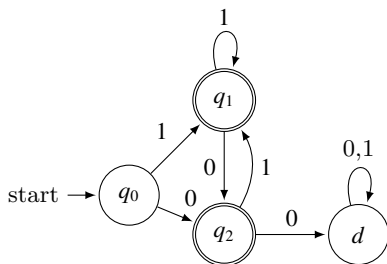
Solution 5.10. 不妨将这些 FA 均看成 DFA

取 DFA $M = (Q_1 \times Q_2, \Sigma_1 \cap \Sigma_2, \delta, [q_{01}, q_{02}, F_1 \times F_2])$ 。

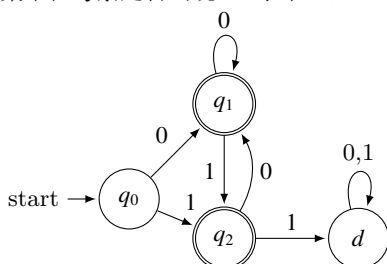
$\forall a \in \Sigma_1 \cap \Sigma_2, (q, p) \in Q_1 \times Q_2,$

$\delta([q, p], a) = [\delta_1(q, a), \delta_2(p, a)]$

step 1: 构造 $\{x|x \in \{0,1\}^+ \text{ 且 } x \text{ 中不含形如 } 00 \text{ 的子串}\}$ 语言的 DFA 。构造要点是，自动机启动并读入一个字符后，就将”精力”集中在考察是否出现 00 子串上，一旦发现子串 00 ，就进入陷阱状态。

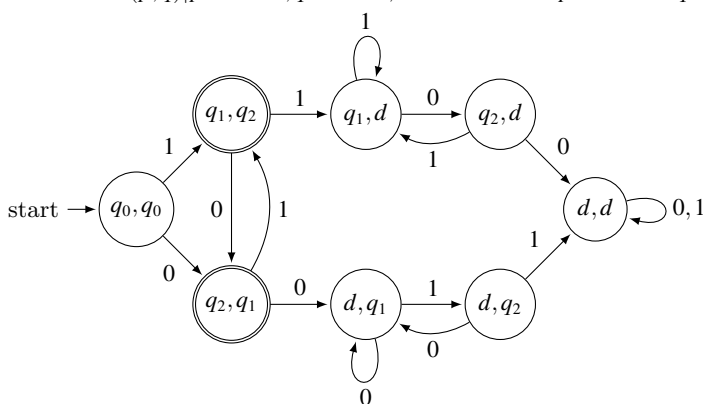


step 2: 构造 $\{x|x \in \{0,1\}^+ \text{ 且 } x \text{ 中不含形如 } 11 \text{ 的子串}\}$ 语言的 DFA 。构造要点是，自动机启动并读入一个字符后，就将”精力”集中在考察是否出现 11 子串上，一旦发现子串 11 ，就进入陷阱状态。



step 3: 构造 $DFA_1 \cap DFA_2$, 其中 step 1 $DFA = DFA_1$, step 2 $DFA = DFA_2$ 。

state: $\forall (p,q) | p \in DFA_1, q \in DFA_2$; marked state: $p \in F_1 \text{ and } q \in F_2$



step 4: 归约 (reduce) $DFA_1 \cap DFA_2$

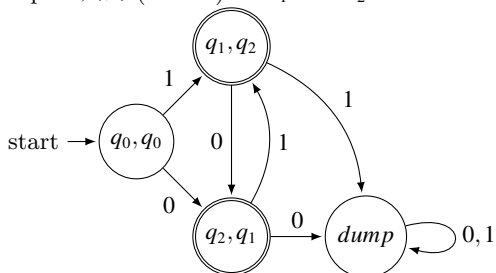


图 5.17: $\{x|x \in \{0,1\}^+ \text{ 且 } x \text{ 中不含形如 } 00 \text{ 的子串}\} \cap \{x|x \in \{0,1\}^+ \text{ 且 } x \text{ 中不含形如 } 11 \text{ 的子串}\}$ 语言的 $\varepsilon - NFA$

Chapter 6

[蒋宗礼 2013](第 4 章正则表达式)

主要内容

- 典型 RE 的构造。
- 与 RE 等价 FA 的构造方法。
- 与 DFA 等价的 RE 的构造。
- 重点
 - RE 的概念。
 - RE 与 DFA 的等价性。
- 难点
 - RE 与 DFA 的等价性证明。

6.1 RE 的形式定义

Definition 6.1. 正则表达式 (regular expression, RE)

设 Σ 是一个字母表, 以递归形式定义正则表达式:

1. \emptyset 是 Σ 上的 RE , 它表示语言 \emptyset ;
2. ε 是 Σ 上的 RE , 它表示语言 $\{\varepsilon\}$;
3. 对于 $\forall a \in \Sigma, a$ 是 Σ 上的 RE , 它便是语言 $\{a\}$.
4. 如果 r, s 分别是 Σ 上表示语言 R, S 的 RE , 则:
 - r, s 的“和” $(r+s)$ 是 Σ 上的 RE , $(r+s)$ 表达的语言为 $R \cup S$;
 - r, s 的“乘积” (rs) 是 Σ 上的 RE , (rs) 表达的语言为 RS ;
 - r 的克林闭包 (r^*) 是 Σ 上的 RE , (r^*) 表达的语言为 R^* ;
5. 只有满足 1,2,3,4 的才是 Σ 上的 RE .

