Acronyms

Use the template *acronym.tex* together with the Springer document class SVMono (monograph-type books) or SVMult (edited books) to style your list(s) of abbreviations or symbols in the Springer layout.

Lists of abbreviations, symbols and the like are easily formatted with the help of the Springer-enhanced `description` environment.

suffices to 就可以...,e.g. In order to check whether a nonempty word belongs to L, it suffices to verify that its first letter is in P

BABI Spelled-out abbreviation and definition

CABR Spelled-out abbreviation and definition

Symbols

$\in$	$x \in S$	x is an element of set S
$\notin$	$x \notin S$	x is not an element of set S
$\subseteq$	$A \subseteq B$	A is a subset of B
$\mathbb{N}$	$\{0,1,2,\ldots\}$	Set of non-negative integers
$\mathbb{N}^+$	$\{1,2,\ldots\}$	Set of positive integers
$\mathbb{N}_k$	$\{1,2,\ldots,k\}$	Index set
$\mathbb{Z}$	$\{\ldots,-2,-1,0,1,2,\ldots\}$	Set of integers
$\mathbb{Q}$	$\{p/q \mid p \in \mathbb{Z}, q \in \mathbb{Z}, q \neq 0\}$	Set of rational numbers
$\mathbb{Q}^+$		Set of rational positive numbers
$\mathbb{R}$		Set of real numbers
$\mathbb{C}$		Set of complex numbers
$\Leftrightarrow$	$\ldots \Leftrightarrow$ —	… if and only if —
$\Rightarrow$	$\ldots \Rightarrow$ —	… implies —

Chapter 1
automata abstract

P P ¶ P $\mathscr{P}$ P P P **P**

[WATSON93a, p6] **the signatures of the transition relations:**
$T \in \mathbb{P}(Q \times V \times Q)$
$T \in V \to P(Q \times Q)$
$T \in Q \times Q \to P(V)$
$T \in Q \times V \to P(Q)$
$T \in Q \to P(V \times Q)$
for example, the function $T \in Q \to P(V \times Q)$ is defined as $T(p) = \{(a,q) : (p,a,q) \in T\}$

ε-transition relation:
$E \in P(Q \times Q)$
$E \in Q \to P(Q)$

$T \in P(Q \times V \times Q), T = \{(s,a,q)\}$
$T(s) \in Q \to P(V \times Q), T(s) = \{(a,q) : (s,a,q) \in T\}$
$Q_{map} : P(Q \times V), Q_{map} = \{(q,a) : (s,a,q) \in T\}$

$Q_{map}(q) = \{a : (s,a,q) \in T\}$
$Q_{map}{}^{-1} : V \nrightarrow P(Q), Q_{map}{}^{-1} = \{(a,q) : (s,a,q) \in T\}$
According to [WATSON93a, Convention A.4] (Tuple projection):
$\bar{\pi}_2(T) = \{(s,q) : (s,a,q) \in T\}$
$Q_{map} = (\bar{\pi}_1(T))^R, Q_{map} = \{(a,q) : (s,a,q) \in T\}^R = \{(q,a) : (s,a,q) \in T\}$

$f(a) = (f(a^R))^R$

Definition 1.1 (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.

$L_1 = \{\varepsilon, a, aa\}, L_1 = \overline{L_1}, L_1$ is prefix-closed.
$L_2 = \{a, b, ab\}, \overline{L_2} = \{\varepsilon, a, b, ab\}, L_2 \subset \overline{L_2}, L_2$ is not prefix closed.

Definition 1.2 (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 $s \notin \overline{L}$.

Definition 1.3 (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 1.1. $A=\{a,aab,baa\}, a^{-1}A=D_aA=\frac{dA}{da}=\{\varepsilon,ab,\emptyset\}=\{\varepsilon,ab\}$ □

Example 1.2. $L=\{ba,baa,baab,ca\}, w=\{ba\},$

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

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

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

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

Example 1.3. $a^{-1}\{a\}=\{\varepsilon\};\quad a^{-1}\{b\}=\emptyset,\quad \Leftarrow if(a\neq b)$ □

