第二部分

编译自动化技术

- 词法分析的自动化
- 语法分析的自动化
- 其它自动化技术
  - 编译优化

第三章:词法分析

3.1 词法分析的功能

－ －识别单词，返回单词的类别和值

3.2 词法分析程序的设计与实现

－ －状态图：有穷自动机的非形式表示

－ －－正则文法和状态图

3.3 词法分析程序的自动生成

－ －正则表达式和有穷自动机

八月十五日夜湓亭望月

白居易

昔年八月十五夜曲
江池畔杏园边今年
八月十五夜湓浦沙
头水馆前西北望乡
何处是东南见月几
回圆昨风一吹无人
会今夜清光似往年

八月十五日夜湓亭望月

白居易

昔年八月十五夜，曲江池畔杏园边。

今年八月十五夜，湓浦沙头水馆前。

西北望乡何处是，东南见月几回圆。

昨风一吹无人会，今夜清光似往年。

八月十五日夜湓亭望月

白居易

昔年八月十五夜，曲江池畔杏园边。

今年八月十五夜，湓浦沙头水馆前。

西北望乡何处是，东南见月几回圆。

昨风一吹无人会，今夜清光似往年。

词法分析： 断词，并给出词性（分类）

语法分析： 断句，并给出句的结构、分类

程序

int main( ) ‘\n’ { int count=read( ); ‘\n’
//if number of entries read is greater than 1 ‘\n’ //then sort( ) and compact( ) ‘\n’ if (count > 1) { sort( ); compact( ); } ‘\n’ if (count ==0) ‘\n’ count << “no sales for this month\n”; ‘\n’ else write( ); ‘\n’ return; ‘\n’ }

程序

```
int main( )
{  int count=read( );
   //if number of entries read is greater than 1
   //then sort( ) and compact( )
   if (count > 1) {  sort( ); compact( ); }
   if (count ==0)
      count << "no sales for this month\n";
   else write( );
   return;
}
```

内 容

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 |V1

U ::=Z1 |1

V ::=Z0 | 0

左线性文法。该文法所定义的语言为：

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

例：正则文法

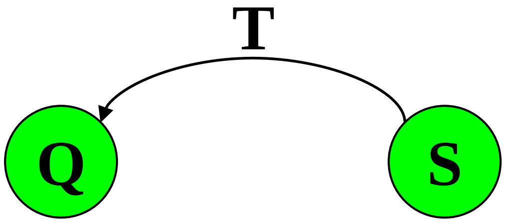
Z::= U0 |V1

U ::=Z1 |1

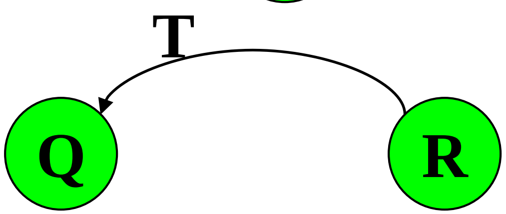
V ::=Z0 | 0

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 ∈Vn,T ∈Vt, 则：



4. 若Q::=RT, Q、R∈Vn,T ∈Vt, 则：



5. 按自动机方法，可加上开始状态和终止状态标志。

1. 每个非终结符设一个状态；
2. 设一个开始状态S；
3. 若Q::=T，Q ∈Vn,T ∈Vt,
4. 若Q::=RT, Q、R∈Vn,T ∈Vt,
5. 加上开始状态和终止状态标志