Example 6.1. 设 $\Sigma = \{0,1\}$

- 0 , 表示语言 $\{0\}$
- 1 , 表示语言 $\{1\}$
- $(0+1)$, 表示语言 $\{0,1\}$
- (01) , 表示语言 $\{01\}$
- $(0+1)^*$, 表示语言 $\{0,1\}^*$

- $(00)(00)^*$, 表示语言 $\{00\}\{00\}^*$
- $(0+1)^*(0+1)(0+1)^*$, 表示语言 $\{0,1\}^+$
- $(0+1)^*000(0+1)^*$, 表示 $\{0,1\}$ 上的至少含有 3 个连续 0 的串组成的语言
- $(0+1)^*01$, 表示所有以 01 结尾的 0,1 字符串组成的语言
- $1(0+1)^*0$, 表示所有以 1 开头, 并且以 0 结尾的 0,1 字符串组成的语言

约定

1. r 的正闭包 r^+ 表示 r 与 (r^*) 的乘积以及 (r^*) 与 r 的乘积:

$$r^+ = rr^* = r^*r$$

2. 闭包运算的优先级最高, 乘运算的优先级次之, 加运算 “+” 的优先级最低。
3. 在意义明确时, RE r 表示的语言记为 $L(r)$, 也可以直接地记为 r 。
4. 加、乘、闭包运算均执行左结合规则。

Definition 6.2. 设 r, s 是字母表 Σ 上的正则表达式, 如果 $L(r) = L(s)$, 则称 r 和 s 相等 (*equivalence*, 也称为等价)。

几个基本结论

1. 结合律:

$$(rs)t = r(st)$$

$$(r+s)+t = r+(s+t)$$
2. 分配律:

$$r(s+t) = rs+rt$$

$$(s+t)r = sr+tr$$
3. 交换律: $r+s = s+r$
4. 幂等律: $r+r = r$
5. 加法运算零元素: $r+\emptyset = r$
6. 乘法运算单位元: $r\varepsilon = \varepsilon r = r$
7. 乘法运算零元素: $r\emptyset = \emptyset r = \emptyset$
8. $L(\emptyset) = \emptyset$
9. $L(\varepsilon) = \{\varepsilon\}$
10. $L(a) = \{a\}, a \in \Sigma$
11. $L(rs) = L(r)L(s)$
12. $L(r+s) = L(r) \cup L(s)$
13. $L(r^*) = (L(r))^*$
14. $L(\emptyset^*) = \{\varepsilon\}$
15. $L((r+\varepsilon)^*) = L(r^*)$
16. $L((r^*)^*) = L(r^*)$
17. $L((r^*s^*)^*) = L((r+s)^*)$
18. 如果 $L(r) \subseteq L(s)$, 则 $r+s = s$
19. $L(r^n) = (L(r))^n$
20. $r^n r^m = r^{n+m}$

Definition 6.3. 设 r 是字母表 Σ 上的一个正则表达式, r 的 n 次幂定义为:

1. $r^0 = \varepsilon$
2. $r^n = r^{n-1}r, (n \geq 1)$

Note 6.1. 一般地, $r + \varepsilon \neq r, (rs)^n \neq r^n s^n, rs \neq sr$.

Example 6.2. 设 $\Sigma = \{0, 1\}$

00 表示语言 $\{00\}$;

$(0+1)^*00(0+1)^*$ 表示所有的至少含两个连续 0 的 0,1 串组成的语言;

$(0+1)^*1(0+1)^9$ 表示所有的倒数第 10 个字符为 1 的串组成的语言;

$L((0+1)^*011) = \{x|x \text{ 是以 } 011 \text{ 结尾的 } 0,1 \text{ 串}\}$;

$L(0^+1^+2^+) = \{0^n1^m2^k | m, n, k \geq 1\}$;

$L(0^*1^*2^*) = \{0^n1^m2^k | m, n, k \geq 0\}$;

$L(1(0+1)^*1+0(0+1)^*0)) = \{x|x \text{ 的开头字符与尾字符相同}\}$ 。

6.2 正则表达式 RE 与 FA 等价

Definition 6.4. 正则表达式 r 称为与 FA M 等价, 如果 $L(r) = L(M)$ 。

从开始状态出发, 根据状态之间按照转移所确定的后继关系, 依次计算出所给 FA 的各个状态 q 对应的 $set(q)$, 并且最终得到相应的 FA 接受的语言的 RE 表示。寻找一种比较“机械”的方法, 使得计算机系统能够自动完成 FA 与 RE 之间的转换。

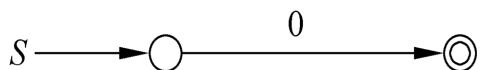


图 6.1: 0 对应的 FA

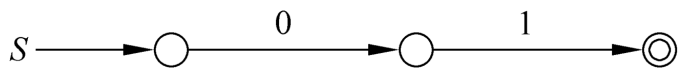


图 6.2: 01 对应的 FA

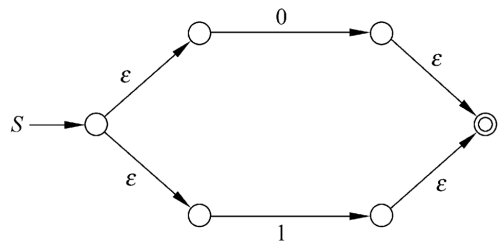
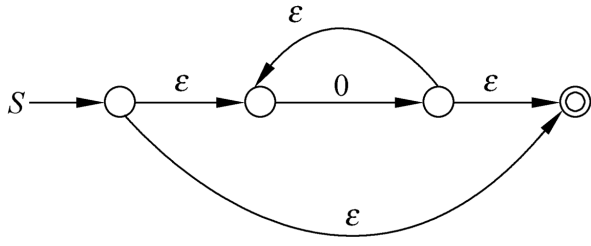


图 6.3: $0+1$ 对应的 FA

图 6.4: 0^* 对应的 FA

Theorem 6.1. 正则表达式 RE 表示的语言是正则语言 RL 。

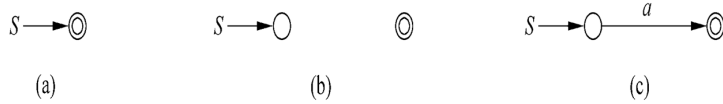
证明. 根据递归表达式的递归定义 (6.1), 施归纳于正则表达式中所含的运算符的个数 n , 证明对于字母表 Σ 上的任意正则表达式 r , 存在 FA M , 使得 $L(M) = L(r)$ 。

- M 恰有一个终止状态。
- M 在终止状态下不做移动。

当 $n = 0$ 时, 即 r 中不含运算符时, 有以下 3 种情况:

1. $r = \varepsilon$, 此时图 6.5(a) 所示的 ε -NFA 满足要求。
2. $r = \emptyset$, 此时图 6.5(b) 所示的 ε -NFA 满足要求。
3. $\forall a \in \Sigma$, 此时图 6.5(c) 所示的 ε -NFA 满足要求。