Example 1.4. $L_0=\{ab\}, L_1=\{ac\}, L_0L_1=\{abac\}$

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

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

$=\{b\}L_1=\{bac\}$ □

Example 1.5. $L_0=\{\varepsilon,ab\}, L_1=\{ac\}, L_0L_1=\{ac,abac\}$

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

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

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

证明. $a^{-1}(L_0L_1)$

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

$L_0=(L_0\backslash\{\varepsilon\})\cup\{\varepsilon\}$

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

$=a^{-1}(L_0L_1\cup L_1)$

$a^{-1}L_0=a^{-1}((L_0\backslash\{\varepsilon\})\cup\{\varepsilon\})$

$=a^{-1}(L_0\backslash\{\varepsilon\})\cup a^{-1}\{\varepsilon\}$

$=a^{-1}L_0\cup\emptyset=a^{-1}L_0$

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 $L/a=\{\varepsilon,ba\}$, 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 $a\backslash L$ 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 $a\backslash L=\{\varepsilon,ab\}$, 证明：如果 L 是正则的，那么 $a\backslash L$ 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 1.4 (Kleene-closure[Chrison2007]). Let $L\subseteq V^*$, then

$$L^* := \{\varepsilon\}\cup L\cup LL\cup LLL\cup\cdots$$

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: $(L^*)^* = L^*$.

$$\begin{aligned} L^* &= \{\varepsilon\}+L^+ \\ &= \{\varepsilon\}\cup(L\backslash\{\varepsilon\})L^* \\ &= \{\varepsilon\}+L+LL+LLL+\cdots \end{aligned}$$

1.1 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 $X = K^\star L$ 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). see[1]

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 word[2] and if $u \in X_1 = KX_1 + L$, then necessarily $u \in L$ since $\varepsilon \notin K$. But in this case, $u \in KX_2 + L = X_2$. see[3]

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 $x \in X_2$.[see[4]] It follows that $u \in KX_2$ and finally $u \in X_2$. This conclude the induction and the proof of the proposition. □

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

---

[1]

$$\begin{aligned} K^* &= \{\varepsilon\} + K^+ \\ &= \{\varepsilon\} + (K \backslash \{\varepsilon\})K^* \\ &= \{\varepsilon\} + K + KK + KKK + \cdots \end{aligned}$$

[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,$\varepsilon$ 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 $|x| = 0, |x| = 1, \cdots, n, x \in X_2$.

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* $L = M^*N$ □

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.

Preliminaries :

$$\begin{aligned}
M^* &= M^k + M^{k-1} + \cdots + M^1 + M^0 \qquad (k \geq 0) \\
&= M^k + M^{k-1} + \cdots + M^1 + \varepsilon \\
&= M^+ + \varepsilon \\
&= MM^* + \varepsilon \\
&= (M \backslash \{\varepsilon\})M^* + \varepsilon \\
M^+ &= M^k + M^{k-1} + \cdots + M^1 \qquad (k > 0) \\
&= M(M^k + M^{k-1} + \cdots + M^1 + M^0) \\
&= MM^* \\
M^0 &= \{\varepsilon\} = 1 \\
M\varepsilon &= \varepsilon M = M \\
\varepsilon + \varepsilon &= \varepsilon \\
M + M &= M
\end{aligned}$$

证明.

$$L = ML + N \Rightarrow$$

$$M^0 L = M^1 L + M^0 N \tag{1.2}$$

$$M^1 L = M^2 L + M^1 N \tag{1.3}$$

$$M^2 L = M^3 L + M^2 N \tag{1.4}$$

$$\tag{1.5}$$

$$\cdots$$

$\Rightarrow$

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

$\Rightarrow$

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

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

$\Rightarrow$

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

(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: $L=\{\varepsilon\}=\{\text{empty word}\}\subseteq N$.

References

[Hopcroft2008] John E. Hopcroft,Rajeev Motwani,Jeffrey D. Ullman 著, 孙家骕等译, 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

1. S. Lipschutz and M. L. Lipson, *Schaum's Outline of Theory and Problems of Discrete Mathematics*, Third Edition, New York: McGraw-Hill, 2007.
2. K. H. Rosen, *Discrete Mathematics and Its Applications*, Seventh Edition, New York: McGraw-Hill, 2007.
3. S. Maclane and G. Birkhoff, *Algebra*, Third Edition, New York: Bchelsea Publishing Company, 1988.

