# 第2章 词法分析



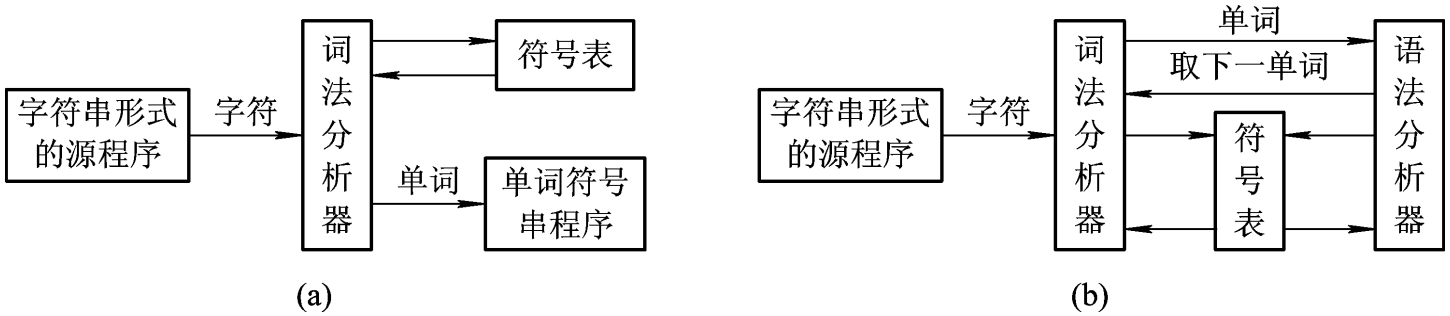
BACK

词法分析可以采用如下两种处理结构：

(1) 把词法分析程序作为主程序。将词法分析工作作为独立的一遍来完成，即把词法分析与语法分析明显分开，由词法分析程序将字符串形式的源程序改造成单词符号串形式的中间程序，以这个中间程序作为语法分析程序的输入。在这种处理结构中，词法分析和语法分析是分别实现的，如图2-1(a) 所示。

(2) 把词法分析程序作为语法分析程序调用的子程序。在进行语法分析时，每当语法分析程序需要一个单词时便调用词法分析程序，词法分析程序每一次调用便从字符串源程序中识别出一个单词交给语法分析程序。在这种处理结构中，词法分析和语法分析实际上是交替进行的，如图2-1(b) 所示。

由于把词法分析器安排成一个子程序比较自然，因此，词法分析程序通常采用第二种处理结构。




(a) 词法分析程序作为主程序；(b) 词法分析程序作为子程序

图2-1 词法分析的两种处理结构

# 2.1 词法分析器的设计方法

## 2.1.1 单词符号的分类与输出形式

### 1. 单词符号分类

词法分析程序简单地说就是读单词程序，该程序扫描用高级语言编写的源程序，将源程序中由单词符号组成的字符串分解出一个个单词来。因此，单词符号是程序语言的基本语法单位，具有确定的语法意义。程序语言的单词符号通常可分为下面五种。

(1) 保留字(也称基本字)：如C语言中的if、else、while和do等，这些字保留了语言所规定的含义，是编译程序识别各类语法成分的依据。几乎所有程序语言都限制用户使用保留字来作为标识符。

(2) 标识符：用来标记常量、数组、类型、变量、过程或函数名等，通常由用户自己定义。

(3) 常数：包括各种类型的常数，如整型常数386、实型常数0.618、布尔型常数TRUE等。

(4) 运算符：如“+”、“–”、“*”、“/”、“>”、“<”等。

(5) 界符：在语言中是作为语法上的分界符号使用的，如“，”、“；”、“(”、“)”等。

**注意：**一个程序语言的保留字、运算符和界符的个数是确定的，而标识符或常数的使用则不限定个数。

**2．词法分析程序输出单词的形式**

我们知道，词法分析程序的输入是源程序字符串，而输出是与源程序等价的单词符号序列，并且所输出的单词符号通常表示成如下的二元式：

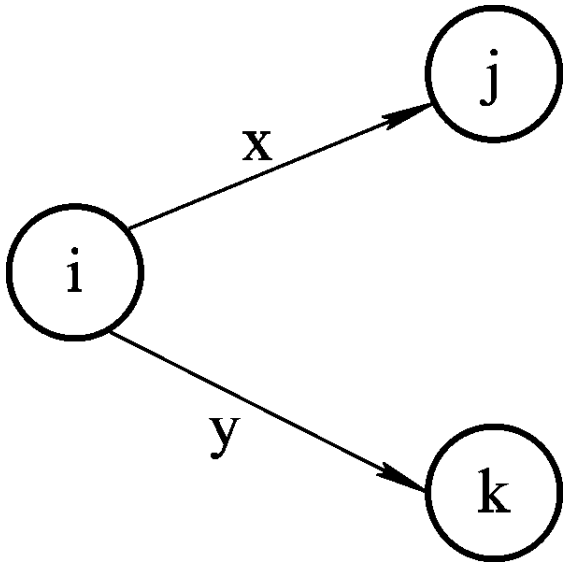
(单词种别，单词自身的值)

(1) 单词种别。单词种别表示单词的种类，它是语法分析所需要的信息。一个语言的单词符号如何划分种类、分为几类、如何编码都属于技术性问题，主要取决于处理上的方便。通常让每种单词对应一个整数码，这样可最大限度地把各个单词区别开来。对于保留字，可将其全体视为一种，也可一字一种，采用一字一种的分类方法处理起来比较方便；标识符一般统归为一种；常数可统归为一种，也可按整型、实型、布尔型等分为几种；运算符和界符可采用一符一种的分法，也可统归为一种。

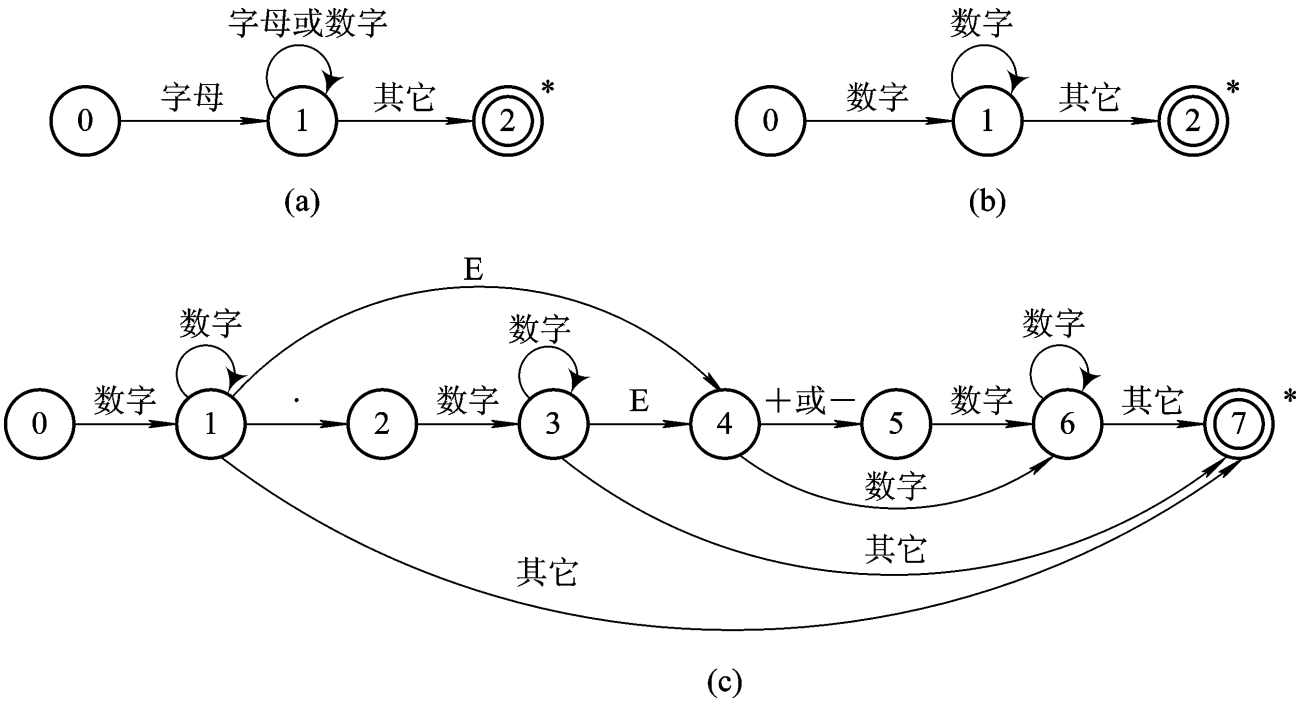
(2) 单词自身的值。单词自身的值是编译中其它阶段所需要的信息。对于单词符号来说，如果一个种别只含有一个单词符号，那么对于这个单词符号，其种别编码就完全代表了它自身的值。如果一个种别含有多个单词符号，那么对于它的每个单词符号，除了给出种别编码之外还应给出单词符号自身的值，以便把同一种类的不同单词区别开来。注意，标识符自身的值就是标识符自身的字符串，而常数自身的值是常数本身的二进制数值。此外，我们也可用指向某类表格中一个特定项目的指针来区分同类中的不同单词。例如，对于标识符，可以用它在符号表的入口指针作为它自身的值；而常数也可用它在常数表的入口指针作为它自身的值。

### 2.1.2 状态转换图

在词法分析中，可以用状态转换图来识别单词。状态转换图是有限的有向图，结点代表状态，用圆圈表示；结点之间可由有向边连接，有向边上可标记字符。例如，图2-2表示在状态i下，若输入字符为x，则读入x并转换到状态j；若输入字符为y，则读入y并转换到状态k。


j
x
i
y
k

状态(即结点)数是有限的，其中必有一初始状态以及若干终止状态，终止状态(终态)的结点用双圈表示以区别于其它状态。图2-3给出了用于识别标识符、无符号整数、无符号数的状态转换图，其初始状态均用0状态表示。




(a) 标识符；(b) 无符号整数；(c) 无符号数

图2-3 标识符及无符号数的状态转换图

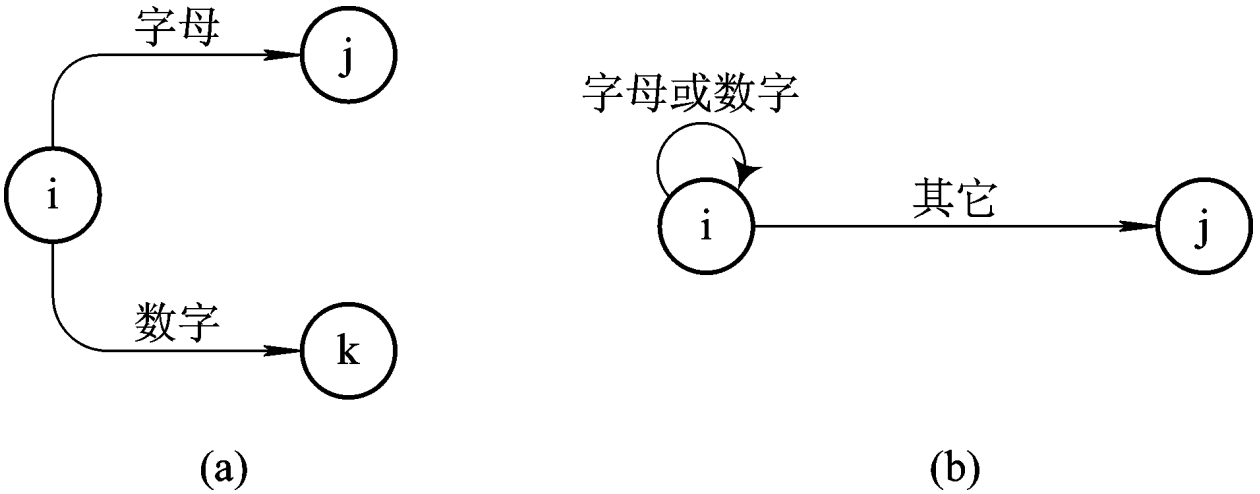
当到达一类单词符号的终止状态时即可给出相应的单词编码。某些终止状态是在读入了一个其它不属于该单词的符号后才得到相应的单词编码的，这表明在识别单词的过程中多读入了一个符号，所以识别出单词后应将最后多读入的这个符号予以回退；我们对此类情况的处理是在终态上以“*”作为标识。