所以, 结论对 $n = 0$ 时成立。

图 6.5: $r = \varepsilon, r = \emptyset, r = a$ 对应的 ε -NFA

假设结论对 $n \leq k$ ($k \geq 0$) 成立, 此时有如下 FA:

$$M_1 = (Q_1, \Sigma, \delta_1, q_{01}, \{f_1\})$$

$$M_2 = (Q_2, \Sigma, \delta_2, q_{02}, \{f_2\})$$

$$L(M_1) = L(r_1), L(M_2) = L(r_2)$$

$$Q_1 \cap Q_2 = \emptyset$$

当 $n = k + 1$ 时, r 有以下 3 种运算:

1. $r = r_1 + r_2$

取 $q_0, f \notin Q_1 \cup Q_2$, 令

$$M = (Q_1 \cup Q_2 \cup \{q_0, f\}, \Sigma, \delta, q_0, \{f\})$$

$$\text{a. } \delta(q_0, \varepsilon) = \{q_{01}, q_{02}\}$$

$$\text{b. 对 } \forall q \in Q_1, a \in \Sigma \cup \{\varepsilon\}, \delta(q, a) = \delta_1(q, a);$$

$$\text{对 } \forall q \in Q_2, a \in \Sigma \cup \{\varepsilon\}, \delta(q, a) = \delta_2(q, a);$$

$$\text{c. } \delta(f, \varepsilon) = \{f\}$$

d. $\delta(f_2, \varepsilon) = \{f\}$

这里构造的 M , 如图 (6.6)

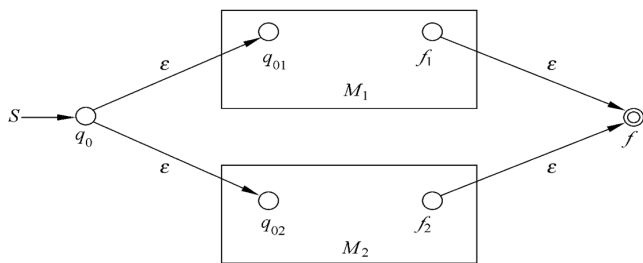


图 6.6: 与 $r_1 + r_2$ 等价的满足要求的 ε -NFA

2. $r = r_1 r_2$

$$M = (Q_1 \cup Q_2, \Sigma, \delta, q_{01}, \{f_2\})$$

a. 对 $\forall q \in Q_1 - \{f_1\}, a \in \Sigma \cup \{\varepsilon\}, \delta(q, a) = \delta_1(q, a);$

b. 对 $\forall q \in Q_2 - \{f_2\}, a \in \Sigma \cup \{\varepsilon\}, \delta(q, a) = \delta_2(q, a);$

c. $\delta(f_1, \varepsilon) = \{q_{02}\}$

这里构造的 M , 如图 (6.7)

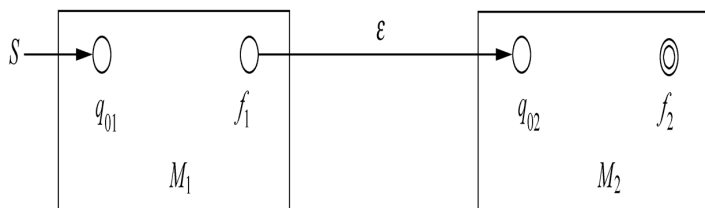


图 6.7: 与 $r_1 r_2$ 等价的满足要求的 ε -NFA

3. $r = r_1^*$

$$M = (Q_1 \cup \{q_0, f\}, \Sigma, \delta, q_0, \{f\})$$

其中 $q_0, f \notin Q_1$, 定义 δ 为

a. 对 $\forall q \in Q_1 - \{f_1\}, a \in \Sigma, \delta(q, a) = \delta_1(q, a);$

b. $\delta(f_1, \varepsilon) = \{q_{01}, f\}$

c. $\delta(q_0, \varepsilon) = \{q_{01}, f\}$

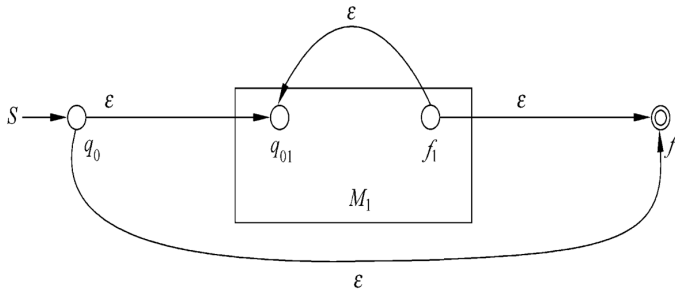
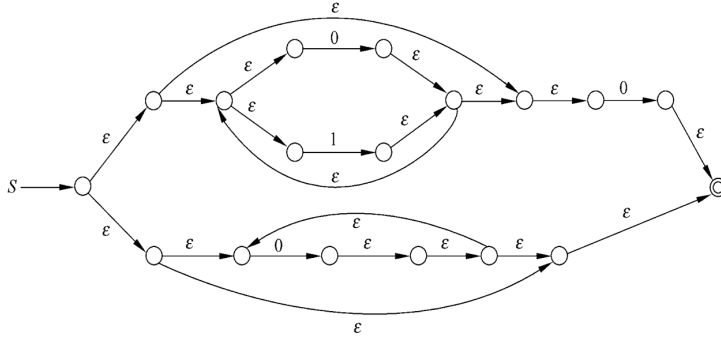
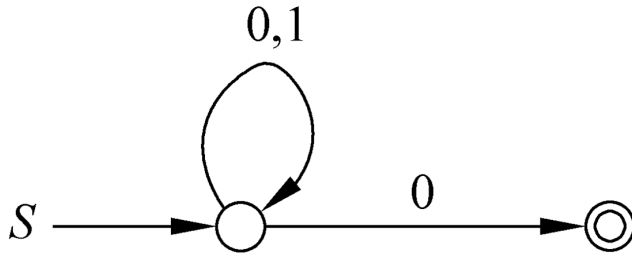
这里构造的 M , 如图 (6.8)

因此结论对 $n = k + 1$ 成立。由归纳法原理, 结论对 Σ 上的任意正则表达式成立。 \square

Example 6.3. 构造与 $(0+1)^*0 + (00)^*$ 等价的 FA。如下图 (6.9)。

按照对 $(0+1)^*0 + (00)^*$ 的”理解”, ”直接地”构造出的 FA。

因为, 如果 $L(r) \subseteq L(s)$, 则 $r + s = s$, 这里 $s = (0+1)^*0, r = (00)^*$, 因此直接构造出的 FA 如下图 (6.10)。

