

第三章:词法分析

3.1 词法分析的功能

3.2 词法分析程序的设计与实现

- - 状态图

3.3 词法分析程序的自动生成

- - 有穷自动机

内 容

- 词法分析程序的功能及实现方案
- 单词的种类及词法分析程序的输出形式
- 正则文法和状态图
- 词法分析程序的设计与实现
- 正则表达式与有穷自动机
- 词法分析程序的自动生成器

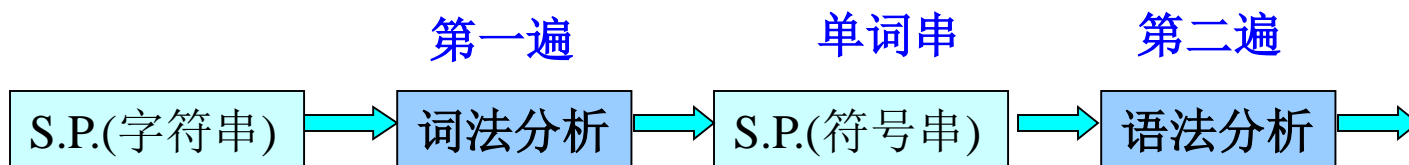
3.1 词法分析程序的功能及实现方案

∞ 词法分析程序的功能

- ◆ 词法分析：根据词法规则识别及组合单词，进行词法检查。
- ◆ 对数字常数完成数字字符串到数值的转换。
- ◆ 删去空格字符和注释。

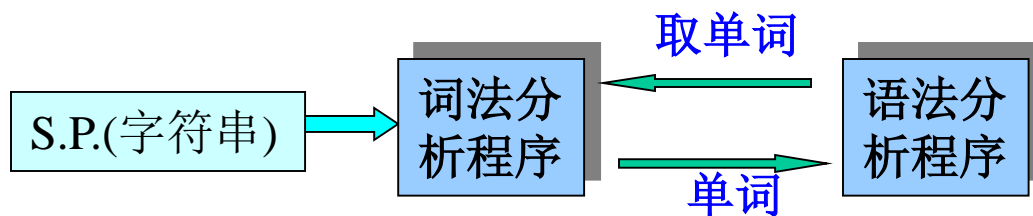
实现方案：基本上有两种

1. 词法分析单独作为一遍



**优点: 结构清晰、
各遍功能单一**
缺点: 效率低

2. 词法分析程序作为单独的子程序



优点: 效率高

3.2 单词的种类及词法分析程序的输出形式

单词的种类

1. **保留字**: begin、end、for、do...
2. **标识符**: 由用户定义, 表示各种名字
3. **常 数**: 无符号数、布尔常数、字符串常数等
4. **分界符**: +、-、*、/、...

词法分析程序的输出形式

表示单词的种类，可用
整数编码或记忆符表示

不同的单词不同的值

单词类别	单词值
------	-----

几种常用的单词内部形式：

- 1、按单词种类分类
- 2、保留字和分界符采用一符一类
- 3、标识符和常数的单词值又为指示字（指针值）

1、按单词种类分类

单词名称	类别编码	单词值
标识符	1	内部字符串
无符号常数(整)	2	整数值
无符号浮点数	3	数值
布尔常数	4	0 或 1
字符串常数	5	内部字符串
保留字	6	保留字或内部编码
分界符	7	分界符或内部编码

2、保留字和分界符采用一符一类

单词名称	类别编码	单词值
标识符	1	内部字符串
无符号常数(整)	2	整数值
无符号浮点数	3	数值
布尔常数	4	0 或 1
字符串常数	5	内部字符串
BEGIN	6	-
END	7	-
FOR	8	-
DO	9	-
.....
:	20	-
+	21	-
*	22	-
,	23	-
(.....	--

3.3 正则文法和状态图

• 状态图的画法（根据文法画出状态图）

例如：正则文法

$$Z ::= U0 \mid V1$$

$$U ::= Z1 \mid 1$$

$$V ::= Z0 \mid 0$$

左线性文法。该文法所定义的语言为：

$$L(G[Z]) = \{ B^n \mid n > 0 \}, \text{ 其中 } B = \{01, 10\}$$

例：正则文法

$Z ::= U0 \mid V1$

$U ::= Z1 \mid 1$

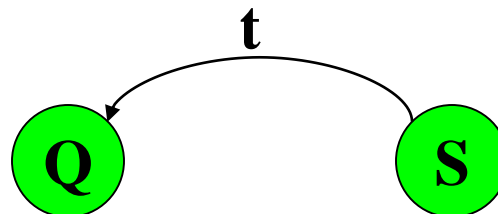
$V ::= Z0 \mid 0$

左线性文法的状态图的画法：

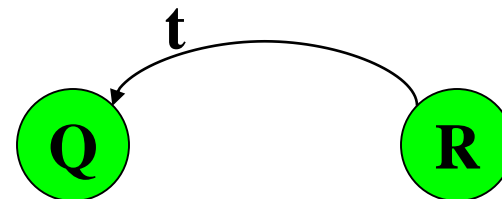
1. 令G的每个非终结符都是一个状态；

2. 设一个开始状态S；

3. 若 $Q ::= t$, $Q \in V_n$, $t \in V_t$, 则：



4. 若 $Q ::= Rt$, $Q, R \in V_n$, $t \in V_t$, 则：



5. 按自动机方法，可加上开始状态和终止状态标志。

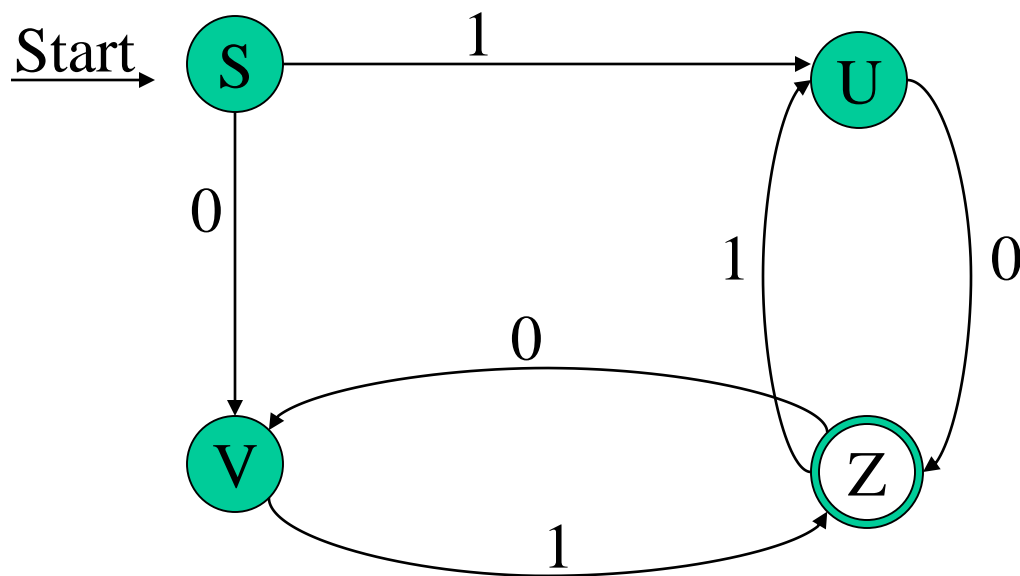
例如：正则文法

$$Z ::= U0 \mid V1$$

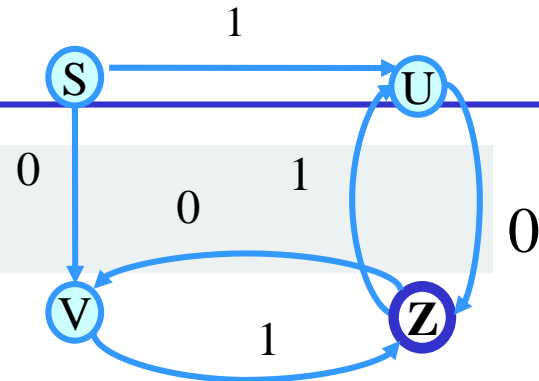
$$U ::= Z1 \mid 1$$

$$V ::= Z0 \mid 0$$

其状态图为：



• 识别算法

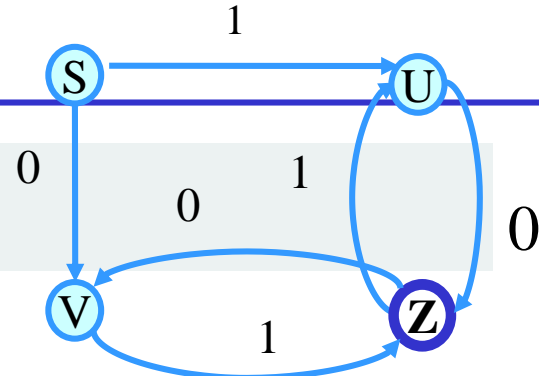


利用状态图可按如下步骤分析和识别字符串 x ：

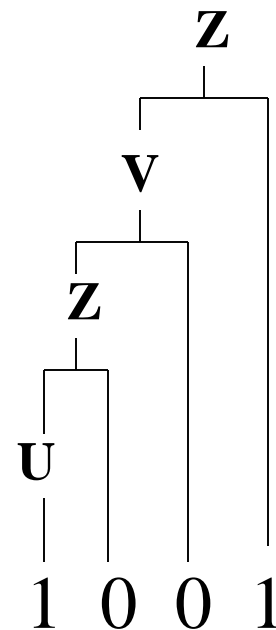
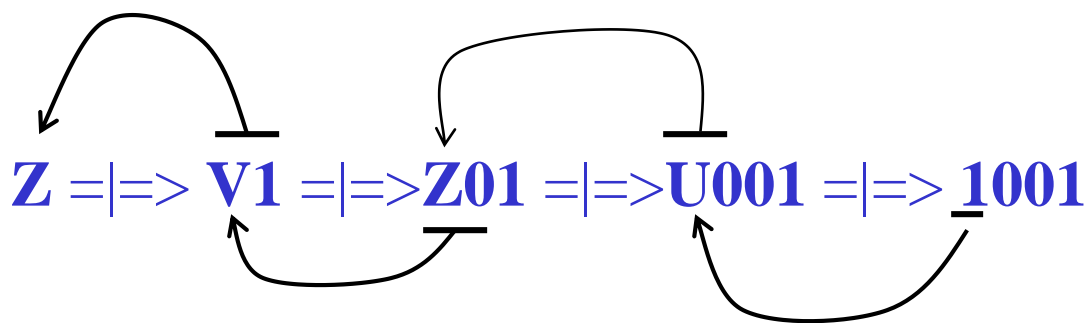
- 1、置初始状态为当前状态，从 x 的最左字符开始，重复步骤2，直到 x 右端为止。
- 2、扫描 x 的下一个字符，在当前状态所射出的弧中找出标记有该字符的弧，并沿此弧过渡到下一个状态；如果找不到标有该字符的弧，那么 x 不是句子，过程到此结束；如果扫描的是 x 的最右端字符，并从当前状态出发沿着标有该字符的弧过渡到下一个状态为终止状态 Z ，则 x 是句子。

例： $x=1001$ 和 1011

•问题:



- 1、上述分析过程是属于自底向上分析？还是自顶向下分析？
- 2、怎样确定句柄？



右线性正则文法例

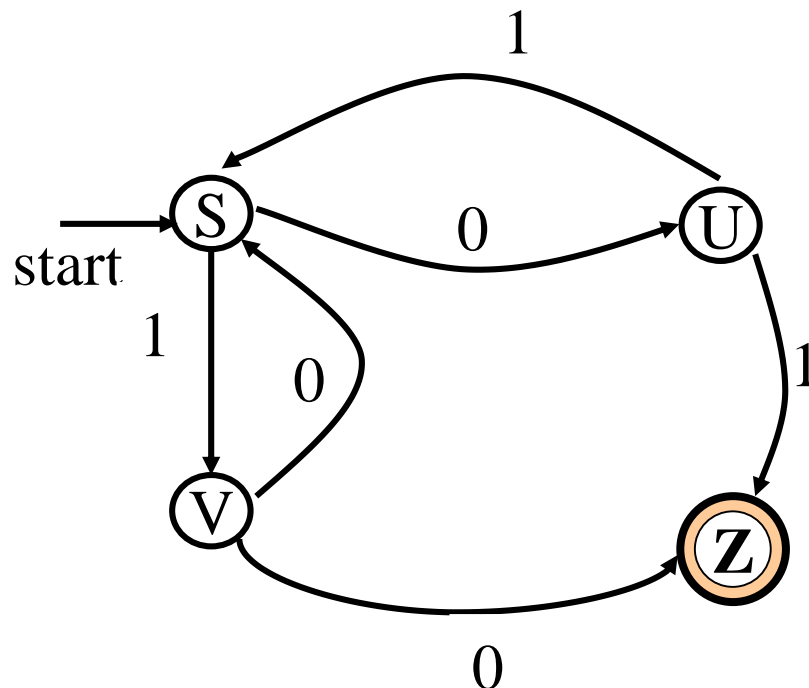
$$S \rightarrow 0U \mid 1V$$

$$U \rightarrow 1S \mid 1$$

$$V \rightarrow 0S \mid 0$$

$$L(G[S]) = \{ B^n \mid n > 0 \}$$

其中 $B = \{ 01, 10 \}$



$$R = (01|10)(01|10)^*$$

作业： 新教材： P73:1, 2

3.4 词法分析程序的设计与实现

词法规则 → 状态图 → 词法分析程序

3.4.1 文法及其状态图

语言的单词符号

标识符

保留字(标识符的子集)

无符号整数

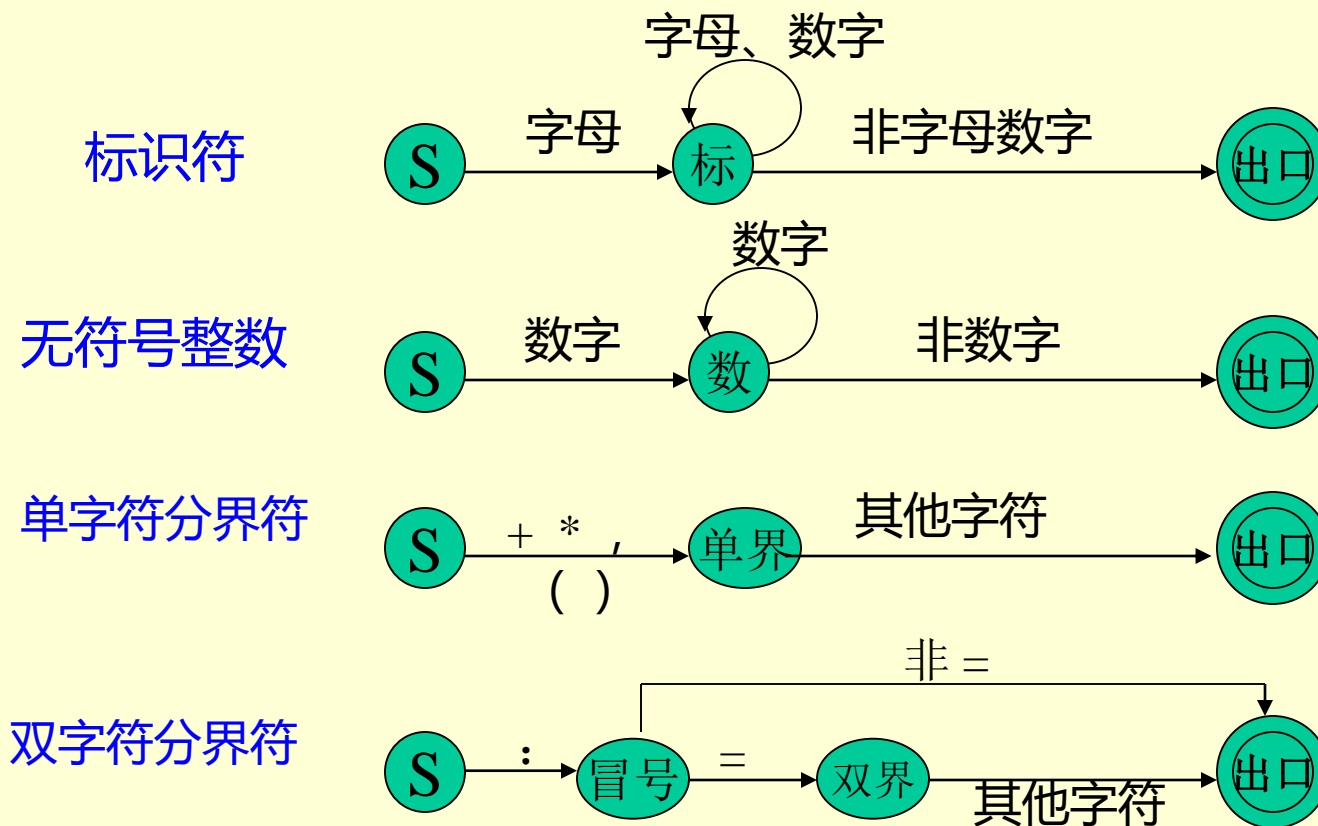
单分界符 + * : , ()