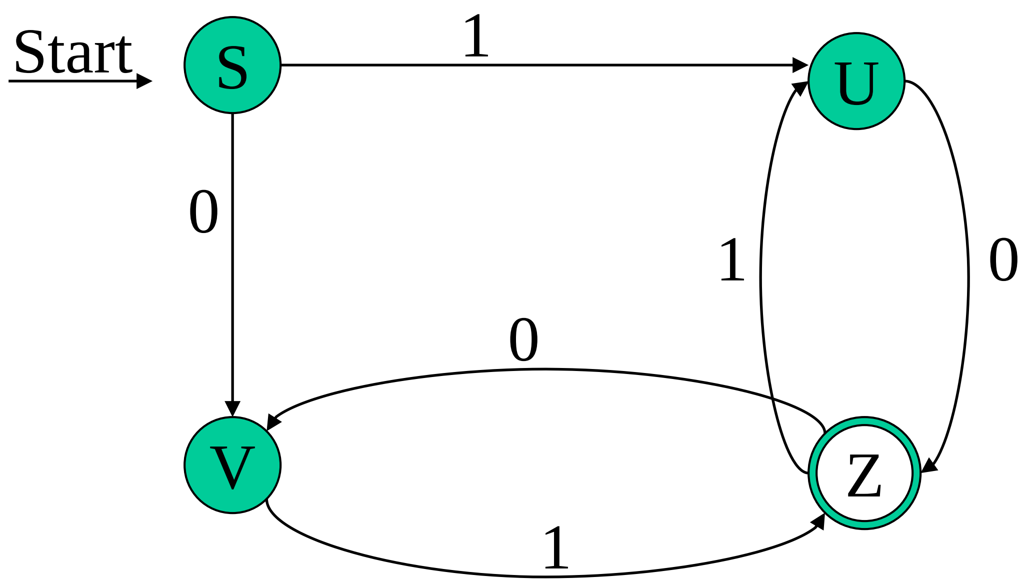
例如：正则文法

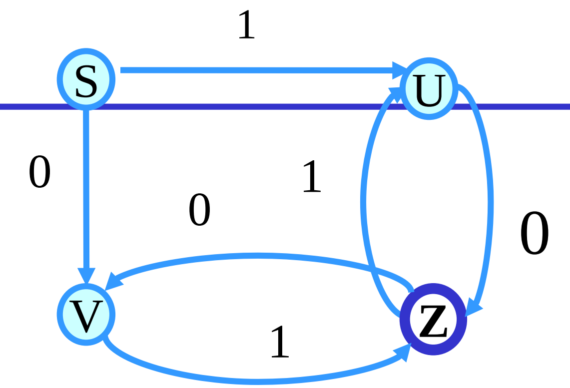
Z::= U0 |V1

U ::=Z1 |1

V ::=Z0 | 0

其状态图为：





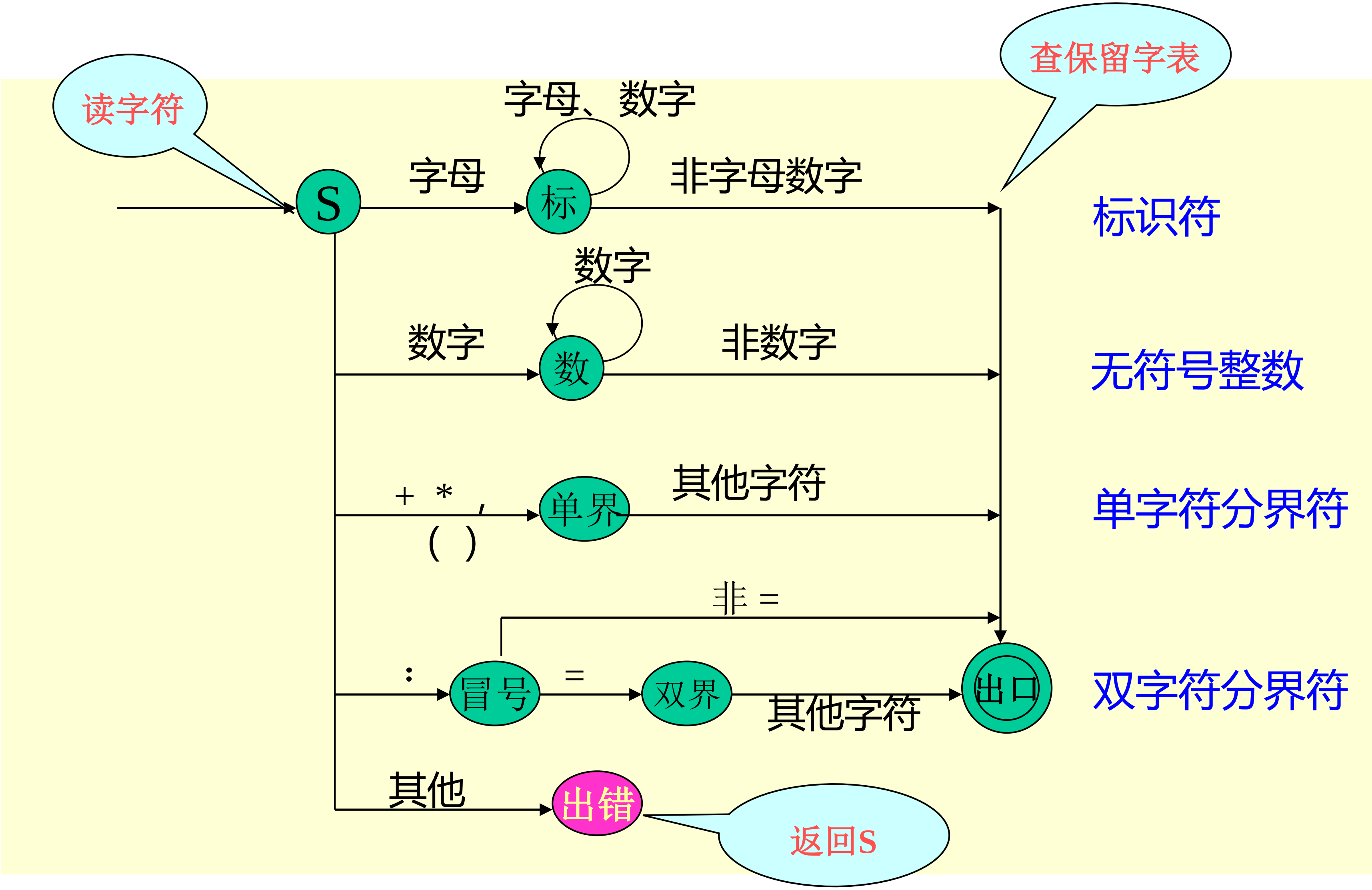
• 识别算法

利用状态图可按如下步骤分析和识别字符串x：

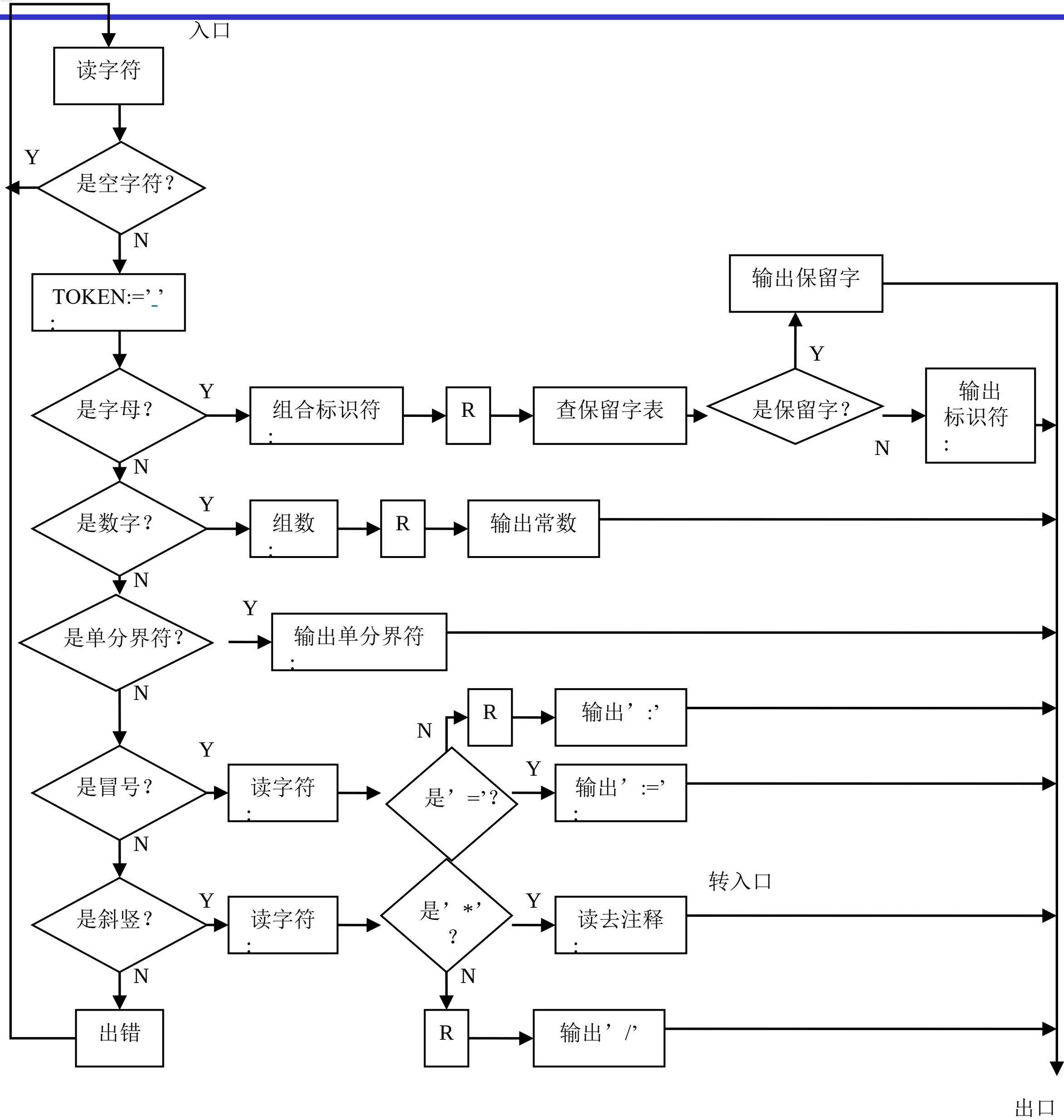
1、置初始状态为当前状态，从x的最左字符开始，重复步骤2，直到x右端为止。

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。

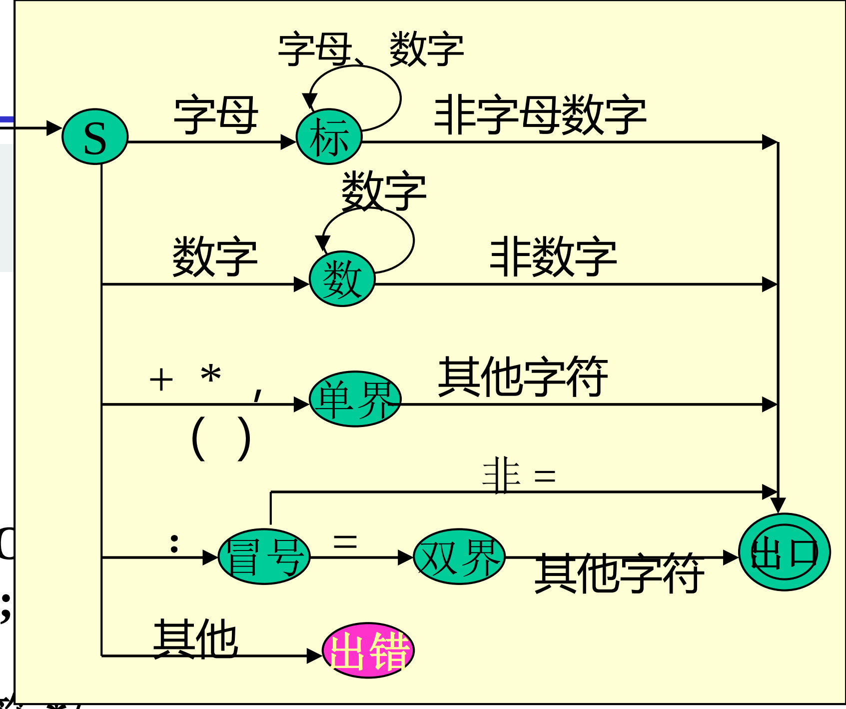
例：x=0110 和1011


读字符
查保留字表
字母、数字
字母
非字母数字
S
标
标识符
数字
数字
非数字
数
无符号整数
+ * ,
( )
单界
其他字符
单字符分界符
非 =
:
冒号
=
双界
其他字符
出口
双字符分界符
其他
出错
返回S

见教材


入口
读字符
是空字符?
Y
N
TOKEN:=' ' ;
是字母?
组合标识符 ;
R
查保留字表
是保留字?
输出保留字
输出标识符 ;
是数字?
组数 ;
输出常数
是单分界符?
输出单分界符 ;
是冒号?
读字符 ;
是' ='?
输出' :'
输出' :=' ;
是斜竖?
是' *' ?
读去注释 ;
转入口
输出' /'
出错
出口