图 6.8: 与 r_1^* 等价的满足要求的 ϵ -NFA图 6.9: 与 $(0+1)^*0+(00)^*$ 等价的 FA图 6.10: 与 $(0+1)^*0+(00)^*$ 等价的 FA

6.3 正则语言 RL 可以用正则表达式 RE 表示

Theorem 6.2. 正则语言 RL 可以用正则表达式 RE 表示。

设 DFA

$$M = (\{q_1, q_2, \dots, q_n\}, \Sigma, \delta, q_1, F)$$

令

$$R_{ij}^k = \{x \mid \delta(q_i, x) = q_j, \text{ 而且对于 } x \text{ 的任意前缀 } y (y \neq x, y \neq \epsilon), \text{ 如果 } \delta(q_i, y) = q_l, \text{ 则 } l \leq k\}$$

R_{ij}^k 表示所有那些将 DFA 从给定状态 q_i 引导到状态 q_j , 并且“途中”不经过 (进入并离开) 下标大于 k 的状态的所有字符串。值得提醒的是, i 和 j 的值不受小于等于 k 的限制。

对于 $\forall q_i, q_j \in \{q_1, q_2, \dots, q_n\}$, R_{ij}^k 是所有可以将 DFA 从状态 q_i 引导到状态 q_j 的字符串组成的集合。为了便于计算, 可以将 R_{ij}^k 递归地定义为:

$$R_{ij}^0 = \begin{cases} \{a | \delta(q_i, a) = q_j\} & \text{如果 } i \neq j \\ \{a | \delta(q_i, a) = q_j\} \cup \{\varepsilon\} & \text{如果 } i = j \end{cases}$$

$$R_{ij}^k = R_{ik}^{k-1} (R_{kk}^{k-1})^* R_{kj}^{k-1} \cup R_{ij}^{k-1}$$

显然,

$$L(M) = \bigcup_{q_f \in F} R_{1f}^n$$

当 $R_{ij}^0 = \emptyset$ 时, 它对应的正则表达式为 \emptyset ; 当 $R_{ij}^0 = \{a_1, a_2, \dots, a_n\} \neq \emptyset$ 时, 它对应的正则表达式为 $a_1 + a_2 + \dots + a_n$.

仅当 $i = j$ 时, 集合 R_{ij}^0 中含有一个 ε , 而 R_{ij}^k 的表达式中含的都是定义正则表达式时用的运算, 所以, 容易得到 R_{ij}^k 的正则表达式。

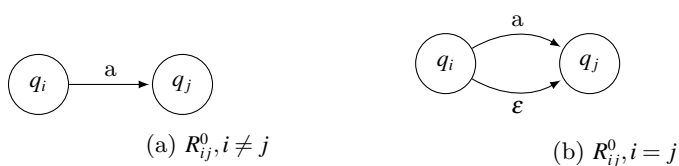


图 6.11: R_{ij}^0 对应的 FA

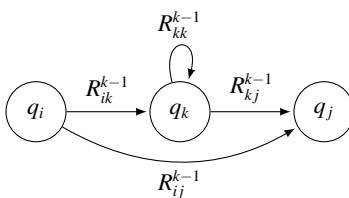


图 6.12: R_{ij}^k 对应的 FA

图上作业法

典型等价变换示例, 见图 (6.13)。

图上作业法操作步骤

1. 预处理:

a. 用标记为 X 和 Y 的状态将 M “括起来”:

- 在状态转移图中增加标记为 X 和 Y 的状态, 从标记为 X 的状态到标记为 q_0 的状态引一条标记为 ε 的弧; 从标记为 $q(q \in F)$ 的状态到标记为 Y 的状态分别引一条标记为 ε 的弧。

b. 去掉所有的不可达状态。

2. 对通过步骤 (1) 处理所得到的状态转移图重复如下操作, 直到该图中不再包含除了标记为 X 和 Y 外的其他状态, 并且这两个状态之间最多只有一条弧。

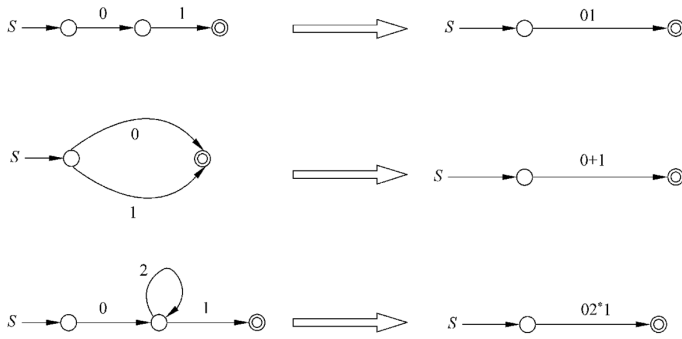


图 6.13: 典型等价变换示例

a. 并弧

- 将从 q 到 p 的标记为 r_1, r_2, \dots, r_g 并行弧用从 q 到 p 的、标记为 $r_1 + r_2 + \dots + r_g$ 的弧取代这 g 个并行弧。

b. 去状态 1

- 如果从 q 到 p 有一条标记为 r_1 的弧，从 p 到 t 有一条标记为 r_2 的弧，不存在从状态 p 到状态 p 的弧，将状态 p 和与之关联的这两条弧去掉，用一条从 q 到 t 的标记为 $r_1 r_2$ 的弧代替。

c. 去状态 2

- 如果从 q 到 p 有一条标记为 r_1 的弧，从 p 到 t 有一条标记为 r_2 的弧，从状态 p 到状态 p 的标记为 r_3 的弧，将状态 p 和与之关联的这三条弧去掉，用一条从 q 到 t 的标记为 $r_1 r_3^* r_2$ 的弧代替。

d. 去状态 3

- 如果图中只有三个状态，而且不存在从标记为 X 的状态到达标记为 Y 的状态的路，则将除标记为 X 的状态和标记为 Y 的状态之外的第 3 个状态及其相关的弧全部删除。