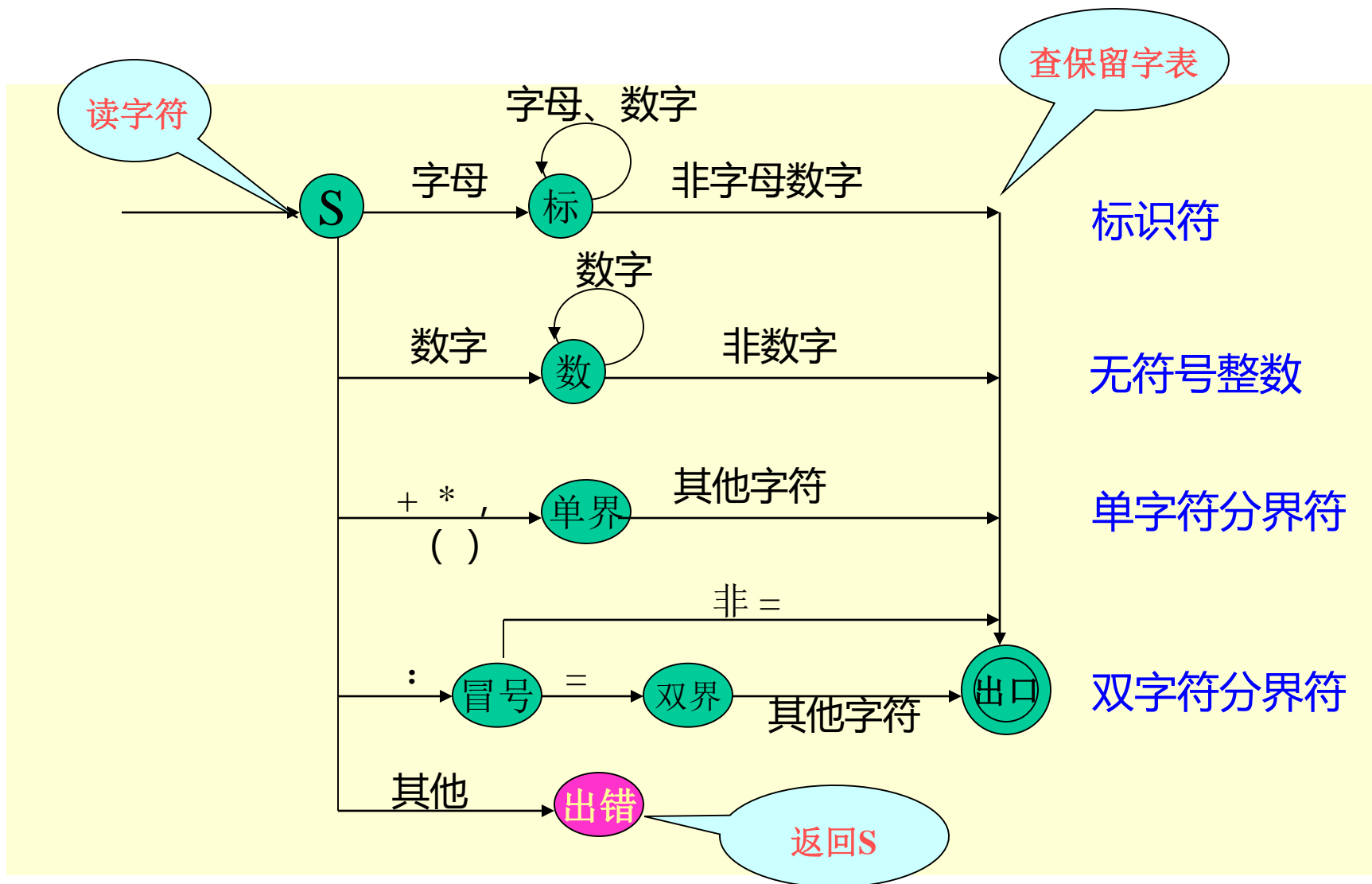
双分界符 :=

两点说明：1. 空格的作用

2. 实数的表示

- 文法: 1. $\langle \text{标识符} \rangle ::= \text{字母} \mid \langle \text{标识符} \rangle \text{字母} \mid \langle \text{标识符} \rangle \text{数字}$
- 2. $\langle \text{无符号整数} \rangle ::= \text{数字} \mid \langle \text{无符号整数} \rangle \text{数字}$
- 3. $\langle \text{单字符分界符} \rangle ::= : \mid + \mid * \mid , \mid (\mid)$
- 4. $\langle \text{双字符分界符} \rangle ::= \langle \text{冒号} \rangle =$
 $\langle \text{冒号} \rangle ::= :$





3.4.2 状态图的实现——构造词法分析程序

1. 单词及内部表示

2. 词法分析程序需要引用的公共（全局）变量和过程

3. 词法分析程序算法

1.单词及内部表示: 保留字和分界符采用一符一类

单词名称	类别编码	记忆符	单词值
BEGIN	1	BEGINSY	-
END	2	ENDSY	-
FOR	3	FORSY	-
DO	4	DOSY	-
IF	5	IFSY	-
THEN	6	THENSY	-
ELSE	7	ELSESY	-
标识符	8	IDSY	内部字符串
常数(整)	9	INTSY	整数值
:	10	COLONSY	-
+	11	PLUSSY	-
*	12	STARSY	-
,	13	COMSY	-
(14	LPARSY	-
)	15	RPARSY	-
:=	16	ASSIGNSY	-

2.词法分析程序需要引用的公共（全局）变量和过程

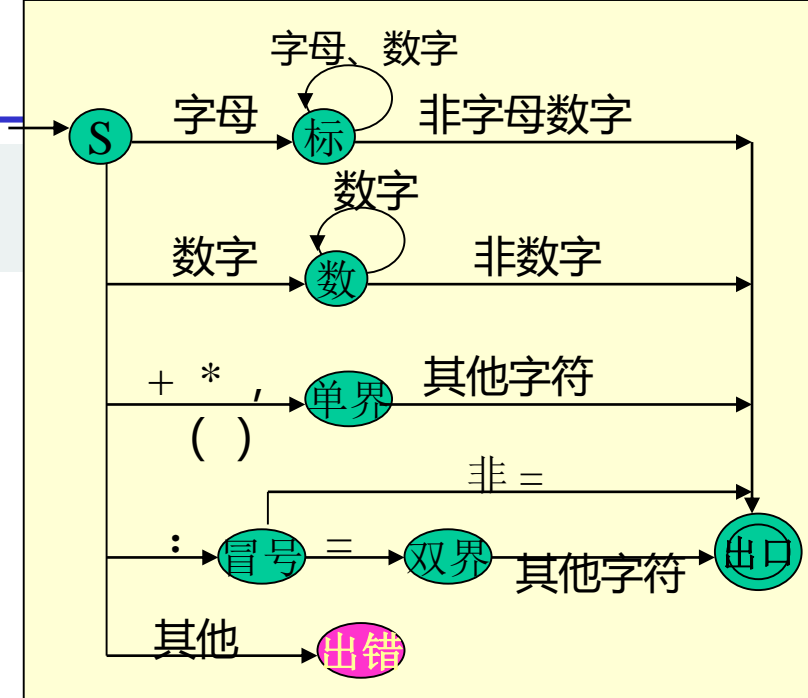
名称	类别	功能
▶ CHAR	• 字符变量	• 存放当前读入的字符
▶ TOKEN	• 字符数组	• 存放单词字符串
▶ GETCHAR	• 读字符过程	• 读字符到CHAR, 移动指针
▶ GETNBC	• 过程	• 反复调用GETCHAR, 直至CHAR进入一个非空白字符
▶ CAT	• 过程	• CHAR与TOKEN连接
▶ ISLETTER 和 ISDIGIT	• 布尔函数	• 判断
▶ UNGETCH	• 过程	• 读字符指针后退一个字符
▶ RESERVE	• 布尔函数	• 判断TOKEN中的字符串 是保留字, 还是标识符
▶ ATOI	• 函数	• 字符串到数字的转换
▶ ERROR	• 过程	• 出错处理

3、词法分析程序算法

```

START: TOKEN := ' '; /*置TOKEN为空串*/
      GETCHAR; GETNBC;
CASE CHAR OF
'A'..'Z': BEGIN
    WHILE ISLETTER OR ISDIGET DO
        BEGIN CAT; GETCHAR END;
    UNGETCH;
    C:= RESERVE; /* 返回0, 为标识符 */
    IF C=0 THEN RETURN('IDSY': TOKEN)
    ELSE RETURN (C,-) /*C为保留字编码*/
END;
'0'..'9': BEGIN
    WHILE DIGIT DO
        BEGIN CAT; GETCHAR END;
    UNGETCH;
    RETURN ('INTSY',ATOI)
END;
'+': RETURN('PLUSSY',-);

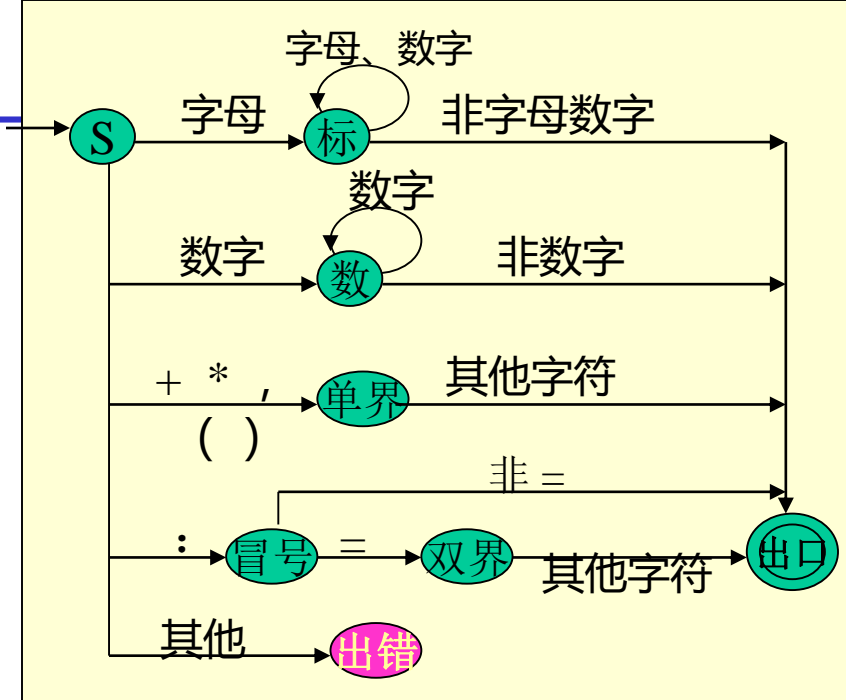
```



```

'*': RETURN('STARSY',-);
',': RETURN('COMMASY',-);
'(': RETURN('LPARSY',-);
')': RETURN('RPARSY',-);
':': BEGIN
    GETCHAR;
    if CHAR='=' THEN RETURN('ASSIGNSY',-);
    UNGETCH;
    RETURN('COLONSY',-);
END
END OF CASE;
ERROR;
GOTO START;
    
```

练习：用C语言实现上述算法。



编程练习作业：

P73： 3 （词法见P66~67）

3.5 正则表达式与有穷自动机

3.5.1 正则表达式和正则集合的递归定义

有字母表 Σ , 定义在 Σ 上的正则表达式和正则集合递归定义如下:

1. ϵ 和 ϕ 都是 Σ 上的正则表达式, 其正则集合分别为: $\{\epsilon\}$ 和 ϕ ;
2. 任何 $a \in \Sigma$, a 是 Σ 上的正则表达式,其正则集合为: $\{a\}$;
3. 假定 U 和 V 是 Σ 上的正则表达式, 其正则集合分别记为 $L(U)$ 和 $L(V)$, 那么 $U|V$, $U \cdot V$ 和 U^* 也都是 Σ 上的正则表达式, 其正则集合分别为 $L(U) \cup L(V)$ 、 $L(U) \cdot L(V)$ 和 $L(U)^*$;
4. 任何 Σ 上的正则表达式和正则集合均由1、 2和3产生。

正则表达式中的运算符：

| -----或（选择） • -----连接
* 或 { } ---重复 () -----括号

运算符的优先级：

先*， 后·， 最后 |
• 在正则表达式中可以省略。

正则表达式相等 \Leftrightarrow 这两个正则表达式表示的语言相等

如： $b\{ab\} = \{ba\}b$

$\{a|b\} = \{\{a\} \{b\}\} = (a^*b^*)^*$

例：设 $\Sigma = \{ a, b \}$, 下面是定义在 Σ 上的正则表达式和正则集合

正则表达式

正则集合

ba^*

以b为首，后跟0个和多个a的符号串

$a(a|b)^*$

Σ 上以a为首的所有符号串

$(a|b)^*(aa|bb)(a|b)^*$

Σ 上含有aa或bb的所有符号串

正则表达式的性质:

设 e_1, e_2 和 e_3 均是某字母表上的正则表达式, 则有:

单位正则表达式: $\varepsilon \quad \varepsilon e = e\varepsilon = e$

交换律: $e_1 \mid e_2 = e_2 \mid e_1$

结合律: $e_1 \mid (e_2 \mid e_3) = (e_1 \mid e_2) \mid e_3$

$e_1(e_2e_3) = (e_1e_2)e_3$

分配律: $e_1(e_2 \mid e_3) = e_1e_2 \mid e_1e_3$

$(e_1 \mid e_2)e_3 = e_1e_3 \mid e_2e_3$

此外: $r^* = (r \mid \varepsilon)^* \quad r^{**} = r^*$

$(r \mid s)^* = (r^*s^*)^*$

正则表达式与3型文法等价

例如：

正则表达式： ba^*

3型文法： $Z ::= Za|b$

$a(a|b)^*$

$Z ::= Za|Zb|a$

例：

3型文法

正则表达式

$S ::= aS|aB$

$B ::= bC$

$C ::= aC|a$

$aS|aba^*a \rightarrow a^*aba^*a$
 \uparrow
 ba^*a
 a^*a

3.5.2 确定的有穷自动机 (DFA) — 状态图的形式化 (Deterministic Finite Automata)

一个确定的有穷自动机 (DFA) M 是一个五元式:

$$M = (S, \Sigma, \delta, s_0, Z)$$

其中:

1. S — 有穷状态集
2. Σ — 输入字母表
3. δ — 映射函数(也称状态转换函数)

$$S \times \Sigma \rightarrow S$$

$$\delta(s, a) = s', \quad s, s' \in S, \quad a \in \Sigma$$

4. s_0 — 初始状态 $s_0 \in S$
5. Z — 终止状态集 $Z \subseteq S$

例如: $M: (\{0, 1, 2, 3\}, \{a, b\}, \delta, 0, \{3\})$

$\delta(0, a) = 1$	$\delta(0, b) = 2$
$\delta(1, a) = 3$	$\delta(1, b) = 2$
$\delta(2, a) = 1$	$\delta(2, b) = 3$
$\delta(3, a) = 3$	$\delta(3, b) = 3$

状态转换函数 δ 可用一矩阵来表示:

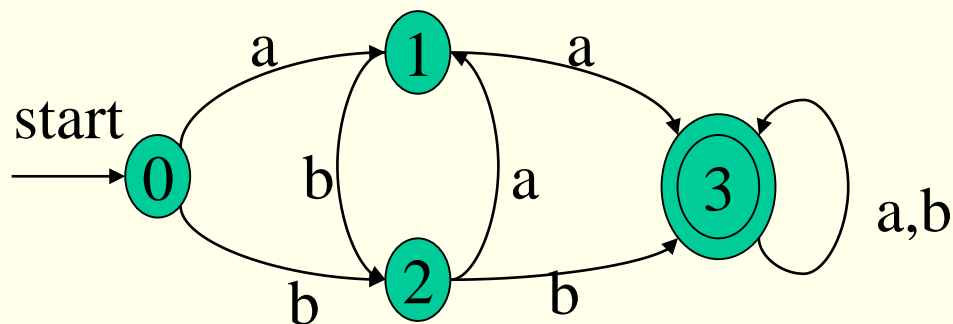
输入 字符 状态	a	b
0	1	2
1	3	2
2	1	3
3	3	3

所谓确定的状态机, 其确定性表现在状态转换函数是单值函数!