对于不含回路的分支状态来说，可以让它对应一个switch( ) 语句或一组if-else语句。例如，图2-4(a) 的状态i所对应的switch语句如下：

```
s=getchar( );
switch(s)
{    case 'a':
     case 'b':
       …
     case 'z':
       … ;                    //实现状态 j 功能的语句
     case '0':
     case '1':
       …
     case '9':
       … ;                    //实现状态 k 功能的语句
}
```

对于含回路的状态来说，可以让它对应一个 while 语句。例如，图 2-4(b) 的状态 i 所对应的 while 语句如下：

```
getchar( );
while( letter( )||digit( ))
     getchar( );
…;                            //实现状态 j 功能的语句
```




(a) 含分支的状态i；(b) 含回路的状态i

图2-4 含有分支或回路的状态示意

# 2.2 一个简单的词法分析器示例

## 2.2.1 C语言子集的单词符号表示

一个非常重要的事实是：大多数程序语言的单词符号都可以用状态转换图予以识别。作为一个综合例子，我们来构造一个C语言子集的简单词法分析器。表2.1列出了这个C语言子集的所有单词符号以及它们的种别编码和内码值。由于直接使用整数编码不利于记忆，故该例中用一些特殊符号来表示种别编码。

表 2.1 C 语言子集的单词符号及内码值

| 单词符号 | 种别编码 | 助记符 | 内码值 |
|---|---|---|---|
| while | 1 | while | — |
| if | 2 | if | — |
| else | 3 | else | — |
| switch | 4 | switch | — |
| case | 5 | case | — |
| 标识符 | 6 | id | id 在符号表中的位置 |
| 常数 | 7 | num | num 在常数表中的位置 |
| + | 8 | + | — |
| − | 9 | − | — |
| * | 10 | * | — |
| <= | 11 | relop | LE |
| < | 11 | relop | LT |
| == | 11 | relop | EQ |
| = | 12 | = | — |
| ; | 13 | ; | — |

### 2.2.2 C语言子集对应的状态转换图

在设计的状态转换图中，首先对输入串做预处理，即剔除多余的空白符(在实际的词法分析中，预处理还包括剔除注释和制表换行符等编辑性字符的工作)，使词法分析工作既简单又清晰。其次，将保留字作为一类特殊的标识符来处理，也即对保留字不专设对应的状态转换图，当转换图识别出一个标识符时就去查对表2.1的前五项，确定它是否为一个保留字。当然，也可以专设一个保留字表来进行处理。

图2-5就是对应表2.1这个简单词法分析的状态转换图。

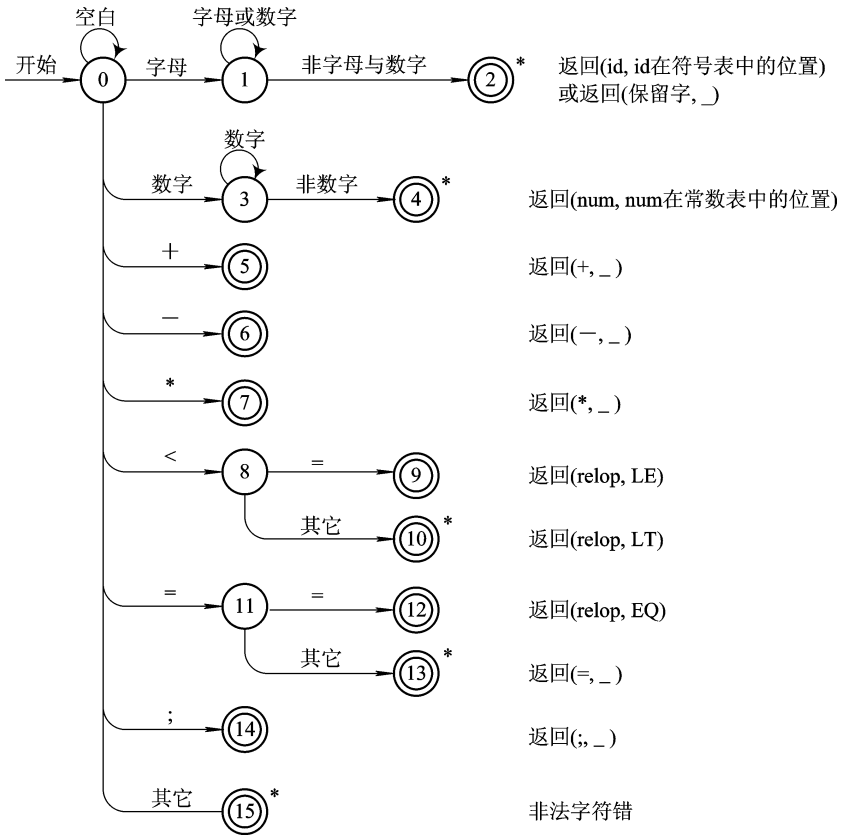



图2-5 简单词法分析的状态转换图

**注意：**在状态2时，所识别出的标识符应先与表2.1的前五项逐一比较，若匹配，则该标识符是一个保留字，否则就是标识符。如果是标识符，应先查符号表，看表中是否有此标识符。若表中无此标识符，则将它登录到符号表中，然后返回其在符号表中的入口指针(地址)作为该标识符的内码值；若表中有此标识符，则给出重名错误信息。在状态4时，应将识别的常数转换成二进制常数并将其登录到常数表中，然后返回其在常数表中的入口指针作为该常数的内码值。

### 2.2.3 状态转换图的实现

状态转换图非常容易用程序实现，最简单的办法是让每个状态对应一小段程序。对于图2-5所示的状态转换图，我们首先引进一组变量和函数如下：

(1) character：字符变量，存放最新读入的源程序字符。

(2) token：字符数组，存放构成单词符号的字符串。

(3) getbe( )：若character中的字符为空白，则调用getchar( )，直至character为非空白字符为止。

(4) concatenation( )：将token中的字符串与character中的字符连接并作为token中新的字符串。

(5) letter( )和digit( )：判断character中的字符是否为字母和数字的布尔函数，是则返回true(即1)，否则返回false(即0)。

(6) reserve( )：按token数组中的字符串查表2.1中的前五项(即判别其是否为保留字)，若是保留字则返回它的种别编码，否则返回0值。

(7) retract( )：扫描指针回退一个字符，同时将character置为空白。

(8) buildlist( )：将标识符登录到符号表中或将常数登录到常数表中。

(9) error( )：出现非法字符，显示出错信息。

相对于图2-5的词法分析器构造如下：

```
token=' ';                          //对 token 数组初始化
character=getchar( );
getbe( );                           //滤除空格
switch(character)
{
    case 'a':
    case 'b':
    …
    case 'z':
        while( letter( ) || digit( ))
        {
            concatenation( );       //将当前读入的字符送入 token 数组
            character =getchar( );
        }
        retract( );                 //扫描指针回退一个字符
        c=reserve( );
        if(c==0)
        {
```

```
            buildlist( );                    //将标识符登录到符号表中
            return(id, 指向 id 的符号表入口指针);
        }
        else
             return(C 语言子集中的保留字种别编码, null);
        break;
case '0':
case '1':
…
case '9':
        while(digit( ))
        {
            concatenation( );
            character=getchar( );
```

```
        }
        retract( );
        buildlist( );                          //将常数登录到常数表中
        return(num, num的常数表入口指针);
        break;
case '+':
        return('+', null);
        break;
case '-':
        return('-', null);
        break;
case '*':
        return('*', null);
        break;
case '<':
        character=getchar( );
        if(character=='=')
                return(relop, LE);
        else
        {
```

```
            retract( );
            return(relop, LT);
        }
        break;
case '=':
        character=getchar( );
        if(character== '=')
            return(relop, EQ);
        else
        {
            retract( );
            return('=', null);
        }
        break;
case ';':
        return(';', null);
        break;
default:
        error( );
```

# 2.3 正规表达式与有限自动机简介

## 2.3.1 正规表达式与正规集

状态转换图对构造词法分析程序是行之有效的，为了便于词法分析器的自动生成，还须将状态转换图的概念加以形式化。正规表达式就是一种形式化的表示法，它可以表示单词符号的结构，从而精确地定义单词符号集。正规表达式简称为正规式，它表示的集合即为正规集。

为了理解正规式与正规集的含义，我们以程序语言中的标识符为例予以说明。程序语言中使用的标识符是一个以字母开头的字母数字串，如果字母用letter表示，数字用digit表示，则标识符可表示为

letter (letter｜digit)*

其中，letter与 (letter｜digit)*的并置表示两者的连接；括号中的“｜”表示letter或digit两者选一；“*”表示零次或多次引用由“*”标记的表达式；(letter｜digit)*是letter｜digit的零次或多次并置，即表示一长度为0、1、2、…的字母数字串；letter (letter｜digit)*表示以字母开头的字母数字串，也即标识符集。letter (letter｜digit)*就是表示标识符的正规式，而标识符集就是这个正规式所表示的正规集。

对于给定的字母表Σ，正规式和正规集的递归定义如下：

(1) ε和Φ都是Σ上的正规式，它们所表示的正规集分别为{ε}和Φ。

(2) 对任一个a∈Σ，a是Σ上的一个正规式，它所表示的正规集为{a}。

(3) 如果R和S是Σ上的正规式，它们所表示的正规集分别为L(R) 和L(S)，则：

① R｜S是Σ上的正规式，它所表示的正规集为L(R)∪L(S)；

② R·S是Σ上的正规式，它所表示的正规集为L(R) L(S)；

③ (R)*是Σ上的正规式，它所表示的正规集为(L(R))*；

④ R也是Σ上的正规式，它所表示的正规集为L(R)。

(4) 仅由有限次使用规则(1)～(3)得到的表示式是Σ上的正规式，它所表示的集合是Σ上的正规集。

在上述定义中，规则(1)、(2)为基础规则，规则(3)为归纳规则，规则(4)是界限规则或终止规则。此外，Σ上的一个字是指由Σ中的字符所构成的一个有穷序列；不包含任何字符的序列称为空字，用ε表示。我们用Σ*表示Σ上所有字的全体，则空字ε也在其中。例如，若Σ = {a, b}，则Σ* = {ε, a, b, aa, ab, ba, bb, aaa, …}。我们还用Φ表示不含任何元素的空集{}。这里需要注意ε、{ }和{ε}的区别：{ε}是由空字组成的集合，而{ }则表示不含任何字的集合。

正规式间的运算符“｜”表示或，“•”表示连接(通常可省略)，“＊”表示闭包，使用括号可以改变运算的次序。如果规定“＊”优先于“•”，“•”优先于“｜”，则在不出现混淆的情况下括号也可以省去。注意，Σ*的正规式R和S的连接可以形式化地定义为