3. 从标记为 X 的状态到标记为 Y 的状态的弧的标记为所求的正则表达式。如果此弧不存在，则所求的正则表达式为 \emptyset 。

Example 6.4. 求图 (6.14) 所示的 DFA 等价的 RE。

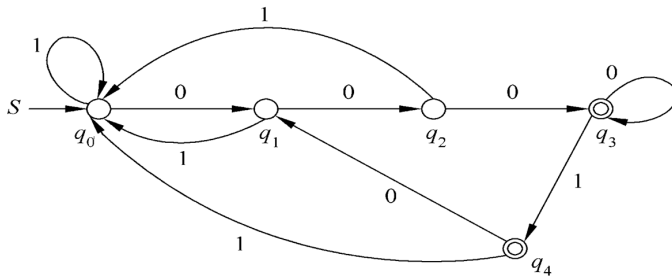


图 6.14: 由 DFA 构造等价正则表达式的示例 DFA

1. 预处理，得到图6.15。
2. 去掉状态 q_3 ，得到图6.16
3. 去掉状态 q_4 ，得到图6.17
4. 合并从标记为 q_2 的状态到标记为 Y 的状态的两条并行弧。得到图6.18。

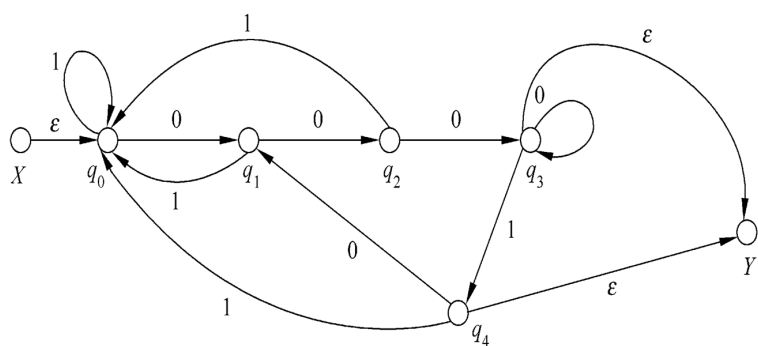


图 6.15: 执行步骤 (1) 后的 *DFA*

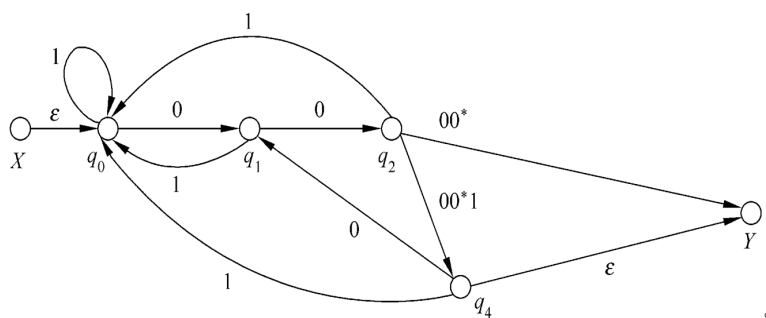


图 6.16: 去掉状态 q_3 后的 DFA

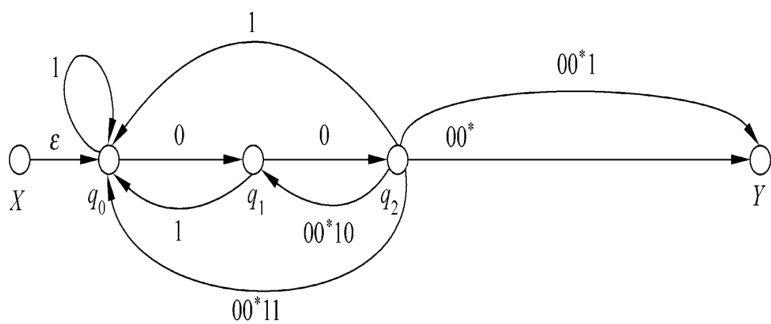


图 6.17: 去掉状态 q_4 后的 DFA

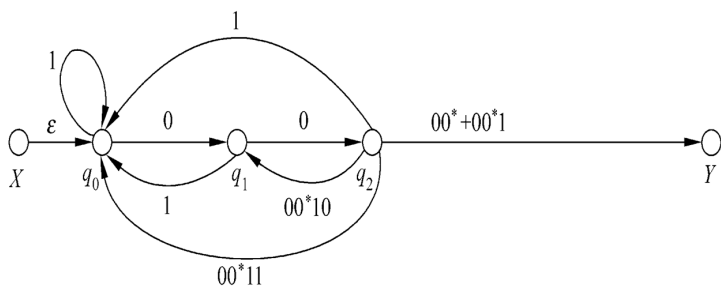
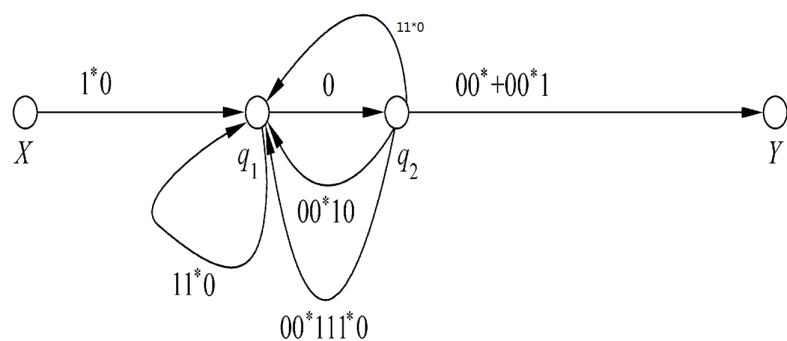


