第 3 章 有穷状态自动机

在乔姆斯基体系中，根据文法的不同，语言被分成 4 类。在这里，语言的文法描述提供了生成语言的手段，但是，对语言句子的识别来说，还存在一些不方便。从本章开始，将按照语言的分类，介绍一些识别语言的模型。

3.1 语言的识别

第 2 章定义由正则文法产生的语言为正则语言。当给定一个正则文法 $G=(V,T,P,S)$，相应的正则语言 $L(G)$也就给定了。此时，任意给定一个字符串 w，w 是这个语言的句子吗？为回答这个问题，需要考查 $S\overset{*}{\Rightarrow}w$ 在 G 中是否成立。按照文法的要求，如果 w 能由给定文法产生，就必须找出它的推导，或者将它归约成 S；如果 w 不能由文法产生，就必须证明 G 中不存在 w 的推导，或者 w 在 G 中不能归约成 S。这件事并不是对所有的文法都很容易做到的，正则文法也不例外。在第 2 章中已经了解到，给定文法所定义的语言的一个句子在该文法中可能存在许多不同的推导。更为严重的是，有时可以选择的直接推导不止一个，它们中有的是对的，有的是错的。如果选择了错误的推导，则在后续的推导中发现错误而无法继续进行下去时，就需要回过头来重新选择推导。归约也是一样，也会在某一步遇到多个不同的直接归约，它们中间也可能存在当前看似正确而实际是错误的归约。当不幸选中错误的归约，同样需要回过头来重新进行选择。这个过程称为“回溯”，是高级程序设计语言编译系统的设计中要解决的重要问题之一。当 w 不能由文法产生时，必须证明 G 中不存在 w 的推导，或者 w 在 G 中不能归约成 S。也就是说，要证明所有的推导都无法得到 w，或者所有的归约都不能将 w 最终归约成 S。下面看一个简单的例子。设文法 G 有如下产生式：

$$S \rightarrow aA \mid aB$$
$$A \rightarrow aA \mid c$$
$$B \rightarrow aB \mid d$$

字符串 $aaad$ 是该文法所定义的语言的句子吗？现在按推导的方法来回答此问题，归约方法类似，请读者自己练习。

一般系统在处理当前字符时，并不知道后面将出现的字符，所以，对字符串 $aaad$ 的分析，一开始并不知道句子的尾字符是 d。除非通过多遍扫描来完成句子的分析，否则就难以保证推导的每一步一定都是该句子的正确推导。只有在多遍扫描中，才有可能在进行句子

的推导前收集到可用于推导的信息。假设在推导的第一步选择 S 产生式的第一个候选式，于是

$$\begin{aligned} S &\Rightarrow aA && \text{使用产生式 } S\to aA\\ &\Rightarrow aaA && \text{使用产生式 } A\to aA\\ &\Rightarrow aaaA && \text{使用产生式 } A\to aA\\ &\Rightarrow aaaaA && \text{使用产生式 } A\to aA \end{aligned}$$

到这里，才发现这一推导无法推出 $aaad$。首先要考虑的问题是在上述推导中是否有选择错误的时候，因为上述推导的每一步各有两个不同的产生式可供选择(这样，共有 $2^4=16$ 种“需要考虑的”不同推导，注意仅仅在这种意义下的推导已经是文法的语法变量的候选式个数的推导步数次幂)，而我们只选择了其中的一种。是第四步应该用产生式 $A\to c$ 吗？要回答此问题，需要回到第三步推导的结果：

$$\begin{aligned} &\Rightarrow aaaA && \text{使用产生式 } A\to aA\\ &\Rightarrow aaac && \text{使用产生式 } A\to c \end{aligned}$$

结果仍然不对，因此又要重新回到第二步的推导结果……如此下去，最后回到开始，使用产生式 $S\to aB$ 重新开始推导。由此出发，读者自己不难给出 $aaad$ 的正确推导(仍然会出现错误的推导)。

由上例可以看出，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。是否可以从识别的角度去考虑问题的求解呢？不难看出，上例中文法 G 产生的语言的句子都是前面至少有一个 a，最后是一个 c 或者是一个 d，最后一个字符是 c 的可以算一类，最后一个字符是 d 的可以算另一类：

$$\{a^nc\mid n\geqslant 1\}\cup\{a^nd\mid n\geqslant 1\}$$

现在考虑按照如下方式设计一个系统：系统初始时处于开始处理输入字符串的状态(不妨取名为 q_0)。当读入一个 a 时，表示输入串已经满足 a 的个数“至少为 1”的要求(“至少为 1”表明，a 的个数可以有多个)，因此，可用一个状态表示目前已经得到至少一个 a(不妨取名为 q_1)。在 q_1 状态下，如果遇到 a，仍是满足至少一个 a 的要求，系统保持在 q_1 状态；如果遇到 c，则表示当前接受的字符串属于第一类，系统进入状态 q_2；如果在 q_1 状态下遇到 d，则表示当前接受的字符串属于第二类，系统进入状态 q_3。为了实现对该系统(模型)既简洁(抽象)又直观的描述，用顶点表示系统的状态，用带有标志的弧表示系统在某一状态下从输入字符串中读入当前字符后进入到另一个状态。据此，得到图 3-1。

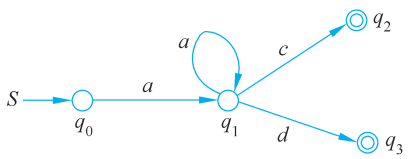

图 3-1　系统识别语言$\{a^nc\mid n\geqslant 1\}\cup\{a^nd\mid n\geqslant 1\}$的字符串过程中状态的变化图示

根据图 3-1，系统对字符串 $aaad$ 的处理就相对“容易”一些：系统在状态 q_0 启动(用标有 S 的箭头表示)，读入字符 a，转到状态 q_1，在 q_1 状态下，读入第二个字符 a，系统仍然处于状态 q_1；再读入第三个字符 a，系统仍然处于状态 q_1；读入第四个字符 d 时，系统进入状态 q_3。到此，发现 $aaad$ 是语言中的第二类句子。如果输入的字符串中还剩余有其他字符，那么这个输入字符串就不是该语言的句子了。这里用双圈顶点表示获得给定语言句子的状态。也就是说，系统在处理输入字符串的过程中，如果能从初始状态开始，最后到达用双圈顶点表示的“终止”状态，就认为这个输入字符串是

给定语言的句子；否则，这个输入字符串就不是给定语言的句子。虽然这种系统（模型）每“启动一次”只完成一个句子的识别，但是它可以识别语言的所有句子。所以，可以称这种系统（模型）为语言的识别系统（模型）。

对上述识别系统（模型），可以归纳出如下几个主要方面：

（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）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，系统有一个读写指针，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）在所有的状态中有一个状态，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）还有一些状态表示系统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，将系统从开始状态引导到这种状态的所有字符串构成的语言就是系统所能识别的语言。

将此系统（模型）对应成如图 3-2 所示的物理模型——称之为有穷状态自动机的物理模型。首先，它有一个**输入带**。该输入带上有一系列的“带方格”，每个带方格可以存放一个字符。为了不让输入带的存储容量影响我们对主要问题的考虑，约定：输入串从输入带的左端点开始存放，输入带的右端是无穷的。也就是说，从左端点的第一个带方格开始，输入带可以存放任意长度的输入字符串。其次，系统有一个**有穷状态控制器**（finite state control，FSC），该控制器的状态只有有穷多个。FSC 控制一个**读头**，用来从输入带上读入字符。每读入一个字符，就将读头指向下一个待读入的字符。