$$RS=\{\alpha\beta \mid \alpha\in R \ \&\ \beta\in S\}$$

即集合RS中的字是由R和S中的字连接而成的，且R自身的n次连接记为

$$R^n=\underbrace{RR\cdots R}_{n\text{个}}$$

对于Σ上的正规式R和S，如果它所表示的正规集L(R) =L(S)，则称R和S等价并记为R=S。不难证明，正规式具有下列性质：

(1) 交换律：R｜S = S｜R。

(2) 结合律：R｜(S｜T) = (R｜S)｜T；R(ST) = (RS)T。

(3) 分配律：R(S｜T) = RS｜RT；　(R｜S)T = RT｜ST。

(4) 同一律：εR = Rε= R。

**例2.1** 令Σ={a, b}，设R=a (a｜b)* 是Σ上的正规式，试求其表示的正规集。

**[解答]**

L(R) =L(a(a｜b)*) =L(a)L((a｜b)*) =L(a) (L(a｜b))*=L(a) (L(a)∪L(b))*

={a}({a}∪{b})*={a}{a,b}*={a}{ε, a, b, aa, ab, ba, bb, aaa, …}

={a, aa, ab, aaa, aab, aba, abb, aaaa, …}

**例2.2** 判断下述正规式之间是否等价：

(1) (a｜b)*与a*｜b*

(2) (ab)*与a*b*

(3) (a｜b)*与(a*b*)*

**[解答]**

(1) (a｜b)*对应的正规集其a、b可任意交替出现，如abbaaaba…；而(a*｜b*)对应的正规集只可出现任意个a或者任意个b；因此两者不等价。

(2) (ab)*对应的正规集是以任意个ab对出现的，即ababab…；而a*b*对应的正规集则是先出现任意个a后接任意个b，即a…ab…b；因此两者不等价。

(3) 由于(a｜b)*对应的正规集其a、b可任意交替出现，如aababbb；而(a*b*)*可采用如下构造方法得到字aababbb：

$(a^*b^*)^2 = (a^*b^*)^0 \cup (a^2b1)1 \cup (a^1b^3)^2 = aababbb$

反之，对(a*b*)*产生的任意字也可由(a｜b)*得到，即两者是等价的。

**例2.3** 证明：设$L(a+)=\{a\}^*-\{\varepsilon\}$，则有$a+=aa^*$。

**[证明]** $L(a+)=\{a\}^*-\{\varepsilon\}=\{\varepsilon, a, a^2, a^3, \ldots\}-\{\varepsilon\}$

$=\{a, a^2, a^3, \ldots\}=\{a\}\cdot\{\varepsilon, a, a^2, \ldots\}$

$=\{a\}\{a\}^*=L(a)\ L(a^*)=L(aa^*)$

故 $a+=aa^*$

## 2.3.2 有限自动机

有限自动机(FA)是更一般化的状态转换图，它分为确定有限自动机DFA和非确定有限自动机NFA两种。

### 1．确定有限自动机(DFA)

一个确定的有限自动机$M_d$(记为DFA $M_d$)是一个五元组$M_d$= (S，Σ，f，$s_0$，Z)，其中：

(1) S是一个有限状态集，它的每一个元素称为一个状态。

(2) Σ是一个有穷输入字母表，它的每一个元素称为一个输入字符。

(3) f是一个从S × Σ到S的单值映射，即f($s_i$，a) =$s_j$且有$s_i$、$s_j$∈S和a∈Σ。

(4) $s_0$∈S，是唯一的一个初态。

(5) Z S，是一个终态集。

**注意：** f($s_i$，a) =$s_j$表示当前状态为si且输入字符为a时，自动机将转换到下一个状态$s_j$，也即$s_j$称为$s_i$的一个后继状态。状态转换函数f是单值函数，f($s_i$, a) 唯一确定了下一个要转换的状态，即由每个状态发出的有向边(输出边)上所标记的输入字符各不相同。

例如，对图2-6所给出的状态s1有：

$$f(s_1, a) = s_2$$

$$f(s_1, b) = s_3$$

$$f(s_1, c) = s_4$$

因此，f是单值映射函数。

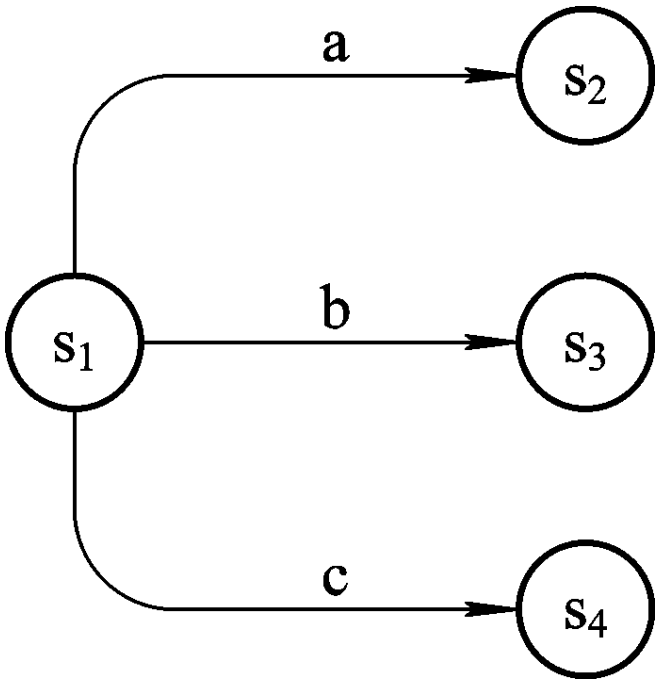



图2-6 DFA的状态转换示意

**2．非确定有限自动机(NFA)**

一个非确定有限自动机Mn(记为NFA $M_n$)是一个五元组$M_n$= (S，Σ，f，Q，Z)，其中：

(1) S、Σ、Z的意义与DFA相同。

(2) f是一个从S × Σ*到S的子集映射。

(3) Q S，是一个非空初态集。

NFA和DFA的区别主要有两点：其一是NFA可以有若干个初始状态，而DFA仅有一个初始状态；其二是NFA的状态转换函数f不是单值函数，而是一个多值函数，即f($s_i$，a) = {某些状态的集合}($s_i$∈S)，它表示不能由当前状态和当前输入字符唯一地确定下一个要转换的状态，也即允许同一个状态对同一个输入字符可以有不同的输出边。

例如，对图2-7所给出的状态$s_1$有：

$$f(s_1, a) = \{s_1, s_2, s_3\}$$

即f是一个从S × Σ*到S的子集映射；Σ*表示输出边上所标记的不仅是字符，也可以是字。此外，NFA还允许$f(s_1, \varepsilon) = \{$某些状态的集合$\}$，即在NFA的状态转换图中输出边上的标记还可是ε (空字)。

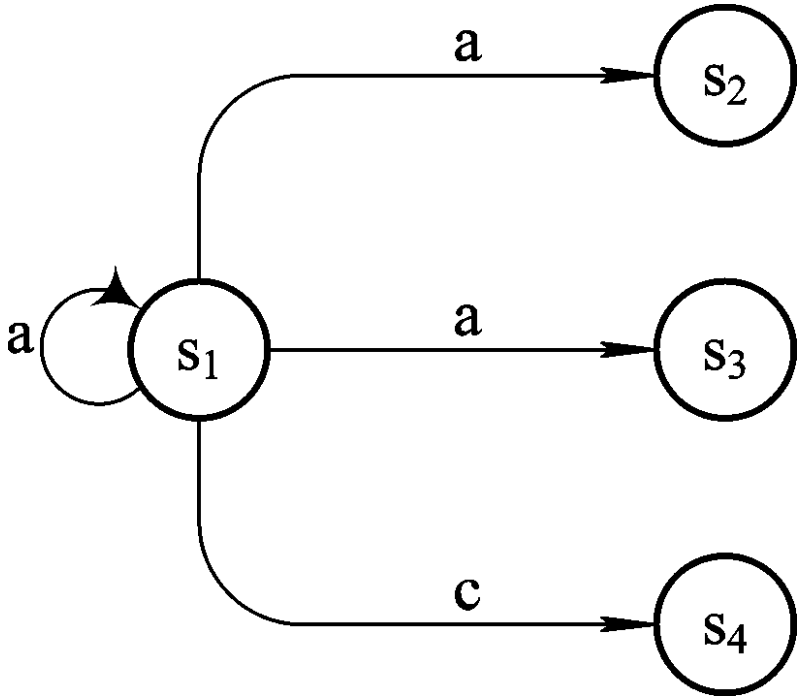



图2-7 NFA的状态转换示意

### 3．状态转换图与状态转换矩阵

DFA和NFA都可以用状态转换图表示。假定DFA(或NFA)有m个状态、n个输入字符(或字)，则这个状态转换图含有m个状态，每个状态最多有n条输出边与其它状态相连接，每一条输出边用Σ(或Σ*)中的一个不同的输入字符(或一个输入字)作标记，整个图含有唯一一个初态(或多个初态)和若干个终态。

DFA和NFA也可以用状态转换矩阵表示。状态转换矩阵的行表示状态，列表示输入符号，矩阵元素表示f(si，a) 的值。

**例2.4** 假定DFA Md= ({$s_0$，$s_1$，$s_2$}，{a，b}，f，$s_0$，{$s_2$})，且有：

f ($s_0$，a) =$s_1$　　f ($s_0$，b) =$s_2$

f ($s_1$，a) =$s_1$　　f ($s_1$，b) =$s_2$

f ($s_2$，a) =$s_2$　　f ($s_2$，b) =$s_1$

试给出DFA $M_d$的状态转换图与状态转换矩阵。

**[解答]** DFA $M_d$的状态转换图见图2-8，状态转换矩阵见表2.2。

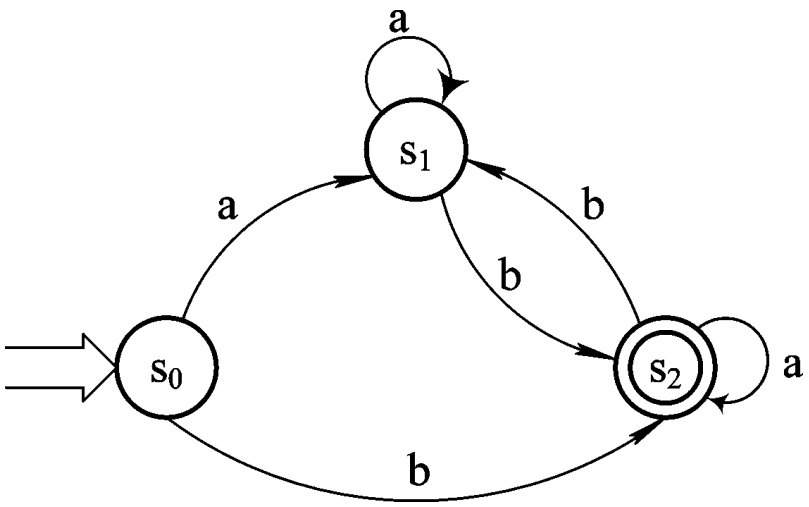



图2-8 例2.4的DFA Md状态转换图

表 2.2 状态转换矩阵

| 状态 \ 字符 | a | b |
|---|---|---|
| $S_0$ | $S_1$ | $S_2$ |
| $S_1$ | $S_1$ | $S_2$ |
| $S_2$ | $S_2$ | $S_1$ |

**例2.5** 假定NFA Mn= ({$s_0$，$s_1$，$s_2$}，{a, b}，f，{$s_0$，$s_2$}，{$s_1$})，且有：

f($s_0$，a) = {$s_2$}　　f($s_0$，b) = {$s_0$, $s_1$}

f($s_1$，a) = Φ　　f($s_1$，b) = {$s_2$}

f($s_2$，a) = Φ　　f($s_2$，b) = {$s_1$}

试给出NFA $M_n$的状态转换图与状态转换矩阵。

**[解答]** NFA Mn的状态转换图见图2-9，状态转换矩阵见表2.3。

对于FA M和任一Σ上的字符串α(即α∈Σ*)，如果存在一条从初始状态到终止状态的通路，通路上有向边(输出边)所标识的字符依次连接所得到的字符串恰为α，则称α可以为FA M所接受或称α为FA M所识别。FA M所能识别的字符串集称为FA M所识别的语言，记为L(M)。

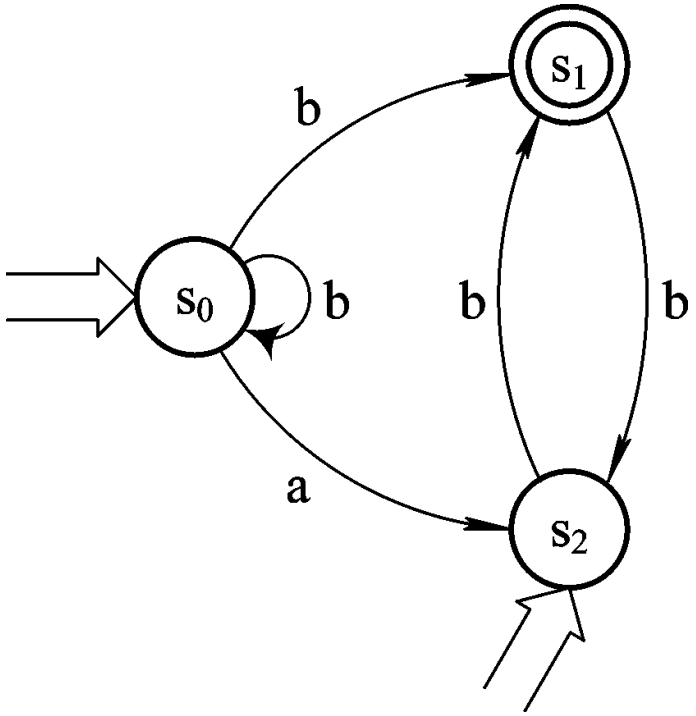



图2-9 例2.5的NFA Mn的状态转换图

表 2.3 状态转换矩阵

| 状态 \ 字 | a | b |
|---|---|---|
| s0 | {s2} | {s0, s2} |
| s1 | Φ | {s2} |
| s2 | Φ | {s1} |