图 6.18: 合并后的 *DFA*

图 6.19: 去掉状态 q_0 后的 DFA

5. 去掉状态 q_0 , 得到图6.19。

6. 并弧, 得到图6.20。

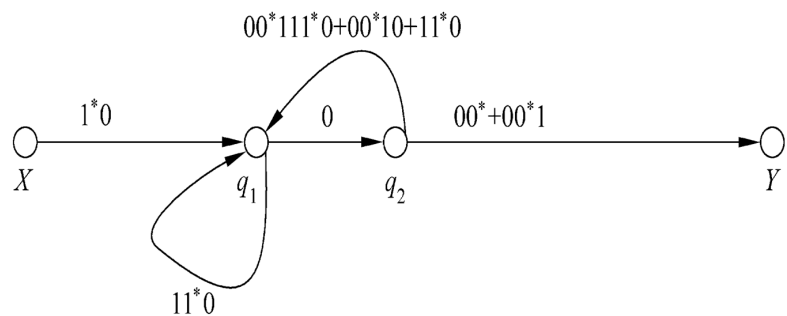
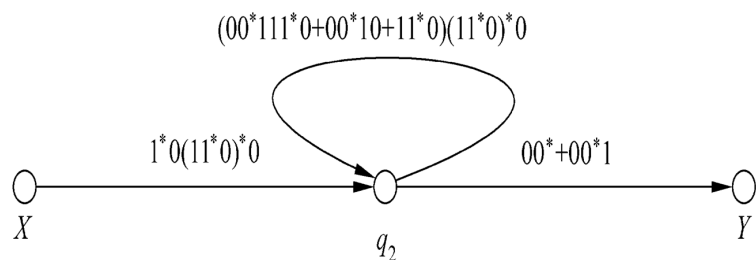


图 6.20: 并弧后的 DFA

7. 去掉状态 q_1 , 得到图6.21。

图 6.21: 去掉状态 q_1 后的 DFA

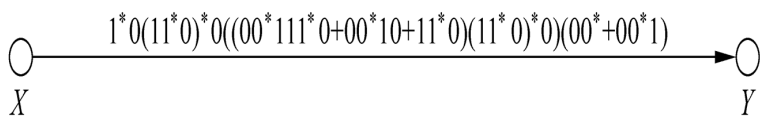
8. 去掉状态 q_1

9. 去掉状态 q_2 , 得到图6.22。

得到以下正则表达式, 就是所求。

$$1^*0(11^*0)^*0((00^*111^*0+00^*10+11^*0)(11^*0)^*0)(00^*+00^*1)$$

Note 6.2. 以下几点需要注意:

图 6.22: 去掉状态 q_2 后的 DFA

1. 如果去状态的顺序不一样，则得到的 RE 可能在形式是不一样，但它们都是等价的。
2. 当 DFA 的终止状态都是不可达的时候，状态转移图中必不存在从开始状态到终止状态的路。此时，相应的 RE 为 \emptyset 。
3. 不计算自身到自身的弧，如果状态 q 的入度为 n ，出度为 m ，则将状态 q 及其相关的弧去掉之后，需要添加 $n \times m$ 条新弧。
4. 对操作的步数施归纳，可以证明它的正确性。

Corollary 6.1. 正则表达式与 FA、正则文法等价，是正则语言的表示模型。

6.4 正则语言等价模型的总结

到此，一共给出了正则语言的 5 种等价描述模型：正则文法 (RG)，确定的有穷状态自动机 (DFA)，不确定的有穷状态自动机 (NFA)，带空移动的有穷状态自动机 (ϵ -NFA)，正则表达式 (RE)。他们之间的等价转换如图 (6.23)。

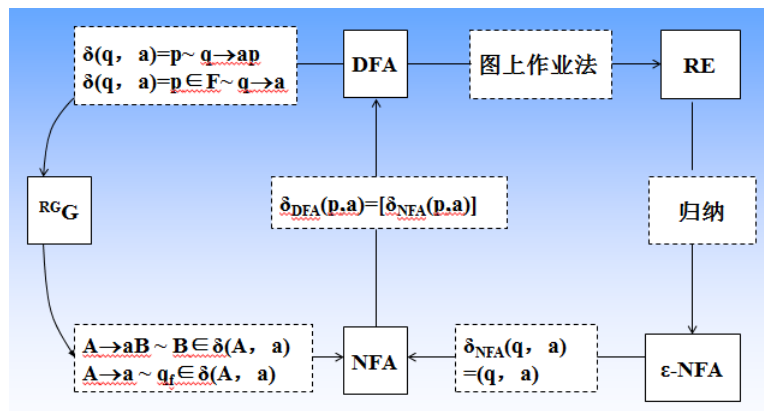


图 6.23: 正则语言 5 中等价模型的转换

小结

本章讨论了 RL 及其与 FA 的等价性。

1. 字母表 Σ 上的 RE 用来表示 Σ 上的 RL 。 $\emptyset, \epsilon, a \in \Sigma$ 是 Σ 上的最基本的 RE ，它们分别表示语言 $\emptyset, \{\epsilon\}, \{a\}$ ，以此为基础，如果 r 和 s 分别是 Σ 上的表示语言 R 和 S 的 RE ，则 $r+s, rs, r^*$ 分别是 Σ 上的表示语言 $R \cup S, RS, R^*$ 的 RE 。如果 $L(r) = L(s)$ ，则称 r 与 s 等价。

2. RE 对乘、加满足结合律；乘对加满足左、右分配律；加满足交换率和幂等率； \emptyset 是加运算的零元素； ε 是乘运算的单位元； \emptyset 是乘运算的零元素。
3. RE 是 RL 的一种描述。容易根据 RE 构造出与它等价的 FA 。反过来，可以用图上作业法构造出与给定的 DFA 等价的 RE 。
4. RL 的 5 种等价描述模型转换图。

6.5 Exercise and Solution

Exercise 6.1. 写出下列语言的正则表达式。

1. $0, 1^+$ 。
2. $\{x | x \in \{0, 1\}^* \text{ 且 } x \text{ 中不含形如 } 00 \text{ 的子串}\}$ 。

Solution 6.1.