Chapter 2
[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 ($\varepsilon - NFA$)
  - 定义。
  - $\varepsilon - NFA$ 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*、$\varepsilon - NFA$、*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。

2.1 语言的识别

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

$S \to aA|aB$

$A \to aA|c$

$B \to aB|d$

分析句子 (word)$aaac$ 的过程中可能需要回溯。

- 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 2.1)

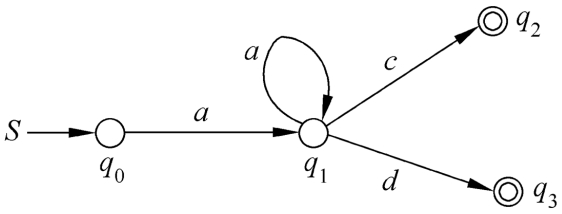

图 2.1 系统识别语言 $\{a^n c|n \geq 1\} \cup \{a^n d|n \geq 1\}$ 的字符串中状态的变化

有穷状态自动机的物理模型

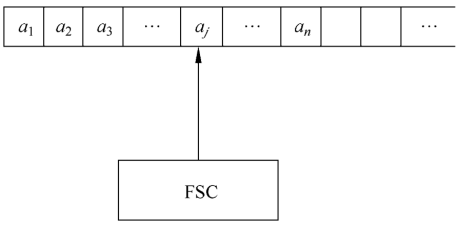

图 2.2 有穷状态自动机的物理模型

2.2 有穷状态自动机 (finite automaton,FA)

$$M = (Q, \Sigma, \delta, q_0, F)$$

Q: 状态的非空有穷集合。$\forall \in Q, q$ 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。$\delta : Q \times \Sigma \to Q$，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)*。

将 δ 扩充为

$$\hat{\delta} : Q \times \Sigma^* \to Q$$

对于任意的 $q \in Q, w \in \Sigma^*, a \in \Sigma$，定义

1. $\hat{\delta}(q, \varepsilon) = q$
2. $\hat{\delta}(q, wa) = \delta(\hat{\delta}(q, w), a)$

$$\begin{aligned}\hat{\delta}(q,a) &= \hat{\delta}(q, \varepsilon a) \\ &= \delta(\hat{\delta}(q, \varepsilon), a) \\ &= \delta(q, a)\end{aligned}$$

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

对于 $\forall x \in \Sigma^*$ 如果 $\delta(q, w) \in F$，则称 x 被 M 接受，如果 $\delta(q, w) \notin F$, 则称 M 不接受 x。

$$L(M) = \{x | x \in \Sigma^*, \text{且} \delta(q, w) \in F\}$$

称为由 M 接受 (识别) 的语言

$$L(M_1) = L(M_2) = \{2^{2^n} | n \geq 1\}$$

如果 $L(M_1) = L(M_2)$, 则称 M_1 与 M_2 等价。

Example 2.1. 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(2.3),table(2.1)

Example 2.2. 构造一个 *DFA*，它接受的语言为

$$\{x000|x\in\{0,1\}^*\}$$

see: figure(2.4)

表 2.1 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

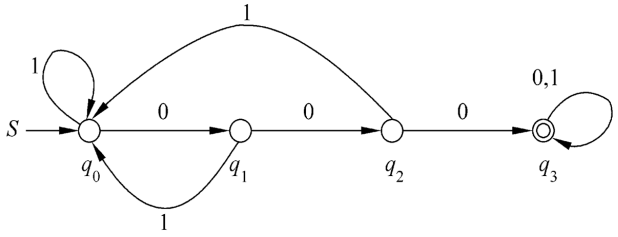

图 2.3 识别语言 $\{x000y|x,y\in\{0,1\}^*\}$ 的 *DFA*

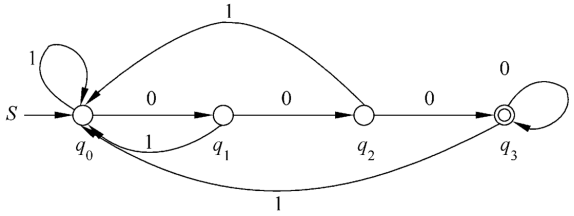

图 2.4 识别语言 $\{x000|x\in\{0,1\}^*\}$ 的 *DFA*

Note 2.1. 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* 可以有多于 1 个的终止状态。

即时描述 (instantaneous description，ID）

- $x,y\in\Sigma^*,\delta(q_0,x)=q,xqy$ 称为 M 的一个即时描述，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 $xqay\vdash_M xapy$。
- $\alpha\vdash_{M^n}\beta$: 表示 M 从即时描述 α 经过 n 次移动到达即时描述 β。
- M 存在即时描述 $\alpha_1,\alpha_2,\alpha_{n-1}$，使得

$$\alpha\vdash_M\alpha_1,\alpha_1\vdash_M\alpha_2,\cdots,\alpha_{n-1}\vdash_M\beta$$