**注意：**对于任何两个FA M和FA M'，如果L(M) =L(M')，则称有限自动机M和M'等价。此外，对于任一给定的NFA M，一定存在一个DFA M'，使L(M) =L(M')。因此，DFA是NFA的特例，NFA可以有DFA与之等价，即两者描述能力相同。DFA便于识别，易于计算机实现，而NFA便于定理的证明。

## 2.4 正规表达式到有限自动机的构造

### 2.4.1 由正规表达式构造等价的非确定有限自动机(NFA)

由正规表达式构造等价的NFA M的方法如下：

(1) 将正规表达式R表示成如图2-10所示的拓广转换图。

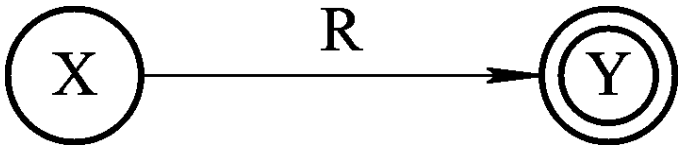



图2-10 拓广转换图

(2) 对正规表达式采用如图2-11所示的三条转换规则来构造NFA M。

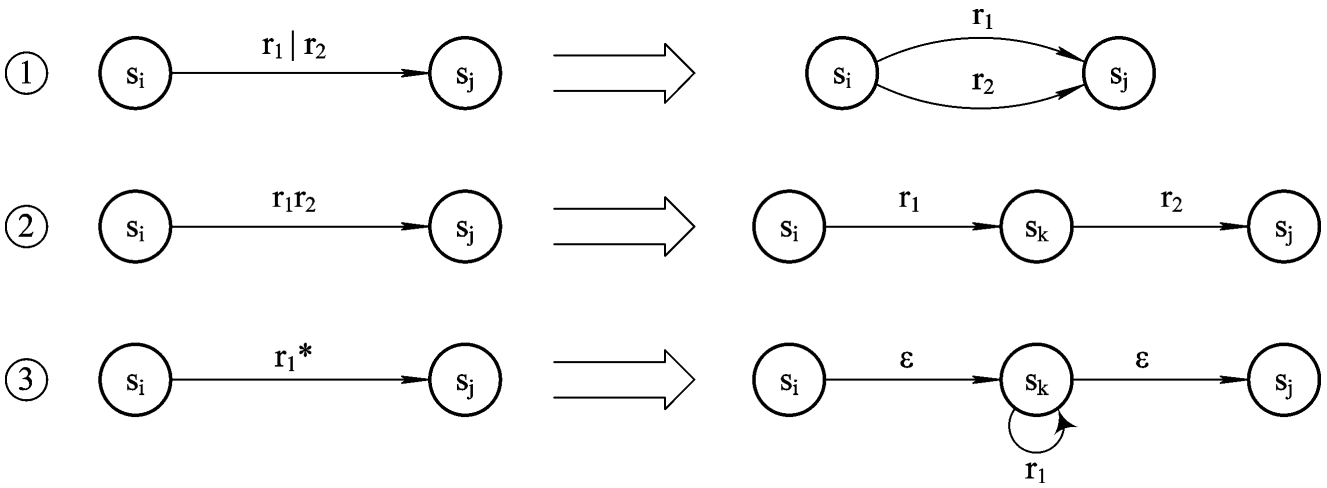



图2-11 转换规则

对于给定的正规表达式R，首先将其表示成如图2-10所示的形式，其中X为初始状态，Y为终止状态；然后逐步将这个拓广转换图运用图2-11的三条转换规则不断加入新结点进行分解，直至每条有向边上仅标识有Σ的一个字母或ε为止，则NFA M构造完成。

**例2.6** 对给定正规表达式b*(d｜ad) (b｜ab)+构造其NFA M。

**[解答]** 先用R+=RR*改造题设的正规表达式为b*(d｜ad) (b｜ab) (b｜ab)*，然后构造其NFA M，如图2-12所示。

注意，拓广后的状态转换图中如果有下面两种情况之一存在，则为NFA M：

(1) 有ε边存在。

(2) 某结点对同一输入字符存在多条输出边(即为多值映射)。

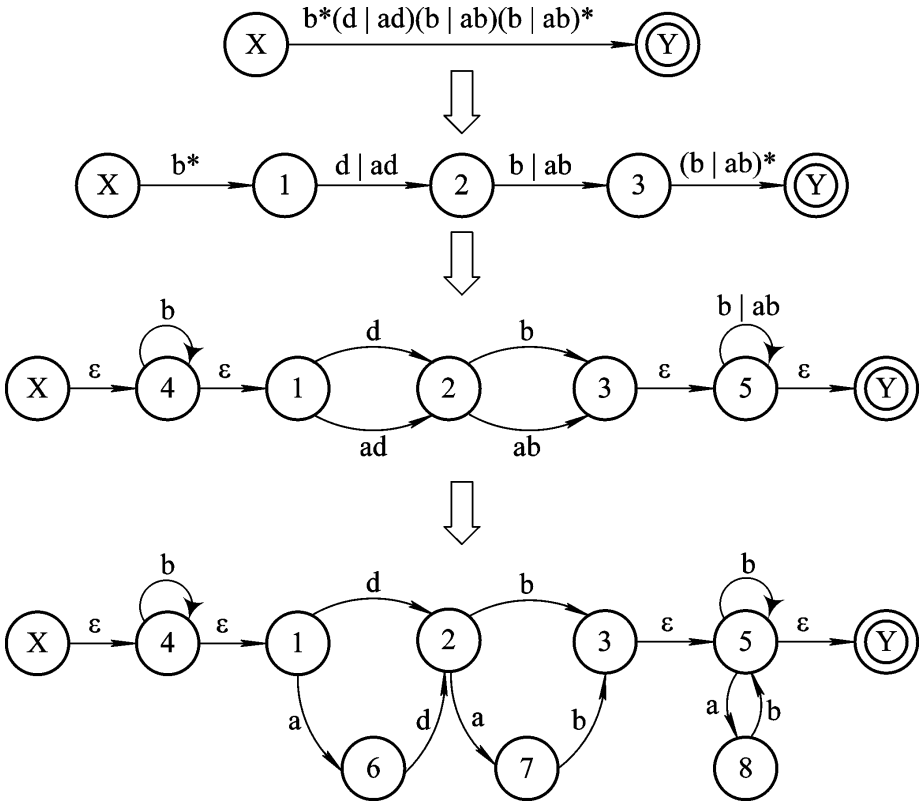



图2-12 例2.6的NFA M

## 2.4.2 NFA的确定化

NFA的确定化是指对给定的NFA M都能相应地构造出一个与之等价的DFA M，使它们能够识别相同的语言。我们采用子集法来对NFA M确定化。

首先定义NFA M的一个状态子集I的ε_闭包，即ε_CLOSURE(I)，则：

(1) 若si∈I，则si∈ε_CLOSURE(I)。

(2) 若$s_i \in I$，则对从si出发经过ε通路所能到达的任何状态sj，都有$s_j \in \varepsilon_CLOSURE(I)$。

其次，对FA M的一个状态子集I，若a是Σ中的一个字符，定义

$$I_a=\varepsilon\_CLOSURE(J)$$

其中，J是所有那些可以从I中的某一状态出发经过有向边a而能到达的状态集。

**例2.7** 已知一状态转换图如图2-13所示，且假定I = ε_{1} = {1, 2}，试求从状态I出发经过一条有向边a而能到达的状态集J和ε_CLOSURE(J)。

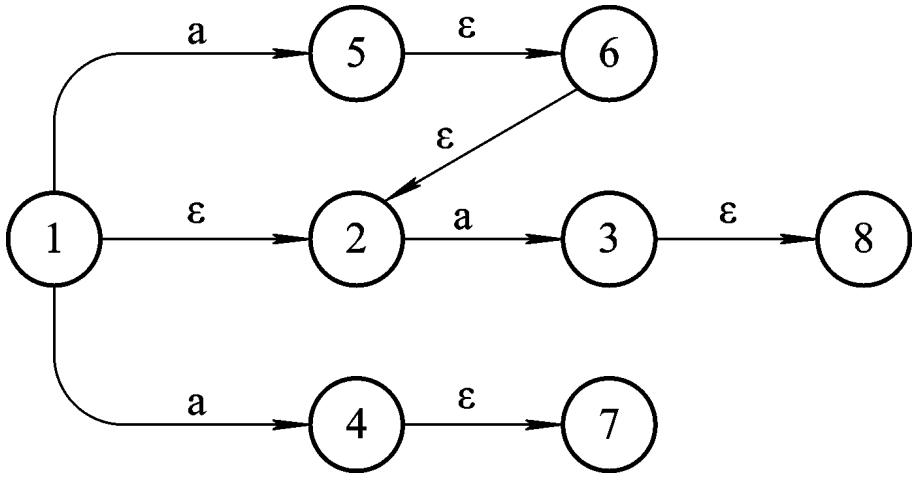



图2-13 例2.7的状态转换图

**[解答]** 从状态I中的状态1或状态2出发经过一条a弧而能到达的状态集J为

{5，3，4}

若$s_i \in J$，则由$s_i$出发经过任意条ε有向边而能到达的任何状态$s_j \in$ε_CLOSURE(J)，因此ε_CLOSURE(J) 为

{5，6，2，3，8，4，7}

用子集法对NFA M确定化的方法如下：

(1) 构造一张转换表，其第一列为状态子集I，对不同的a($a \in \Sigma$)在表中单设一列Ia。

(2) 表的第一行第一列其状态子集I为ε_CLOSURE($s_0$)；其中，$s_0$为初始状态。

(3) 根据第一列中的I为每一个a求其$I_a$并记入对应的$I_a$列中，如果此$I_a$不同于第一列已存在的所有状态子集I，则将其顺序列入空行中的第一列。

(4) 重复步骤(3)直至对每个I及a均已求得$I_a$，并且无新的状态子集Ia加入第一列时为止；此过程可在有限步后终止。

(5) 重新命名第一列的每一状态子集，则转换表便成为新的状态转换矩阵，其状态转换函数f是S × Σ到S的单值映射，即为与NFA M等价的DFA M'。

**例2.8** 正规表达式(a｜b)*(aa｜bb) (a｜b)*的NFA M如图2-14所示，试将其确定化为DFA M'。

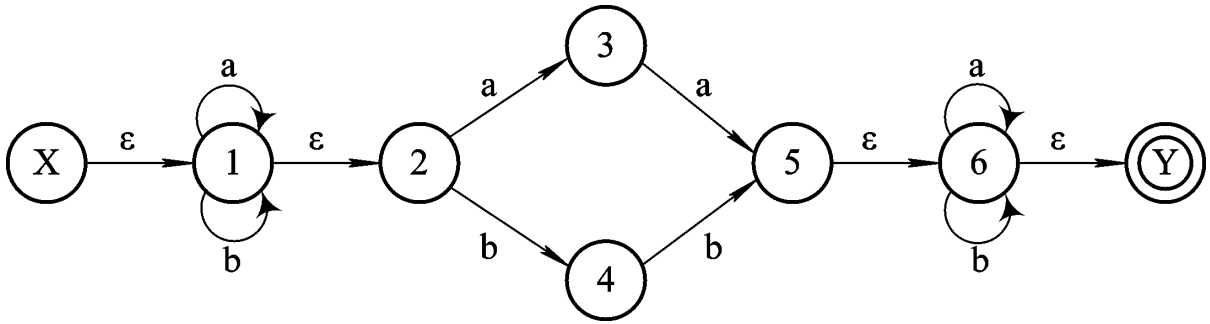



图2-14 例2.8的NFA M

[解答] 用子集法将图2-14所示的NFA M确定化为表2.4。

表 2.4 例 2.8 的转换表

| I | $I_a$ | $I_b$ |
|---|---|---|
| {X,1,2} | {1,2,3} | {1,2,4} |
| {1,2,3} | {1,2,3,5,6,Y} | {1,2,4} |
| {1,2,4} | {1,2,3} | {1,2,4,5,6,Y} |
| {1,2,3,5,6,Y} | {1,2,3,5,6,Y} | {1,2,4,6,Y} |
| {1,2,4,5,6,Y} | {1,2,3,6,Y} | {1,2,4,5,6,Y} |
| {1,2,4,6,Y} | {1,2,3,6,Y} | {1,2,4,5,6,Y} |
| {1,2,3,6,Y} | {1,2,3,5,6,Y} | {1,2,4,6,Y} |