1. $0, 1^+$

$$r = (0 + 1)(0 + 1)^*$$

2. $\{x | x \in \{0, 1\}^* \text{ 且 } x \text{ 中不含形如 } 00 \text{ 的子串}\}$

a. 分析语言，直接构造正则表达式。

$$r_1 = (1 + 01)^* = \{x | x \text{ 无连续的 } 0, \text{ 但以 } 1 \text{ 结尾；当以 } 0 \text{ 开头时，长度不小于 } 2\}$$

$$r_2 = (1 + 10)^* = \{x | x \text{ 无连续的 } 0, \text{ 以 } 1 \text{ 开头，但以 } 0 \text{ 或者以 } 1 \text{ 结尾；当以 } 0 \text{ 结尾时，长度不小于 } 2\}$$

$$r_3 = (1 + 01)^* 0 = \{x | x \text{ 无连续的 } 0, \text{ 以 } 0 \text{ 结尾；长度至少为 } 1\}$$

$$r_4 = 0 = \{x | x \text{ 无连续的 } 0, \text{ 以 } 0 \text{ 开头和结尾；长度为 } 1\}$$

$$\therefore r_4 \subseteq r_3$$

$$\therefore r_3 = r_3 + r_4$$

$$r = r_1 + r_2 + r_3 + r_4 = r_1 + r_2 + r_3$$

$$r = (1 + 01)^* + (1 + 10)^* + (1 + 01)^* 0 = (1 + 10)^* + (1 + 01)^* (\varepsilon + 0)$$

另一种思路是根据 $(1 + 01)^*$ 和 $(1 + 10)^*$ 直接补充所缺部分。

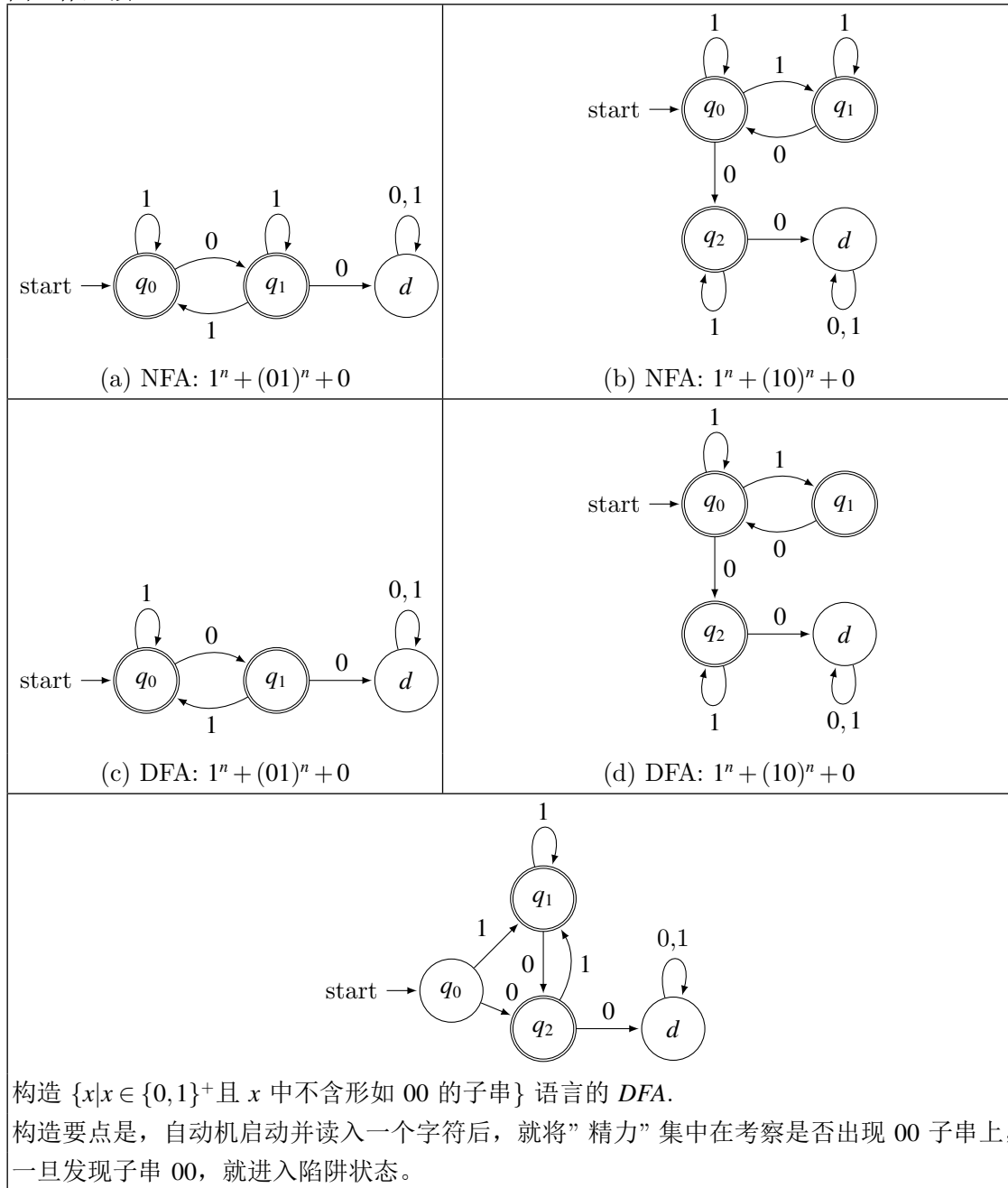
用 $0(1 + 10)^*$ 补上以 0 开头的满足要求的字符串：

$$(1 + 10)^* + 0(1 + 10)^* = (\varepsilon + 0)(1 + 10)^*$$

或者用 $(1 + 01)^* 0$ 补上以 0 结尾的满足要求的字符串：

$$(1 + 01)^* + (1 + 01)^* 0 = (1 + 01)^* (0 + \varepsilon)$$

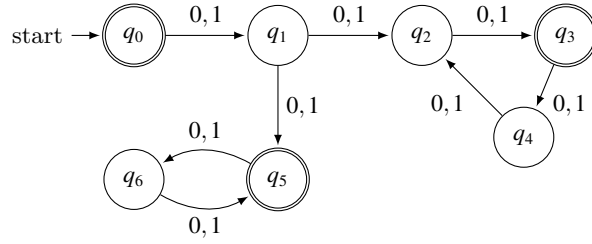
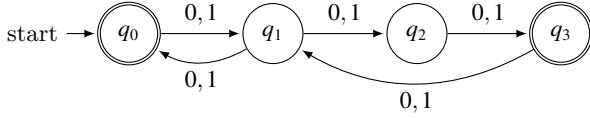
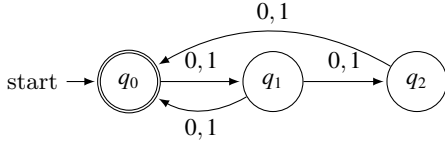
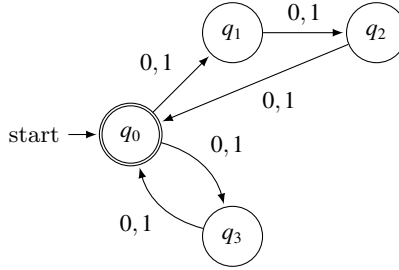
b. 图上作业法。

**Exercise 6.2.** 构造下列正则表达式的等价 *FA*。

$$((0+1)(0+1))^* + ((0+1)(0+1(0+1)))^*$$

Solution 6.2.