  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 2.3. 图 (2.5) 的 DFA 到即时描述 (instantaneous description,ID) 的转换

$$\begin{aligned}
q_0101010001 &\vdash 1q_0010010001\\
&\vdash 10q_110010001\\
&\vdash 101q_00010001\\
&\vdash 1010q_1010001\\
&\vdash 10100q_210001\\
&\vdash 101001q_00001\\
&\vdash 1010010q_1001\\
&\vdash 10100100q_201\\
&\vdash 101001000q_31\\
&\vdash 1010010001q_0
\end{aligned}$$

即

$$q_0101010001 \vdash_{10} 1010010001q_0$$
$$q_0101010001 \vdash_{+} 1010010001q_0$$
$$q_0101010001 \vdash_{*} 1010010001q_0$$

对于 $x \in \Sigma^*$,

$$q_0x1 \vdash_{+} x1q_0$$
$$q_0x10 \vdash_{+} x10q_1$$
$$q_0x100 \vdash_{+} x100q_2$$
$$q_0x000 \vdash_{+} x000q_3$$

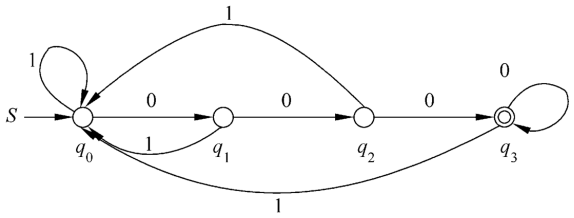

图 2.5 识别语言 $\{x000|x \in \{0,1\}^*\}$ 的 *DFA*

能引导 *FA* 从开始状态 q_0 到达 q 的字符串的集合为：

$$set(q) = \{x|x \in \Sigma^*, \delta(q_0,x) = q\}$$

Example 2.4. 对图 2.5所给的 *DFA* 中的所有 q，求 $set(q)$。

$$set(q_0) = \{x|x \in \Sigma^*, x = \varepsilon\text{或者 } x \text{ 以 1 结尾}\}$$
$$set(q_1) = \{x|x \in \Sigma^*, x = 0\text{或者 } x \text{ 以 10 结尾}\}$$
$$set(q_2) = \{x|x \in \Sigma^*, x = 00\text{或者 } x \text{ 以 100 结尾}\}$$
$$set(q_3) = \{x|x \in \Sigma^*, x \text{ 以 000 结尾}\}$$
$$set(q_4) = \{x|x \in \Sigma^*, x \text{ 以 001 结尾}\}$$

这 5 个集合是两两互不相交。

对于任意一个 *FA*, $M = (Q,\Sigma,\delta,q_0,F)$ 我们可以按照如下方式定义关系 R_M:

对 $\forall x,y \in \Sigma^*, xR_My \Leftrightarrow \exists q \in Q$，使得 $x \in set(q)$ 和 $y \in set(q)$ 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 2.5. 构造一个 *DFA*，它接受的语言为 $\{0^n1^m2^k | n,m,k \geq 1\}$。

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 2.6
- 再补充细节, 见图 2.7

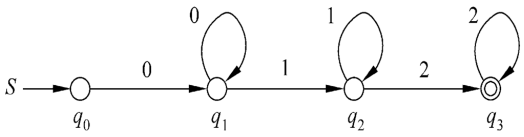

图 2.6 语言 $\{0^n1^m2^k | n,m,k \geq 1\}$ 的主题框架

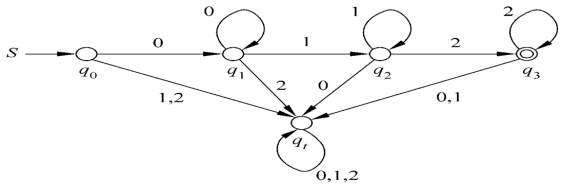

图 2.7 语言 $\{0^n1^m2^k | n,m,k \geq 1\}$ 的细节

1. 当 *FA* 一旦进入状态 q_t，它就无法离开此状态。所以，q_t 相当于一个陷阱状态 (trap)。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”，然后再去考虑画出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。

Example 2.6. 构造一个 *DFA*，它接受的语言为 $\{x|x\in\{0,1\}^*$，且当把 x 看成二进制数时，x 模 3 与 0 同余$\}$。

q_0: 对应除以 3 余数为 0 的 x 组成的等价类；

q_1: 对应除以 3 余数为 1 的 x 组成的等价类；

q_2: 对应除以 3 余数为 2 的 x 组成的等价类；

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：$\delta(q_s,0)=q_0;(q_s,1)=q_1$。

q_0: 能引导 M 到达此状态的 x 除以 3 余 0，所以有：$x=3\times n+0$。

读入 0 时，引导 M 到达下一个状态的字符串为 $x_0, x_0=2\times(3\times n+0)=3\times 2\times n+0$。所以，$\delta(q_0,0)=q_0$；