对表2.4中的所有子集重新命名，得到表2.5的状态转换矩阵及对应的状态转换图(见图2-15)。注意，状态3、4、5、6因其原来对应的子集中含有终态Y而均为终态。

表2.5 例2.8的状态转换矩阵

| S | a | b |
|---|---|---|
| 0 | 1 | 2 |
| 1 | 3 | 2 |
| 2 | 1 | 4 |
| 3 | 3 | 5 |
| 4 | 6 | 4 |
| 5 | 6 | 4 |
| 6 | 3 | 5 |

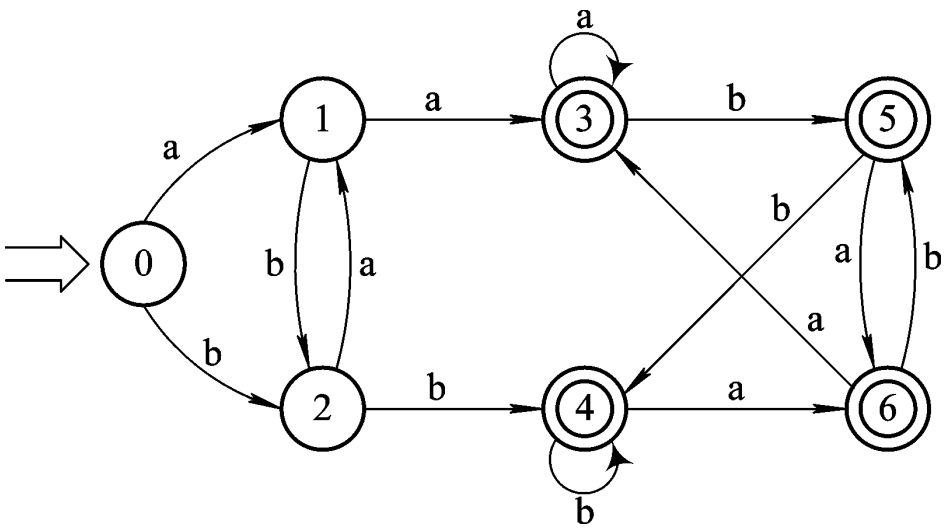



图2-15 例2.8未化简的DFA M'

### 2.4.3 确定有限自动机(DFA)的化简

对NFA确定化后所得到的DFA可能含有多余的状态，因此还应对其进行化简。所谓DFA M的化简，是指寻找一个状态数比M少的DFA M'，使得L(M) =L(M')。化简了的DFA M' 满足下述两个条件：

(1) 没有多余状态。

(2) 在其状态集中，没有两个相互等价的状态存在。

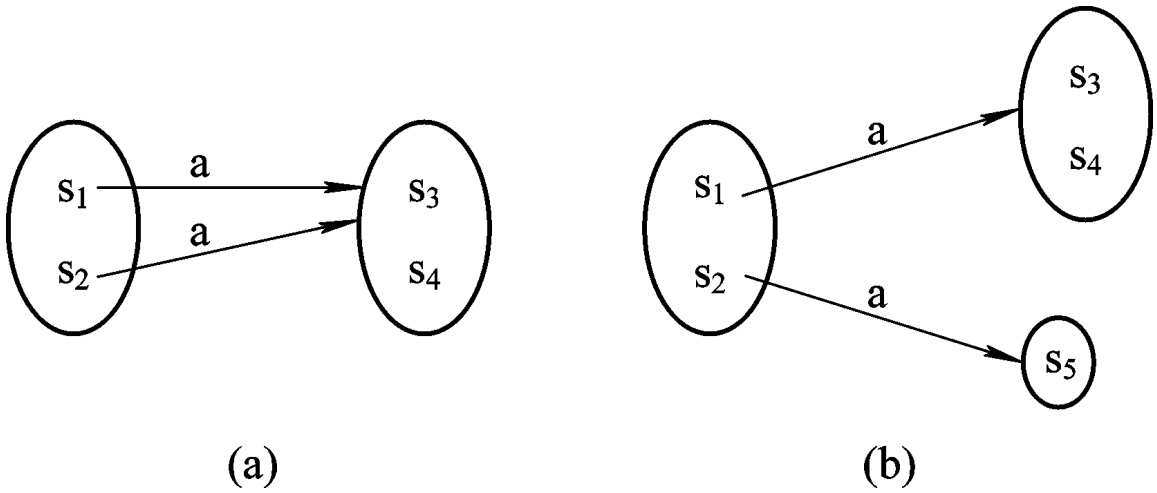
所谓两个状态相互等价是指：对一给定的DFA M，若存在状态s1、s2∈S且s1≠s2，如果从s1出发能识别字符串α而停于终态，从s2出发也同样能够识别这个α而停于终态；反之，若从s2出发能识别字符串β而停于终态，则从s1出发也能识别这个β而停于终态，则称s1和s2是等价的，否则就是可区分的。

一个DFA M的状态最小化过程是将M的状态集分割成一些不相交的子集，使得任何不同的两个子集其状态都是可区分的，而同一子集中的任何两个状态都是等价的。最后，从每个子集中选出一种状态，同时消去其它等价状态，就得到最简的DFA M'且L(M) =L(M')。

DFA M的化简方法如下：

(1) 首先将DFA M的状态集S中的终态与非终态分开，形成两个子集，即得到基本划分。

(2) 对当前已划分出的$I^{(1)}$、$I^{(2)}$、…、$I^{(m)}$子集(属于不同子集的状态是可区分的)，看每一个I是否能进一步划分；也即对某个$I^{(i)}=\{s_1, s_2, \ldots, sk\}$，若存在一个输入字符$a(a\in\Sigma)$使得$Ia^{(i)}$不全包含在当前划分的某一子集$I^{(j)}$中(即跨越到两个子集)，就将$I^{(i)}$一分为二(见图2-16)。




(a) 无需划分；(b) 需要划分

图2-16 是否划分示意

(3) 重复步骤(2)，直到子集个数不再增加为止(即每个子集已是不可再分的了)。所谓不能划分，是指该子集或者仅有一个状态，或者虽有多个状态但这些状态均不可区分(即等价)。

那么，如何进行划分呢？假定当前子集$I^{(i)}=\{s_1，s_2，\ldots\}$，且状态$s_1$和$s_2$经过有向边a分别到达状态$t_1$和$t_2$，而$t_1$和$t_2$又分属于当前已划分出的两个不同子集$I^{(j)}$和$I^{(k)}$，则此时应将$I^{(i)}$分为两部分，使得一部分含有s1：

$$I^{(i1)}=\{s \mid s\in I^{(i)}\text{且}s\text{经有向边a到达}t_1\}$$

而另一部分含有$s_2$：

$$I^{(i2)}=I^{(i)}-I^{(i1)}$$

由于$t_1$和$t_2$是可区分的，即存在一个字符串α，$t_1$将读出α而停于终态，而$t_2$或读不出α或可以读出α但不能到达终态。因此，字符串α将把状态$s_1$和$s_2$区分开来，也就是说，$I^{(i1)}$中的状态与$I^{(i2)}$中的状态是可区分的。至此，我们已将$I^{(i)}$ 划分为两个子集$I^{(i1)}$和$I^{(i2)}$，形成了新的划分。

当子集个数不再增加时，就得到一个最终划分。对最终划分的每一个子集，我们选取子集中的一个状态作为代表。例如，假定$I^{(i)}=\{s_1, s_2, s_3\}$是这样一个子集，且我们挑选了$s_1$代表这个子集。这时，凡在原来DFA M中有指向$s_2$和$s_3$的有向边均改为在新的DFA M'中指向$s_1$。改向之后，就可将$s_2$和$s_3$从原来的状态集S中删除了。若$I^{(i)}$中含有原来的初态，则$s_1$就是DFA M' 中的新初态；若$I^{(i)}$中含有原来的终态，则$s_1$就是DFA M' 中的新终态。此时，DFA M' 已是最简的了(含有最少的状态)。

**例2.9** 化简由例2.8得到的DFA M。

[解答] (1) 首先将状态集 S={0, 1, 2, 3, 4, 5, 6}划分为终态集{3, 4, 5, 6}和非终态集{0, 1, 2}。

(2) 考察$\{0, 1, 2\}_a$：因 $0_a=2_a=\{1\}$，而 $1_a=\{3\}$，分属于非终态集和终态集，故将{0, 1, 2}划分为{0, 2}和{1}。

(3) 考察$\{0, 2\}_b$：$0_b=\{2\}$，$2_b=\{4\}$，它们分属于两个不同的状态集，故{0, 2}划分为{0}和{2}。

(4) 考察$\{3, 4, 5, 6\}_a$：$3_a=6_a=\{3\}\subset\{3, 4, 5, 6\}$；$4_a=5_a=\{6\}\subset\{3, 4, 5, 6\}$，即都属于终态集，故不进行划分。

(5) 考察$\{3, 4, 5, 6\}_b$：$3_b=6_b=\{5\}\subset\{3, 4, 5, 6\}$；$4_b=5_b=\{4\}\subset\{3, 4, 5, 6\}$，即都属于终态集，故不进行划分。

(6) 按顺序重新命名状态子集{0}、{1}、{2}、{3, 4, 5, 6}为 0、1、2、3，则得到化简后的状态转换矩阵(见表 2.6)和 DFA M"(见图 2-17)。

表 2.6 例 2.9 的状态转换矩阵

| S | a | b |
|---|---|---|
| 0 | 1 | 2 |
| 1 | 3 | 2 |
| 2 | 1 | 3 |
| 3 | 3 | 3 |

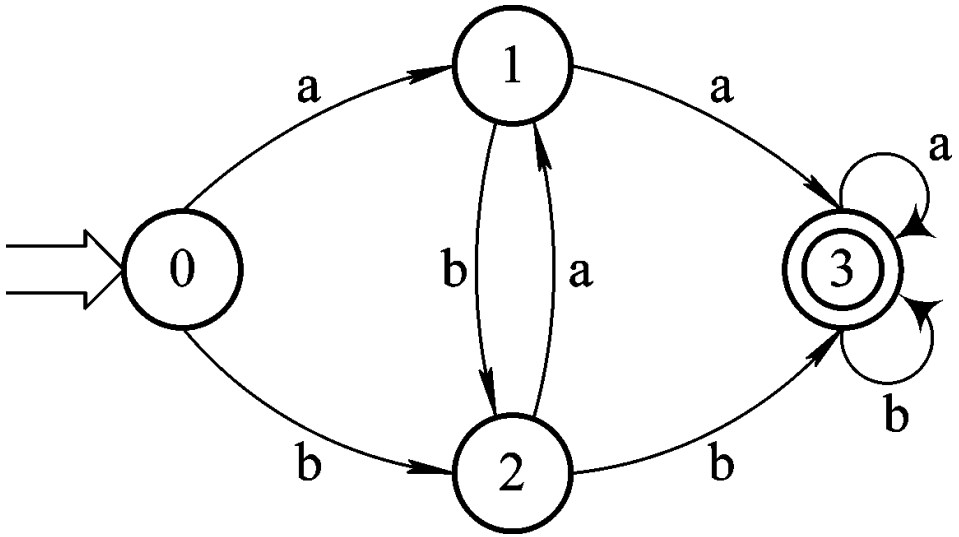



图2-17 例2.9化简后的DFA M"