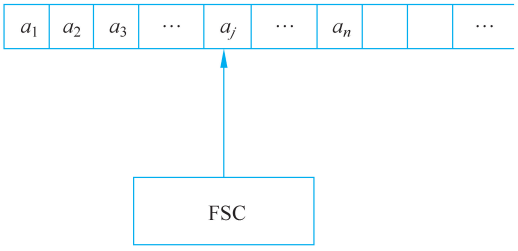

图 3-2　有穷状态自动机的物理模型

系统的每一个动作由 3 个节拍构成：读入读头所指向的字符；根据当前状态和读入的字符改变有穷控制器的状态；将读头向右移动一格。

3.2　有穷状态自动机

CPU、存储器、外部设备组成的基本的计算机硬件系统可以表示成三元组：

计算机硬件系统＝（CPU，存储器，外部设备）

进程管理、设备管理、存储管理、文件管理、网络通信管理组成的计算机操作系统可以表示成五元组：

操作系统=(进程管理,设备管理,存储管理,文件管理,网络通信管理)

同样,将有穷状态自动机表示成一个五元组(quintuple 或 5-tuple),其形式化的定义如下。

定义 3-1 **有穷状态自动机**(finite automaton,FA)M 是一个五元组：

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

其中,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。

δ——**状态转移函数**(transition function),有时又称为**状态转换函数**或者**移动函数**,δ：$Q\times\Sigma\to Q$。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。

q_0——M 的**开始状态**(initial state),也可称为**初始状态**或者**启动状态**,$q_0\in Q$。

F——M 的**终止状态**(final state)集合,$F\subseteq Q$。$\forall q\in F$,q 称为 M 的终止状态。

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**(accept state)。

例 3-1 下面是有穷状态自动机。

$$M_1=(\{q_0,q_1,q_2\},\{0\},\delta_1,q_0,\{q_2\})$$

其中,$\delta_1(q_0,0)=q_1$,$\delta_1(q_1,0)=q_2$,$\delta_1(q_2,0)=q_1$,可以用表 3-1 表示状态转移函数 δ_1。

表 3-1 状态转移函数 δ_1

状态说明	状　态	输入字符
		0
开始状态	q_0	q_1
	q_1	q_2
终止状态	q_2	q_1

$$M_2=(\{q_0,q_1,q_2,q_3\},\{0,1,2\},\delta_2,q_0,\{q_2\})$$

其中,$\delta_2(q_0,0)=q_1$,$\delta_2(q_1,0)=q_2$,$\delta_2(q_2,0)=q_1$,$\delta_2(q_3,0)=q_3$,$\delta_2(q_0,1)=q_3$,$\delta_2(q_1,1)=q_3$,$\delta_2(q_2,1)=q_3$,$\delta_2(q_3,1)=q_3$,$\delta_2(q_0,2)=q_3$,$\delta_2(q_1,2)=q_3$,$\delta_2(q_2,2)=q_3$,$\delta_2(q_3,2)=q_3$。

注意到用表格表达 δ 比逐个列出它的函数式更为清楚,所以,仍然采用表格的形式给出 δ_2,如表 3-2 所示。

定义 FA 的目的是用它来识别语言,所以,需要将 δ 的定义域从 $Q\times\Sigma$ 扩充到 $Q\times\Sigma^*$ 上。按照本节前面的描述,对一个给定的输入字符串,M 从开始状态读入该串的第一个字符,每处理完一个字符,就进入下一个状态,并在此新状态下读入下一个字符。按照这个过程,直到整个字符串被处理完。因此,按照下列定义,将 δ 扩充为 $\hat{\delta}$：$Q\times\Sigma^*\to Q$。

表 3-2　状态转移函数 δ_2

状态说明	状　态	输入字符		
		0	1	2
开始状态	q_0	q_1	q_3	q_3
	q_1	q_2	q_3	q_3
终止状态	q_2	q_1	q_3	q_3
	q_3	q_3	q_3	q_3

对任意 $q\in Q, w\in\Sigma^*, a\in\Sigma$，有

(1) $\hat{\delta}(q,\varepsilon)=q$。

(2) $\hat{\delta}(q,wa)=\delta(\hat{\delta}(q,w),a)$。

由于 δ 的定义域 $Q\times\Sigma$ 是 $\hat{\delta}$ 的定义域 $Q\times\Sigma^*$ 的真子集：$Q\times\Sigma\subset Q\times\Sigma^*$，所以，对于任意 $(q,a)\in Q\times\Sigma$，如果 $\hat{\delta}$ 和 δ 都有相同的值，就不用区分这两个符号了。事实上，对于任意 $q\in Q$，$a\in\Sigma$，有

$$\begin{aligned}\hat{\delta}(q,a)&=\hat{\delta}(q,\varepsilon a) && \varepsilon\text{ 是单位元素}\\ &=\delta(\hat{\delta}(q,\varepsilon),a) && \text{根据定义的第(2)条}\\ &=\delta(q,a) && \text{根据定义的第(1)条}\end{aligned}$$

所以，今后用 δ 代替 $\hat{\delta}$。

另外，由于对于任意 $q\in Q$，$a\in\Sigma$，$\delta(q,a)$均有确定的值，所以，又将这种 FA 称为**确定的有穷状态自动机**(deterministic finite automaton，DFA)。到此，可以给出 DFA 接受的语言的定义。

定义 3-2　设 $M=(Q,\Sigma,\delta,q_0,F)$是一个 DFA。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。

$$L(M)=\{x\mid x\in\Sigma^*\text{ 且 }\delta(q_0,x)\in F\}$$

称为由 M 接受(识别)的语言。

不难看出，例 3-1 中定义的 $M_1=(\{q_0,q_1,q_2\},\{0\},\delta_1,q_0,\{q_2\})$和 $M_2=(\{q_0,q_1,q_2,q_3\},\{0,1,2\},\delta_2,q_0,\{q_2\})$所接受的语言为

$$L(M_1)=L(M_2)=\{0^{2n}\mid n\geqslant 1\}$$

定义 3-3　设 M_1 和 M_2 为 FA。如果 $L(M_1)=L(M_2)$，则称 M_1 与 M_2 等价。

例 3-2　构造一个 DFA，它接受的语言为$\{x000y\mid x,y\in\{0,1\}^*\}$。

设 $L=\{x000y\mid x,y\in\{0,1\}^*\}$。不难看出，这个语言的特点是“每个句子都含有 3 个连续的 0”。显然，对任意给定的一个输入 $x\in\{0,1\}^*$，所构造的 DFA 的主要任务是检查 x 中是否存在子串“000”，一旦发现它含有这样的子串，就表示它是一个合法的句子。所以，在确认它含有“000”后，就应该逐一地读入该输入字符串的剩余后缀，并接受该串。所以，主要问题是如何发现子串“000”。由于字符是逐一被读入的，当从输入串中读入一个 0 时，它可能是子串“000”的第一个 0，因此，需要记住这个 0；如果紧接着读入的是 1，则刚才读入的

“0”并不是子串“000”的第一个0,此时需要重新寻找子串“000”的第一个0;如果紧接着读入的是0,则这个0可能是子串“000”的第二个0,此时也需要记住这个0,且目前已经发现了连续的两个0。DFA继续向前扫描,如果再次读到0,则表明已发现子串“000”。按照这一分析,识别所给语言的DFA M 至少需要如下几个状态:

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”的第一个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”的第二个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。

从而,可得到关于 M 的状态转移函数的如下部分定义:

$\delta(q_0,0)=q_1$——M 读到了一个0,这个0可能是子串“000”的第一个0。

$\delta(q_1,0)=q_2$——M 又读到了一个0,这个0可能是子串“000”的第二个0。

$\delta(q_2,0)=q_3$——M 找到了子串“000”。

之所以说这只是 M 的状态转移函数的部分定义,是因为还缺乏当 M 在状态 q_0,q_1,q_2 遇到输入字符1时,以及 M 在状态 q_3 遇到0和1时的定义。所以,上述3项定义实际上给出了 M 的“主体框架”。现在将缺少的部分补充如下:

$\delta(q_0,1)=q_0$——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”的第一个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,表明读入这个1之前所读入的0并不是子串“000”的第一个0。因此,M 需要重新回到状态 q_0,以寻找子串“000”的第一个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”的第一个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。

到此,得到接受语言 $\{x000y \mid x,y\in\{0,1\}^*\}$ 的DFA:

$M=(\{q_0,q_1,q_2,q_3\},\{0,1\},\{(q_0,0)=q_1,\delta(q_1,0)=q_2,\delta(q_2,0)=q_3,\delta(q_0,1)=q_0,\delta(q_1,1)=q_0,\delta(q_2,1)=q_0,\delta(q_3,0)=q_3,\delta(q_3,1)=q_3\},q_0,\{q_3\})$

为清楚起见,用表3-3表示该 M 的移动函数。

表3-3 M 的移动函数

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

初学者容易忽视的问题是漏填表中某些定义项，尤其当以函数式和状态转移图(定义 3-4)的形式给出 DFA 时更是如此。所以，特别强调：如果要求构造 DFA，就必须给出所有的定义，也就是要将表 3-3 填满，除非给出新的约定。

另外，若 DFA 在扫描完输入串之前已经获得足够的信息，确认一个输入字符串是它接受的句子，则它进入终止状态，并且可以在此状态读完该字符串的剩余部分。

下面用图 3-3 给出所构造的 M 的图示。

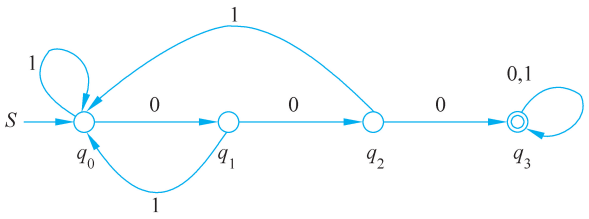

图 3-3　DFA M 的图示

这个图看起来要比 $M=(\{q_0,q_1,q_2,q_3\},\{0,1\},\{\delta(q_0,0)=q_1,\delta(q_1,0)=q_2,\delta(q_2,0)=q_3,\delta(q_0,1)=q_0,\delta(q_1,1)=q_0,\delta(q_2,1)=q_0,\delta(q_3,0)=q_3,\delta(q_3,1)=q_3\},q_0,\{q_3\})$直观得多，所以理解起来也容易许多。因此，将这种图定义为 FA 的状态转移图。注意，这里定义的状态转移图不仅仅用来表示 DFA，还被用来表示后面定义的 NFA 和ε-NFA。

定义 3-4　设 $M=(Q,\Sigma,\delta,q_0,F)$为一个 DFA，满足如下条件的有向图称为 M 的**状态转移图**(transition diagram)：

(1) $q\in Q\Leftrightarrow q$ 是该有向图中的一个顶点。

(2) $\delta(q,a)=p\Leftrightarrow$图中有一条从顶点 q 到顶点 p 的标记为 a 的弧。

(3) $q\in F\Leftrightarrow$标记为 q 的顶点用双层圈标出。

(4) 用标有 S 的箭头指出 M 的开始状态。

状态转移图又称为**状态转换图**。有时，状态转移图中会存在一些并行的弧：它们从同一顶点出发，到达同一个顶点。对这样的并行弧，用一条有多个标记的弧表示。例如，图 3-3 中的从状态 q_3 到状态 q_3 的标记为 0 和 1 的弧表示从状态 q_3 到状态 q_3 有两条并行的弧，它们的标记分别为 0 和 1。

例 3-3　构造一个 DFA，它接受的语言为$\{x000|x\in\{0,1\}^*\}$。

语言$\{x000|x\in\{0,1\}^*\}$与语言$\{x000y|x,y\in\{0,1\}^*\}$的不同之处是，前者要求每个句子都是以 000 结尾的，而不是要求句子中含有子串“000”。所以，在构造中，查到 3 个连续的 0 并不表明此串可以被接受，而需要看这 3 个连续的 0 是否为输入串的最后 3 个字符。因此，通过对图 3-3 的状态赋予新的解释，并进行适当的修改，以获得接受语言$\{x000|x\in\{0,1\}^*\}$的 DFA：

在状态 q_0 读到的 0 可能是输入字符串的最后 3 个 0 的第一个 0。

在状态 q_1 紧接着读到的 0 可能是输入字符串的最后 3 个 0 的第二个 0。

在状态 q_2 紧接着读到的 0 可能是输入字符串的最后 3 个 0 的第三个 0。

在状态 q_3 紧接着读到的 0 也可能是输入字符串的最后 3 个 0 的第三个 0。

如果在状态 q_1,q_2,q_3 读到的是 1，则要重新检查输入串是否以 3 个 0 结尾。

所以,相应的 DFA 的状态转移图如图 3-4 所示。

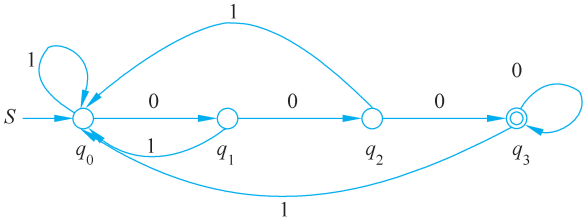

图 3-4　接受语言$\{x000 \mid x \in \{0,1\}^*\}$的 DFA

以下几点值得注意:

(1) 图 3-4 清楚地表示出了接受语言$\{x000 \mid x \in \{0,1\}^*\}$的 DFA,所以,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,并行的弧按弧上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,该有向路上从第一条边到最后一条边的标记依次并置构成字符串 x。简称此路的标记为 x。

(4) 一个 FA 可以有多于一个的终止状态。例如,图 3-5 是接受语言$\{x000 \mid x \in \{0,1\}^*\} \cup \{x001 \mid x \in \{0,1\}^*\}$的 FA,它有两个终止状态。

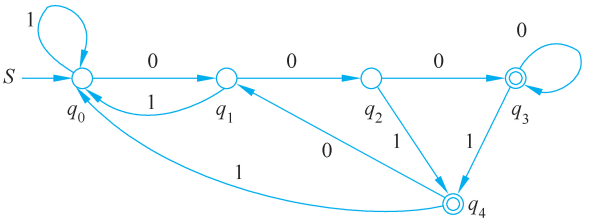

图 3-5　接受语言$\{x000 \mid x \in \{0,1\}^*\} \cup \{x001 \mid x \in \{0,1\}^*\}$的 FA

为了更方便地描述 FA 识别一个输入串的过程,定义 FA 的即时描述。

定义 3-5　设 $M=(Q,\Sigma,\delta,q_0,F)$为一个 FA,$x,y \in \Sigma^*$,$\delta(q_0,x)=q$,$xqy$ 称为 M 的一个**即时描述**(instantaneous description,ID)。它表示 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 的读头当前正指向 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。

将$\vdash_M$看作 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$\left(\vdash_M\right)^n$,$\left(\vdash_M\right)^*$,$\left(\vdash_M\right)^+$,按照二元关系合成的意义,有