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',-) ;
```

内 容

11.1 正则表达式

11.1.1 正则表达式和正则集合的递归定义

有字母表∑, 定义在∑ 上的正则表达式和正则集合递归定义如下:

1. ε和ϕ都是∑ 上的正则表达式, 其正则集合分别为:{ε}和ϕ;
2. 任何a∈ ∑ , a是∑ 上的正则表达式,其正则集合为:{a};
3. 假定U和V是 ∑ 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•V和U^*也都是∑ 上的正则表达式, 其正则集合分别为L(U) ∪L(V)、 L(U) • L(V)和$L(U)^*$;
4. 任何∑ 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

与集合的闭包运算有区别
这里a^*表示由任意个a组成的串，
而$\{a,b\}^* = \{\varepsilon, a, b, aa, ab, ba, bb, \ldots\}$

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⇔ 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 Σ = { 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e1, e2和e3均是某字母表上的正则表达式, 则有:

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

交换律:　$e1 \mid e2 = e2 \mid e1$

结合律:　$e1|(e2|e3) = (e1|e2)|e3$

$e1(e2e3) = (e1e2)e3$

分配律:　$e1(e2|e3) = e1e2|e1e3$

$(e1|e2)e3 = e1e3|e2e3$

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

$(r|s)^* = (r^*s^*)^*$

证明：$A^{+}=AA^{*}=A^{*}A$

证：∵ $A^{*}=A^{0}\cup A^{+}$，$A^{+}=A^{1}\cup A^{2}\cup\ldots\cup A^{n}\cup\ldots$

得：$A^{*}=A^{0}\cup A^{1}\cup A^{2}\cup\ldots\cup A^{n}\cup\ldots$

∴ $AA^{*}=A\ (A^{0}\cup A^{1}\cup A^{2}\cup\ldots\cup A^{n}\cup\ldots)$

$=\ AA^{0}\cup AA^{1}\cup AA^{2}\cup\ldots\cup A\ \ A^{n}\cup\ldots$

$=A\cup A^{2}\cup A^{3}\cup A^{n+1}\cup\ldots$

$=\ A^{+}$

同理可得：$A^{*}A\ =(A^{0}\cup A^{1}\cup A^{2}\cup\ldots\cup A^{n}\cup\ldots)A$

$=A^{0}\ A\cup A^{1}A\cup A^{2}A\cup\ldots\cup A^{N}A\cup\ldots$

$=\ A\cup A^{2}\cup A^{3}\cup A^{N+1}\cup\ldots$

$=\ A^{+}$

因此：$A^{+}=AA^{*}=A^{*}A$

正则表达式与3型文法等价

例如：

正则表达式：	ba*	a(a|b)*
3型文法：	Z ::= Za|b	Z::=Za|Zb|a

例：

3型文法	正则表达式	
S::= aS|aB	aS|aba*a →	a*aba*a
B::= bC	ba*a	
C::= aC|a	a*a	

11.2.1 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)

   $S\times\Sigma\rightarrow S$

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

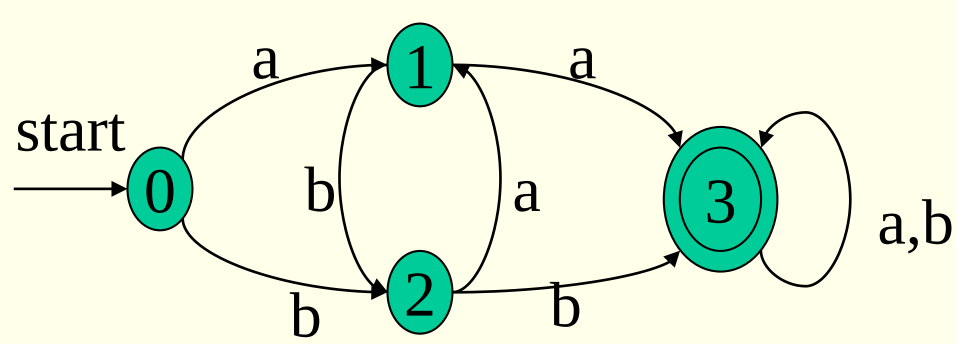
状态 \ 输入字符	a	b
0	1	2
1	3	2
2	1	3
3	3	3

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:

状态 \ 输入字符	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 $\alpha = a_1a_2 \cdots a_n$， $\alpha \in \Sigma^*$，若$\delta(\delta(\cdots\delta(s_0, a_1), a_2)\cdots), a_{n-1}), a_n) = s_n$，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。