### 2.4.4 正规表达式到有限自动机构造示例

**例2.10** 试用DFA的等价性证明正规表达式 ( a | b )*与 ( a*b* )*等价。

**[解答]** (1) 正规表达式 (a | b)*对应的NFA M如图2-18所示。

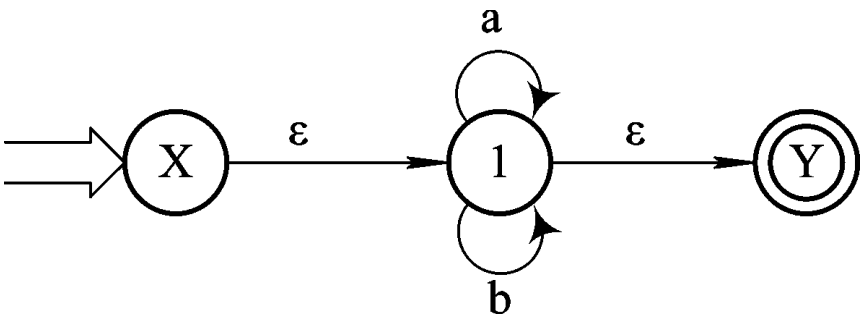



图2-18 (a | b)*的NFA M

用子集法将图2-18所示的NFA M确定化得到如表2.7所列的转换表，重新命名后得到如表2.8所列的状态转换矩阵。

表 2.7 (a | b)*的转换表

| I | $I_a$ | $I_b$ |
|---|---|---|
| {X,1,Y} | {1,Y} | {1,Y} |
| {1,Y} | {1,Y} | {1,Y} |

表 2.8 (a | b)*的状态转换矩阵

| S | a | b |
|---|---|---|
| 0 | 1 | 1 |
| 1 | 1 | 1 |

由于状态0和状态1均为终态，无论输入什么字符，其下一状态仍是终态，故最简DFA M如图2-19所示。

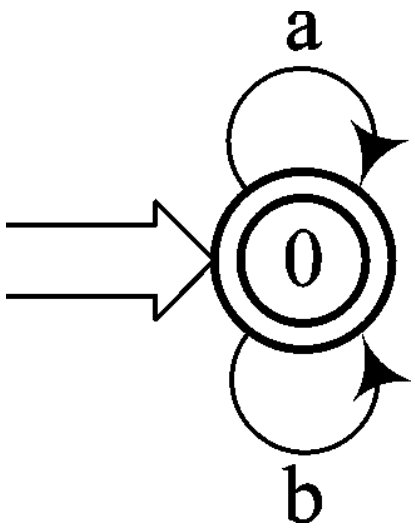



图2-19 (a｜b)*的最简DFA M

用子集法将图2-20所示的NFA M确定化得到如表2.9所列的转换表，重新命名得到如表2.10所列的状态转换矩阵。

表 2.9 (a|b)*的转换表

| I | $I_a$ | $I_b$ |
|---|---|---|
| {X,1,2,3,Y} | {2,3,1,Y} | {2,3,1,Y} |
| {2,3,1,Y} | {2,3,1,Y} | {2,3,1,Y} |

表 2.10 (a|b)*的状态转换矩阵

| S | a | b |
|---|---|---|
| 0 | 1 | 1 |
| 1 | 1 | 1 |

由于表2.8和表2.10的状态转换矩阵相同，故 ( a | b )*和 ( a*b* )*等价。

**注意**，对正规表达式a*b*构造DFA M，则首先画出NFA M如图2-21所示，用子集法将图2-21所示的NFA M确定化得到如表2.11所列的转换表，重新命名得到如表2.12所列的状态转换矩阵。

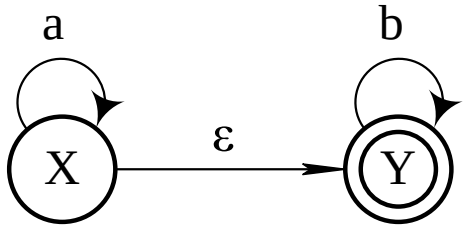



图2-21 a*b*的NFA M

表 2.11 a*b*的转换表

| I | $I_a$ | $I_b$ |
|---|---|---|
| {X,Y} | {X,Y} | {Y} |
| {Y} | — | {Y} |

表 2.12 a*b*的状态转换矩阵

| S | a | b |
|---|---|---|
| 0 | 0 | 1 |
| 1 | — | 1 |

(注：表中的空集Φ一律用“—”表示)

虽然状态0和状态1都是终态，但两者面对字符a转换的下一状态是不一样的：$0_a=0$，$1_a=\Phi$(即“—”)；也即状态0和状态1不等价，故不可将图2-21的DFA M合并成图2-19的DFA M。因此，正规表达式a*b*根据表2.12最终得到的DFA M如图2-22所示。

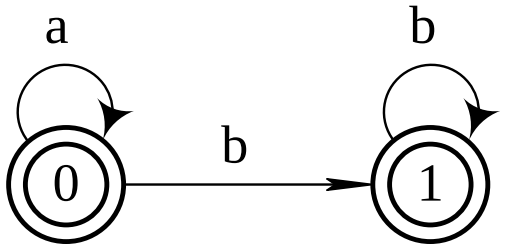



图2-22 a*b* 的DFA M

**例2.11** C语言可接受的合法的文件名为device: name. extension，其中第一部分(device:)和第三部分(.extension)可缺省。若device、name和extension都是字母串，长度不限，但至少为1，试画出识别这种文件名的DFA M。

**[解答]** 以字母“c”代表字母，则所求正规式为

( cc*:｜ε) cc* ( .cc*｜ε)

应用例2.10由图2-18化简为图2-19的结论，去掉多余的ε弧，得到的NFA M如图2-23所示。

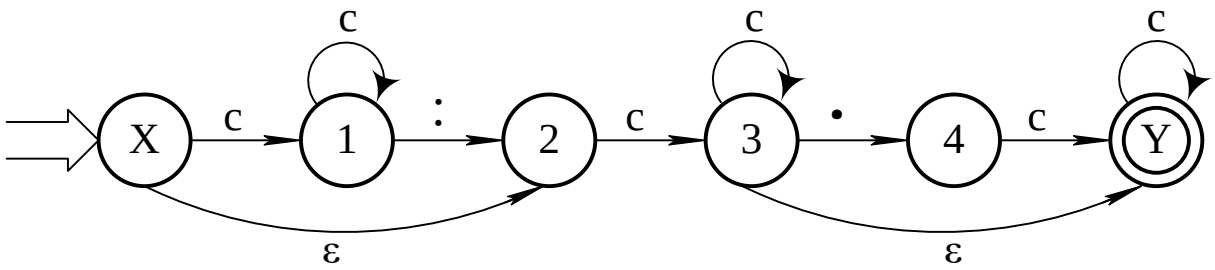



图2-23 例2.11的NFA M

用子集法将NFA M确定化，得到如表2.13所列的转换表；重新命名后得到如表2.14所列的状态转换矩阵和如图2-24所示的DFA M'。

表2.13 例2.11的转换表

| I | I$_c$ | I: | I. |
|---|---|---|---|
| {X, 2} | {1, 3, Y} | — | — |
| {1, 3, Y} | {1, 3, Y} | {2} | {4} |
| {2} | {3, Y} | — | — |
| {4} | {Y} | — | — |
| {3, Y} | {3, Y} | — | {4} |
| {Y} | {Y} | — | — |

重新命名

表2.14 例2.11的状态转换矩阵

| S | c | : | . |
|---|---|---|---|
| 0 | 1 | — | — |
| 1 | 1 | 2 | 3 |
| 2 | 4 | — | — |
| 3 | 5 | — | — |
| 4 | 4 | — | 3 |
| 5 | 5 | — | — |

由表2.14可以看出，终态集中的状态1对应三种输入字符的下一状态均存在，状态4对应三种输入字符的下一状态存在两个，而状态5对应三种输入字符的下一状态仅有一个存在，故状态1、4、5都是可区分的。相应地，非终态0、2、3对应输入字母c的下一状态也都是可区分的了。因此，图2-24的DFA M' 已为最简。

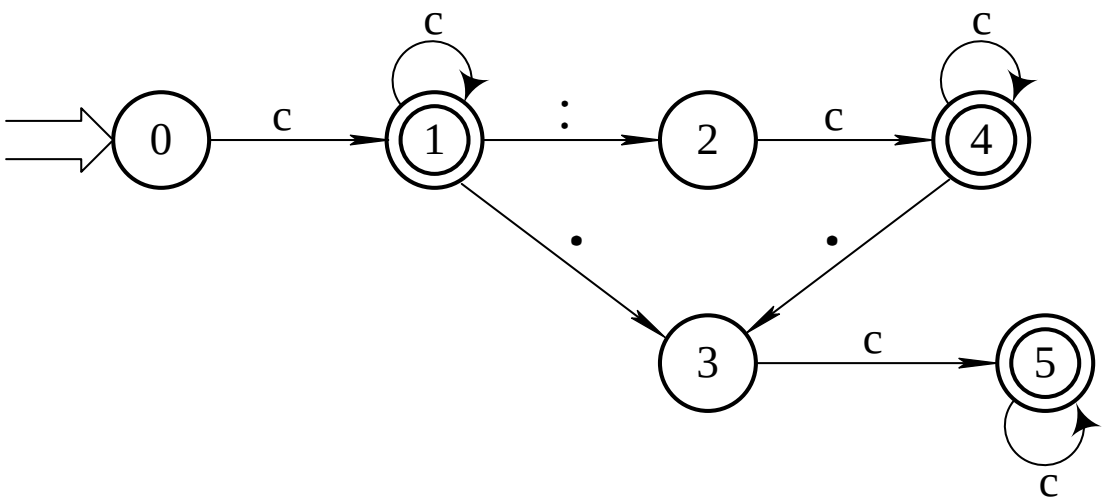



图2-24 例2.11的DFA M'

**例2.12** 某高级程序语言无符号数的正规表达式为

digit+ [ .digit+ ] [ e [ + | − ] digit+ ]

其中，digit表示数字，“[ ]”表示“[ ]”中的内容可有可无，试给出其DFA M。

**[解答]** 我们用d代表digit，“[ ]”中的内容为无时用ε表示，则本题的正规式又可表示为

dd*( . dd* | ε) ( e ( + | - | ε) dd* | ε)

由此，用状态转换图表示接受无符号数的NFA M如图2-25所示。

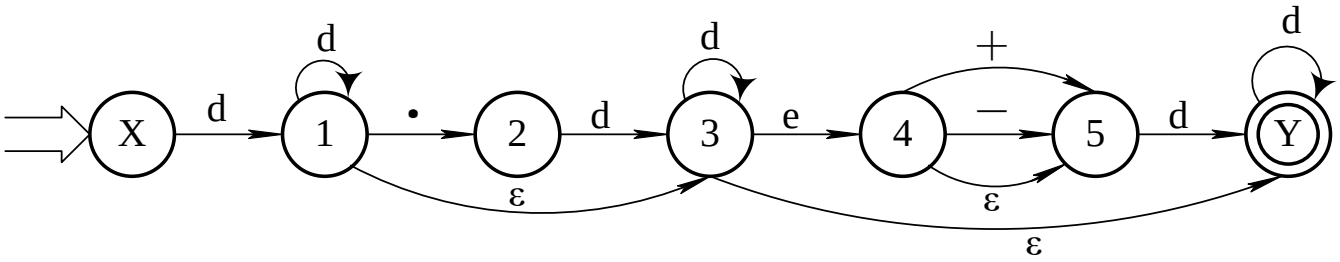



图2-25 例2.12的NFA M

用子集法将NFA M确定化，得到如表2.15所列的转换表(I+和I-用I±表示在同一列中)；重新命名后得到如表2.16所列的状态转换矩阵和如图2-26所示的DFA M'。

表 2.15 例 2.12 的转换表

| I | $I_{\pm}$ | $I_d$ | I. | $I_e$ |
|---|---|---|---|---|
| {X} | — | {1, 3, Y} | — | — |
| {1, 3, Y} | — | {1, 3, Y} | {2} | {4, 5} |
| {2} | — | {3, Y} | — | — |
| {4, 5} | {5} | {Y} | — | — |
| {3, Y} | — | {3, Y} | — | {4, 5} |
| {5} | — | {Y} | — | — |
| {Y} | — | {Y} | — | — |

重新命名

表 2.16 例 2.12 的状态转换矩阵

| S | ± | d | . | e |
|---|---|---|---|---|
| 0 | — | 1 | — | — |
| 1 | — | 1 | 2 | 3 |
| 2 | — | 4 | — | — |
| 3 | 5 | 6 | — | — |
| 4 | — | 4 | — | 3 |
| 5 | — | 6 | — | — |
| 6 | — | 6 | — | — |

根据表2.16的化简过程如下：首先分为非终态和终态两个集合{0，2，3，5}和{1，4，6}。由非终态集{0，2，3，5}可知，状态3面对输入符号“+”、“–”的下一状态与状态0、2、5不同，故将状态3划分出来。终态集{1，4，6}中的状态1、4、6面对不同输入符号的下一非空状态分别有3、2、1种，故它们都是可区分的，并由此导致状态0、2、5面对输入符号“d”的下一状态也不相同，即状态0、2、5均可区分。故图2-26的DFA M'已为最简。