$\alpha \vdash_M^n \beta$：表示 M 经过 n 次移动，即时描述从 α 变成 β。即 M 存在即时描述 $\alpha_1, \alpha_2, \cdots, \alpha_{n-1}$，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 经过至少 1 次移动，即时描述从 α 变成 β。

$\alpha \vdash_M^* \beta$：表示 M 经过若干步移动，即时描述从 α 变成 β。

当讨论的问题中只有唯一的 FA M 时，所进行的移动只能是 M 中的移动，此时符号中的 M 就可以省略了。一般地，为了简洁起见，当意义清楚时，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。

例如，用例 3-3 所定义的 M 处理输入串 1010010001，有

$$\begin{aligned} q_0 1010010001 &\vdash 1q_0 010010001 \\ &\vdash 10q_1 10010001 \\ &\vdash 101q_0 0010001 \\ &\vdash 1010q_1 010001 \\ &\vdash 10100q_2 10001 \\ &\vdash 101001q_0 0001 \\ &\vdash 1010010q_1 001 \\ &\vdash 10100100q_2 01 \\ &\vdash 101001000q_3 1 \\ &\vdash 1010010001q_0 \end{aligned}$$

即

$$q_0 1010010001 \vdash^{10} 1010010001q_0$$

$$q_0 1010010001 \vdash^+ 1010010001q_0$$

$$q_0 1010010001 \vdash^* 1010010001q_0$$

由于 M 在处理完 1010010001 后到达状态 q_0，所以 1010010001 不是 M 接受的语言的句子，虽然 M 在处理该字符串的过程中曾经经过终止状态 q_3。

不难证明，对于 $x \in \Sigma^*$，

$$q_0 x1 \vdash^+ x1q_0$$

$$q_0 x10 \vdash^+ x10q_1$$

$$q_0 x100 \vdash^+ x100q_2$$

$$q_0x000 \vdash^{+} x000q_3$$

定义 3-6 设 $M=(Q,\Sigma,\delta,q_0,F)$为一个 FA,对 $\forall q\in Q$,能引导 FA 从开始状态到达 q 的字符串的集合为

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

对图 3-5 所给的 DFA,有

$$set(q_0)=\{x\mid x\in\Sigma^*,x=\varepsilon \text{ 或者 } x \text{ 以 1 但不是 001 结尾}\}$$

$$set(q_1)=\{x\mid x\in\Sigma^*,x=0 \text{ 或者 } x \text{ 以 10 结尾}\}$$

$$set(q_2)=\{x\mid x\in\Sigma^*,x=00 \text{ 或者 } x \text{ 以 100 结尾}\}$$

$$set(q_3)=\{x\mid x\in\Sigma^*,x \text{ 以 000 结尾}\}$$

$$set(q_4)=\{x\mid x\in\Sigma^*,x \text{ 以 001 结尾}\}$$

细心观察会发现,上述 5 个集合是两两互不相交的,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。也就是说,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 $M=(Q,\Sigma,\delta,q_0,F)$,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。进一步的讨论放在后面进行。

例 3-4 构造一个 DFA,它接受的语言为$\{0^n1^m2^k\mid n,m,k\geqslant 1\}$。

该语言的句子的特点是:0 在最前面,1 在中间,2 在最后,它们不可以交叉出现,也不可以颠倒顺序,字符 0,1,2 的个数均不可少于 1。根据此,需要用 4 个状态:

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,接着读到了至少一个 2。

根据上述分析,可以得到如图 3-6 所示的状态转移图。

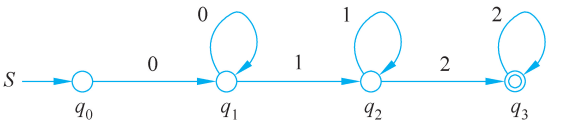

图 3-6 接受语言$\{0^n1^m2^k\mid n,m,k\geqslant 1\}$的 DFA 的主体框架

显然,图 3-6 不是要求的 DFA 的状态转移图,因为从此图中可以看出相应 FA 的输入字母表至少要包含字符 0,1,2,但是图中每个状态处并没有给出所有这 3 个字符的移动。按照语言的要求,如果未被标出的字符在这些状态出现,相应的输入串就肯定不是语言的句子。因此,我们引入第五个状态 q_t,一旦先前引入的状态发现输入串肯定不是语言的句子,就进入此状态,完成对输入字符串剩余部分的读入,即进行相应的例外处理。从而得到识别语言$\{0^n1^m2^k\mid n,m,k\geqslant 1\}$的 DFA(图 3-7)。

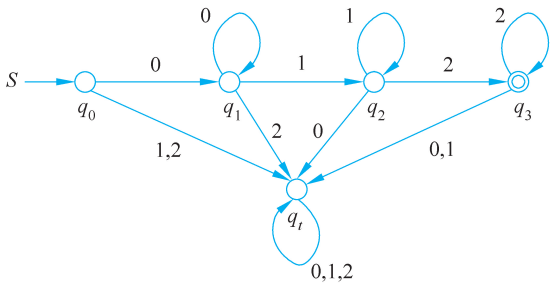

图 3-7　接受语言$\{0^n1^m2^k \mid n,m,k\geqslant 1\}$的 DFA

本例中有以下几点值得注意：

(1) 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，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所进入的状态。在此状态下，FA 读完输入串中剩余的字符。陷阱状态的引入，使得 FA 的构造更方便。

(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。可以先根据语言的主要特征画出该 FA 的“主体框架”，然后再去考虑一些细节要求。

(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，肯定是无法构造出相应的 FA 的。例如，对语言$\{0^n1^n \mid n\geqslant 1\}$，当扫描到 n 个 0 时，需要去寻找 n 个 1，由于 n 有无穷多个，所以，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$\{0^n1^n \mid n\geqslant 1\}$的 FA。这就是说，语言$\{0^n1^n \mid n\geqslant 1\}$不属于 FA 可以接受的语言类。相应的证明留到研究 FA 接受的语言类的性质时进行。

例 3-5　构造一个 DFA，它接受的语言为$\{x \mid x\in\{0,1\}^*$，且当把 x 看成二进制数时，x 模 3 与 0 同余$\}$。

从题目要求来看，该语言的句子都是 0,1 字符串。当把它们看成二进制数表示时，它们就是二进制数。先不考虑以 0 开头的非 0 数问题。作为一个二进制数，x 应该是非空的字符串。解答此题的第一个想法可能是希望构造的 DFA M 在读入字符串的过程中按照二进制的解释计算出它的值，然后计算出它除以 3 后的余数，当余数为 0 时就接受该字符串，否则就不接受。由于 $x\in\{0,1\}^*$，所以 x 有无穷多种，要计算出每个 x 的值，就要求 M 能够记忆这些值。但是，M 要想记忆这些值，就必须有无穷多个状态，每个状态只能对应一个值。所以，这种方法是不可取的。在定义 3-6 中曾经提到过，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。现在的问题是如何确定这些等价类。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 个状态分别与这 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。

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。

此外，由于要求 x 是非空的，所以还需要一个开始状态：

q_s——M 的开始状态。

下面逐一考虑在一个状态下读入一个字符时应该如何改变状态：

q_s——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 $x=3n+0$。

当 M 在此状态下读入 0 时，引导 M 到达下一个状态的字符串为 $x0$，此时 $x0=2(3n+0)=3\times 2n+0$。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，应该继续保持状态 q_0。即 $\delta(q_0,0)=q_0$。