$$\delta(s_0, a_1) = s_1$$
$$\delta(s_1, a_2) = s_2$$
$$\cdots\cdots\cdots$$
$$\delta(s_{n-2}, a_{n-1}) = s_{n-1}$$
$$\delta(s_{n-1}, a_n) = s_n$$

换言之：若存在一条初始状态到某一终止状态的路径，且这条路径上所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： $L(M) = \{\alpha \mid \delta(s_0, \alpha) = s_n, s_n \in Z\}$

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

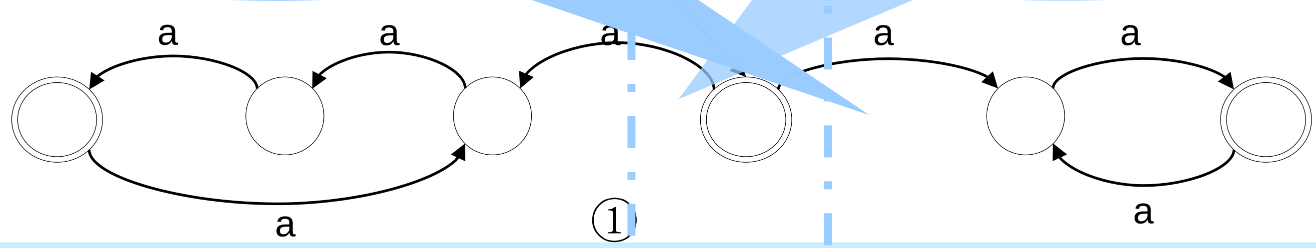
若δ是一个多值函数，且输入可允许为ε，则有穷自动机是不确定的，即在某个状态下，对于某个输入字符存在多个后继状态。

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。

在始态，输入字母a时，自动机既可以向

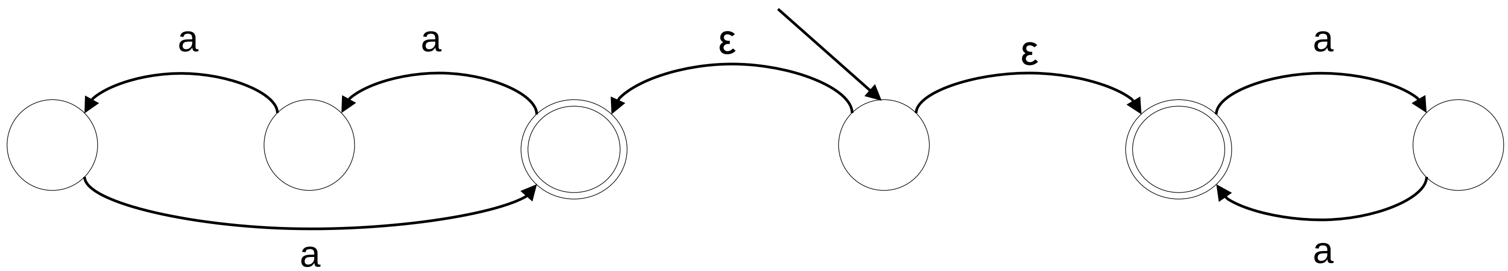
如果向右，则可接受由a组成长度为偶数的字符串。

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：



②

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, Σ∪{ε}, δ, s_0, Z)

其中 S—有穷状态集

Σ∪{ε}—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 $S \times \Sigma \cup \{\varepsilon\} \rightarrow 2^S$

(2^S --S的幂集—S的子集构成的集合)

NFA M'所接受的语言为:

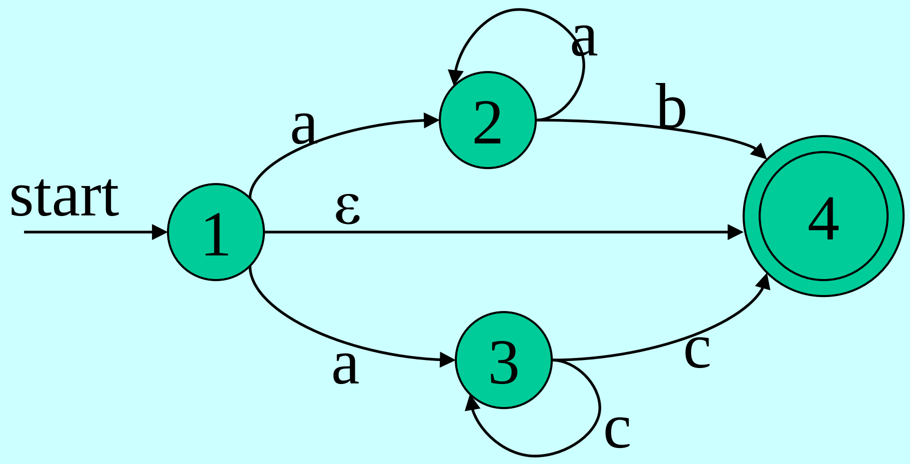
$$L(M')=\{\alpha|\delta(S_0,\alpha)=S' \quad S'\cap Z\neq\Phi\}$$

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

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|ε

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。

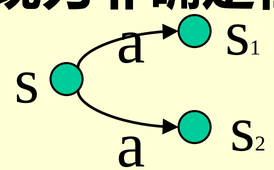
有一个正则表达式则对应一个正则集合。

若V是正则集合，iff V=L（M）

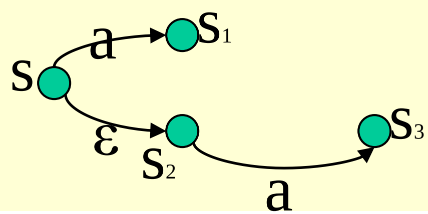
即一个正则表达式对应一个DFA M

2. NFA的定义,δ非单值函数, 且有ε弧, 表现为非确定性

如： $\delta(s,a)=\{s_1,s_2\}$



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



11.2.3 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。但在多数计算机硬件并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：

$$\text{NFA}\ \ M' \xrightarrow{\text{构造成一个}} \text{DFA}\ \ \ M$$

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∈I，则s∈ε-closure（I）；

2）若s∈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。

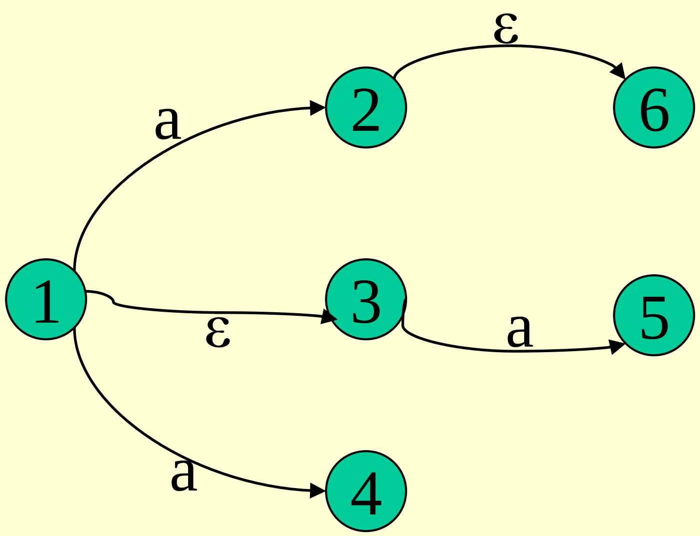
可以通过一例子来说明状态子集的ε-闭包的构造方法

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，$a \in \Sigma$

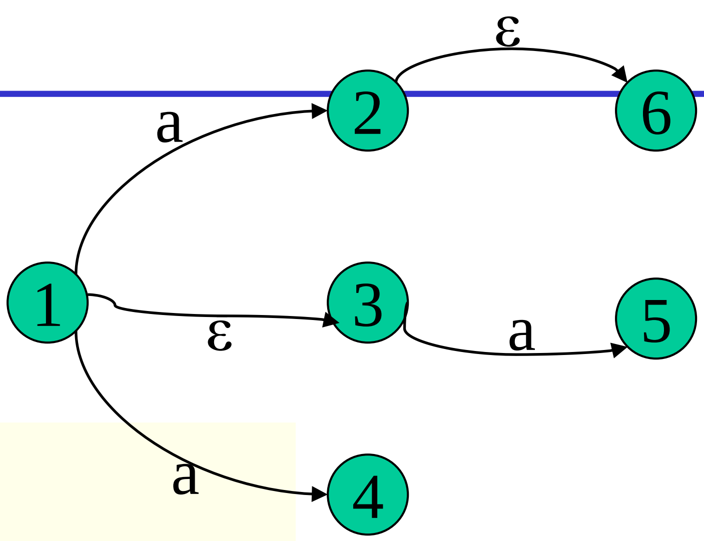
定义：$I_a = \varepsilon\text{-closure}(J)$

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 =ε-closure(J)

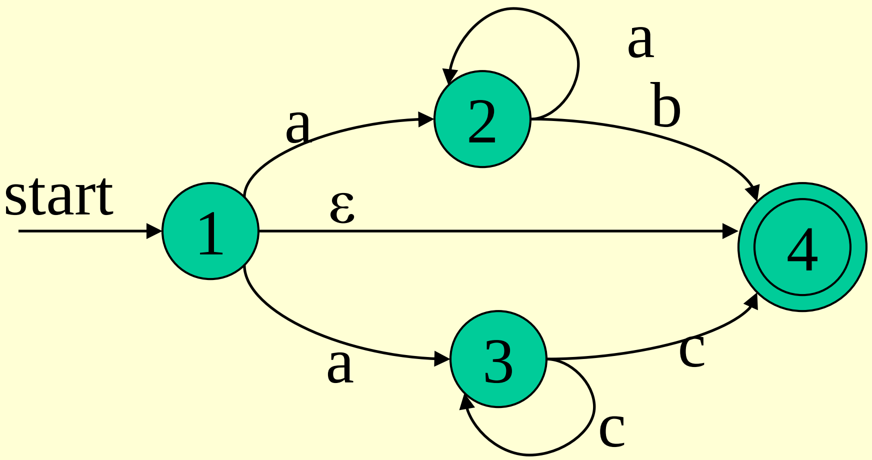
=ε-closure(δ（1，a））

=ε-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'



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