DFA也可以用一状态转换图表示：

输入 字符 状态	a	b
0	1	2
1	3	2
2	1	3
3	3	3

DFA的状态图表示：



DFA M所接受的符号串:

令 $\alpha = a_1 a_2 \dots a_n$, $\alpha \in \Sigma$, 若 $\delta(\delta(\dots \delta(s_0, a_1), a_2) \dots a_{n-1}), a_n) = s_n$, 且 $s_n \in Z$, 则可以写成 $\delta(s_0, \alpha) = s_n$, 我们称 α 可为 M 所接受。

$$\delta(s_0, a_1) = s_1$$

$$\delta(s_1, a_2) = s_2$$

.....

$$\delta(s_{n-2}, a_{n-1}) = s_{n-1}$$

$$\delta(s_{n-1}, a_n) = s_n$$

换言之：若存在一条初始状态到某一终止状态的路径，且这条路径上能有弧的标记符号连接成符号串 α ，则称 α 为 DFA M（接受）识别。

DFA M 所接受的语言为： $L(M) = \{ \alpha \mid \delta(s_0, \alpha) = s_n, s_n \in Z \}$

3.5.3 不确定的有穷自动机(NFA) (Nondeterministic Finite Automata)

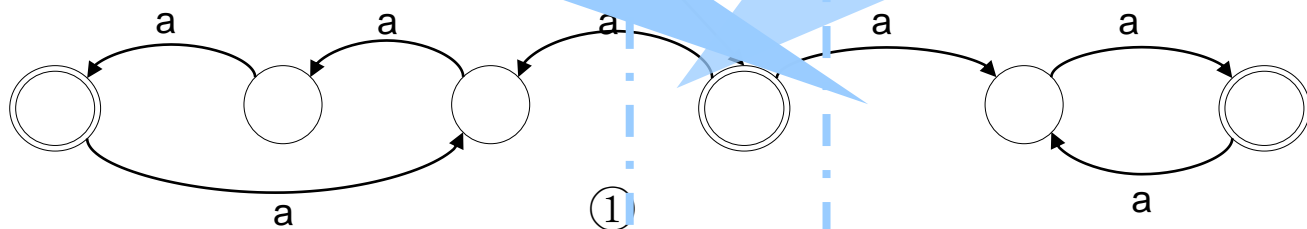
若 δ 是一个多值函数，且输入可允许为 ε ，则有穷自动机是不确定的，即在某个状态下，对于某个输入字符存在多个后继状态。

从同一状态出发，有以同一字符标记的多条边，或者有以 ε 标记的特殊边的自动机。

在当前始态，输入字母a时，自动机既可

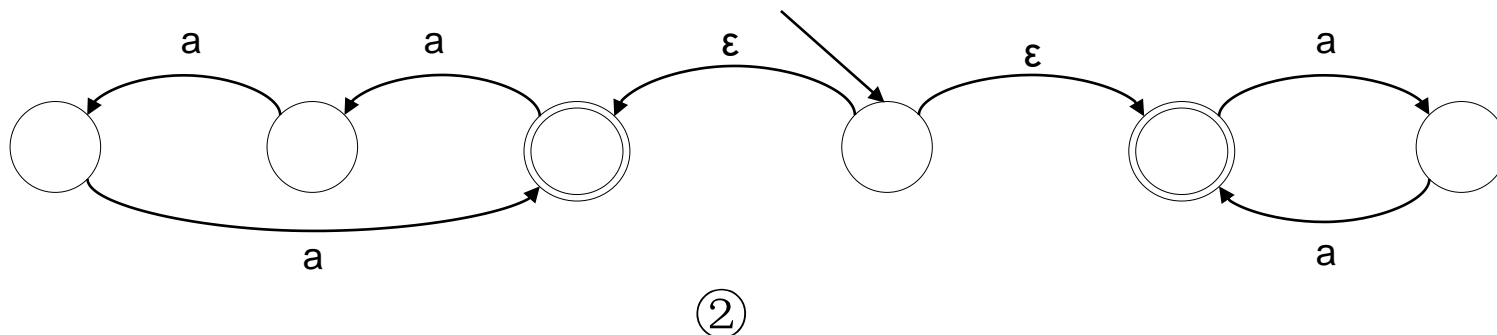
如果向右，则可接受由a组成长度为偶数的字符串。

如果向左，则可接受由a组成长度为3的倍数的字符串；



因此，该NFA所能接受的语言是所有由a组成，长度为2和3的倍数的字符串的集合。在第一次状态转换中，自动机需要选择要走的路径。只要有任何路径可匹配输入字符串，该串就必须被接受，因此NFA必须正确“猜测”所需的路径。

经过以 ϵ 标记的边无须任何字符输入。这里是接受同一语言的另一个NFA:



图示自动机需要选择沿哪一条标记有 ϵ 的边前进。如果一个状态同时引出以 ϵ 标记的边和以其它字符标记的边, 则自动机可以选择处理一个输入字符并沿其对应的边前进, 或者仅沿 ϵ 边前进。

NFA的形式定义为:

一个非确定的有穷自动机NFA M' 是一个五元式:

$NFA\ M' = (S, \Sigma \cup \{\epsilon\}, \delta, s_0, Z)$

其中 S — 有穷状态集

$\Sigma \cup \{\epsilon\}$ — 输入符号加上 ϵ ,

即自动机的每个结点所射出的弧可以是 Σ 中的一个字符或是 ϵ

s_0 — 初态 $s_0 \in S$

Z — 终态集 $Z \subseteq S$

δ — 转换函数 $S \times \Sigma \cup \{\epsilon\} \rightarrow 2^S$

(2^S -- S 的幂集 — S 的子集构成的集合)

NFA M' 所接受的语言为:

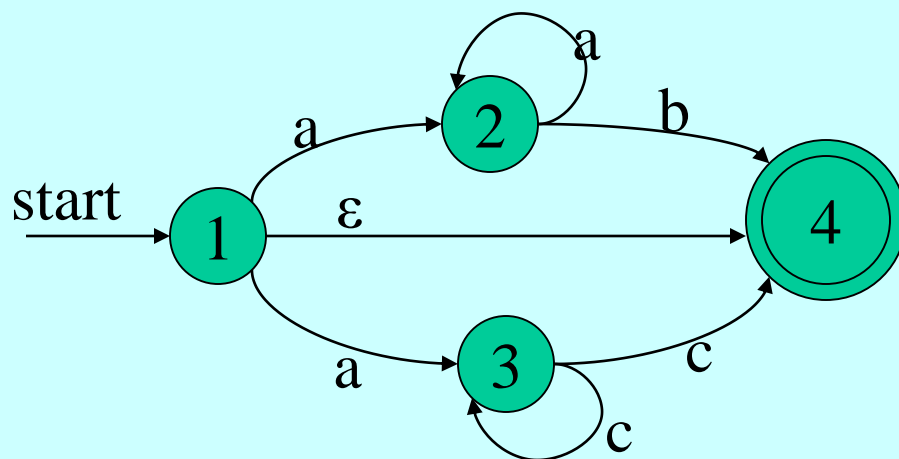
$$L(M') = \{\alpha \mid \delta(S_0, \alpha) = S', S' \cap Z \neq \Phi\}$$

例: NFA $M' = (\{1, 2, 3, 4\}, \{a, b, c\} \cup \{\epsilon\}, \delta, 1, \{4\})$

符号 状态	ϵ	a	b	c
1	{4}	{2, 3}	Φ	Φ
2	Φ	{2}	{4}	Φ
3	Φ	Φ	Φ	{3, 4}
4	Φ	Φ	Φ	Φ

状态 \ 符号	ϵ	a	b	c
1	{4}	{2, 3}	Φ	Φ
2	Φ	{2}	{4}	Φ
3	Φ	Φ	Φ	{3, 4}
4	Φ	Φ	Φ	Φ

上例题相应的状态图为：



M'所接受的语言（用正则表达式） **$R = aa^*b | ac^*c | \epsilon$**

复习:

1. 正则表达式与有穷自动机，给出了两者的定义。

用3型文法所定义的语言都可以用正则表达式描述，
用正则表达式描述单词是为了自动生成词法分析程序。

有一个正则表达式则对应一个正则集合。

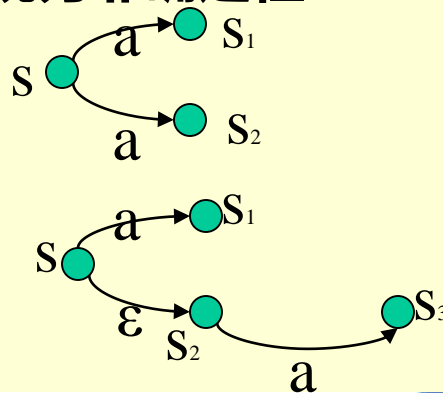
若V是正则集合，iff $V = L(M)$

即一个正则表达式对应一个DFA M

2. NFA的定义, δ 非单值函数, 或有 ϵ 弧, 表现为非确定性

如: $\delta(s, a) = \{s_1, s_2\}$

$\delta(s, a) = \{s_1, s_3\}$



3.5.4 NFA的确定化

正如我们所学到的，用计算机程序实现DFA是很容易的。但在不能正确猜测路径的情况下，NFA的实现就有些困难了。

已证明：不确定的有穷自动机与确定的有穷自动机从功能上来说是等价的，也就是说能够从：

NFA M $\xrightarrow{\text{构造一个}}$ DFA M'
使得 $L(M)=L(M')$

为了使得NFA确定化，首先给出两个定义：

定义1、集合I的 ϵ -闭包：

令I是一个状态集的子集，定义 ϵ -closure (I) 为：

- 1) 若 $s \in I$ ，则 $s \in \epsilon$ -closure (I) ；
 - 2) 若 $s \in I$ ，则从s出发经过任意条 ϵ 弧能够到达的任何状态都属于 ϵ -closure (I) 。
- 状态集 ϵ -closure (I) 称为I的 ϵ -闭包。

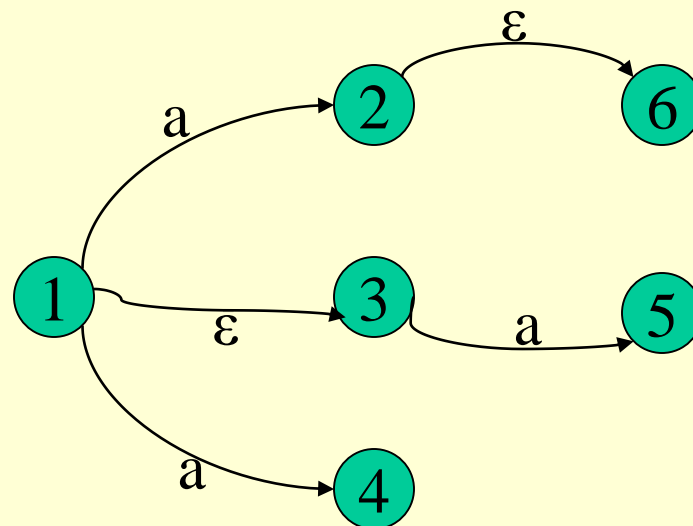
可以通过一例子来说明状态子集的 ϵ -闭包的构造方法

例：

如图所示的状态图：

令 $I = \{1\}$ ，

求 ϵ -closure (I) = ?



根据定义：

ϵ -closure (I) = {1, 3}

定义2: 令 I 是NFA M' 的状态集的一个子集, $a \in \Sigma$

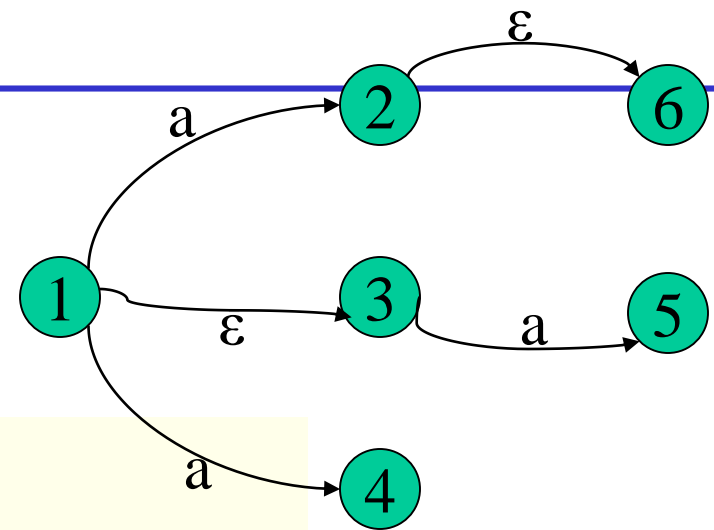
定义: $I_a = \varepsilon\text{-closure}(J)$

其中 $J = \bigcup_{s \in I} \delta(s, a)$

-- J 是从状态子集 I 中的每个状态出发,经过标记为 a 的弧而达到的状态集合。

-- I_a 是状态子集,其元素为 J 中的状态,加上从 J 中每一个状态出发通过 ε 弧到达的状态。

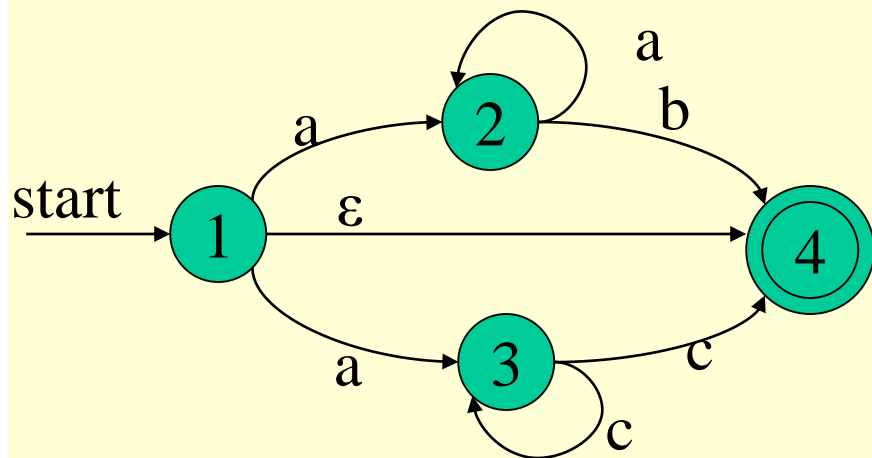
同样可以通过一例子来说明上述定义,仍采用前面给定的状态图为例



例：令 $I = \{1\}$
 $I_a = \epsilon\text{-closure}(J)$
 $= \epsilon\text{-closure}(\delta(1, a))$
 $= \epsilon\text{-closure}(\{2, 4\})$
 $= \{2, 4, 6\}$