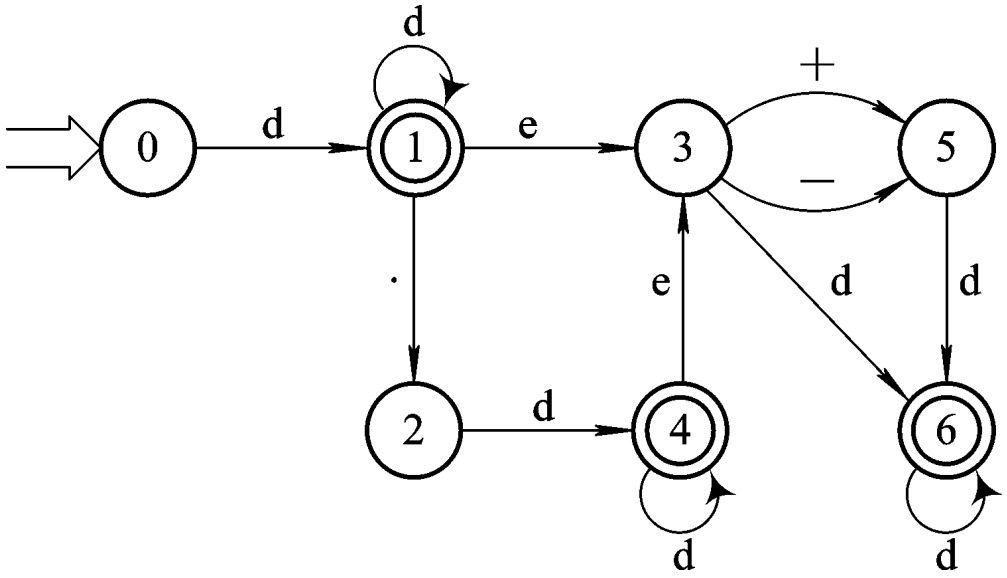



图2-26 例2.12的DFA M'

**例2.13** 构造一个DFA M，它接收Σ={a, b}上所有满足下述条件的字符串：该字符串中的每个a都有至少一个b直接跟在其右边。

**[解答]** 已知Σ= {a, b}，根据题意得到正规表达式为b* ( abb* )*。根据此正规表达式画出相应的NFA M如图2-27所示。

用子集法将NFA M确定化，得到如表2.17所列的转换表；重新命名后得到如表2.18所列的状态转换矩阵和如图2-28所示的DFA M'。

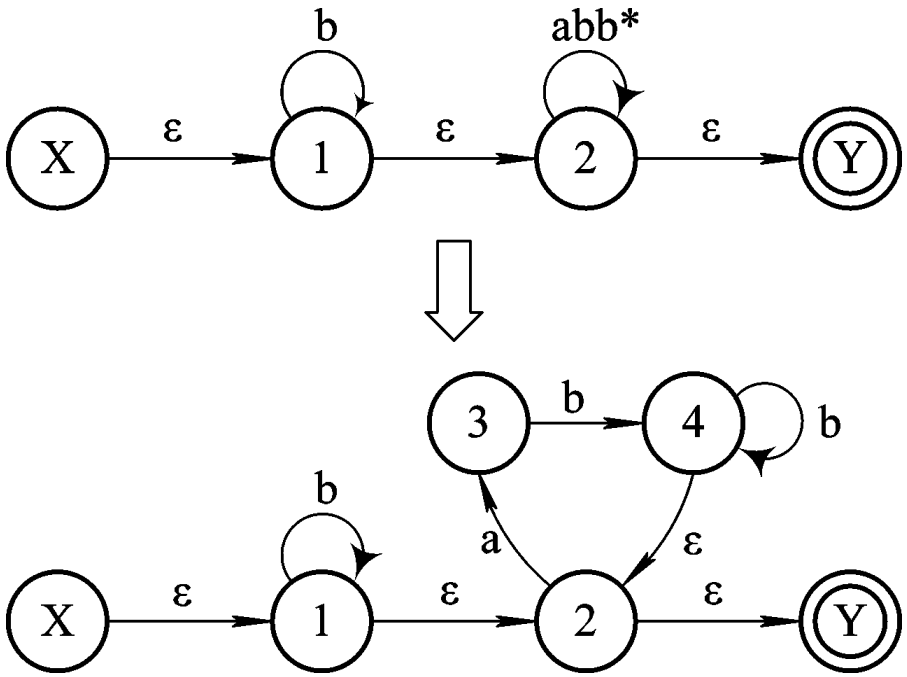



图2-27 例2.13的NFA M(一)

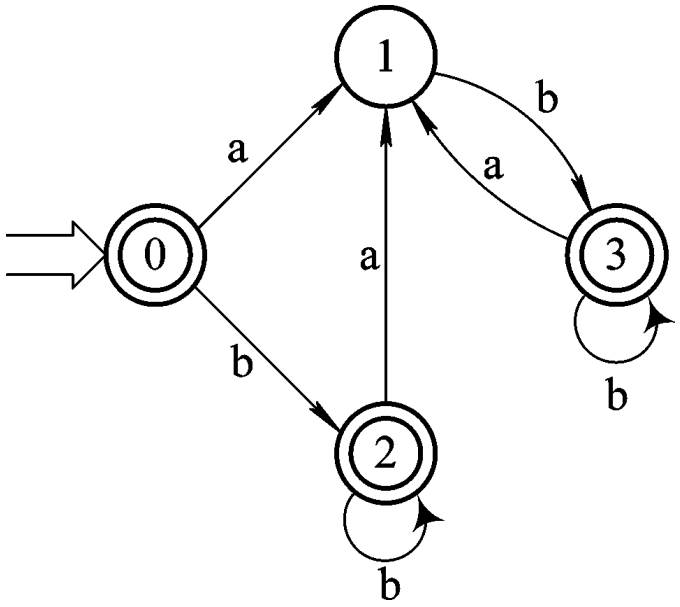



图2-28 例2.13的DFA M'

表 2.17 例 2.13 的转换表(一)

| I | $I_a$ | $I_b$ |
|---|---|---|
| {X, 1, 2, Y} | {3} | {1, 2, Y} |
| {3} | — | {4, 2, Y} |
| {1, 2, Y} | {3} | {1, 2, Y} |
| {4, 2, Y} | {3} | {4, 2, Y} |

重新命名

表 2.18 例 2.13 的状态转换矩阵

| S | a | b |
|---|---|---|
| 0 | 1 | 2 |
| 1 | — | 3 |
| 2 | 1 | 2 |
| 3 | 1 | 3 |

用DFA M的化简方法先得到一个初始划分，即终态集为{0, 2, 3}，非终态集为{1}；由化简方法可知这已是最终划分(状态0、状态2、状态3均为等价状态)，重新命名终态集为0、非终态集为1后得到最终化简的DFA M" 如图2-29所示。

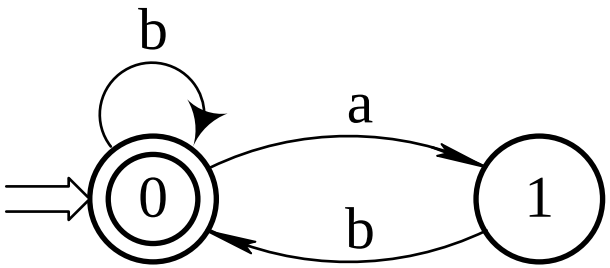



图2-29 例2.13最终化简后的 DFA M"

实际上，根据正规表达式b*(abb*)可以看出，当b*和(abb*)*(在此指括号外的这个“*”)的闭包*都取0时，则由初态X到终态Y有一条ε通路，即得到图2-30。由图2-30的状态Y上的正规式abb*可以看出，当b*的闭包*取0时，则由状态Y出发，经过a和b又应回到状态Y，而b*则可描述为由状态Y出发又回到状态Y的一条标记为b的有向边，这样就得到图2-31。

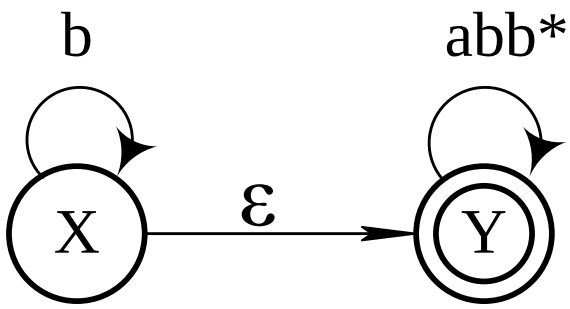



图2-30 例2.13的NFA M(二)

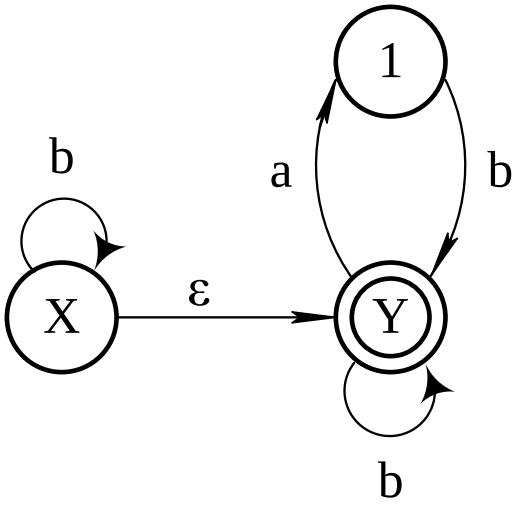



图2-31 例2.13的NFA M(三)

用子集法对图2-31进行确定化，得到表2.19，重新命名并进行化简，最终可得到图2-29。

表 2.19　例 2.13 的转换表(二)

| I | $I_a$ | $I_b$ |
|---|---|---|
| {X,Y} | {1} | {X,Y} |
| {1} | — | {Y} |
| {Y} | {1} | {Y} |

此题根据题意也可得到正规表达式为 ( ab | b )*，最终同样可化简为图2-29。

**例2.14** 构造一个DFA M，它接收Σ={a, b}上所有含奇数个a的字符串。

**[解答]** 根据题意，我们首先可以构造出字符串中含偶数个a的正规表达式：( b | ab*a )*，然后在其之前添加一个a即为奇数个a。因此，得到含奇数个a的字符串的正规表达式为：b*a ( b | ab*a ) *。根据此正规表达式画出相应的NFA M如图2-32所示。图2-32中每条边上为单个字符且是单值映射，并且无ε边出现，故已是DFA M；体现在表2-20中，对每个输入字符的输出，即子集映射此时已全部是单个字符，即为单值映射。