$I_a=\varepsilon\text{-closure}(\delta(1,a)\cup\delta(4,a))$

$=\varepsilon\text{-closure}(\{2,3\}\cup\varphi)$

$=\varepsilon\text{-closure}(\{2,3\})$

$=\{2,3\}$

$I_b=\varepsilon\text{-closure}(\delta(1,b)\cup\delta(4,b))$

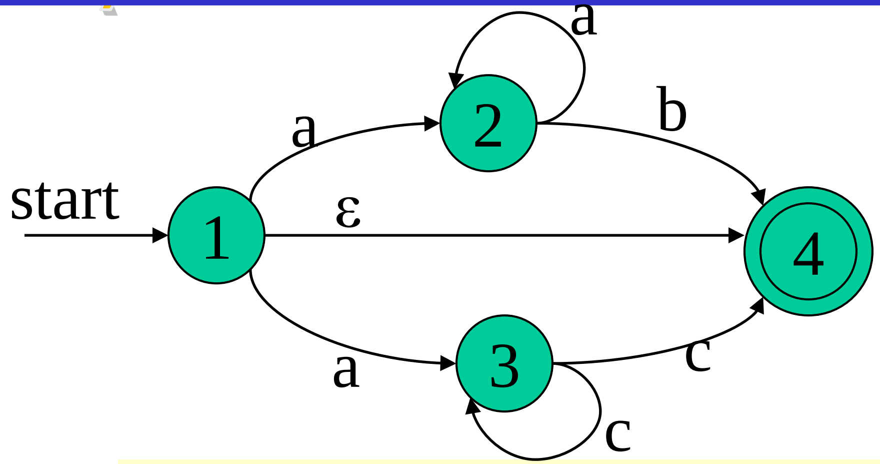
$=\varepsilon\text{-closure}(\varphi)$

$=\varphi$

$I_c=\varepsilon\text{-closure}(\delta(1,c)\cup\delta(4,c))$

$=\varphi$

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



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}

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}

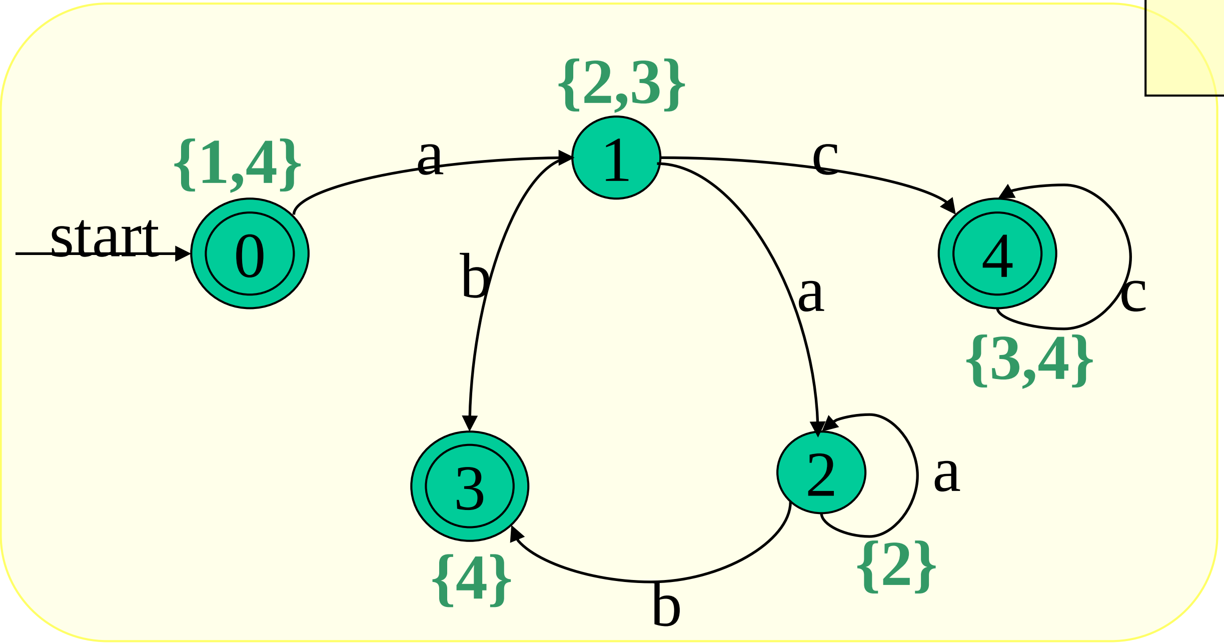
将求得的状态转换矩阵重新编号

DFA M状态转换矩阵：

符号 / 状态	a	b	c
0	1	–	–
1	2	3	4
2	2	3	–
3	–	–	–
4	–	–	4

状态 \ 符号	a	b	c
0	1	−	−
1	2	3	4
2	2	3	−
3	−	−	−
4	−	−	4

DFA M的状态图：



★ **注意：原初始状态的ε-closure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。

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')**

构造出来的DFA M唯一吗？

11.2.4 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 ⇔ 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
S_0	S_1	S_5
S_1	S_2	S_7
S_2	S_2	S_5
S_3	S_5	S_7
S_4	S_5	S_6
S_5	S_3	S_1
S_6	S_8	S_0
S_7	S_0	S_1
S_8	S_3	S_6

画状态图可以看出S_4,S_6,S_8为不可达状态应该消除

	0	1
S_0	S_1	S_5
S_1	S_2	S_7
S_2	S_2	S_5
S_3	S_5	S_7
S_5	S_3	S_1
S_7	S_0	S_1

(2)等价状态⟺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。

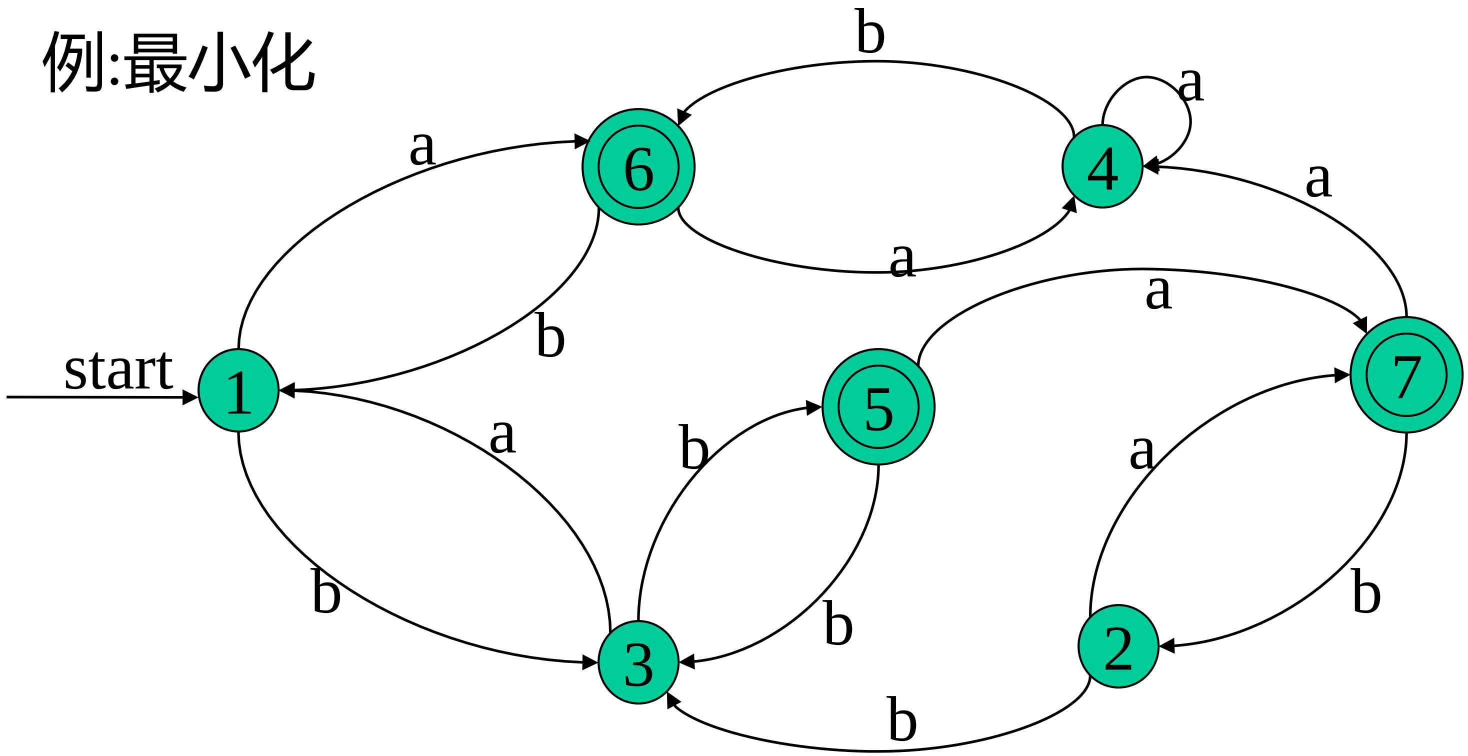
对于所有输入符号c，$I_c(s)=I_c(t)$，即状态s、t对于c具有相同的后继，则称s，t是等价的。

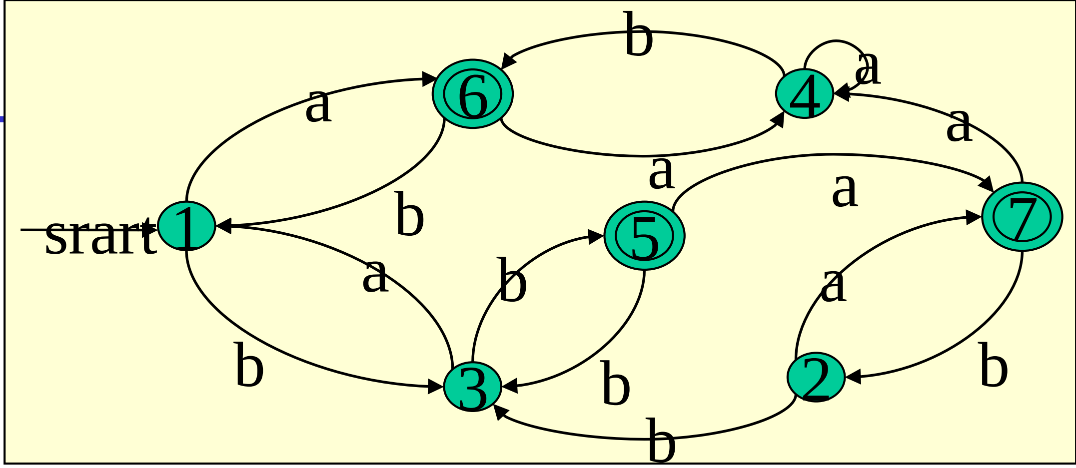
（任何有后继的状态和任何无后继的状态一定不等价）

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。

“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别的，而同一个子集中的任何状态都是等价的。

例:最小化



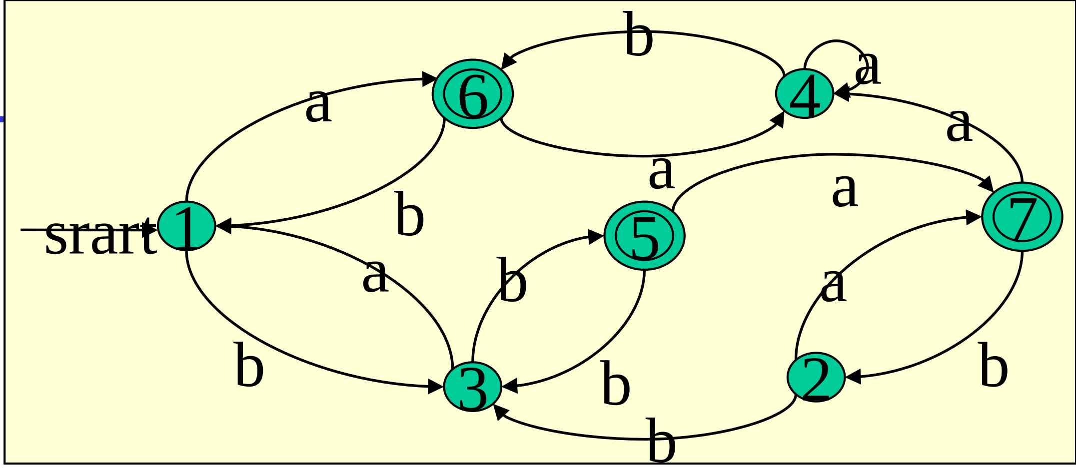


解:

(一)区分终态与非终态

	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	1
3	1	5	2
4	4	6	3
5	7	3	4
6	4	1	4
7	4	2	4

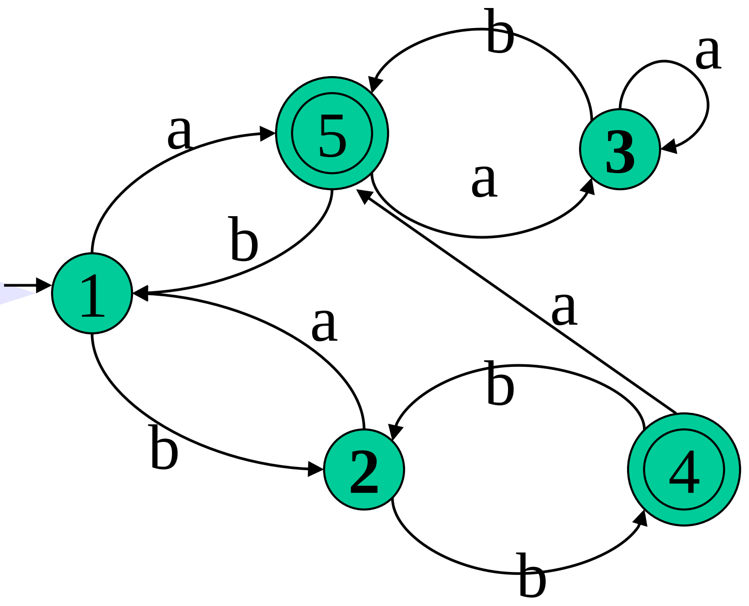
	a	b	区号
1	6	3	1
2	7	3	1
3	1	5	2
4	4	6	3
5	7	3	4
6	4	1	5
7	4	2	5

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

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

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



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’)**

下面：

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.

11.2.5 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 ⇔ DFA M
V=L(R) L(M)=L(R)

所以 正则表达式R ⇒ NFA M' ⇒ DFA M