根据定义1, 2, 可以将上述的M'确定化（即可构造出状态转换矩阵）

例：有NFA M'



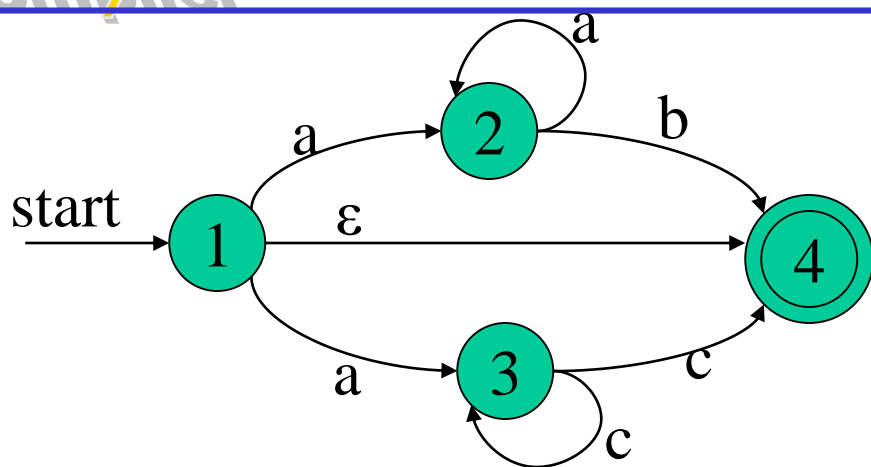
$$I = \varepsilon\text{-closure}(\{1\}) = \{1, 4\}$$

$$\begin{aligned} I_a &= \varepsilon\text{-closure}(\delta(1, a) \cup \delta(4, a)) \\ &= \varepsilon\text{-closure}(\{2, 3\} \cup \varnothing) \\ &= \varepsilon\text{-closure}(\{2, 3\}) \\ &= \{2, 3\} \end{aligned}$$

$$\begin{aligned} I_b &= \varepsilon\text{-closure}(\delta(1, b) \cup \delta(4, b)) \\ &= \varepsilon\text{-closure}(\varnothing) \\ &= \varnothing \end{aligned}$$

$$\begin{aligned} I_c &= \varepsilon\text{-closure}(\delta(1, c) \cup \delta(4, c)) \\ &= \varnothing \end{aligned}$$

$$I = \{2, 3\}, I_a = \{2\}, I_b = \{4\}, I_c = \{3, 4\} \dots$$



I	I_a	I_b	I_c
{1,4}	{2,3}	\varnothing	\varnothing
{2,3}	{2}	{4}	{3,4}
{2}	{2}	{4}	\varnothing
{4}	\varnothing	\varnothing	\varnothing
{3,4}	\varnothing	\varnothing	{3,4}

将求得的状态转换矩阵重新编号

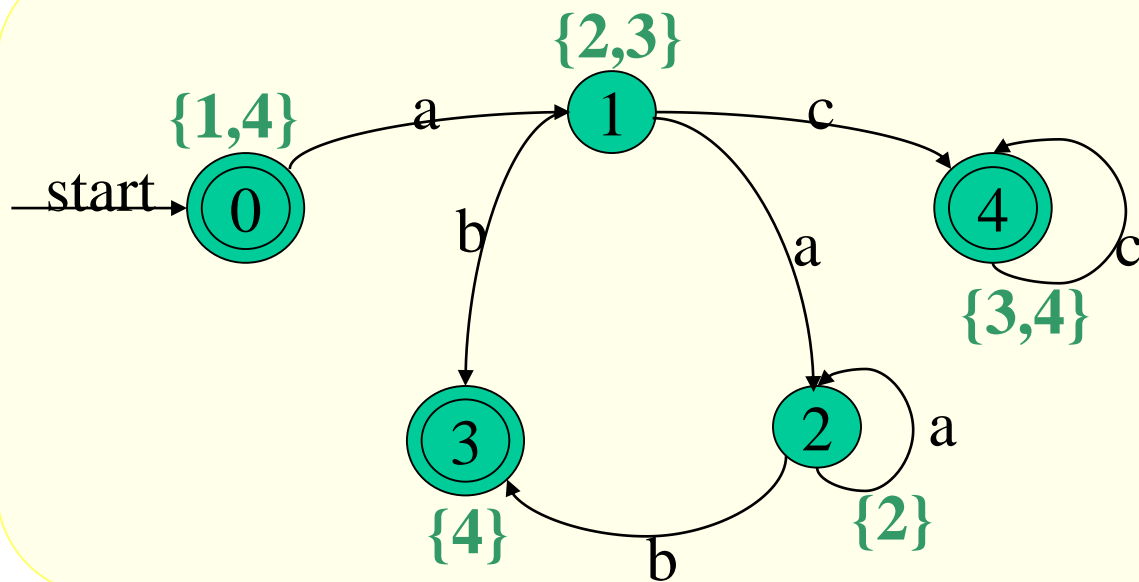
DFA M状态转换矩阵：

I	I _a	I _b	I _c
{1,4}	{2,3}	φ	φ
{2,3}	{2}	{4}	{3,4}
{2}	{2}	{4}	φ
{4}	φ	φ	φ
{3,4}	φ	φ	{3,4}

<div>符号</div> <div>状态</div>	a	b	c
0	1	—	—
1	2	3	4
2	2	3	—
3	—	—	—
4	—	—	4

符号 状态	a	b	c
0	1	—	—
1	2	3	4
2	2	3	—
3	—	—	—
4	—	—	4

DFA M的状态图:



★ 注意：原初始状态1的 ϵ -closure子集为DFA M的初态
包含原终止状态4的状态子集为DFA M的终态。

3.5.5 正则表达式与DFA的等价性

定理： 在 Σ 上的一个子集 V ($V \subseteq \Sigma^*$) 是正则集合，当且仅当存在一个DFA M ，使得 $V=L(M)$

V 是正则集合，
 R 是与其相对应的正则表达式 \Leftrightarrow DFA M
 $V=L(R)$ $L(M)=L(R)$

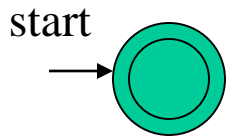
所以 正则表达式 $R \Rightarrow$ NFA $M' \Rightarrow$ DFA M
 $L(R) = L(M') = L(M)$

证明: 根据定义。

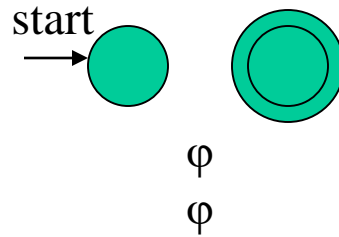
正则表达式和正则集合的递归定义

有字母表 Σ , 定义在 Σ 上的正则表达式和正则集合递归定义如下:

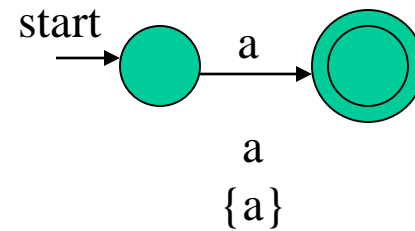
1. ϵ 和 ϕ 都是 Σ 上的正则表达式, 其正则集合分别为: $\{\epsilon\}$ 和 ϕ ;
2. 任何 $a \in \Sigma$, a 是 Σ 上的正则表达式,其正则集合为: $\{a\}$;
3. 假定 U 和 V 是 Σ 上的正则表达式, 其正则集合分别记为 $L(U)$ 和 $L(V)$, 那么 $U|V$, $U \cdot V$ 和 U^* 也都是 Σ 上的正则表达式, 其正则集合分别为 $L(U) \cup L(V)$ 、 $L(U) \cdot L(V)$ 和 $L(U)^*$;
4. 任何 Σ 上的正则表达式和正则集合均由1、 2和3产生。



正则表达式 ε
正则集合 $\{\varepsilon\}$

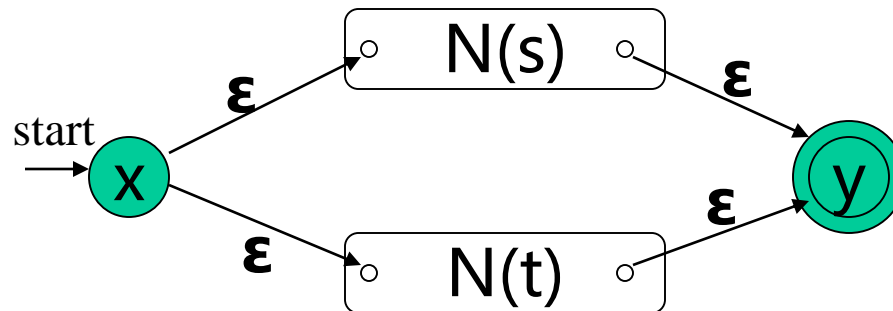


φ
 φ

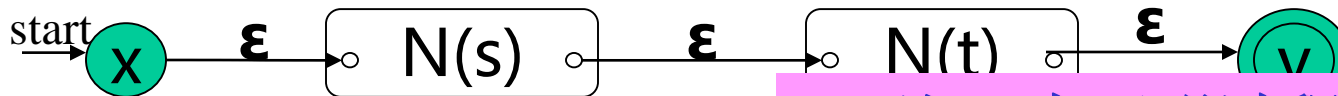


a
 $\{a\}$

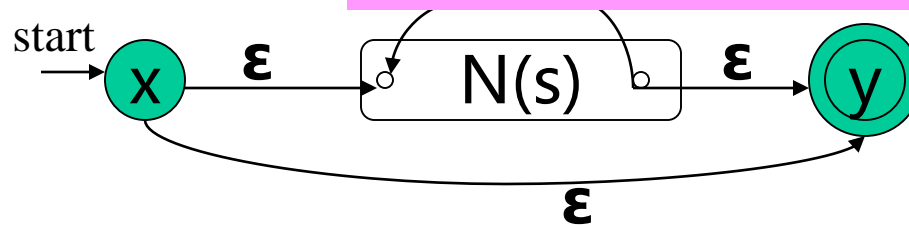
$R=s|t$ NFA(R):



$R=st$ NFA(R):



$R=s^*$ NFA(R):



注：本证明过程也给出了
将正则表达式转换为NFA的算法

复习:

1. 正则表达式与有穷自动机, 给出了两者的定义。
用3型文法所定义的语言都可以用正则表达式描述,
用正则表达式描述单词是为了自动生成词法分析程序。
有一个正则表达式则对应一个正则集合。
2. NFA M' 的定义、确定化 → **对任何一个NFA M' , 都可以构造出一个DFA M , 使得 $L(M) = L(M')$**
3. **正则表达式与DFA的等价性**
我们证明了对任何一个正则表达式, 都可以构造出等价的NFA.

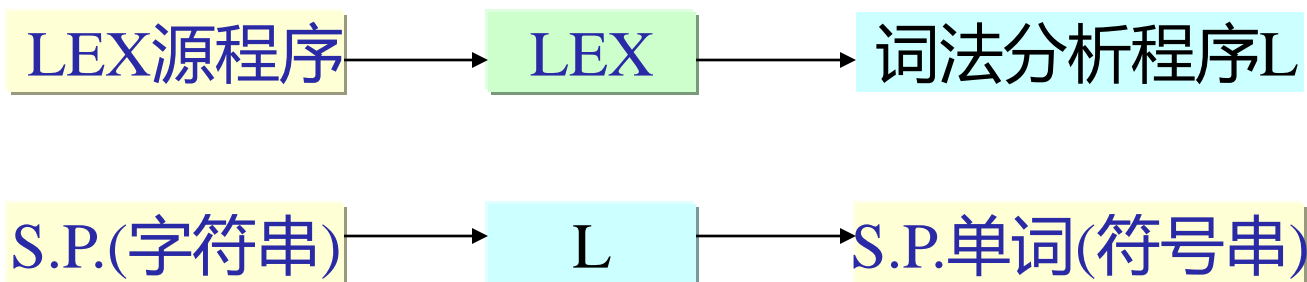
3.6 词法分析程序的自动生成器—LEX (LEXICAL)

LEX的原理:

正则表达式与DFA的等价性

根据给定的正则表达式自动生成相应的词法分析程序。

LEX的功能:



3.6.1 LEX源程序

一个LEX源程序主要由三个部分组成:

1. 辅助定义式
2. 识别规则
3. 用户子程序

各部分之间用%%隔开

辅助定义式是如下形式的LEX语句：

$$D_1 \longrightarrow R_1$$

$$D_2 \longrightarrow R_2$$

$$\vdots$$

$$\vdots$$

$$D_n \longrightarrow R_n$$

其中：

R_1, R_2, \dots, R_n 为正则表达式。
 D_1, D_2, \dots, D_n 为正则表达式名字，称简名。

例：标识符：
 $\text{letter} \rightarrow A|B|\dots|Z$
 $\text{digit} \rightarrow 0|1|\dots|9$
 $\text{iden} \rightarrow \text{letter}(\text{letter}|\text{digit})^*$

带符号整数：
 $\text{integer} \rightarrow \text{digit}(\text{digit})^*$
 $\text{sign} \rightarrow +|-|\epsilon$
 $\text{sign_integer} \rightarrow \text{sign integer}$



识别规则：是一串如下形式的LEX语句：

$$\begin{array}{ll} P_1 & \{A_1\} \\ P_2 & \{A_2\} \\ & \vdots \\ & \vdots \\ P_m & \{A_m\} \end{array}$$

P_i ：定义在 $\Sigma \cup \{D_1, D_2, \dots, D_n\}$ 上的正则表达式，也称词形。

$\{A_i\}$ ： A_i 为语句序列，它指出，在识别出词形为 P_i 的单词以后，词法分析器所应作的动作。

其基本动作是返回单词的类别编码和单词值。

下面是识别某语言单词符号的LEX源程序：

例：LEX 源程序

```
AUXILIARY DEF
    letter → A|B|...|Z
    digit  → 0|1|...|9
%%
RECOGNITION RULES
```

1.BEGIN

2.END

3.FOR

{RETURN(1,—) }

{RETURN(2,—) }

{RETURN(3,—) }

RETURN是LEX过程，该过程将单词传给语法分析程序

RETURN (C, LEXVAL)

其中C为单词类别编码

LEXVAL:

标识符: TOKEN (字符数组)

整常数: DTB (数值转换函数, 将TOKEN
中的数字串转换二进制值)

其他单词: 无定义

/* 识别规则 */

4.DO	{RETURN(4,—) }
5.IF	{RETURN(5,—) }
6.THEN	{RETURN(6,—) }
7.ELSE	{RETURN(7,—) }
8.letter(letter digit)*	{RETURN(8,TOKEN) }
9.digit(digit)*	{RETURN(9,DTB) }
10. :	{RETURN(10,—) }
11. +	{RETURN(11,—) }
12. “*”	{RETURN(12,—) }

13. ,	{RETURN(13,—) }
14. “ (”	{RETURN(14,—) }
15. “) ”	{RETURN(15,—) }
16. :=	{RETURN(16,—) }
17. =	{RETURN(17,—) }

3.6.2 LEX的实现

LEX的功能是根据LEX源程序构造一个词法分析程序，该词法分析器实质上是一个有穷自动机。

LEX生成的词法分析程序由两部分组成：

词法分析程序

状态转换矩阵(DFA)