当 M 在此状态下读入 1 时，引导 M 到达下一个状态的字符串为 $x1$，此时 $x1=2(3n+0)+1=3\times 2n+1$。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 $x=3n+1$。

当 M 在此状态下读入 0 时，引导 M 到达下一个状态的字符串为 $x0$，此时 $x0=2(3n+1)=3\times 2n+2$。这表明，$x0$ 应该属于 q_2 对应的等价类。所以，M 在 q_1 状态下读入 0 后，应该转到状态 q_2。即 $\delta(q_1,0)=q_2$。

当 M 在此状态下读入 1 时，引导 M 到达下一个状态的字符串为 $x1$，此时 $x1=2(3n+1)+1=3\times 2n+2+1=3(2n+1)$。这表明，$x1$ 应该属于 q_0 对应的等价类。所以，M 在 q_1 状态下读入 1 后，应该转到状态 q_0。即 $\delta(q_1,1)=q_0$。

q_2——能引导 M 到达此状态的 x 除以 3 余 2，所以有 $x=3n+2$。

当 M 在此状态下读入 0 时，引导 M 到达下一个状态的字符串为 $x0$，此时 $x0=2(3n+2)=3\times 2n+4=3(2n+1)+1$。这表明，$x0$ 应该属于 q_1 对应的等价类。所以，M 在 q_2 状态下读入 0 后，应该转到状态 q_1。即 $\delta(q_2,0)=q_1$。

当 M 在此状态下读入 1 时，引导 M 到达下一个状态的字符串为 $x1$，此时 $x1=2(3n+2)+1=3\times 2n+4+1=3(2n+1)+2$。这表明，$x1$ 应该属于 q_2 对应的等价类。所以，M 在 q_2 状态下读入 1 后，应该继续保持状态 q_2。即 $\delta(q_2,1)=q_2$。

按照上述分析，可以得到所求的 DFA M 的状态转移图，如图 3-8 所示。

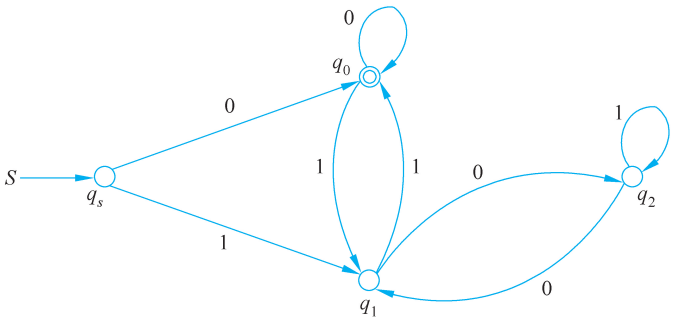

图 3-8　接受语言$\{x \mid x\in\{0,1\}^*$，且当把 x 看成二进制数时，x 模 3 与 0 同余$\}$的 DFA

注意，M 所接受的语言中，含有形如 000 的 0 和形如 000011 的非 0 二进制数。这些与通常的习惯是不一样的。请读者对图 3-8 所给出的 M 进行修改，使得 M 接受的语言中不含 $0^n(n>1)$的串和以 0 开头的非 0 串。

例 3-6 构造一个 DFA，它接受的语言 $L=\{x \mid x\in\{0,1\}^*$，且对 x 中任意一个长度不大于 5 的子串 $a_1a_2\cdots a_n$，$a_1+a_2+\cdots+a_n\leqslant 3(n\leqslant 5)\}$。

对 $\{0,1\}^*$ 中的任意一个字符串 $a_1a_2\cdots a_m$，当 $m\leqslant 3$ 时，肯定是满足要求的。当串的长度大于或等于 4 时，必须进行适当的考查，看是否存在不满足要求的子串。如果发现这样的子串，相应的字符串就一定不是 L 的句子，此时，可以让 DFA M 进入“陷阱状态”q_t，读完输入串中剩余的字符。为了叙述方便，按照子串的定义及 DFA 处理输入串的过程——从左到右逐个扫描输入串中的每个字符，来讨论 DFA 的构造。设输入串为

$$a_1a_2\cdots a_i\cdots a_{i+4}a_{i+5}\cdots a_m$$

M 对输入串的考查应按下列过程进行：

当 $i=1,2,3$，也就是 M 读到输入串的第 1、第 2、第 3 个字符时，它需要将这些字符记下来。因为 $a_1a_2\cdots a_i$ 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 $a_1+a_2+\cdots+a_i\leqslant 3$ 的情况下，M 需要将 $a_1a_2\cdots a_i$ 记下来；在 $a_1+a_2+\cdots+a_i>3$ 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，此时，以前读到的第 1 个字符 a_1 就没有用了，它要看 $a_2+a_3+\cdots+a_6\leqslant 3$ 是否成立，如果成立，M 需要将 $a_2a_3\cdots a_6$ 记下来；在 $a_2+a_3+\cdots+a_6>3$ 时，M 应该进入陷阱状态 q_t。

当 $i=7$，也就是 M 读到输入串的第 7 个字符，此时，以前读到的第 2 个字符 a_2 就没有用了，它要看 $a_3+a_4+\cdots+a_7\leqslant 3$ 是否成立，如果成立，M 需要将 $a_3a_4\cdots a_7$ 记下来；在 $a_3+a_4+\cdots+a_7>3$ 时，M 应该进入陷阱状态 q_t。

……

以此类推，当 M 完成对子串 $a_1a_2\cdots a_i\cdots a_{i+4}$ 的考查，并发现它满足语言的要求时，M 记下来的是 $a_ia_{i+1}\cdots a_{i+4}$，此时 M 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，它要看 $a_{i+1}+a_{i+2}+\cdots+a_{i+5}\leqslant 3$ 是否成立，如果成立，M 需要将 a_{i+1}，a_{i+2}，…，a_{i+5} 记下来；在 $a_{i+1}+a_{i+2}+\cdots+a_{i+5}>3$ 时，M 应该进入陷阱状态 q_t。

从上述分析来看，M 需要记忆的内容有：

什么都未读入——$2^0=1$ 种。

记录 1 个字符——$2^1=2$ 种。

记录 2 个字符——$2^2=4$ 种。

记录 3 个字符——$2^3=8$ 种。

记录 4 个字符——$2^4-1=15$ 种。

记录 5 个字符——$2^5-6=26$ 种。

记录当前的输入串不是句子——1 种。

可见 M 需要记忆的内容是有限多种。分别用不同的状态对应要记忆的不同内容。为了便于理解，直接用要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。

$q[a_1a_2\cdots a_i]$——M 记录有 i 个字符，$1\leqslant i\leqslant 5$。其中，$a_1,a_2,\cdots,a_i\in\{0,1\}$。