$$L(R) = L(M') = L(M)$$

证明: 根据定义。

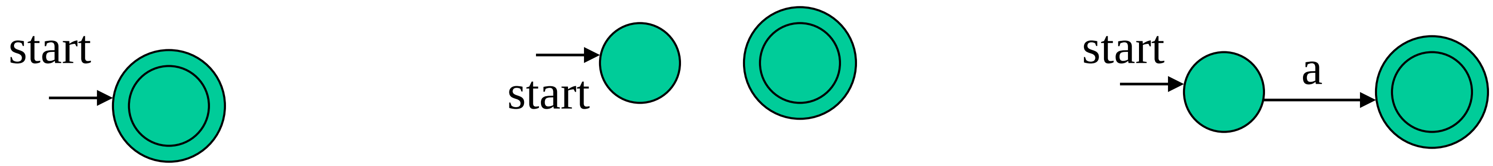
复习

正则表达式和正则集合的递归定义

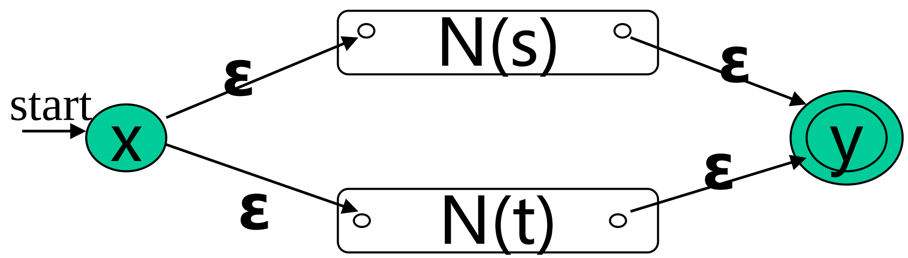
有字母表Σ, 定义在Σ 上的正则表达式和正则集合递归定义如下:

1. ε和φ都是Σ 上的正则表达式, 其正则集合分别为:{ε}和φ;
2. 任何a∊ Σ , 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•V和U^*也都是Σ 上的正则表达式, 其正则集合分别为L(U) ∪L(V)、 L(U) • 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。

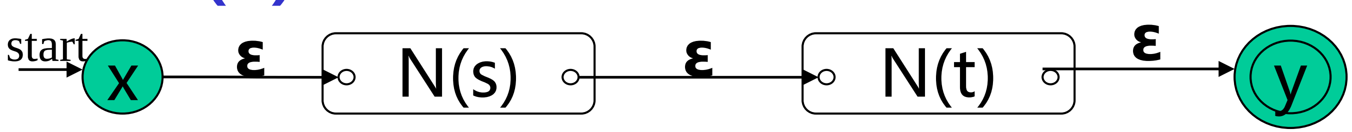
正则表达式	ε	φ	a
正则集合	{ε}	φ	{a}



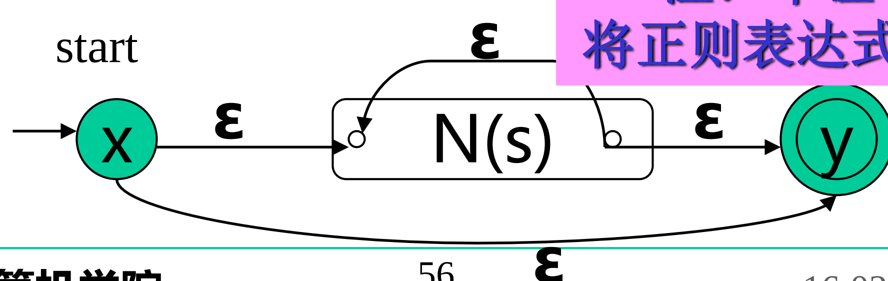
R=s|t NFA(R):



R=st NFA(R):



R=s* NFA(R):

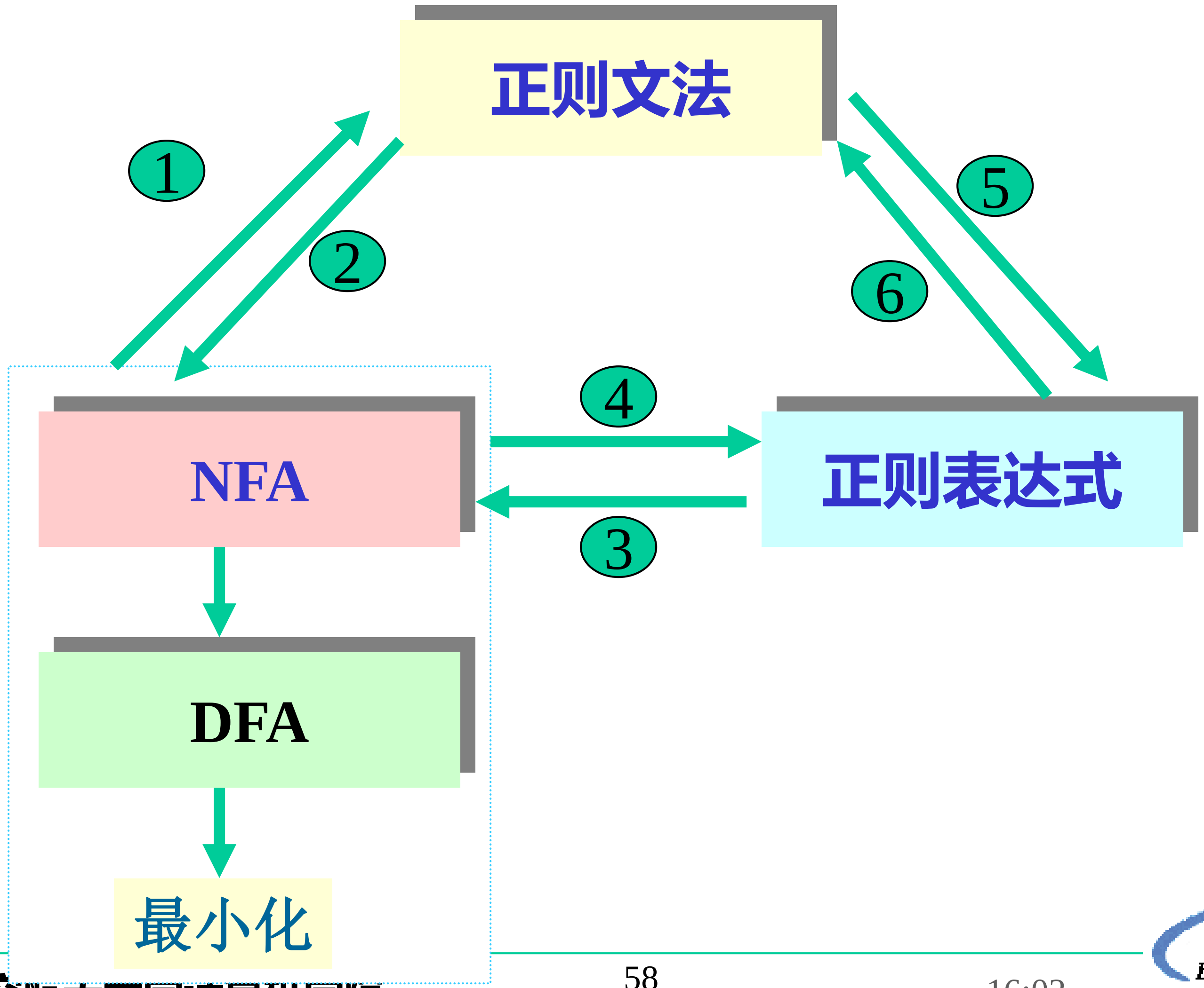


注：本证明过程也给出了将正则表达式转换为NFA的算法

判断题：

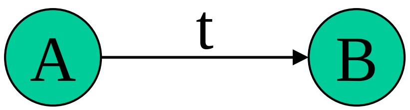
1. 对任意一个右线性文法G，都存在一个NFA M，满足L(G) = L(M)　( )
2. 对任意一个右线性文法G，都存在一个DFA M，满足L(G) = L(M)　( )
3. 对任何正则表达式e，都存在一个NFA M，满足L(M) = L(e)　( )
4. 对任何正则表达式e，都存在一个DFA M，满足L(M) = L(e)　( )

补充



（1）有穷自动机⇒正则文法

算法:

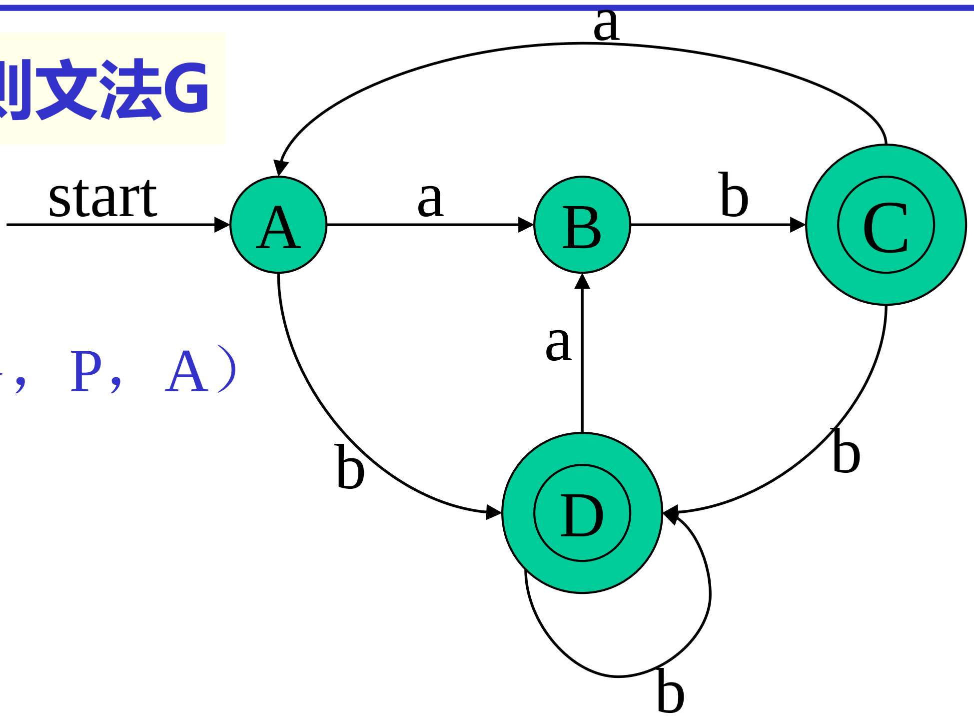


1.对转换函数f(A,t)=B，可写成一个产生式:A→tB

2.对可接受状态Z,增加一个产生式:Z→ε

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：$A \rightarrow aB$

$A \rightarrow bD$

$B \rightarrow bC$

$C \rightarrow aA$

$C \rightarrow bD$

$C \rightarrow \varepsilon$

$D \rightarrow aB$

$D \rightarrow bD$

$D \rightarrow \varepsilon$

1.对转换函数f(A,t)=B，可写成一个产生式:A→tB

2.对可接受状态Z,增加一个产生式:Z→ε

(2) 正则文法⇒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→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→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→aA|bB|ε

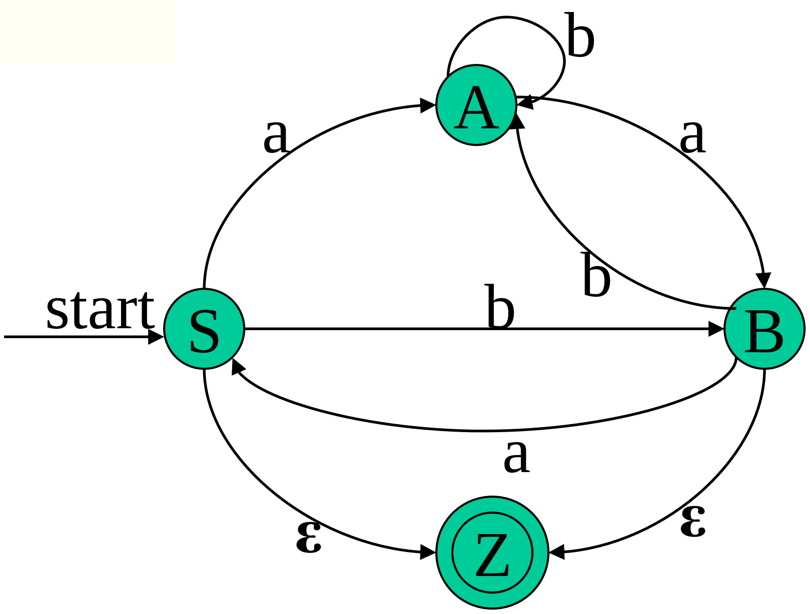
A→aB|bA

B→aS|bA|ε

4.对G中的形如A→tB,
构造M的一个转换函数f(A,t)=B;

5.对G中的形如A→t的产生式,
构造M的一个转换函数f(A,t)=Z。

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

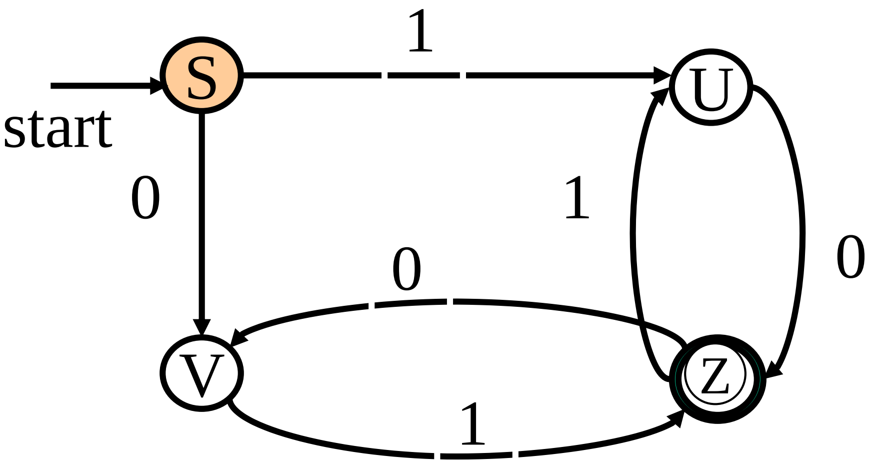
$Z \rightarrow U0 \mid V1$

$U \rightarrow Z1 \mid 1$

$V \rightarrow Z0 \mid 0$

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

其中 $B= \{01,10\}$



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

右线性正则文法例

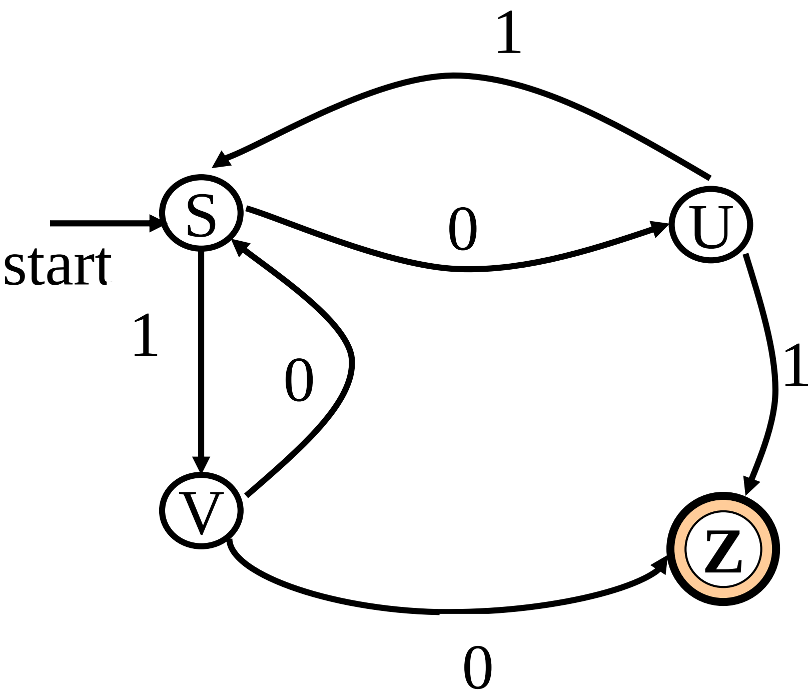
$S \rightarrow 0U \mid 1V$

$U \rightarrow 1S \mid 1$

$V \rightarrow 0S \mid 0$

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

其中 $B = \{ 01, 10 \}$

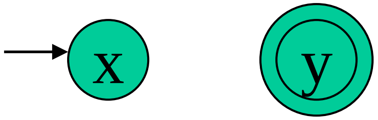


$R = (01|10)(01|10)^*$

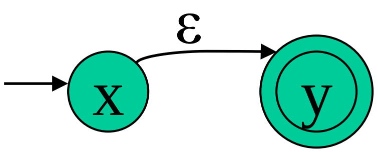
（3）正则式⇒有穷自动机

语法制导方法

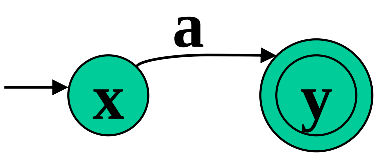