1. $(0+1)(0+1)^* + ((0+1)(0+1(0+1)))^*$ 的等价 *FA* 见图 (6.24)。
2. $(0+1)(0+1)^* + ((0+1)(0+1(0+1)))^*$ 的不等价 *FA* 见图 (6.25)。
3. $(0+1)(0+1)^* + ((0+1)(0+1(0+1)))^*$ 的不等价 *FA* 见图 (6.26)。
4. $(0+1)(0+1)^* + ((0+1)(0+1(0+1)))^*$ 的不等价 *FA* 见图 (6.27)。

图 6.24: $((0+1)(0+1))^* + ((0+1)(0+1(0+1)))^*$ 的等价 FA图 6.25: $((0+1)(0+1))^* + ((0+1)(0+1(0+1)))^*$ 的不等价 FA图 6.26: $((0+1)(0+1))^* + ((0+1)(0+1(0+1)))^*$ 的不等价 FA图 6.27: $((0+1)(0+1))^* + ((0+1)(0+1(0+1)))^*$ 的不等价 FA

6.6 正则代换 (regular substitution)

设 Σ, Δ 是两个字母表, 映射

$$f: \Sigma \rightarrow 2^{\Delta^*}$$

被称为是从 Σ 到 Δ 的 **代换**。如果对于 $\forall a \in \Sigma, f(a)$ 是 Δ 上的 **RL**, 则称 f 为**正则代换**。

- 现将 f 的定义域扩展到 Σ^* 上:

1. $f(\epsilon) = \{\epsilon\}$
2. $f(xa) = f(x)f(a)$

- 再将 f 的定义域扩展到 2^{Δ^*}

对于 $\forall L \subseteq \Sigma^*$

$$f(L) = \bigcup_{x \in L} f(x)$$

- f 是正则代换, 则

1. $f(\emptyset) = \emptyset$
2. $f(\varepsilon) = \varepsilon$
3. 对于 $\forall a \in \Sigma, f(a)$ 是 Δ 上的 RE
4. 如果 r, s 是 Σ 上的 RE , 则

$$f(r+s) = f(r) + f(s)$$

$$f(rs) = f(r)f(s)$$

$$f(r^*) = f(r)^*$$
 是 Δ 上的 RE

Example 6.5. 设 $\Sigma = 0, 1, \Delta = a, b, f(0) = a, f(1) = b^*$, 则

$$\begin{aligned}
 f(010) &= f(0)f(1)f(0) = ab^*a \\
 f(11, 00) &= f(11) \cup f(00) \\
 &= f(1)f(1) \cup f(0)f(0) \\
 &= b^*b^* + aa = b^* + aa \\
 f(L(0^*(0+1)1^*)) &= L(a^*(a+b^*)(b^*)^*) \\
 &= L(a^*(a+b^*)b^*) \\
 &= L(a^*ab^* + a^*b^*b^*) \\
 &= L(a^*b^*)
 \end{aligned}$$

Theorem 6.3. 设 L 是 Σ 上的一个 RL

$$f: \Sigma \rightarrow 2^{\Delta^*}$$

是正则代换, 则 $f(L)$ 也是 RL . □

证明. 描述工具 RE

对 r 中运算符的个数 n 施以归纳, 证明 $f(r)$ 是表示 $f(L)$ 的 RE .

- 当 $n = 0$ 时, 结论成立。
- 当 $n \leq k$ 时, 定理成立, 即当 r 中运算符的个数不大于 k 时: $f(L(r)) = L(f(r))$ 。
- 当 $n = k + 1$ 时,

1. $r = r_1 + r_2$

$$\begin{aligned}
 f(L) &= f(L(r)) \\
 &= f(L(r_1 + r_2)) \\
 &= f(L(r_1) \cup L(r_2)) && RE \text{ 的定义} \\
 &= f(L(r_1)) \cup f(L(r_2)) && \text{正则代换的定义} \\
 &= L(f(r_1)) \cup L(f(r_2)) && \text{归纳假设} \\
 &= L(f(r_1) + f(r_2)) && RE \text{ 的定义} \\
 &= L(f(r_1 + r_2)) && RE \text{ 的正则代换的定义} \\
 &= L(f(r))
 \end{aligned}$$

2. $r = r_1 r_2$

$$\begin{aligned}
 f(L) &= f(L(r)) \\
 &= f(L(r_1 r_2)) \\
 &= f(L(r_1) L(r_2)) && RE \text{ 的定义} \\
 &= f(L(r_1)) f(L(r_2)) && \text{正则代换的定义} \\
 &= L(f(r_1)) L(f(r_2)) && \text{归纳假设} \\
 &= L(f(r_1) f(r_2)) && RE \text{ 的定义} \\
 &= L(f(r_1 r_2)) && RE \text{ 的正则代换的定义} \\
 &= L(f(r_1 r_2))
 \end{aligned}$$

3. $r = r_1^*$

$$\begin{aligned}
 f(L) &= f(L(r)) \\
 &= f(L(r_1^*)) \\
 &= f(L(r_1)^*) && RE \text{ 的定义} \\
 &= (f(L(r_1)))^* && \text{正则代换的定义} \\
 &= (L(f(r_1)))^* && \text{归纳假设} \\
 &= L(f(r_1)^*) && RE \text{ 的定义} \\
 &= L(f(r_1^*)) && RE \text{ 的正则代换的定义} \\
 &= L(f(r))
 \end{aligned}$$

□

Chapter 7

Supervisor

$$L(SUP) = \overline{K}, L_m(SUP) = K \quad (7.1)$$

$$L(G) \cap L(LOC) = L(SUP) \quad (7.2)$$

$$L_m(G) \cap L_m(LOC) = L_m(SUP) \quad (7.3)$$

$$SUP_1 = Sync(G_1, LOC_1) \quad (7.4)$$

$$SUP_2 = Sync(G_2, LOC_2) \quad (7.5)$$

$$(7.6)$$