取 DFA $M=(Q,\{0,1\},\delta,q[\varepsilon],F)$，其中，

$$F=\{q[\varepsilon]\}\cup\{q[a_1a_2\cdots a_i]\mid a_1,a_2,\cdots,a_i\in\{0,1\}\text{且}\ 1\leqslant i\leqslant 5\ \text{且}\ a_1+a_2+\cdots+a_i\leqslant 3\}$$

$$Q=\{q_t\}\cup F$$

$$\delta(q[\varepsilon],a_1)=q[a_1]$$

$$\delta(q[a_1],a_2)=q[a_1a_2]$$

$$\delta(q[a_1a_2],a_3)=q[a_1a_2a_3]$$

$$\delta(q[a_1a_2a_3],a)=\begin{cases}q[a_1a_2a_3a] & \text{如果}\ a_1+a_2+a_3+a\leqslant 3\\ q_t & \text{如果}\ a_1+a_2+a_3+a>3\end{cases}$$

$$\delta(q[a_1a_2a_3a_4],a)=\begin{cases}q[a_1a_2a_3a_4a] & \text{如果}\ a_1+a_2+a_3+a_4+a\leqslant 3\\ q_t & \text{如果}\ a_1+a_2+a_3+a_4+a>3\end{cases}$$

$$\delta(q[a_1a_2a_3a_4a_5],a)=\begin{cases}q[a_2a_3a_4a_5a] & \text{如果}\ a_2+a_3+a_4+a_5+a\leqslant 3\\ q_t & \text{如果}\ a_2+a_3+a_4+a_5+a>3\end{cases}$$

$$\delta(q_t,a_1)=q_t$$

在以上各式中,$a,a_1,a_2,a_3,a_4,a_5\in\{0,1\}$。

在本例中,没有采用状态转移图的表示方法,也没有逐个地写出每一个具体输入字符和状态的定义,而是以变量(模式)的形式给出的,可以算作新的表示方式。因为本例给出的状态较多,共计57个,如果逐一给出每一个具体输入字符和状态的定义,会显得比较繁杂,也不宜理解。建议读者在今后描述FA时,根据具体的情况使用适当的方式。另外,也请读者考虑,是否可以去掉那些记录5个字符的状态。

此外,请读者注意,本例中使用的描述方式也可以理解成FA的状态具有有穷的存储功能。对本例而言,FA的状态最多可以存放5个字符。在开始未读入字符时,它的存储器是空的,然后每读入一个字符,就按照排队的顺序将当前读入的字符存入存储器,直到存储器中存满5个字符。在存储器中存满5个字符后又读入新的字符,此时将排在队首的字符"挤掉",并将当前读入的新的字符放入队尾。

与前面曾给出的FA相比,本例所给的FA的状态明显要多很多。在一些实际问题的求解中,会遇到需要更多状态的FA。本例所给的FA的状态数是否可以减少呢?粗略地考虑,这种可能性是存在的。在处理字符串的过程中,是在考查当前5个字符的和是否不大于3,这就是说,可以根据和的情况以及1在长度为5的子串中的分布来给出新的状态,而且这个新的状态有可能代表现在所给FA的若干状态。很明显,这种状态的设置是比较困难的。实际上,目前也不用从这样"颇费脑筋"的途径去寻找状态更少的等价FA。后面,将从FA所接受语言的一般特征出发,给出获取含有最少状态的FA的方法。

3.3 不确定的有穷状态自动机

3.3.1 作为对DFA的修改

考虑这样一个问题:设$\Sigma=\{0,1\}$,现在需要构造一个接受$L=\{x\mid x\in\Sigma^*$,且x含有子串00或11$\}$的DFA。按照例3-6所给的构造方法,可以比较容易地构造出如表3-4所示的DFA。

表 3-4　接受 L 的 DFA

状态说明	状　态	输入字符	
		0	1
开始状态	$q[\varepsilon]$	$q[0]$	$q[1]$
刚读入的字符为 0	$q[0]$	$q[00]$	$q[1]$
刚读入的字符为 1	$q[1]$	$q[0]$	$q[11]$
含 00，终止状态	$q[00]$	$q[00]$	$q[00]$
含 11，终止状态	$q[11]$	$q[11]$	$q[11]$

类似地，也可以给出接受语言$\{x \mid x \in \Sigma^*$，且 x 的倒数第 10 个字符为 1$\}$的 DFA。在没有掌握上述方法时，构造会显得非常困难。是否可以考虑图 3-9 和图 3-10 所示的 FA 的构造呢？如果可以，也就是说，这两个图给出的确实也是 FA，那么，今后在构造 FA 时就更方便了。

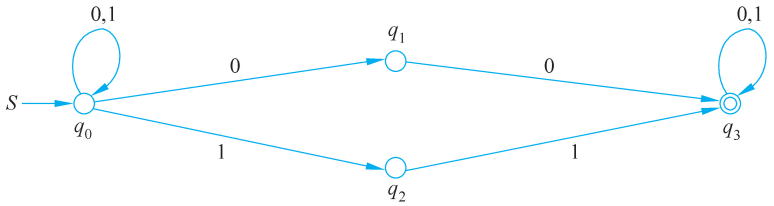

图 3-9　希望是接受语言$\{x \mid x \in \{0,1\}^*$，且 x 含有子串 00 或 11$\}$的 FA

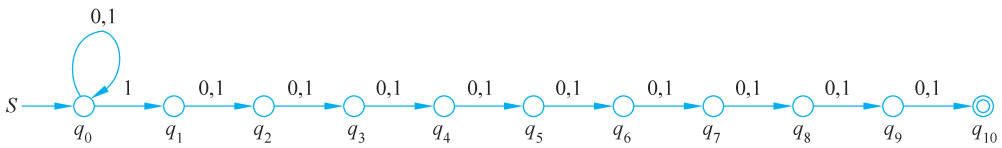

图 3-10　希望是接受语言$\{x \mid x \in \{0,1\}^*$，且 x 的倒数第 10 个字符为 1$\}$的 FA

这两个图所给的 FA 与前面所定义的 DFA 的区别在于：

(1) 并不是对于所有的$(q,a) \in Q \times \Sigma$，$\delta(q,a)$都有一个状态与之对应。

(2) 并不是对于所有的$(q,a) \in Q \times \Sigma$，$\delta(q,a)$只对应一个状态。

在这种 FA 中，对于所有的$(q,a) \in Q \times \Sigma$，$\delta(q,a)$对应的是 Q 的一个子集。对此可以理解成：FA 在任意时刻可以处于有穷多个状态，由于 Q 是有穷的，所以，2^Q 也是有穷的。由此可见，这种 FA 仍然是具有有穷个状态——它在任意时刻处于的多个状态构成 Q 的一个子集。因此，可以认为这种 FA 的一个状态对应的是先前定义的 DFA 的一个状态集合。

实际上，甚至还可以认为，这种 FA 具有“智能”：在一个状态下，它可以根据当前从输入字符串读入的字符自动地在集合 $\delta(q,a)$ 中选择进入一个正确的状态。例如，在图 3-9 所示的 FA 中，一旦它发现当前读入的 0 是要寻找的 00 中的第一个 0(或者当前读

入的1是要寻找的11中的第一个1),它就进入与当前状态不同的下一个新的状态(q_1或q_2);否则,它一直保持在开始状态(q_0)。在图3-10所示的FA中,一旦它发现当前读入的1是倒数第10个字符,它就进入与当前状态不同的下一个状态(q_1);否则,就停留在开始状态(q_0)。

在这种前提下,只要在如图3-9和图3-10所示的FA中存在一条从开始状态出发,最终到达某一个终止状态的标记为x的路径,那么就认为它接受了串x;否则,认为它不接受串x。

按照上述分析,这种FA可以看成对DFA的扩充,而且它很可能是与DFA等价的。下面将首先给出这种FA相应的定义,然后证明它与DFA的等价性。

3.3.2 NFA的形式定义

定义3-7 **不确定的有穷状态自动机**(non-deterministic finite automaton,NFA)M是一个五元组:

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

其中,Q,Σ,q_0,F的意义同DFA。

δ——**状态转移函数**,又称为**状态转换函数**或者**移动函数**,δ:$Q\times\Sigma\to 2^Q$。对$\forall(q,a)\in Q\times\Sigma,\delta(q,a)=\{p_1,p_2,\cdots,p_m\}$表示$M$在状态$q$读入字符$a$,可以选择地将状态变成$p_1,p_2,\cdots$,或者$p_m$,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。