1.(a)对于正则式φ,所构造NFA:



(b)对于正则式ε,所构造NFA:

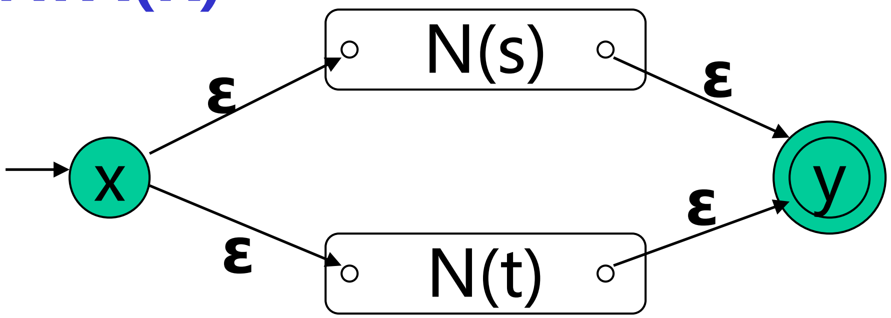


(c)对于正则式a,a∈Σ,则 NFA:

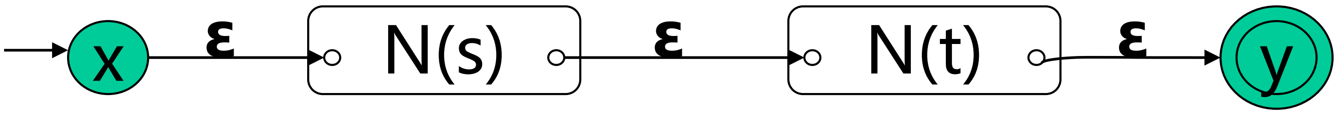


2.若s,t为Σ上的正则式,相应的NFA分别为N(s)和N(t);

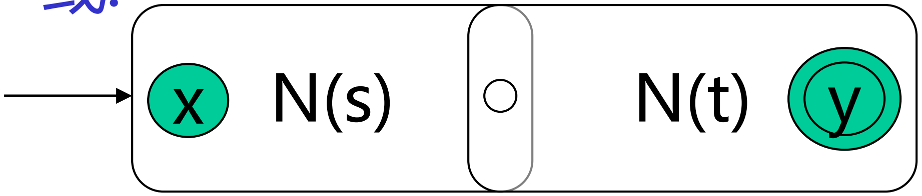
(a)对于正则式R=s|t,　NFA(R)



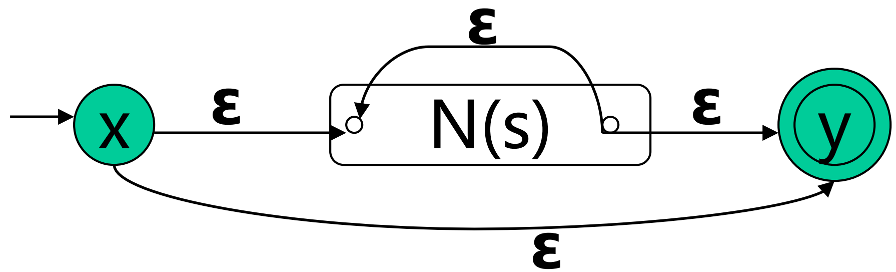
(b)对正则式R=st,　NFA(R)



或:



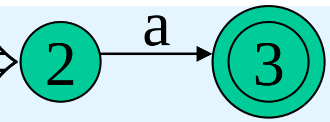
(c)对于正则式R=s*， NFA(R)



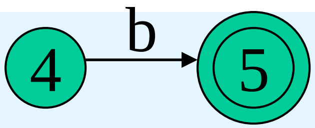
(d)对R=(s),与R=s的NFA一样.

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⇒

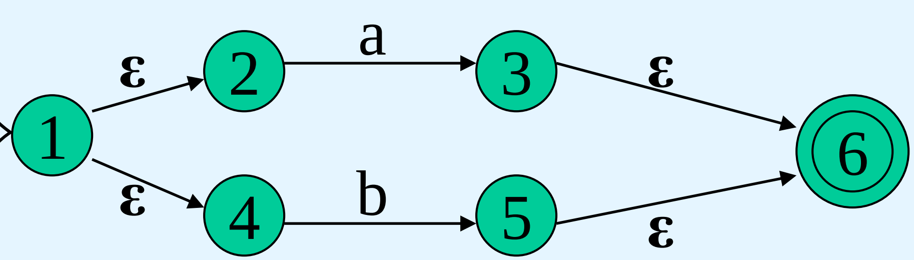


令r_2=b,则有⇒



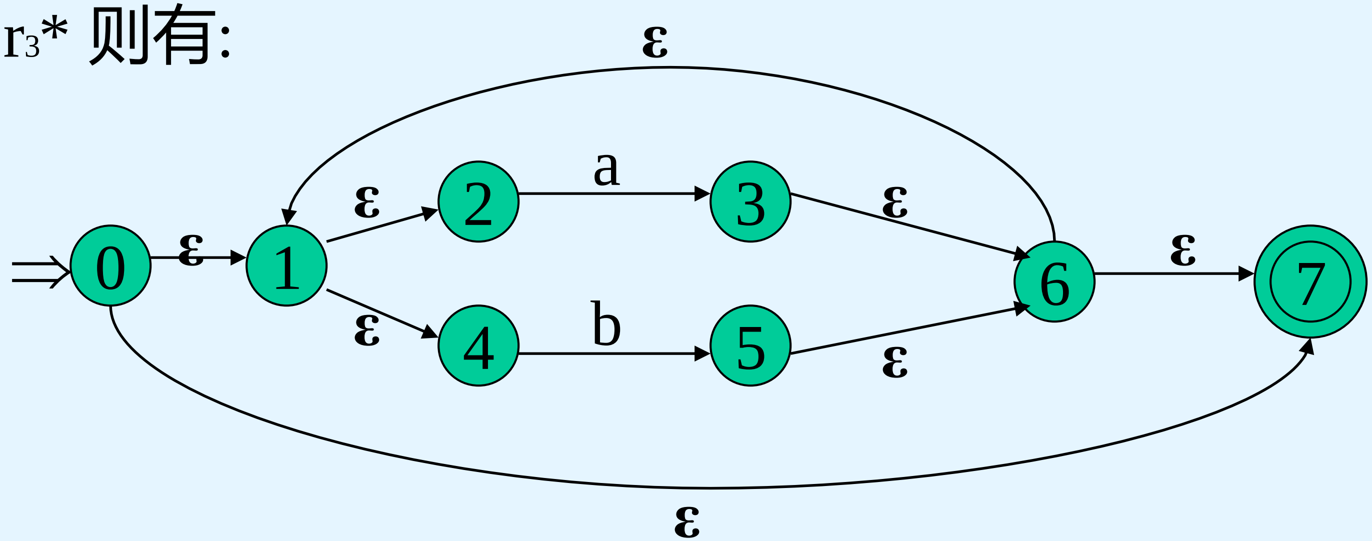
令r_3= r_1|r_2,则有

⇒

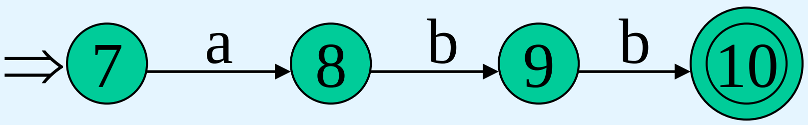


$R=(a|b)^*abb$

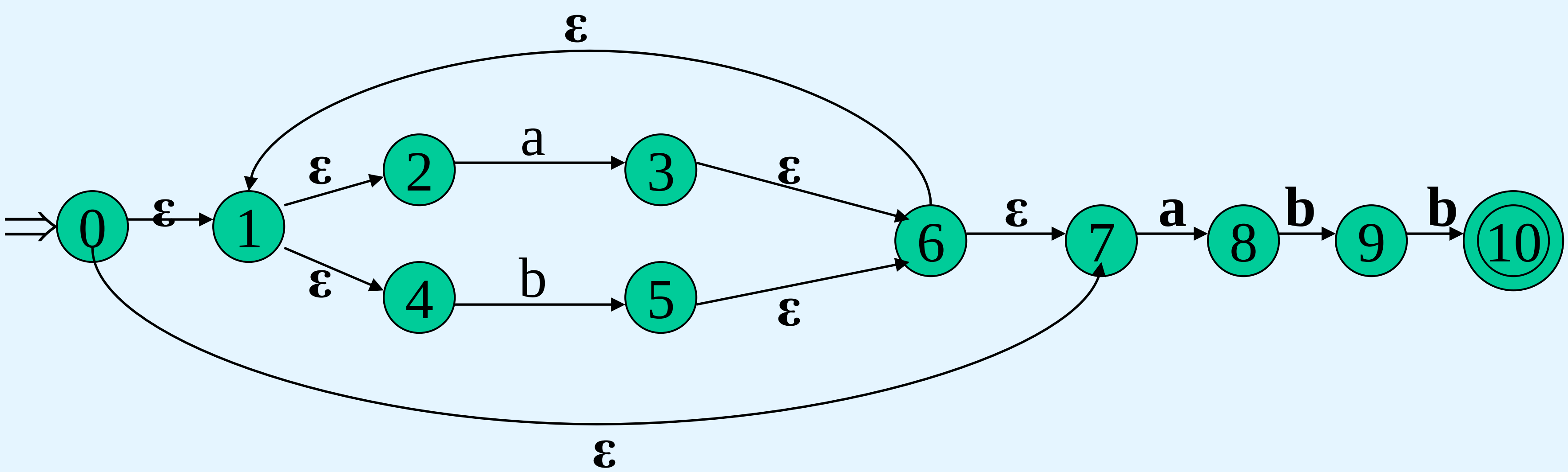
令$r_4= r_3^*$ 则有:



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

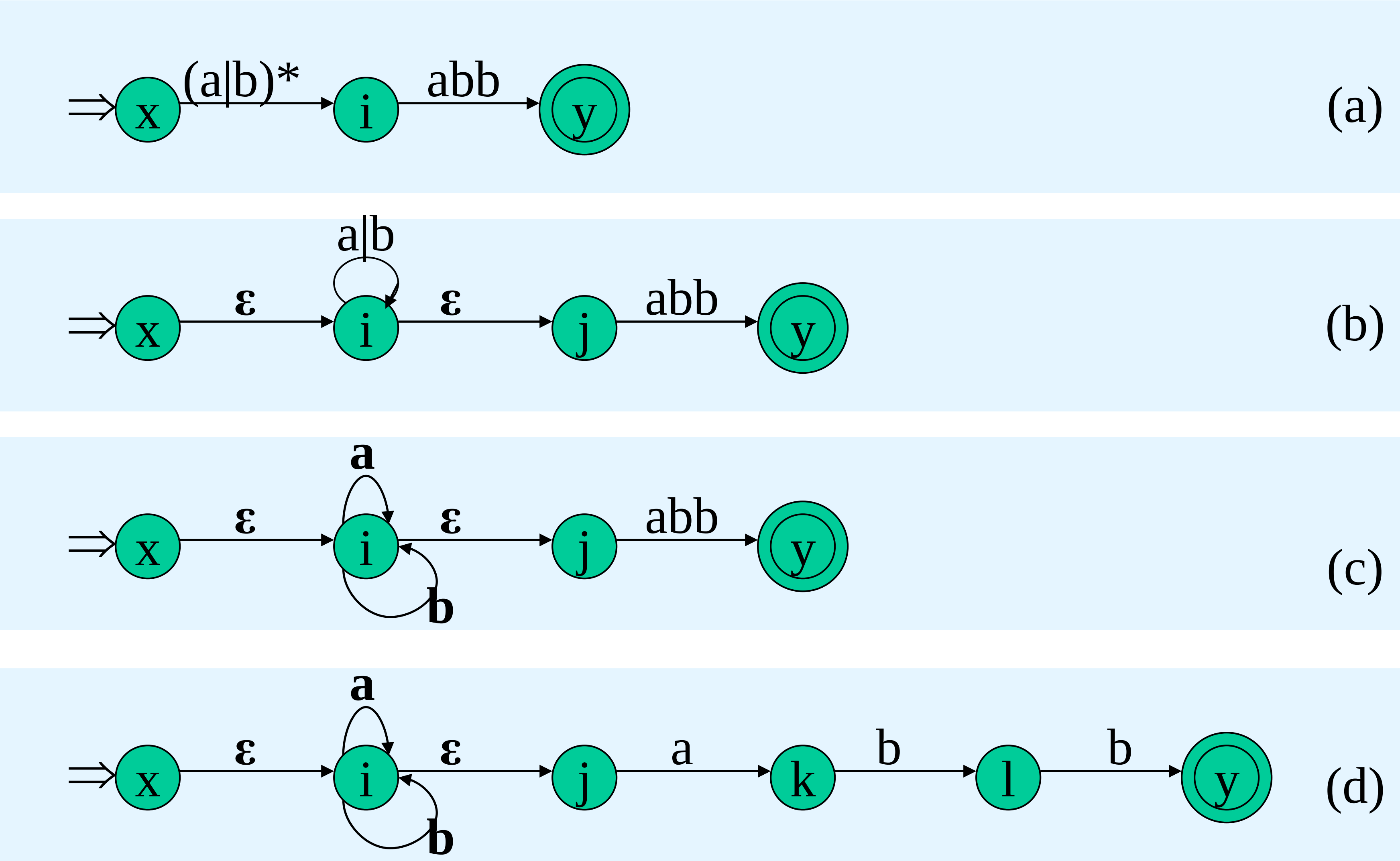


令$r_{10}= r_4 r_9$ 则最终得到NFA N:



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

R=(a|b)*abb

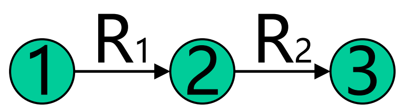


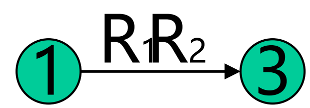
(4) 有穷自动机⇒正则式R

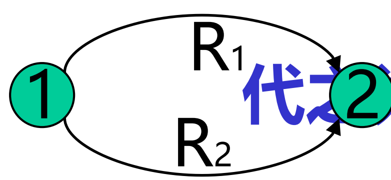
算法:

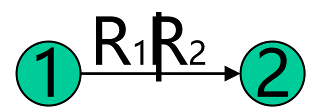
1)在M上加两个结点x,y,从x结点用ε弧到M的所有初态,从M的所有终态用ε到y结点形成与M等价的M’,M’只有一个初态x和一个终态y。

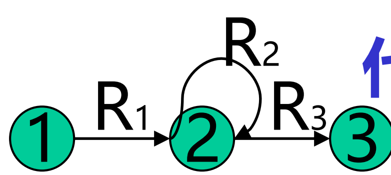
2)逐步消去M’中的所有结点,直至剩下x和y结点,在消结过程中,逐步用正则式来记弧,其消结规则如下:

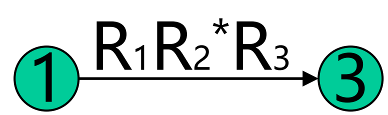
1.对于 

代之为 

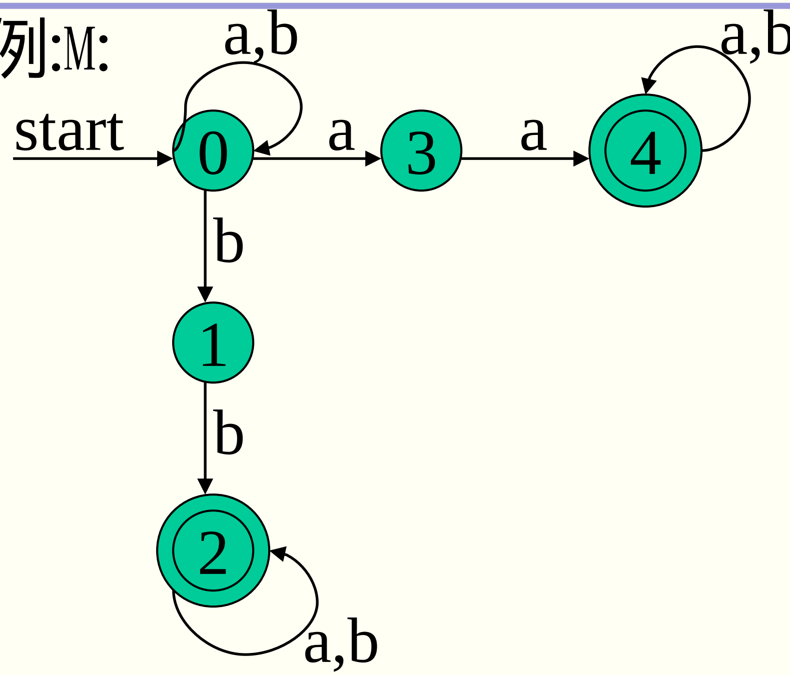
2.对于 

代之为 

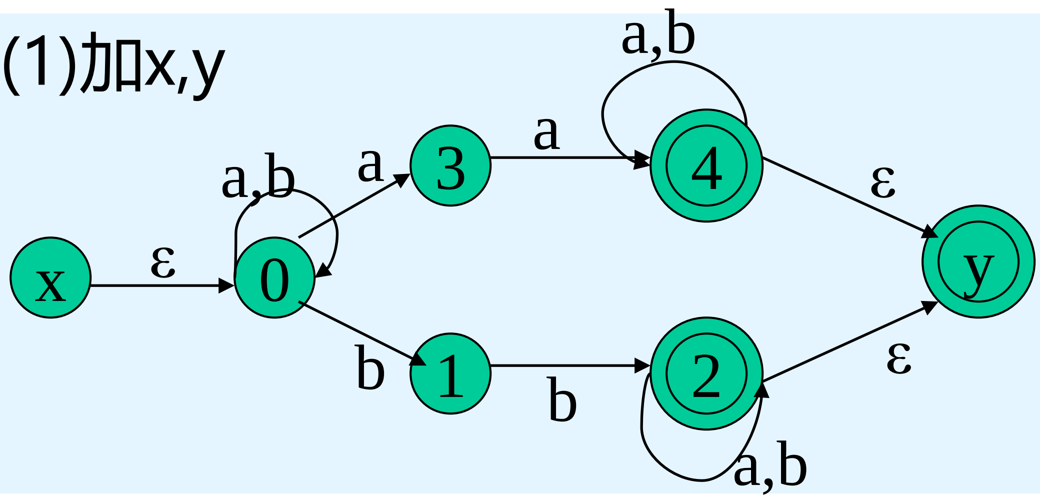
3.对于 

代之为 

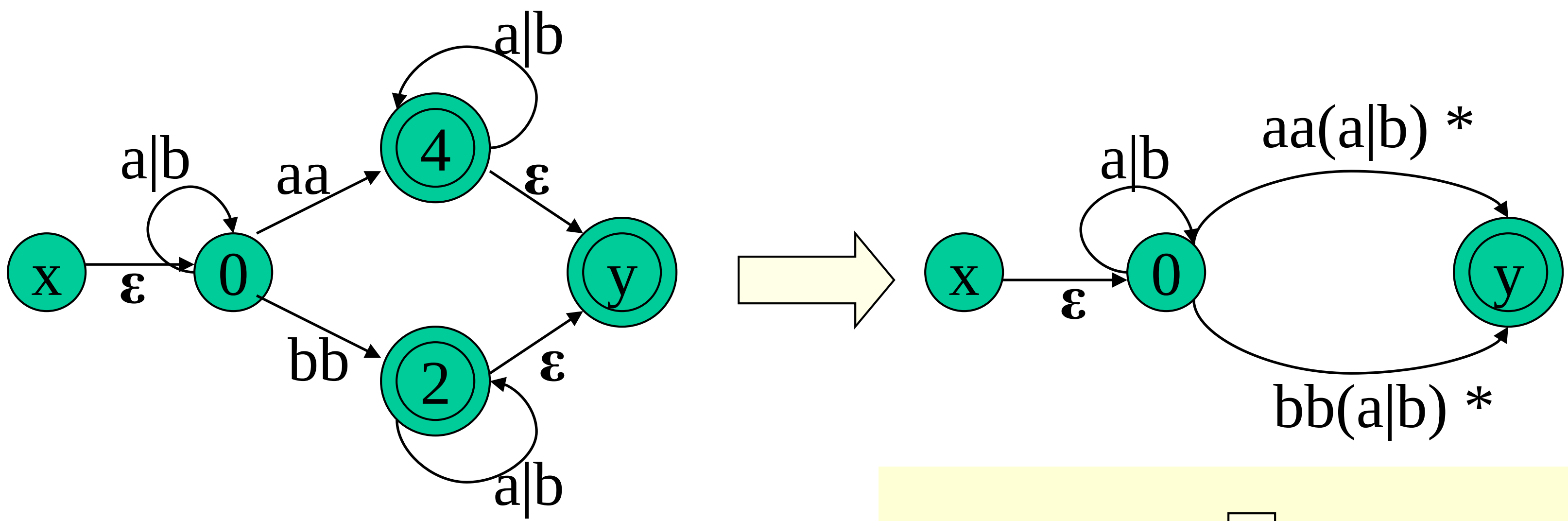
例:M:



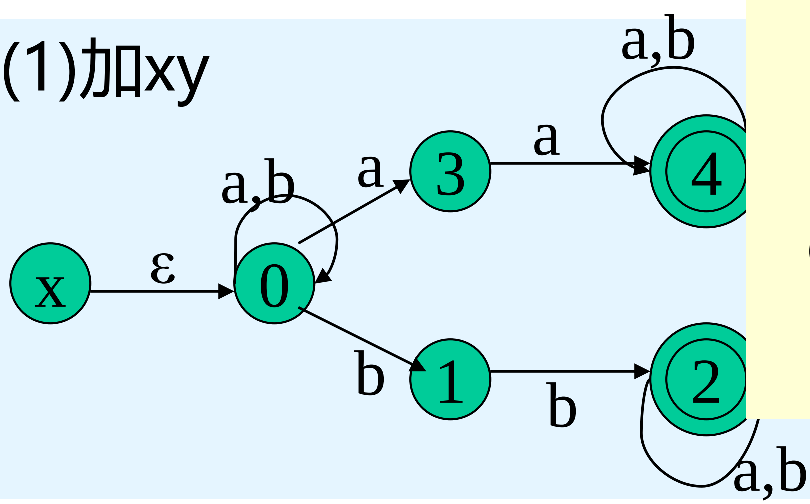
解: (1)加x,y

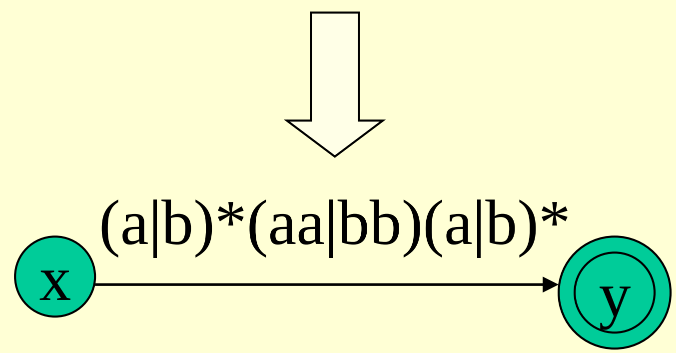


(2)消除M中的所有结点



解: (1)加xy





（5）正则文法⇒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→xB, B→y	A=xy
规则2	A→xA│y	A=x*y
规则3	A→x, A→y	A=x│y

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]

$S\rightarrow aA|a,$

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

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: $S=a((a|d)^*(a|d)|\varepsilon)$

(6) 正则式⇒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 $S \to r$

2) 若x,y是正则式,对形为$A \to xy$的产生式,重写为 $A \to xB \quad B \to y$,其中B为新的非终结符,$B \in V_n$

同样:对于 $A \to x^*y \Rightarrow A \to xA \quad A \to y$

$A \to x|y \Rightarrow A \to x \quad A \to y$

例:将R=a(a|d)*转换成相应的正则文法