读入 1 时，M 到达下一个状态的字符串为 $x_1, x_1=2\times(3\times n+0)+1=3\times 2\times n+1$。所以，$\delta(q_0,1)=q_1$；

q_1: 能引导 M 到达此状态的 x 除以 3 余 1，所以有：$x=3\times n+1$。读入 0 时，引导 M 到达下一个状态的字符串为 $x_0, x_0=2\times(3\times n+1)=3\times 2\times n+2$。所以即：$\delta(q_1,0)=q_2$；

读入 1 时，引导 M 到达下一个状态的字符串为 $x_1, x_1=2\times(3\times n+1)+1=3\times 2\times n+2+1=3\times(2\times n+1)$。所以 $(q_1,1)=q_0$。

q_2: 能引导 M 到达此状态的 x 除以 3 余 2，所以：$x=3\times n+2$。读入 0 时，引导 M 到达下一个状态的字符串为 $x_0, x_0=2\times(3\times n+2)=3\times 2\times n+4=3\times(2\times n+1)+1$。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 $x_1, x_1=2\times(3\times n+2)+1=3\times 2\times n+4+1=3\times(2\times n+1)+2$。所以，$\times(q_2,1)=q_2$。

接受的语言 $\{x|x\in\{0,1\}^*$，且当把 x 看成二进制数时，x 模 3 与 0 同余$\}$ 的 *DFA*。如图 2.8。

2.3 正则代换 (regular substitution)

设 Σ,Δ 是两个字母表，映射

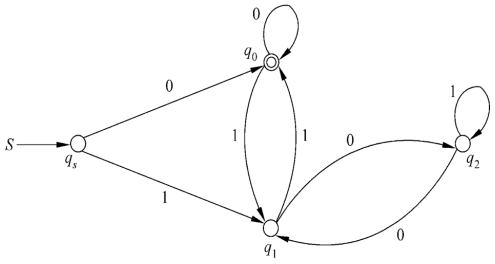

图 2.8 接受的语言 $\{x|x \in \{0,1\}^*$,且当把 x 看成二进制数时，x 模 3 与 0 同余$\}$ 的 *DFA*

$$f : \Sigma \to 2^{\Delta^*}$$

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. $f(\varepsilon) = \{\varepsilon\}$
  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 2.7. 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 2.1. 设 L 是 Σ 上的一个 *RL*

$$f : \Sigma \to 2^{\Delta^*}$$

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. $r = r_1 + r_2$

$$
\begin{aligned}
f(L) &= f(L(r)) \\
&= f(L(r_1 + r_2)) \\
&= f(L(r_1) \cup L(r_2)) && \textit{RE} \text{ 的定义} \\
&= f(L(r_1)) \cup f(L(r_2)) && \text{正则代换的定义} \\
&= L(f(r_1)) \cup L(f(r_2)) && \text{归纳假设} \\
&= L(f(r_1) + f(r_2)) && \textit{RE}\text{的定义} \\
&= L(f(r_1 + r_2)) && \textit{RE}\text{的正则代换的定义} \\
&= L(f(r))
\end{aligned}
$$

2. $r = r_1 r_2$

$$
\begin{aligned}
f(L) &= f(L(r)) \\
&= f(L(r_1 r_2)) \\
&= f(L(r_1)L(r2)) && RE \text{ 的定义} \\
&= f(L(r_1))f(L(r_2)) && \text{正则代换的定义} \\
&= L(f(r_1))L(f(r_2)) && \text{归纳假设} \\
&= L(f(r_1)f(r_2)) && RE \text{ 的定义} \\
&= L(f(r_1 r_2)) && RE \text{ 的正则代换的定义} \\
&= L(f(r_1 r_2))
\end{aligned}
$$

3. $r = {r_1}^*$

$$
\begin{aligned}
f(L) &= f(L(r)) \\
&= f(L({r_1}^*)) \\
&= f(L(r_1)^*) && RE \text{ 的定义} \\
&= (f(L(r_1)))^* && \text{正则代换的定义} \\
&= (L(f(r_1)))^* && \text{归纳假设} \\
&= L(f(r_1)^*) && RE \text{ 的定义} \\
&= L(f({r_1}^*)) && RE \text{ 的正则代换的定义} \\
&= L(f(r))
\end{aligned}
$$

□