定义3-4(FA的状态转移图)对NFA有效。根据实际情况,通过明确相应的表达对象,定义3-5(FA的即时描述)也可以使用。

根据上述定义,图3-9和图3-10表示的都是NFA。将NFA的状态转移图与DFA的状态转移图进行比较,可以看出,DFA是NFA的特例。将图3-9和图3-10表示的NFA分别记为M_1和M_2,它们的状态转移函数可以用表3-5和表3-6的形式给出。

表3-5 NFA M_1的状态转移函数

状态说明	状态	输入字符	
		0	1
开始状态	q_0	$\{q_0,q_1\}$	$\{q_0,q_2\}$
	q_1	$\{q_3\}$	$\varnothing$
	q_2	$\varnothing$	$\{q_3\}$
终止状态	q_3	$\{q_3\}$	$\{q_3\}$

在NFA中,同样需将δ扩充为$\hat{\delta}$:$Q\times\Sigma^*\to 2^Q$。对任意$q\in Q,w\in\Sigma^*,a\in\Sigma$,有

(1) $\hat{\delta}(q,\varepsilon)=\{q\}$。

(2) $\hat{\delta}(q,wa)=\{p\mid\exists r\in\hat{\delta}(q,w)$,使得$p\in\delta(r,a)\}$。

表 3-6　NFA M_2 的状态转移函数

状态说明	状　态	输入字符	
		0	1
开始状态	q_0	$\{q_0\}$	$\{q_0,q_1\}$
	q_1	$\{q_2\}$	$\{q_2\}$
	q_2	$\{q_3\}$	$\{q_3\}$
	q_3	$\{q_4\}$	$\{q_4\}$
	q_4	$\{q_5\}$	$\{q_5\}$
	q_5	$\{q_6\}$	$\{q_6\}$
	q_6	$\{q_7\}$	$\{q_7\}$
	q_7	$\{q_8\}$	$\{q_8\}$
	q_8	$\{q_9\}$	$\{q_9\}$
	q_9	$\{q_{10}\}$	$\{q_{10}\}$
终止状态	q_{10}	$\varnothing$	$\varnothing$

与 DFA 类似，由于 δ 的定义域 $Q\times\Sigma$ 是 $\hat{\delta}$ 的定义域 $Q\times\Sigma^*$ 的真子集：$Q\times\Sigma\subset Q\times\Sigma^*$，所以，对于任意 $(q,a)\in Q\times\Sigma$，如果 $\hat{\delta}$ 和 δ 都有相同的值，就不用区分这两个符号了。事实上，对于任意 $q\in Q$，$a\in\Sigma$，有

$$\begin{aligned}
\hat{\delta}(q,a)&=\hat{\delta}(q,\varepsilon a) && \varepsilon \text{ 是单位元素}\\
&=\{p\mid \exists r\in\hat{\delta}(q,\varepsilon),\text{使得 } p\in\delta(r,a)\} && \text{根据定义的第(2)条}\\
&=\{p\mid \exists r\in\{q\},\text{使得 } p\in\delta(r,a)\} && \text{根据定义的第(1)条}\\
&=\{p\mid p\in\delta(q,a)\} && q \text{ 是 } r \text{ 的唯一值}\\
&=\delta(q,a) && \text{集合的不同描述}
\end{aligned}$$

所以，今后用 δ 代替 $\hat{\delta}$。

由于对 $\forall(q,w)\in Q\times\Sigma^*$，$\delta(q,w)$ 是一个集合，因此，为了叙述方便，进一步扩充 δ 的定义域，δ：$2^Q\times\Sigma^*\to 2^Q$。对任意 $P\subseteq Q$，$w\in\Sigma^*$，有

$$\delta(P,w)=\bigcup_{q\in P}\delta(q,w)$$

由于，对 $\forall(q,w)\in Q\times\Sigma^*$，有

$$\begin{aligned}
\delta(\{q\},w)&=\bigcup_{q\in\{q\}}\delta(q,w)\\
&=\delta(q,w)
\end{aligned}$$

所以，并不一定严格地区分 δ 的第一个分量是一个状态还是一个含有一个元素的集合。这样，对任意 $q\in Q$，$w\in\Sigma^*$，$a\in\Sigma$，有

$$\delta(q,wa)=\delta(\delta(q,w),a)$$

可见,NFA M 从状态 q 出发,处理字符串 wa 的过程为:M 先处理 w,然后从处理 w 后所到达的状态出发,处理字符 a,并进入处理字符 a 后所能到达的所有状态。即对一个字符串 w,NFA M 与 DFA 类似,从左到右逐一地处理每个字符,它从处理完字符串中的第 i 个字符后所进入的状态集合中的任意状态开始,处理字符串中的第 $i+1$ 个字符。也就是说,对输入字符串 $a_1a_2\cdots a_n$,有

$$\delta(q,a_1a_2\cdots a_n)=\delta((\cdots\delta(\delta(q,a_1),a_2),\cdots),a_n)$$

定义 3-8 设 $M=(Q,\Sigma,\delta,q_0,F)$是一个 NFA。对于 $\forall x\in\Sigma^*$,如果 $\delta(q_0,x)\cap F\neq\varnothing$,则称 x 被 M 接受;如果 $\delta(q_0,x)\cap F=\varnothing$,则称 M 不接受 x。

$$L(M)=\{x\mid x\in\Sigma^*\text{ 且 }\delta(q_0,x)\cap F\neq\varnothing\}$$

称为由 M 接受(识别)的语言。

关于 FA 的等价定义 3-3 也适应 NFA。

NFA 和 DFA 是什么关系?我们希望从识别 $L=\{x\mid x\in\{0,1\}^*$ 且 x 的倒数第 3 个字符是 $1\}$的 NFA 和 DFA 的构造得到启发。

例 3-7 构造识别 $L=\{x\mid x\in\{0,1\}^*$ 且 x 的倒数第 3 个字符是 $1\}$的 NFA 和 DFA。

图 3-10 给出了$\{x\mid x\in\{0,1\}^*$ 且 x 的倒数第 10 个字符是 $1\}$的 NFA,参照图 3-10,很容易得到图 3-11 所示识别 L 的 NFA 的状态转移图。我们将此 NFA 记作 M_1。

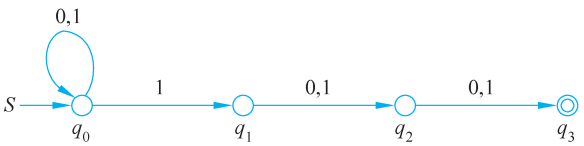

图 3-11 识别 $L=\{x\mid x\in\{0,1\}^*$ 且 x 的倒数第 3 个字符是 $1\}$的 NFA M_1 的状态转移图

为了构造出识别该语言的 DFA,一种策略是按照如下思路进行构造。构造的结果如图 3-12 所示。不妨将此 DFA 记作 M_2。

考虑到要识别的是倒数第 3 个字符是 1 的 0、1 串,利用状态的有穷记忆功能记录可能是输入串的倒数第 3 个字符"1"及相关信息。不难看出,状态最多只需要存放当前最新读入的 3 个字符。

为了表达清晰和简洁起见,这里使用一个新的命名规则:直接用当前存储器中存放的内容作为状态名称,从而有形如$[\varepsilon]$、$[a]$、$[ab]$、$[abc]$的状态,其中 $a,b,c\in\{0,1\}$为最新读入的字符,$[\varepsilon]$对应开始"记录"当前字符是否可能为输入串的倒数第 3 个字符"1"的状态,也就是启动状态。