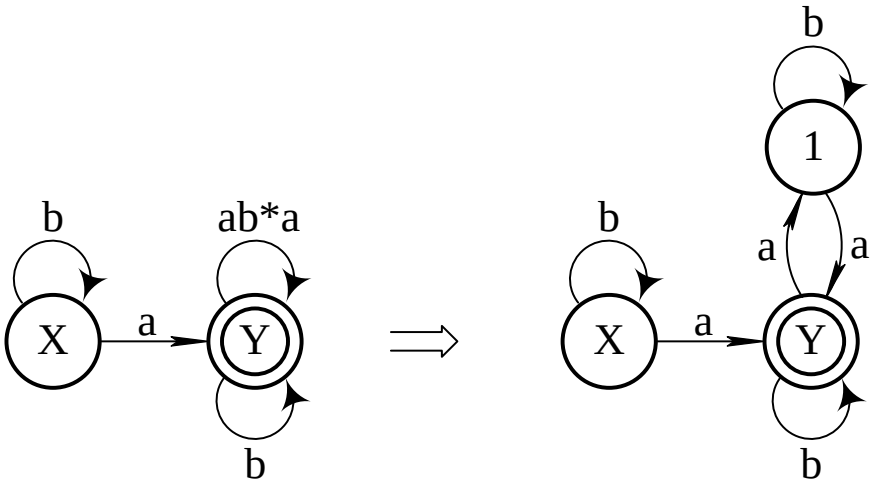



图2-32 例2.14的NFA M

用子集法将图2-32的NFA M(由于是单值映射，故已为DFA M)确定化，得到如表2.20所列的转换表；重新命名后得到如表2.21所列的状态转换矩阵。

表 2.20 例 2.14 的转换表

| I | $I_a$ | $I_b$ |
|---|---|---|
| {X} | {Y} | {X} |
| {Y} | {1} | {Y} |
| {1} | {Y} | {1} |

重新命名

表 2.21 例 2.14 的状态转换矩阵

| S | a | b |
|---|---|---|
| 0 | 1 | 0 |
| 1 | 2 | 1 |
| 2 | 1 | 2 |

根据表2.21的状态转换矩阵进行最小化，得到两个初始划分：{0, 2}和{1}；由于状态0和状态2为等价状态，删去状态2即得到最简DFA M，如图2-33所示。

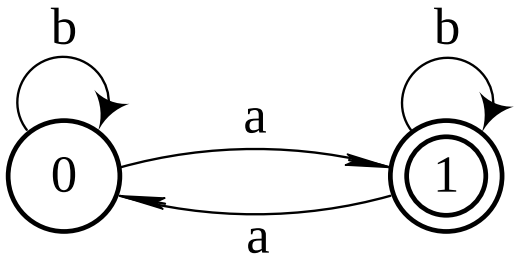



图2-33 例2.14最终化简后的 DFA M

最后需要说明的是，采用如图2-34所示的三条转换规则可以实现从FA M到正规表达式的转换。

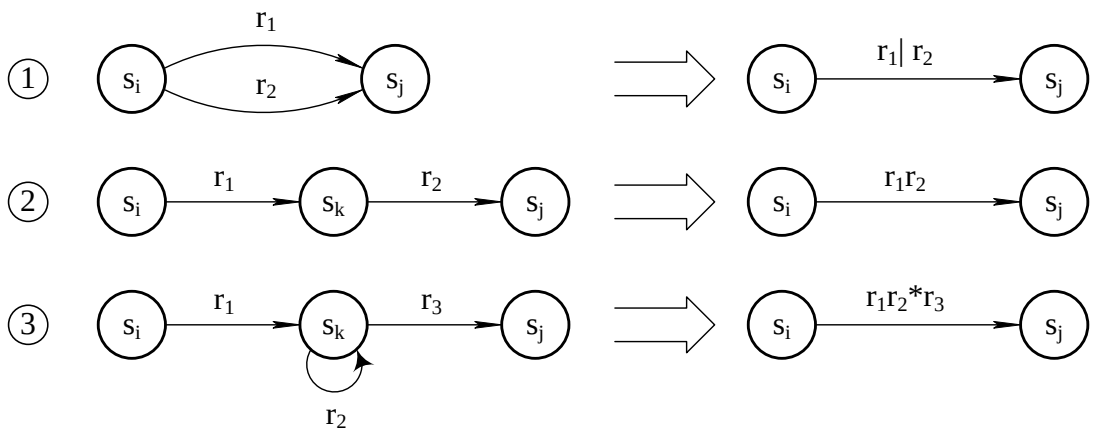



图2-34 转换规则

## 2.5 词法分析器的自动生成

Lex是由美国Bell实验室的M.Lesk和Schmidt于1975年用C语言研制的一个词法分析程序的自动生成工具。对任何高级程序语言，用户必须用正规表达式描述该语言的各个词法类(这一描述称为Lex的源程序)，Lex就可以自动生成该语言的词法分析程序。Lex及其编译系统的作用如图2-35所示。

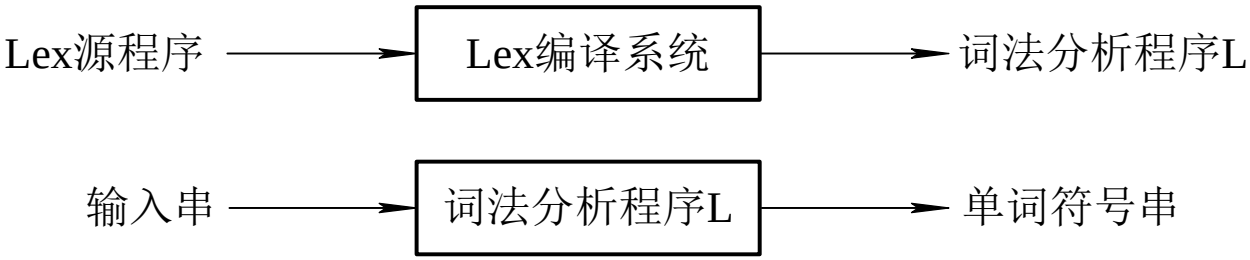



图2-35 Lex及其编译系统的作用

一个Lex源程序由用“%%”分隔的三部分组成：第一部分为正规式的辅助定义式，第二部分为识别规则，最后一部分为用户子程序。其书写格式为

辅助定义式

%%

识别规则

%%

用户子程序

其中，辅助定义式和用户子程序是任选的，而识别规则是必需的。如果用户子程序缺省，则第二个分隔符号“%%”可以省去；但如果无辅助定义式部分，第一个分隔符号“%%”不能省去，因为第一个分隔符号用于指示识别规则部分的开始。

下面给出一个简单语言的单词符号的Lex源程序例子，其输出单词的类别编码用整数编码表示：

```
Auxiliary Definitions                          /*辅助定义*/
letter→A | B | C | … | Z | a | b | c | … | z
digit→0 | 1 | 2 | 3 | … | 9
%%
Recognition Rules                              /*识别规则*/
1 while          {return ( 1, null )}
2 do             {return ( 2, null )}
3 if             {return( 3, null )}
4 else           {return ( 4, null )}
5 switch         {return ( 5, null )}
6 {              {return ( 6, null )}
7 }              {return ( 7, null )}
8 (              {return ( 8, null )}
9 )              {return ( 9, null )}
10 +             {return ( 10, null )}
11 −             {return ( 11, null )}
12 *             {return ( 12, null )}
13 /             {return (13, null )}
14 =             {return ( 14, null )}
15 ；            {return ( 15, null )}
```

```
16 letter (letter | digit)*        {if(keyword(id)==0)
                                      {return ( 16, null );
                                       return ( id ) };
                                    else return ( keyword(id) ) }
17 digit (digit)*                  {val=int(id);
                                    return ( 17, null );
                                    return ( val )}
18 ( letter | digit | { | } | ( | ) | + | − | * | / | = | ; )*
                                   {return ( 18, null );
                                    inslit(id);
                                    return ( pointer, lenth )}
```

该Lex源程序中用户子程序为空；其中识别规则{A18}语句中调用过程“inslit(id) ”是指将字符串常量id存放到字符表中，“pointer”中存放该串的起始位置，“lenth”存放该串的长度。

Lex有两种使用方式：一种是将Lex作为一个单独的工具，用以生成所需的识别程序；另一种是将Lex和语法分析器自动生成工具(如YACC)结合起来使用，以生成一个编译程序的扫描器和语法分析器。

# 习 题 2

**2.1 完成下列选择题：**

(1) 词法分析所依据的是　　。

A．语义规则　　B．构词规则

C．语法规则　　D．等价变换规则

(2) 词法分析器的输入是　　。

A．单词符号串　　B．源程序

C．语法单位　　D．目标程序

(3) 词法分析器的输出是　　。

A. 单词的种别编码

B. 单词的种别编码和自身的值

C. 单词在符号表中的位置

D. 单词自身值

(4) 状态转换图(见图2-36)接受的字集为 _____。

A. 以0开头的二进制数组成的集合

B. 以0结尾的二进制数组成的集合

C. 含奇数个0的二进制数组成的集合

D. 含偶数个0的二进制数组成的集合

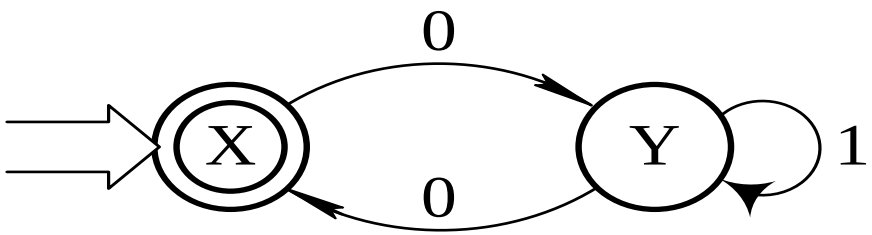



图2-36 习题2.1的DFA M

(5) 对于任一给定的NFA M，　　一个DFA M'，使L(M) =L(M')。

A．一定不存在　　B．一定存在

C．可能存在　　D．可能不存在

(6) DFA适用于　　。

A．定理证明　　B．语法分析

C．词法分析　　D．语义加工

(7) 下面用正规表达式描述词法的论述中，不正确的是　　。

A．词法规则简单，采用正规表达式已足以描述

B．正规表达式的表示比上下文无关文法更加简洁、直观和易于理解

C．正规表达式描述能力强于上下文无关文法

D．有限自动机的构造比下推自动机简单且分析效率高

(8) 与 (a｜b)*(a｜b) 等价的正规式是　　。

A．(a｜b)(a｜b )*　　B．a*｜b*

C．(ab)*(a｜b)*　　D．(a｜b)*

(9) 在状态转换图的实现中，____一般对应一个循环语句。

A．不含回路的分叉结点

B．含回路的状态结点

C．终态结点

D．A～C都不是

(10) 已知DFA Md= ($\{s_0, s_1, s_2\}$, {a, b}, f, $s_0$, $\{s_2\}$)，且有：

$f(s_0, a) = s_1$　　$f(s_1, a) = s_2$

$f(s_2, a) = s_2$　　$f(s_2, b) = s_2$

则该DFA M所能接受的语言可以用正规表达式表示为____。

A．( a | b )*　　B．aa ( a | b )*

C．( a | b )*aa　　D．a ( a | b )*a

2.2 什么是扫描器？扫描器的功能是什么？

2.3 设M= ({x, y}, {a, b}, f, x, {y}) 为一非确定的有限自动机，其中f定义如下：

$$f(x, a) = \{x, y\} \qquad f(x, b) = \{y\}$$

$$f(y, a) = \Phi \qquad f(y, b) = \{x, y\}$$

试构造相应的确定有限自动机M'。

2.4 正规式 (ab)*a与正规式a (ba)*是否等价？请说明理由。

2.5 设有$L(G)=\{a^{2n+1}b^{2m}a^{2p+1} \mid n\geq 0, p\geq 0, m\geq 1\}$。

(1) 给出描述该语言的正规表达式；

(2) 构造识别该语言的确定有限自动机(可直接用状态图形式给出)。

2.6 有语言$L=\{w \mid w\in(0, 1)+$，并且w中至少有两个1，又在任何两个1之间有偶数个0}，试构造接受该语言的确定有限状态自动机(DFA M)。

2.7 已知正规式 ((a｜b)*｜aa)*b和正规式 (a｜b)*b。

(1) 试用有限自动机的等价性证明这两个正规式是等价的；

(2) 给出相应的正规文法。

2.8 构造一个DFA M，它接收Σ = {a, b}上所有不含子串abb的字符串。

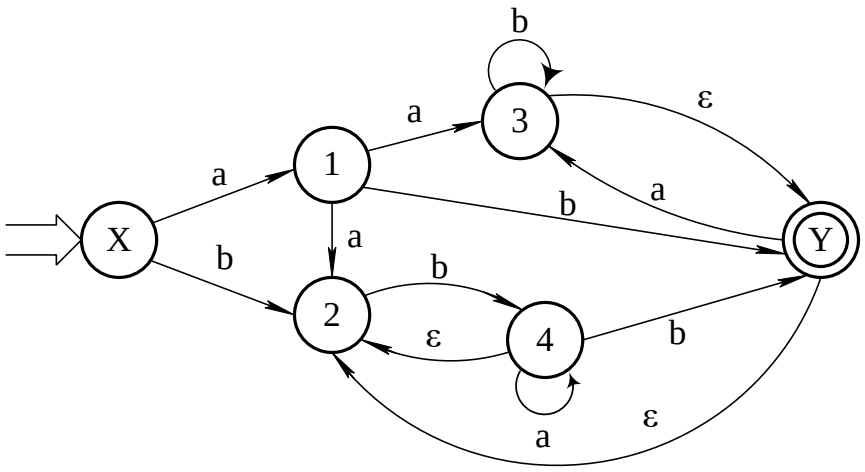
2.9 构造一个DFA M，它接收Σ = {a, b}上所有含偶数个a的字符串。

2.10 下列程序段以B表示循环体，A表示初始化，I表示增量，T表示测试：

```
I=1;
while( I<=n )
{
    sum=sum+a[I];
    I=I+1;
 }
```

请用正规表达式表示这个程序段可能的执行序列。

2.11 将图2-37所示的非确定有限自动机(NFA M)变换成等价的确定有限自动机
(DFA M′)。其中，X为初态，Y为终态。



2.12 有一台自动售货机，接受1分和2分硬币，出售3分钱一块的硬糖。顾客每次向机器中投放大于等于3分的硬币，便可得到一块糖(注意：只给一块并且不找钱)。

(1) 写出售货机售糖的正规表达式；

(2) 构造识别上述正规式的最简DFA M。