控制执行程序

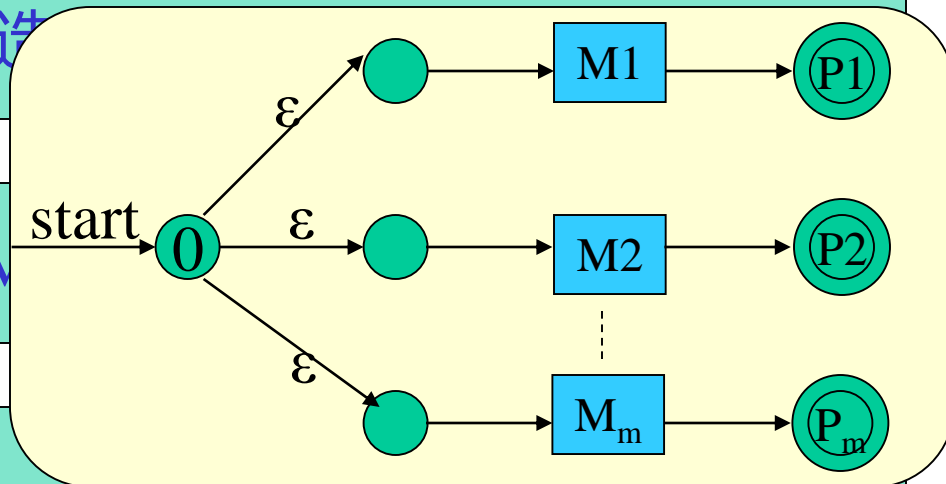
∴ LEX的功能是根据LEX源程序生成状态转换矩阵和控制程序

LEX的工作过程:

· 扫描每条识别规则 P_i , 构造

· 将各条规则的有穷自动机NFA

· 确定化 $NFA \Rightarrow DFA$



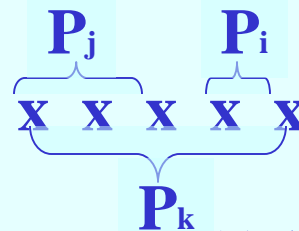
生成该DFA的状态转换矩阵和控制执行程序

如:BEGIN, :=

LEX二义性问题的两条原则:

1.最长匹配原则

在识别单词过程中, 有一字符串
根据最长匹配原则, 应识别为这是一个符合 P_k 规则的单词, 而不是 P_j 和 P_i 规则的单词。



2.优先匹配原则

如有一字符串, 有两条规则可以同时匹配时, 那么用规则序列中位于前面的规则相匹配, 所以排列在最前面的规则优先权最高。

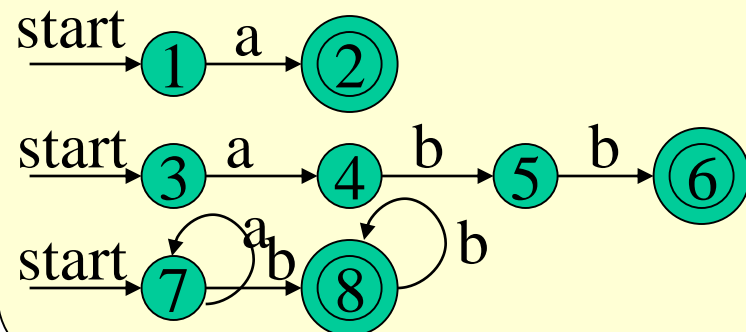
例：字符串 $\cdot \underbrace{\text{BEGIN} \cdot}_{P_8}^{P_1}$

根据原则，应该识别为关键字BEGIN，所以在写LEX源程序时应注意规则的排列顺序。另要注意的是，优先匹配原则是在符合最长匹配的前提下执行的。

可以通过一个例子来说明这些问题：

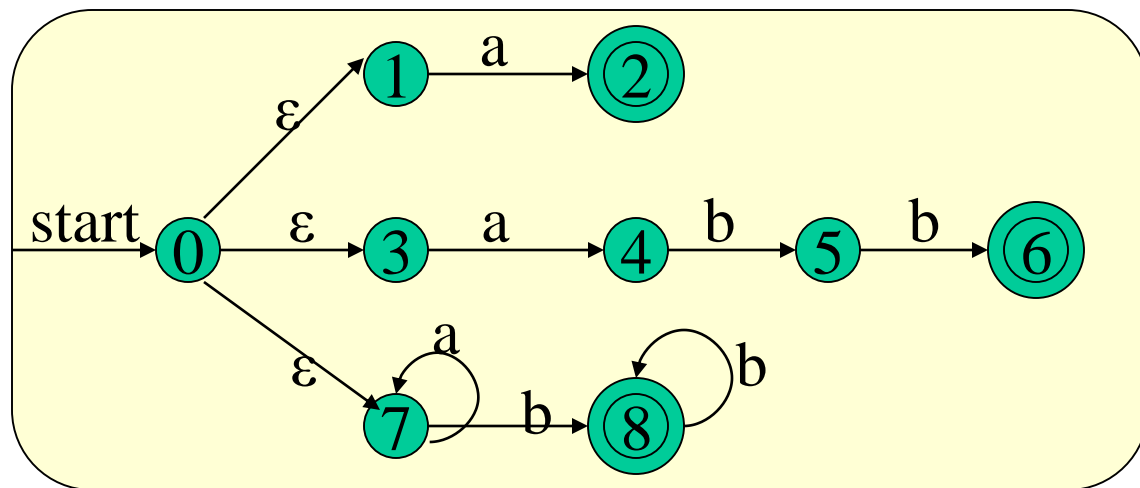
例： LEX源程序

a	{ }
abb	{ }
a*bb*	{ }



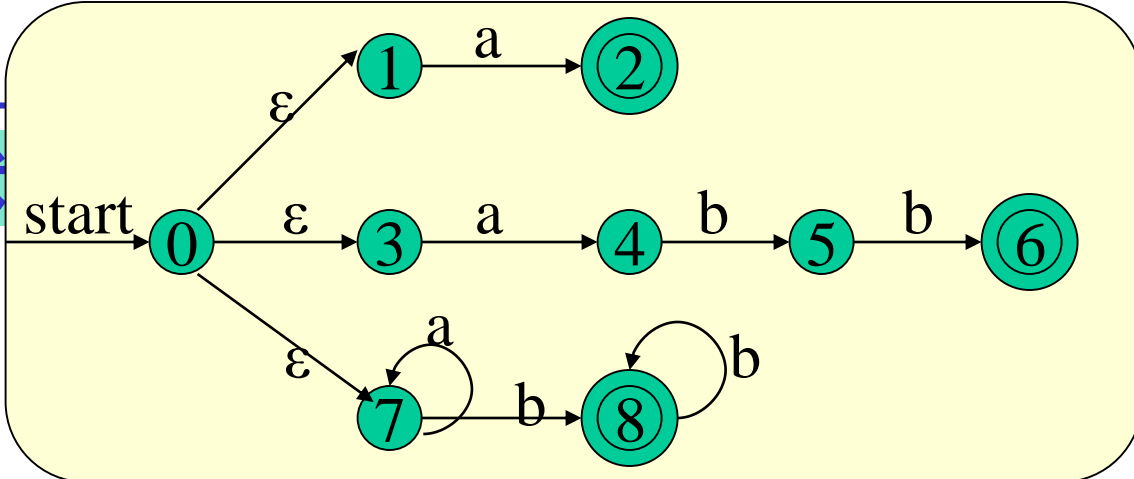
一.读LEX源程序，分别生成NFA，用状态图表示为：

二.合并成一个NFA：



三.确定化

给出状



状态	a	b	到达终态所识别的单词
初态 {0,1,3,7}	{2,4,7}	{8}	
终态 {2,4,7}	{7}	{5,8}	a
终态 {8}	\varnothing	{8}	$a^* bb^*$
终态 {7}	{7}	{8}	
终态 {5,8}	\varnothing	{6,8}	$a^* bb^*$
终态 {6,8}	\varnothing	{8}	abb

在此DFA中 初态为{0,1,3,7}

终态为{2,4,7},{8},{5,8},{6,8}

词法分析程序的分析过程

令输入字符串为aba...

- (1) 吃进字符ab
- (2) 按反序检查状态子集
检查前一次状态是否含有原
NFA的终止状态

读入字符	进入状态
开始	{0,1,3,7}
a	{2,4,7}
b	{5,8}
a	无后继状态(退掉输入字符a)

- 即检查{5,8},含有终态8, 因此断定所识别的单词ab是属于 $a*bb*$ 中的一个。
- 若在状态子集中无NFA的终态, 则要从识别的单词再退掉一个字符(b), 然后再检查上一个状态子集。
- 若一旦吃进的字符都退完, 则识别失败, 调用出错程序, 一般是跳过一个字符然后重新分析。(应打印出错信息)

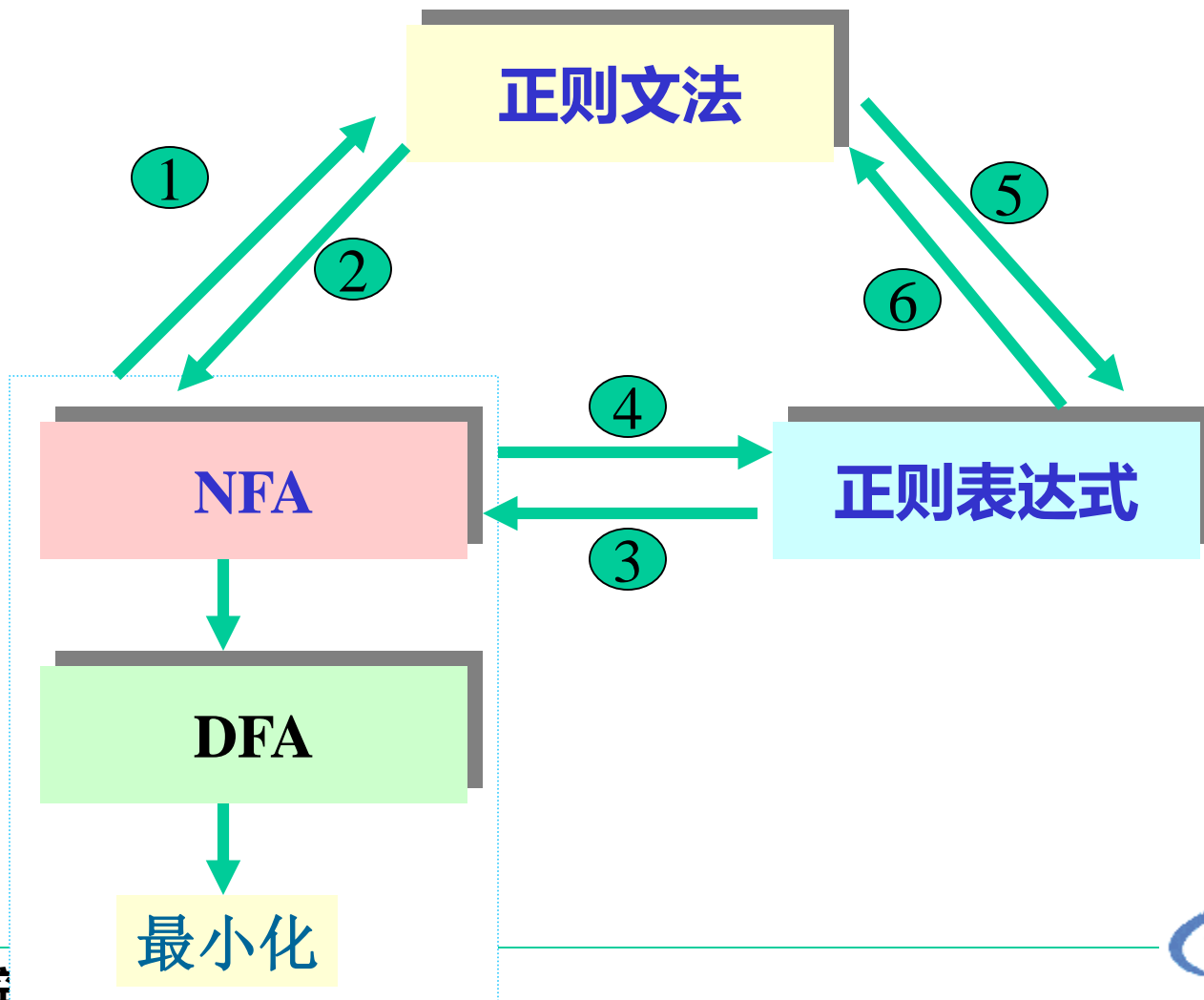
三点说明:

- 1) 以上是LEX的构造原理，虽然是原理性的，但据此就不难将LEX构造出来。
- 2) 所构造出来的LEX是一个通用的工具，用它可以生成各种语言的词法分析程序，只需要根据不同的语言书写不同的LEX源文件就可以了。
- 3) LEX不但能自动生成词法分析器，而且也可以产生多种模式识别器及文本编辑程序等。

第11章作业:

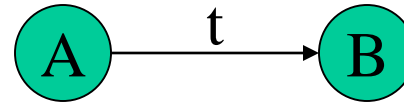
P254 (76-77) : 1、2、4、
5

补充



(1) 有穷自动机 \Rightarrow 正则文法

算法:



1.对转换函数 $f(A,t)=B$, 可写成一个产生式: $A \rightarrow tB$

2.对可接受状态 Z ,增加一个产生式: $Z \rightarrow \epsilon$

3.有穷自动机的初态对应于文法的开始符号(识别符号),
有穷自动机的字母表为文法的终结符号集。

例:给出如图NFA等价的正则文法G

$G = (\{A, B, C, D\}, \{a, b\}, P, A)$

其中P:

$A \rightarrow aB$

$A \rightarrow bD$

$B \rightarrow bC$

$C \rightarrow aA$

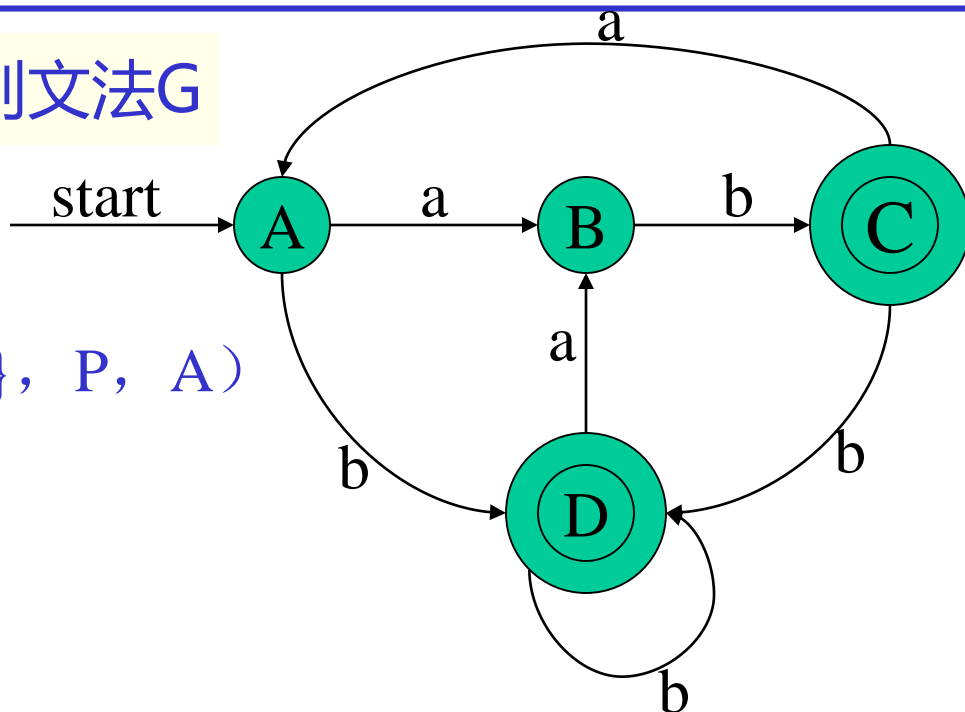
$C \rightarrow bD$

$C \rightarrow \varepsilon$

$D \rightarrow aB$

$D \rightarrow bD$

$D \rightarrow \varepsilon$



(2) 正则文法 \Rightarrow 有穷自动机M

算法:

- 1.字母表与G的终结符号相同;
- 2.为G中的每个非终结符生成M的一个状态,G的开始符号S是开始状态S;
- 3.增加一个新状态Z,作为NFA的终态;
- 4.对G中的形如 $A \rightarrow tB$,其中t为终结符或 ϵ ,A和B为非终结符的产生式,构造M的一个转换函数 $f(A,t)=B$;
- 5.对G中的形如 $A \rightarrow t$ 的产生式,构造M的一个转换函数 $f(A,t)=Z$ 。