从启动状态$[\varepsilon]$开始,如果读到的是 0,则它肯定不是要找的输入串的倒数第 3 个字符"1",因此,停留在状态$[\varepsilon]$;如果读到的是 1,则它有可能是要找的输入串的倒数第 3 个字符"1",需要将这个 1 记录下来,因此进入状态[1]。从而有

$$\delta([\varepsilon],0)=[\varepsilon]$$
$$\delta([\varepsilon],1)=[1]$$

在状态[1],如果读到的字符是 0,表明到目前为止,读到的倒数第 2 和倒数第 1 个字符依次是 1 和 0,而它们都可能是该输入串的倒数第 3 个字符,虽然 0 肯定不是要找的倒数第 3 个字符"1",但它需要用来表明它前面的 1 是目前的倒数第 2 个字符,所以需要记录下来,

因此进入状态[10]；如果读入的字符是1，表明到目前为止，读到的倒数第2个字符和倒数第1个字符都是1，而它们都有可能是要找的输入串的倒数第3个字符“1”，都需要记录下来，因此进入状态[11]。从而有

$$\delta([1],0)=[10]$$
$$\delta([1],1)=[11]$$

在状态[10]，如果读到的字符是0，表明到目前为止，读到的倒数第3个字符、倒数第2个字符和倒数第1个字符依次是1,0,0，而它们都很可能是输入串的倒数第3个字符，而且这两个0都需要用来表明它们前面的这个1目前是倒数第3个字符，所以，新读入的0也需要记录下来，因此进入状态[100]；如果读入的字符是1，表明到目前为止，读到的倒数第3个字符、倒数第2个字符和倒数第1个字符依次是1,0,1，这两个1都有可能是要找的输入串的倒数第3个字符“1”。中间的0需要用来表明它前面的1的位置，所以，都需要记下来，因此进入状态[101]。从而有

$$\delta([10],0)=[100]$$
$$\delta([10],1)=[101]$$

在状态[11]，如果读到的字符是0，表明到目前为止，读到的倒数第3个字符、倒数第2个字符和倒数第1个字符依次是1,1,0，这两个1都有可能是要找的输入串的倒数第3个字符“1”，虽然新读入的0不可能是倒数第3个字符“1”，它同样需要记录下来，用于体现[11]中的两个1的位置，因此进入状态[110]；如果读入的字符是1，表明到目前为止，读到的倒数第3个字符、倒数第2个字符和倒数第1个字符依次是1,1,1，而它们都有可能是要找的输入串的倒数第3个字符“1”，都需要记下来，因此进入状态[111]。从而有

$$\delta([11],0)=[110]$$
$$\delta([11],1)=[111]$$

在状态[100]，再读入一个新的字符，无论这个字符是0还是1，都表明状态[100]所记录的第一个字符1肯定不是倒数第3个字符，不用再保留。如果读到的字符是0，而到目前为止，读到的倒数第3个字符、倒数第2个字符和倒数第1个字符都是0，此时，无论是不是倒数第3个字符，它们肯定都不是要找的输入串的倒数第3个字符“1”，而且它们都不需要用来表明某个1的位置，此时需要重新开始寻找输入串的倒数第3个字符“1”，因此回到状态[ε]；如果读入的字符是1，表明到目前读到的字符为止，倒数第3个字符、倒数第2个字符和倒数第1个字符依次是0,0,1，前面的两个0都不可能是输入串的倒数第3个字符“1”，而且它们对表明当前读入的字符1是不是倒数第3个字符也没有意义，所以不用再存储，而这里的1则可能是输入串的倒数第3个字符，因此回到状态[1]。从而有

$$\delta([100],0)=[\varepsilon]$$
$$\delta([100],1)=[1]$$

在状态[101]，再读入一个新的字符，无论这个字符是0还是1，都表明状态[101]所记录的第一个字符1肯定不是要找的输入串的倒数第3个字符“1”，所以，无须保留；此时状态[101]中的0也肯定不是要找的输入串的倒数第3个字符“1”，而且这个0对表明当前读入的字符是不是倒数第3个字符也没有意义，所以这个0也不用再保留。状态[101]中的第3个字符1，仍然可能是要找的输入串的倒数第3个字符“1”，所以需要保留。此时，如果新读入的字符是0，则当前倒数第2个字符和倒数第1个字符依次是1和0，因此回到状态[10]；

如果读入的字符是1,则当前倒数第2个字符和倒数第1个字符都是1,因此回到状态[11]。从而有

$$\delta([101],0)=[10]$$
$$\delta([101],1)=[11]$$

在状态[110],再读入一个新的字符,无论这个字符是0还是1,都表明状态[110]所记录的第一个字符1肯定不是倒数第3个字符,所以无须保留。此时如果新读入的是0,它需要用来表明状态[110]中10的位置,所以需要记录下来,因此进入状态[100];如果读入的字符是1,表明目前读到的字符为止,倒数第3个字符和倒数第1个字符都是1,而它们都有可能是要找的输入串的倒数第3个字符"1",而其中的0需要用来表示10中的1的位置,所以这个0也需要保留下来,因此进入状态[101]。从而有

$$\delta([110],0)=[100]$$
$$\delta([110],1)=[101]$$

在状态[111],再读入一个新的字符,无论这个字符是0还是1,都表明状态[111]所记录的第一个字符1肯定不是倒数第3个字符,所以无须保留。此时如果新读入的是0,它需要用来表明状态[111]中后两个1的位置,所以需要记录下来,因此进入状态[110];如果读入的字符是1,表明目前读到的字符为止,倒数第3个字符、倒数第2个字符和倒数第1个字符都是1,而它们都很可能是输入串的倒数第3个字符"1",因此进入状态[111]。从而有

$$\delta([111],0)=[110]$$
$$\delta([110],1)=[111]$$

这里的状态[100],[101],[110],[111]都表达了当前已经读入的串的倒数第3个字符是1,满足 L 的要求,所以它们是终止状态,而其他状态都表达了当前已经读入的串的倒数第3个字符不是1,不满足 L 的要求,所以这些状态就不是终止状态。

这个DFA的状态转移图如图3-12所示,我们将它记作 M_2。

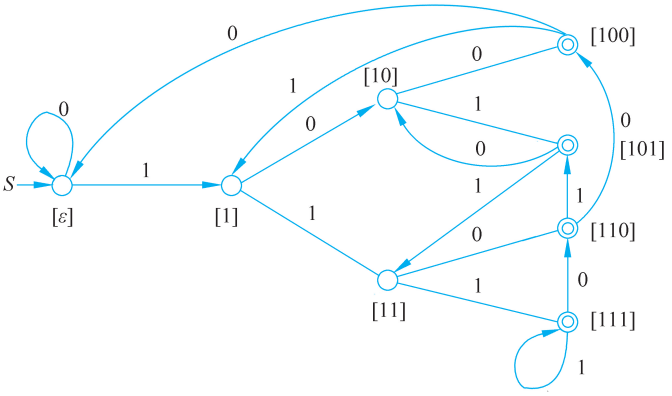

图3-12　识别 $L=\{x \mid x\in\{0,1\}^*$,且 x 的倒数第3个字符是1$\}$ 的DFA M_2 的状态转移图

我们回到图3-11,按照NFA的移动函数 δ 的扩展意义,容易得到如表3-7所示的计算结果。请读者注意此表中第2列中状态集合的出现顺序,以设计出生成此表的算法,对根据给定的NFA构造等价的DFA具有重要意义。