解:1) $S \to a(a|d)^*$

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

3) $S \to aA$
$A \to (a|d)A$
$A \to \varepsilon$

4) $S \to aA$
$A \to aA|dA$
$A \to \varepsilon$

11.3 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：

LEX源程序 → LEX → 词法分析程序L

S.P.(字符串) → L → S.P.单词(符号串)

11.3.1 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：

$$D_1 \longrightarrow R_1$$
$$D_2 \longrightarrow R_2$$
$$\vdots$$
$$\vdots$$
$$D_n \longrightarrow R_n$$

限定：在Ri中只能出现字母表Σ中的字符，以及前面已 定义的正则表达式名字，我们用这种辅助定义式 (相当于规则)来定义程序语言的单词符号。

其中：

R_1, R_2，……，R_n 为正则表达式。

D_1, D_2，……，D_n 为正则表达式名字，称简名。

例：标识符：

letter→A|B|……|Z

digit →0|1| ……|9

iden → letter(letter|digit)*

带符号整数：

integer→digit (digit)*

sign → +| - |ε

sign_integer → 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$\Sigma \cup \{D_1, D_2, \cdots D_n\}$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

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

例：LEX 源程序

AUXILIARY DEFINITIONS /*辅助定义*/

letter → A|B|⋯|Z

digit → 0|1|⋯|9

%%

RECOGNITION RULES /*识别规则*/

1.BEGIN	{RETURN(1,–) }
2.END	{RETURN(2,–) }
3.FOR	{RETURN(3,–) }

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,—) }

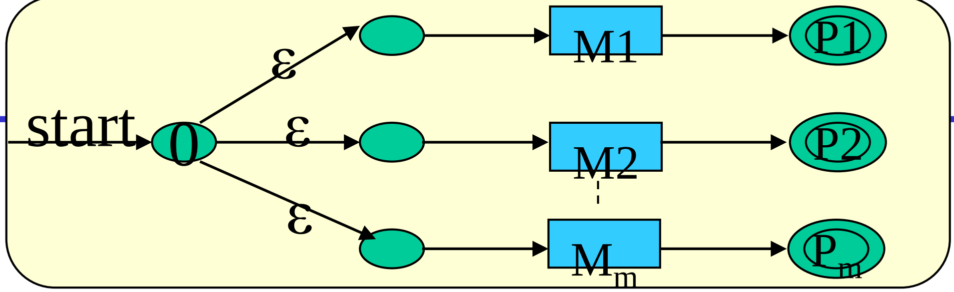
11.3.2 LEX的实现

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Pi，构造相应的不确定有穷自动机M_i
- 将各条规则的有穷自动机M_i合并成一个新的NFA M
- 确定化 NFA⇒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Pj和Pi规则的单词。

$$\underbrace{\overbrace{x\ \ x}^{P_j}\ \ x\ \ \overbrace{x}^{P_i}\ \ x}_{P_k}$$

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。

1.BEGIN {RETURN(1,—) }

8.letter(letter |digit)* {RETURN(8,TOKEN) }

例：字符串·"$\overbrace{\text{begin}}^{P_1}$"·
$\underbrace{\text{begin}}_{P_8}$

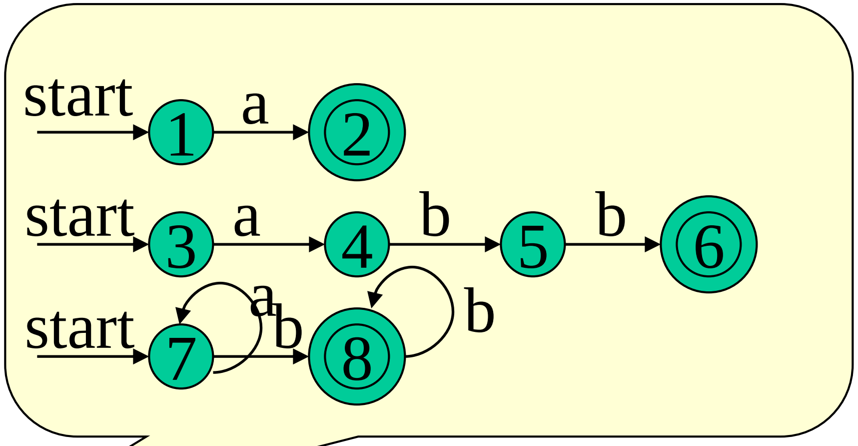
根据原则，应该识别为关键字begin，所以在写LEX源程序时应注意规则的排列顺序。

此外，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：

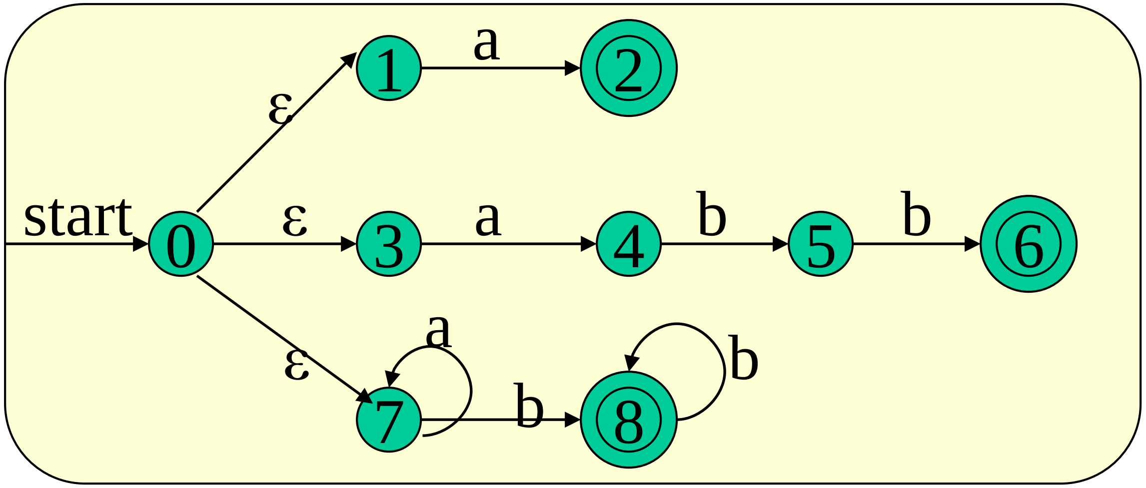
例： LEX源程序

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

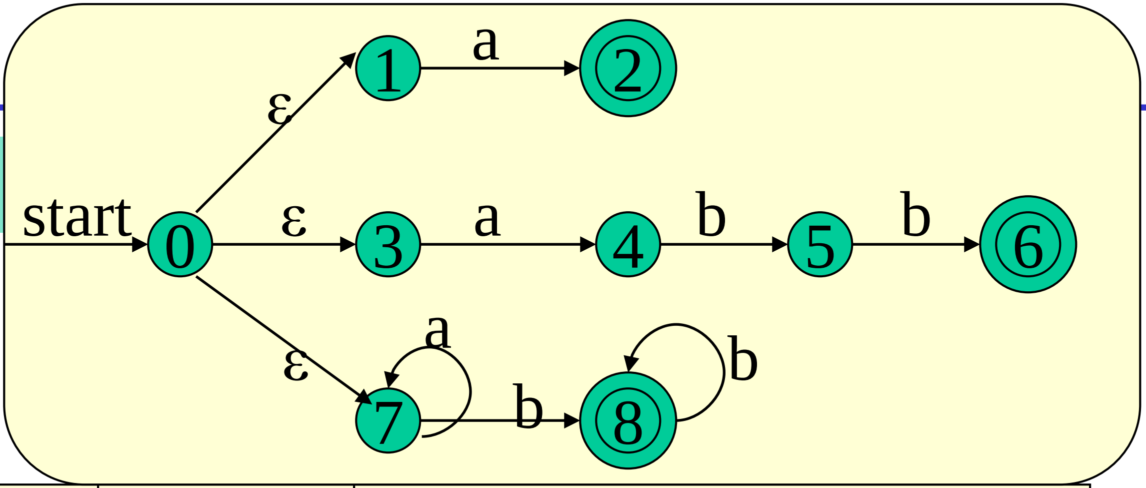


一.读LEX源程序，分别生成NFA，用状态图表示为：

二.合并成一个NFA：



三.确定化 给出状态转换矩阵



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}	φ	{8}	$a^* bb^*$
	{7}	{7}	{8}	
终态	{5,8}	φ	{6,8}	$a^* bb^*$
终态	{6,8}	φ	{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。

第十一章作业：

P254-255 1,2,4,5;