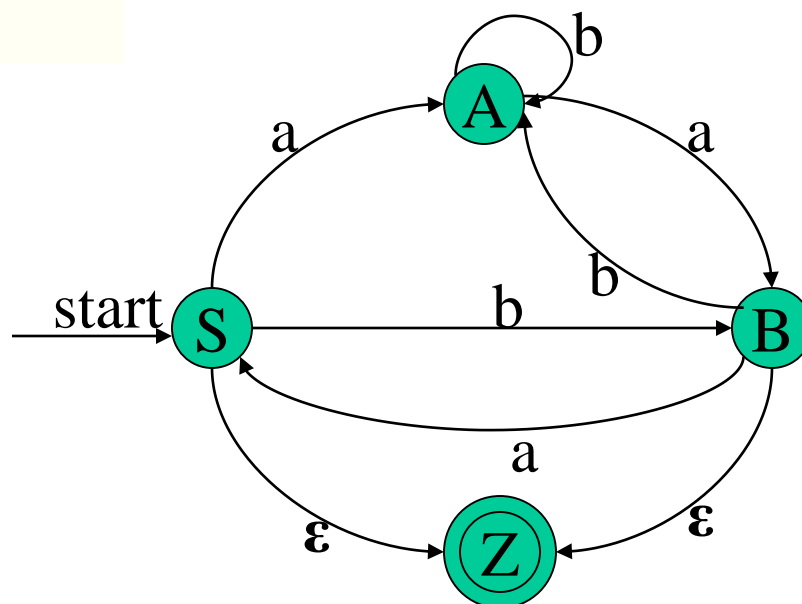
例:求与文法G[S]等价的NFA

G[S]: $S \rightarrow aA | bB | \epsilon$

$A \rightarrow aB | bA$

$B \rightarrow aS | bA | \epsilon$

求得:



左线形正则文法和右线性正则文法的等价

左线性正则文法例

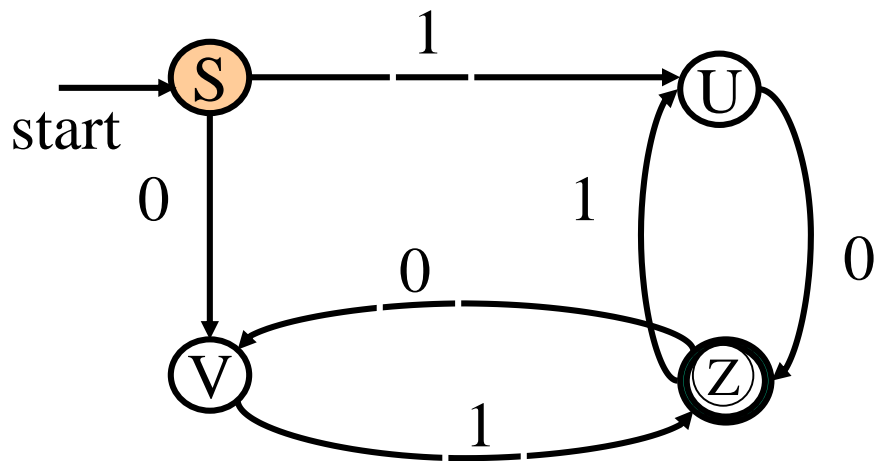
$Z \rightarrow U0 \mid V1$

$U \rightarrow Z1 \mid 1$

$V \rightarrow Z0 \mid 0$

$L(G[Z]) = \{ B^n \mid n > 0 \}$

其中 $B = \{01, 10\}$



$R = (01|10)(01|10)^*$

右线性正则文法例

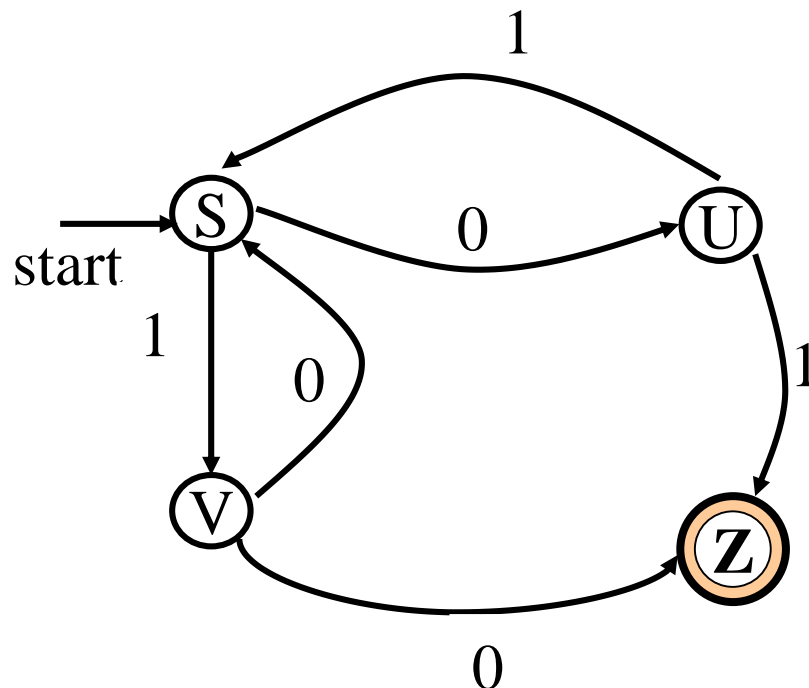
$$S \rightarrow 0U \mid 1V$$

$$U \rightarrow 1S \mid 1$$

$$V \rightarrow 0S \mid 0$$

$$L(G[S]) = \{ B^n \mid n > 0 \}$$

其中 $B = \{ 01, 10 \}$



$$R = (01|10)(01|10)^*$$

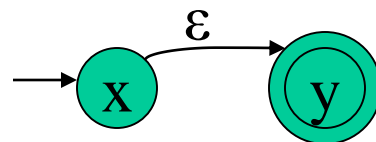

(3) 正则式 \Rightarrow 有穷自动机

语法制导方法

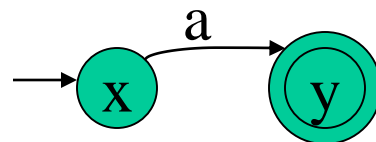
1.(a)对于正则式 ϕ ,所构造NFA:



(b)对于正则式 ϵ ,所构造NFA:

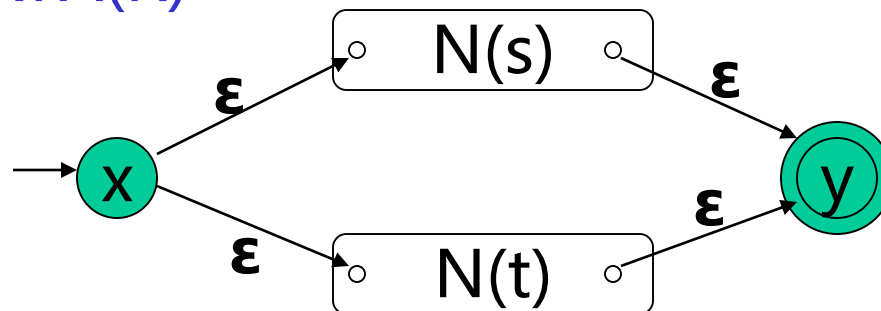


(c)对于正则式 $a, a \in \Sigma$,则 NFA:

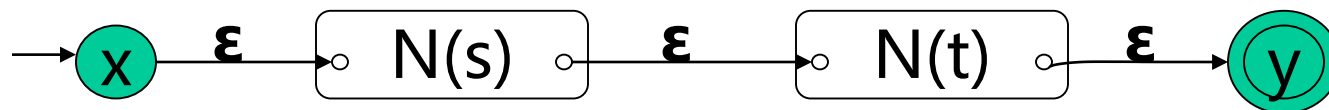


2.若 s, t 为 Σ 上的正则式,相应的NFA分别为 $N(s)$ 和 $N(t)$;

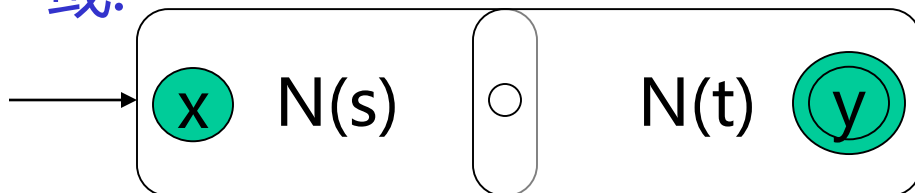
(a)对于正则式 $R = s|t$, $NFA(R)$



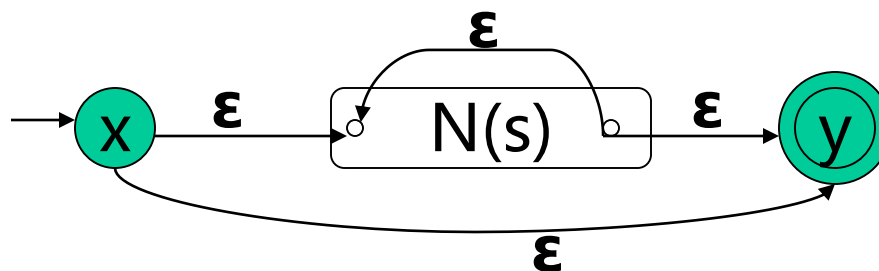
(b)对正则式 $R = st$, $NFA(R)$



或:

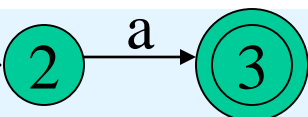


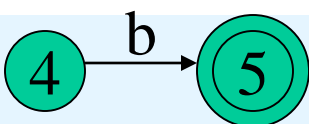
(c)对于正则式 $R=s^*$, $NFA(R)$



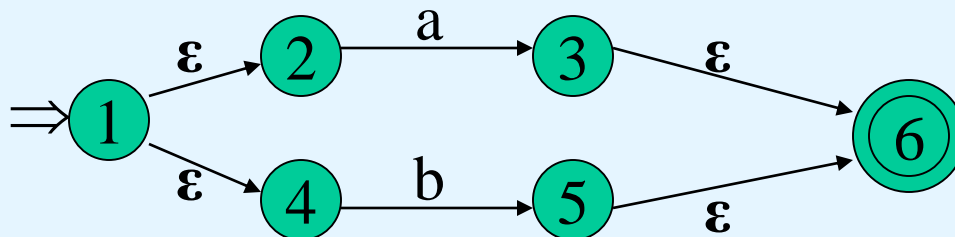
(d)对 $R=(s)$,与 $R=s$ 的NFA一样.

例:为 $R=(a|b)^*abb$ 构造NFA N ,使得 $L(N)=L(R)$

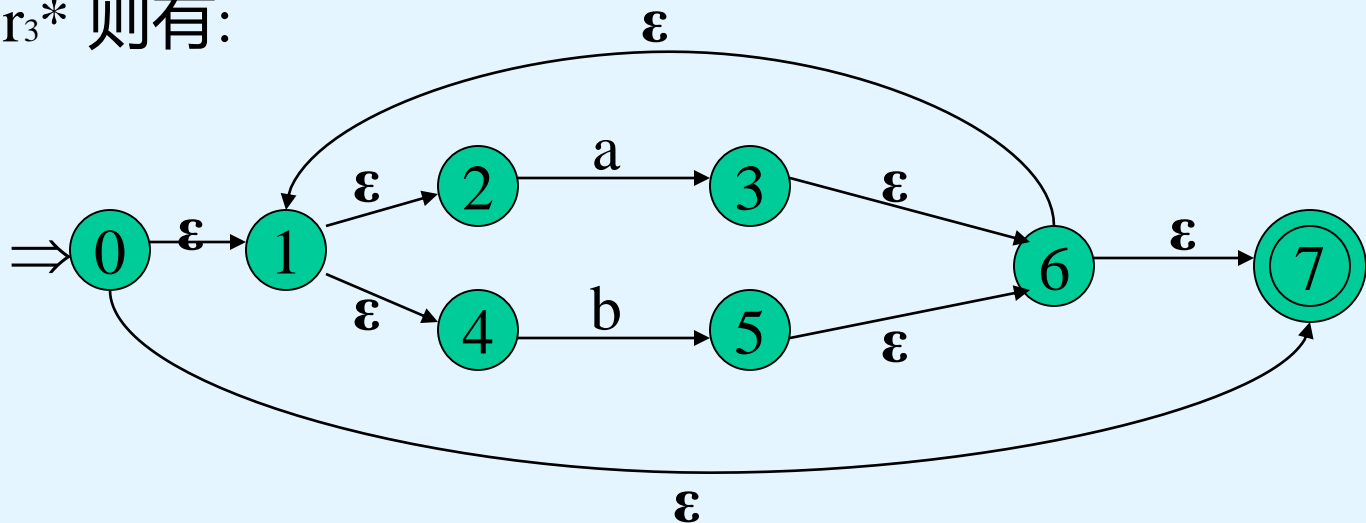
从左到右分解 R ,令 $r_1=a$,第一个 a ,则有 \Rightarrow 

令 $r_2=b$,则有 \Rightarrow 

令 $r_3 = r_1 | r_2$,则有



令 $r_4 = r_3^*$ 则有:



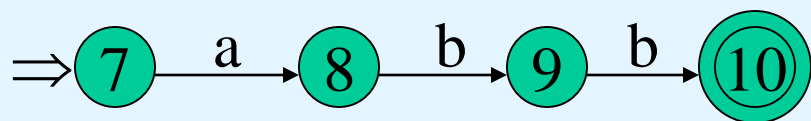
令 $r_5 = a$,

令 $r_6 = b$

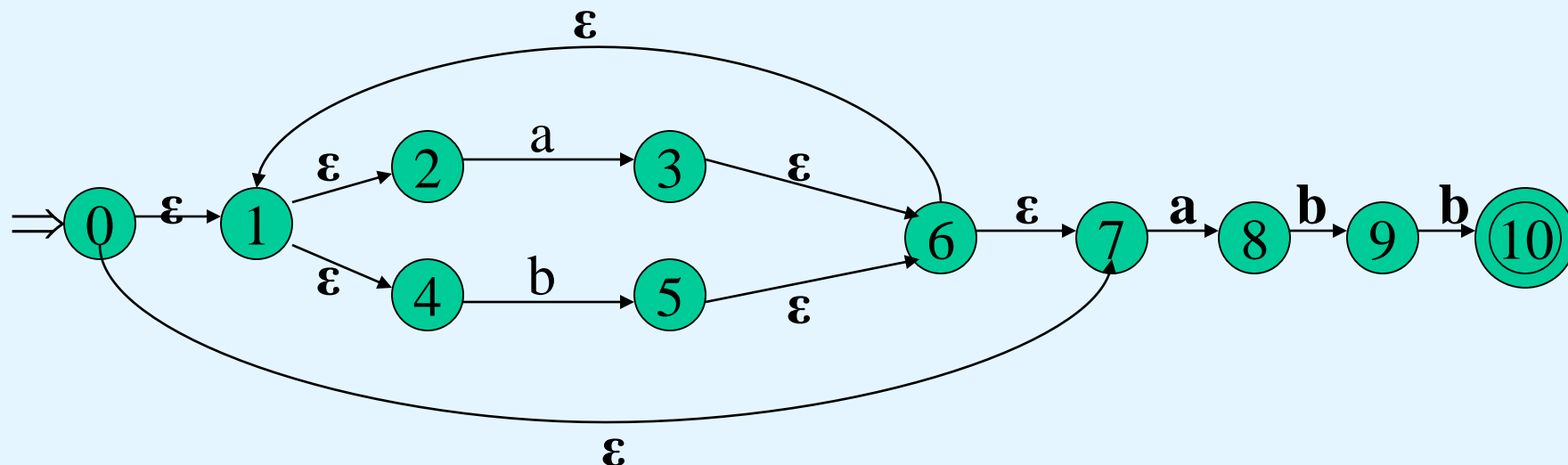
令 $r_7 = b$

令 $r_8 = r_5 r_6$

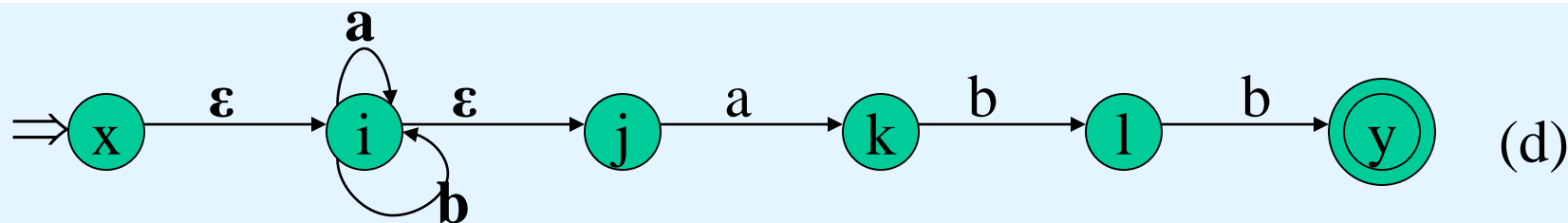
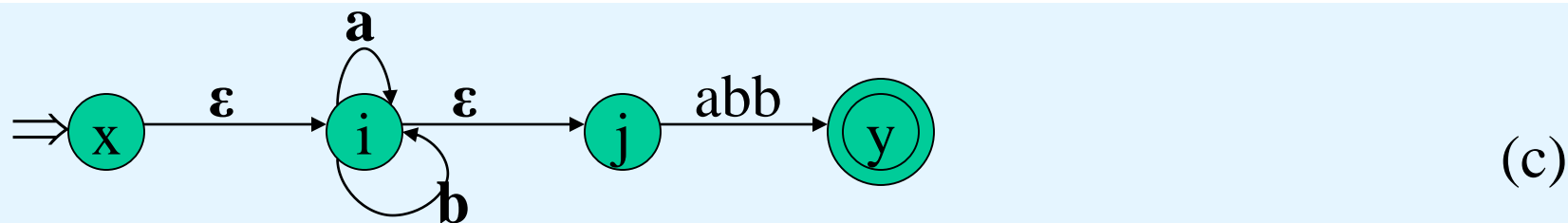
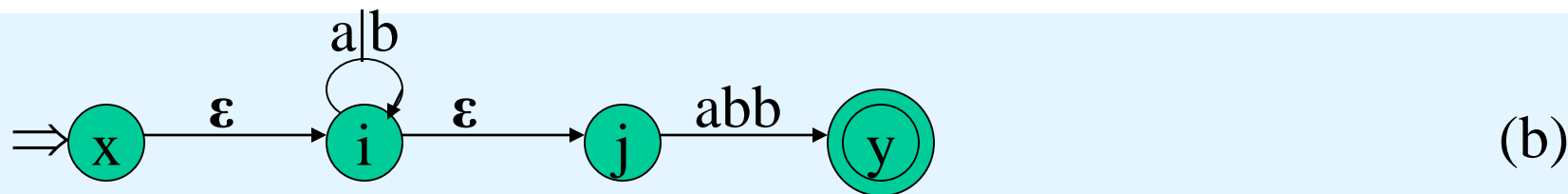
令 $r_9 = r_8 r_7$ 则有



令 $r_{10} = r_4 r_9$ 则最终得到NFA N:



分解R的方法有很多种,下面给出另一种分解方式和所构成的NFA

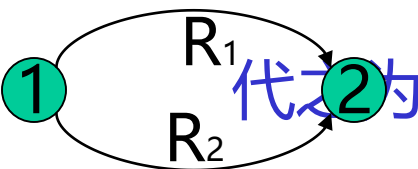
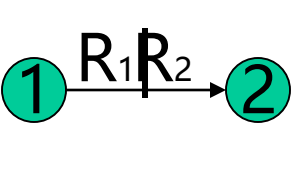


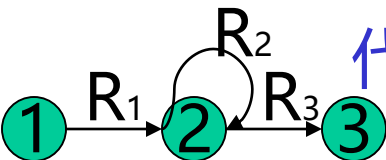
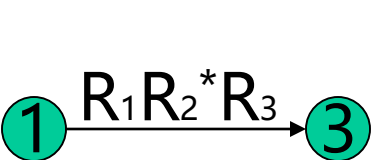
(4) 有穷自动机 \Rightarrow 正则式R

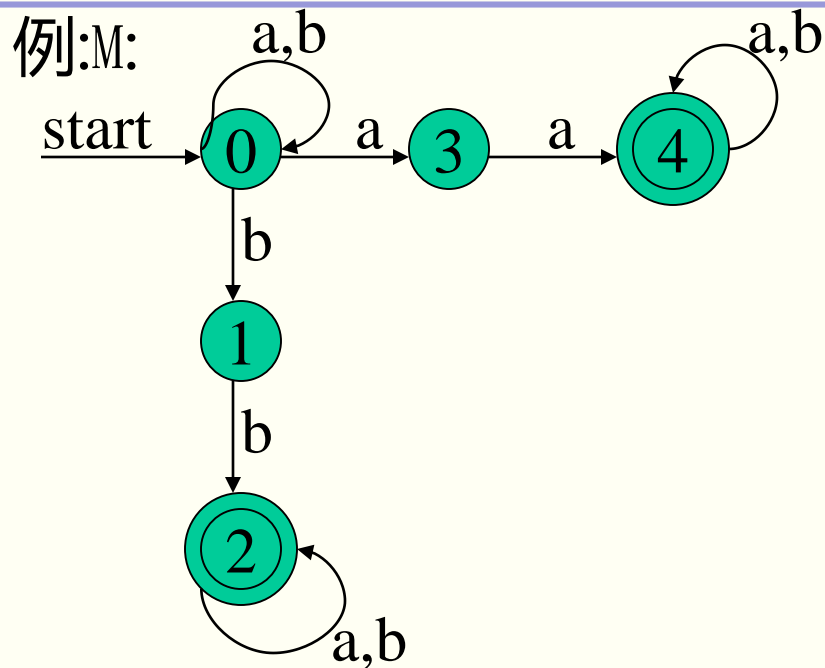
算法:

- 1) 在M上加两个结点x,y,从x结点用 ϵ 弧到M的所有初态,从M的所有终态用 ϵ 到y结点形成与M等价的M',M'只有一个初态x和一个终态y。
- 2) 逐步消去M'中的所有结点,直至剩下x和y结点,在消结过程中,逐步用正则式来记弧,其消结规则如下:

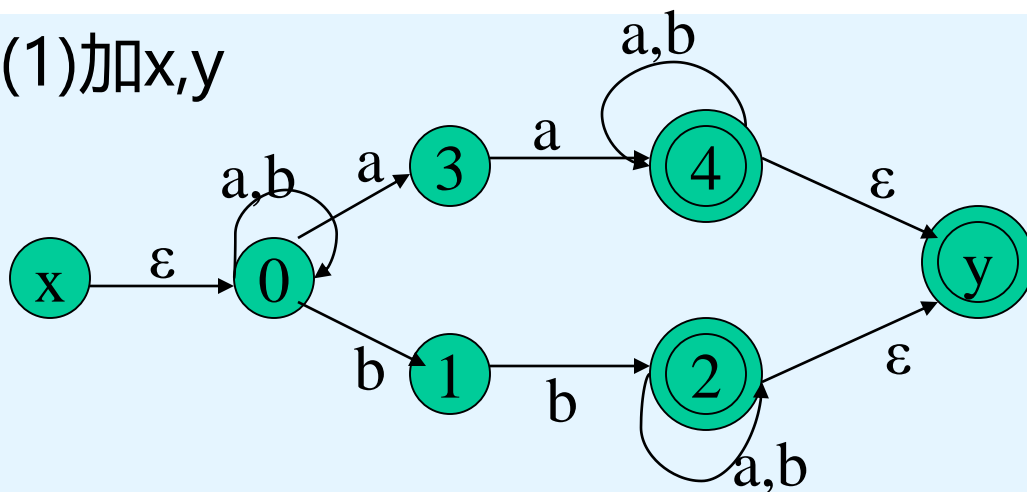
1. 对于  代之为 

2. 对于  代之为 

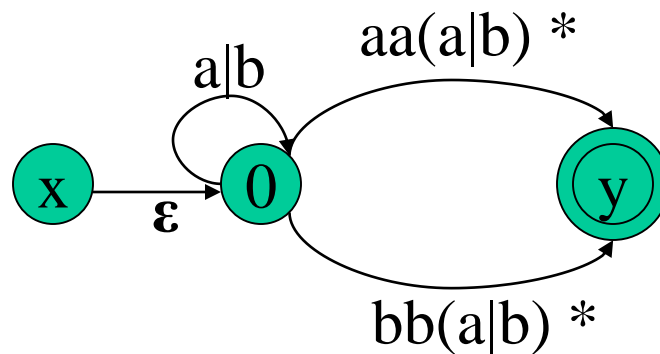
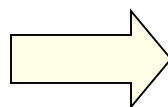
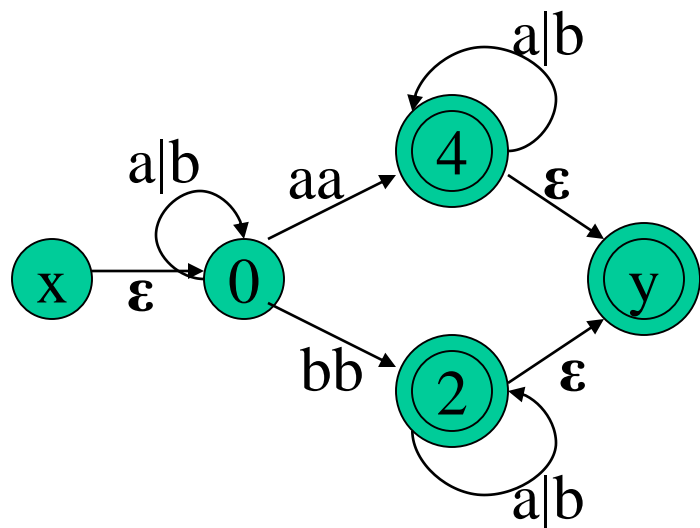
3. 对于  代之为 



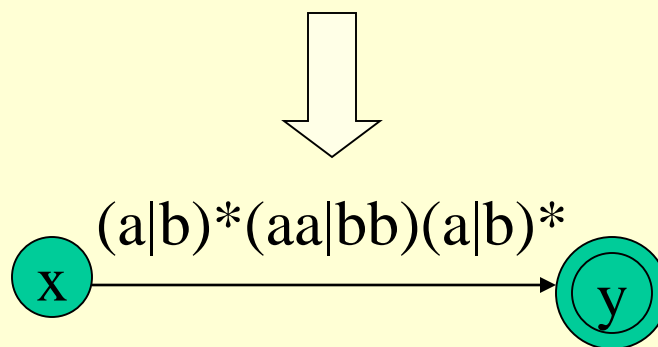
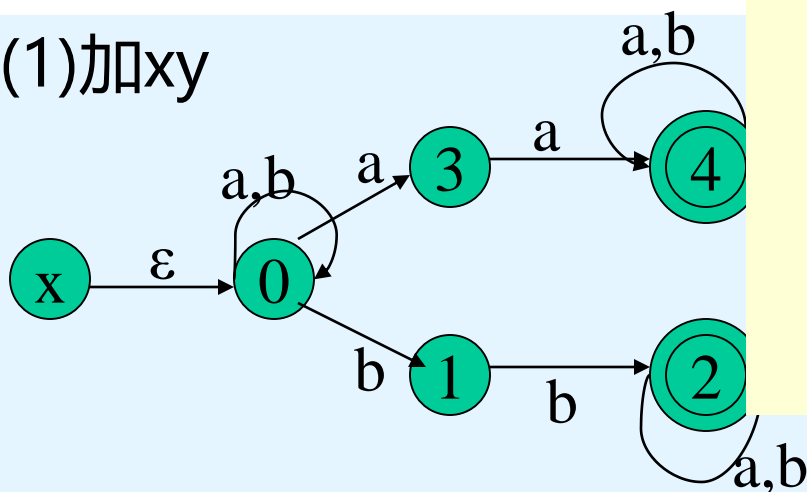
解: (1)加x,y



(2)消除M中的所有结点



解: (1)加xy



(5) 正则文法 \Rightarrow 正则式

利用以下转换规则,直至只剩下一个开始符号定义的产生式,并且产生式的右部不含非终结符。

规则	文法产生式	正则式
规则1	$A \rightarrow xB, B \rightarrow y$	$A = xy$
规则2	$A \rightarrow xA \mid y$	$A = x^*y$
规则3	$A \rightarrow x, A \rightarrow y$	$A = x \mid y$

例:有文法G[s]

$S \rightarrow aA|a,$

$A \rightarrow aA|dA|a|d$

规则

规则1

规则2

规则3

文法产生式

$A \rightarrow xB, B \rightarrow y$

$A \rightarrow xA | y$

$A \rightarrow x, A \rightarrow y$

正则式

$A=xy$

$A=x^*y$

$A=x | y$

于是: $S=aA|a$

$A=(aA|dA)|(a|d) \Rightarrow A=(a|d)A|(a|d)$

由规则二: $A=(a|d)^*(a|d)$

代入: $S=a(a|d)^*(a|d)|a$

于是: $S=a((a|d)^*(a|d)|\epsilon)$



(6) 正则式 \Rightarrow 正则文法

算法:

- 1) 对任何正则式 r ,选择一个非终结符 S 作为识别符号,并产生产生式 $S \rightarrow r$
- 2) 若 x, y 是正则式,对形为 $A \rightarrow xy$ 的产生式,重写为
 $A \rightarrow xB \quad B \rightarrow y$,其中 B 为新的非终结符, $B \in V_n$
 同样: 对于 $A \rightarrow x^*y \Rightarrow A \rightarrow xA \quad A \rightarrow y$
 $A \rightarrow x|y \Rightarrow A \rightarrow x \quad A \rightarrow y$

例:将 $R=a(a|d)^*$ 转换成相应的正则文法

解:1) $S \rightarrow a(a|d)^*$

2) $S \rightarrow aA$
 $A \rightarrow (a|d)^*$

3) $S \rightarrow aA$
 $A \rightarrow (a|d)A$
 $A \rightarrow \epsilon$

4) $S \rightarrow aA$
 $A \rightarrow aA|dA$
 $A \rightarrow \epsilon$



补充: DFA的简化(最小化)

“对于任一个DFA, 存在一个唯一的状态最少的等价的DFA”

一个有穷自动机是化简的 \Leftrightarrow 它没有多余状态并且它的状态中没有两个是互相等价的。

一个有穷自动机可以通过消除多余状态和合并等价状态而转换成一个最小的与之等价的有穷自动机

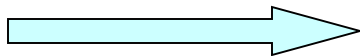
定义:

(1) **有穷自动机的多余状态:** 从该自动机的开始状态出发,任何输入串也不能到达那个状态

例:

	0	1
S0	S1	S5
S1	S2	S7
S2	S2	S5
S3	S5	S7
S4	S5	S6
S5	S3	S1
S6	S8	S0
S7	S0	S1
S8	S3	S6

画状态图可以看出
S₄, S₆, S₈
为不可达状态
应该消除



	0	1
S0	S1	S5
S1	S2	S7
S2	S2	S5
S3	S5	S7
S5	S3	S1
S7	S0	S1

(2)等价状态 \iff 状态s和t的等价条件是:

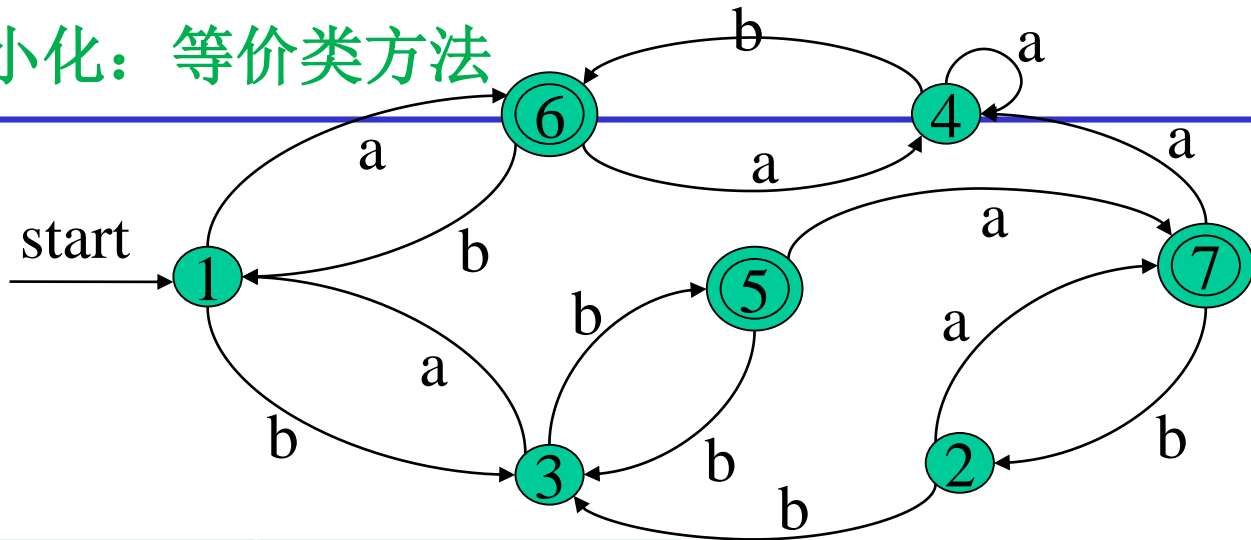
- 1)一致性条件: 状态s和t必须同时为可接受状态或不接受状态。
- 2)蔓延性条件: 对于所有输入符号,状态s和t必须转换到等价的状态里。

对于所有输入符号c, $I_c(s)=I_c(t)$, 即状态s、t对于c具有相同的后继, 则称s, t是等价的。

(任何有后继的状态和任何无后继的状态一定不等价)

有穷自动机的状态s和t不等价,称这两个状态是可区别的。

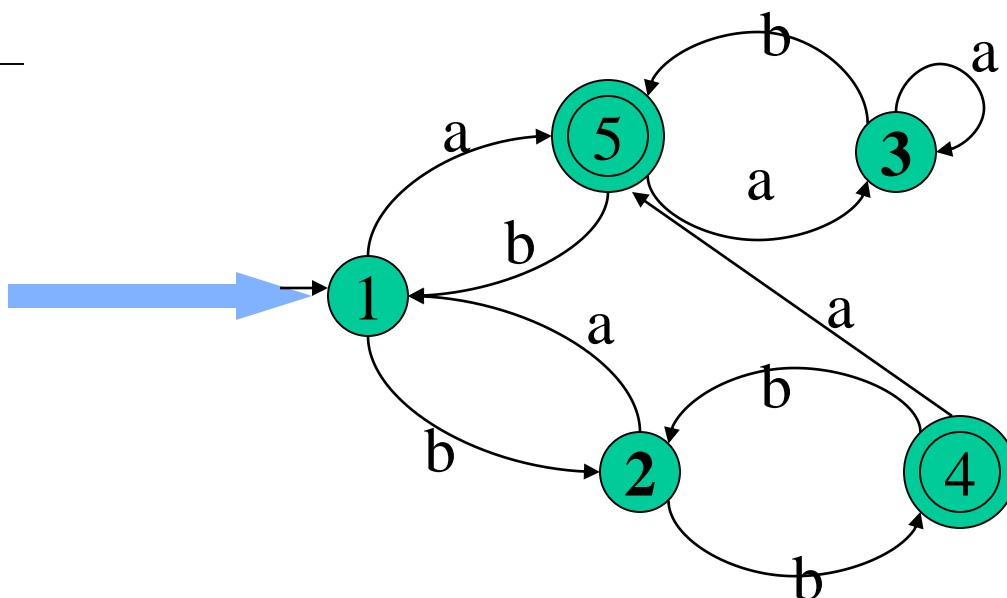
- 非终态：1,2,3,4
- 终态：5,6,7



等价类	a	b
1,2	6,7	3,3
1,3	6,1	
1,4	6,4	
2,3	7,1	
2,4	7,4	
3,4	1,4	5,6
5,6	7,4	
5,7	7,4	
6,7	4,4	1,2

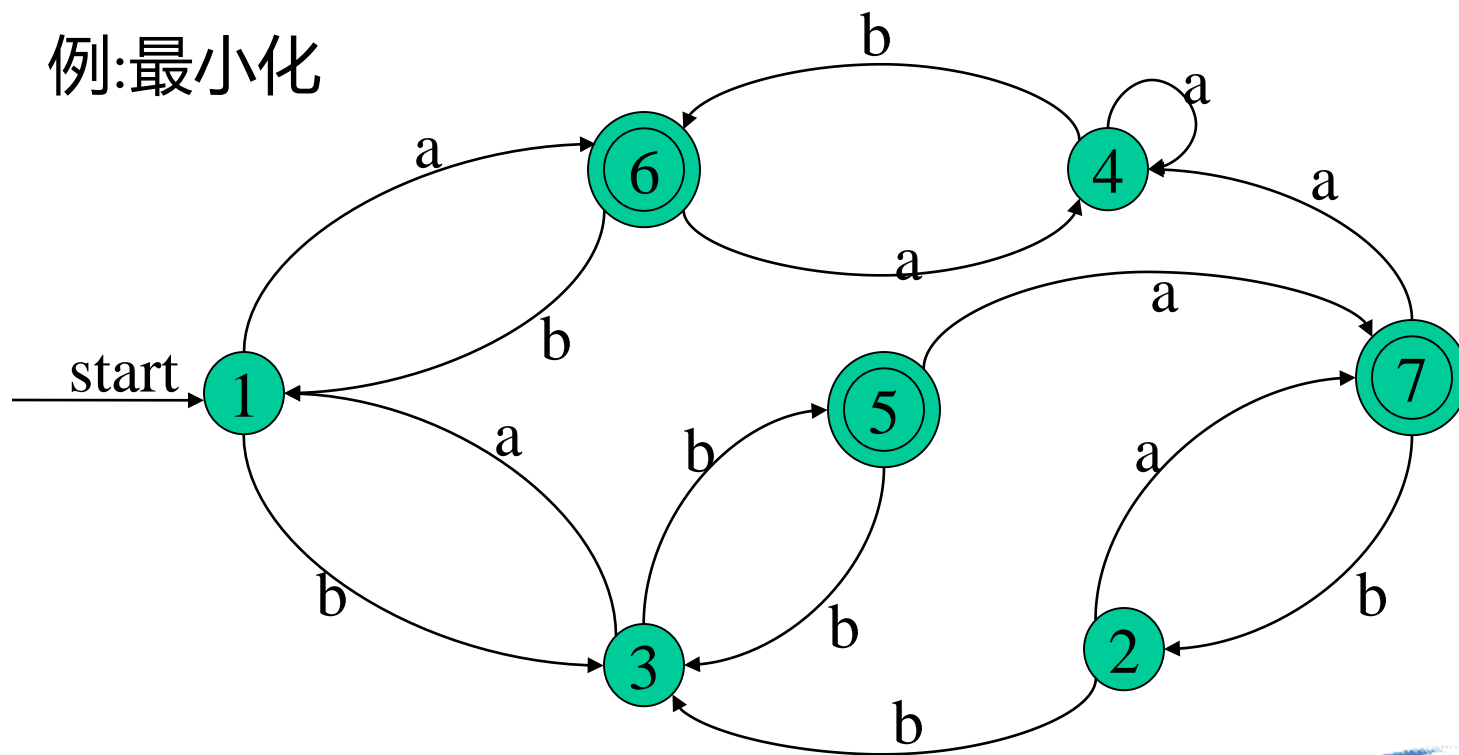
对状态等价类重新编号得到:

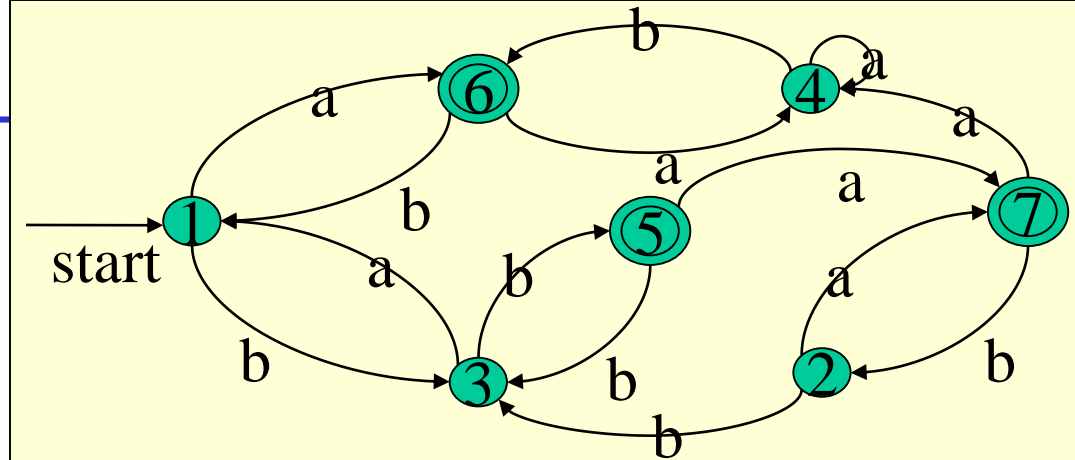
		a	b
<1,2>	1	5	2
<3>	2	1	4
<4>	3	3	5
<5>	4	5	2
<6,7>	5	3	1



“分割法”：把一个DFA(不含多余状态)的状态分割成一些不相关的子集，使得任何不同的两个子集状态都是可区分的，而同一个子集中的任何状态都是等价的。

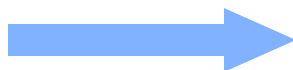
例:最小化



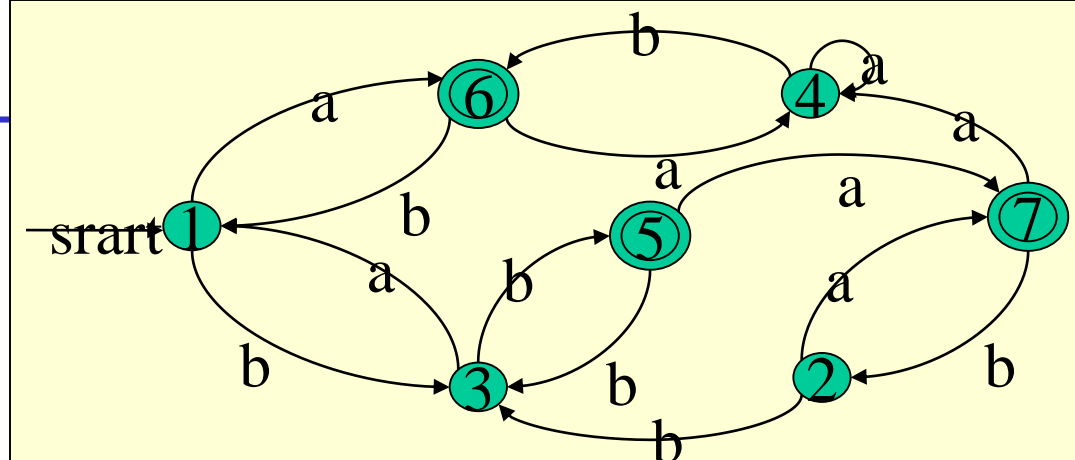


解: (一)区分终态与非终态
区号

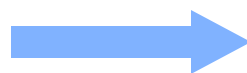
	a	b	区号
1	6	3	1
2	7	3	
3	1	5	
4	4	6	
5	7	3	2
6	4	1	
7	4	2	



	a	b	区号
1	6	3	1
2	7	3	
3	1	5	2
4	4	6	
5	7	3	3
6	4	1	
7	4	2	



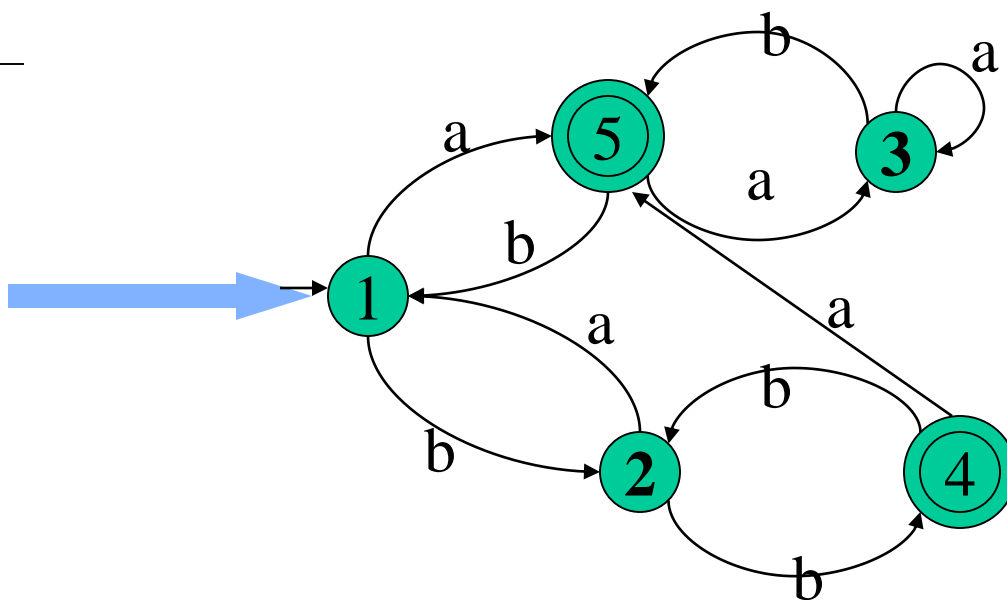
	a	b	区号
1	6	3	1
2	7	3	
3	1	5	2
4	4	6	
5	7	3	4
6	4	1	
7	4	2	5



	a	b	区号
1	6	3	1
2	7	3	
3	1	5	2
4	4	6	
5	7	3	4
6	4	1	
7	4	2	5

将区号代替状态号得:

	a	b
1	5	2
2	1	4
3	3	5
4	5	2
5	3	1

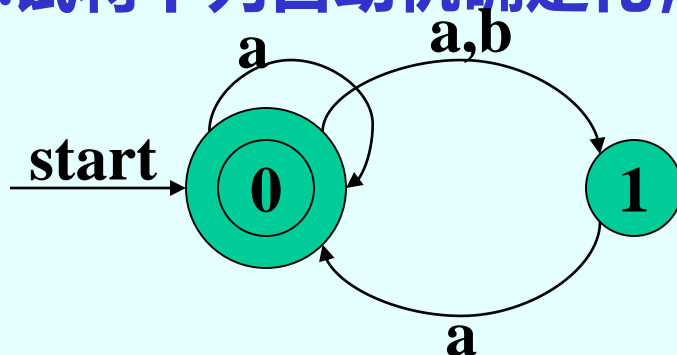




补充习题

1. 有正则表达式 $1(0|1)^*101$, 构造DFA

2. 试将下列自动机确定化, 并最小化:



判断题:

1. 对任意一个右线性文法 G ，都存在一个NFA M ，满足 $L(G) = L(M)$ ()
2. 对任意一个右线性文法 G ，都存在一个DFA M ，满足 $L(G) = L(M)$ ()
3. 对任何正则表达式 e ，都存在一个NFA M ，满足 $L(M) = L(e)$ ()
4. 对任何正则表达式 e ，都存在一个DFA M ，满足 $L(M) = L(e)$ ()