编译技术

胡春明

hucm@buaa.edu.cn

2018.9-2019.1

北京航空航天大学计算机学院

```
while (y < z) {
    int x = a + b;
    y += x;
}
```

w	h	i	l	e		(	y	<	z	)		{	\n
\t	i	n	t		x	=	a	+	b	;	\n	\t	y
	+	=		x	;	\n	}						

```
T_While
T_LeftParen
T_Identifier y
T_Less
T_Identifier z
T_RightParen
T_OpenBrace
T_Int
T_Identifier x
T_Assign
T_Identifier a
T_Plus
T_Identifier b
T_Semicolon
T_Identifier y
T_PlusAssign
T_Identifier x
T_Semicolon
T_CloseBrace
```

保留字: while， int

标识符: x, y, z, a, b

分隔符: _（空格）, \n, \t

token, symbol

单词，符号

规则从哪里来?

（左、右）线性文法

状态图的构造

Source: Stanford CS143 (2012)

正则表达式基础

正则表达式：另一种在Σ*上识别语言的方法

用正则表达式表达语言特征

(R)

R^*

R_1R_2

$R_1 \mid R_2$

ab*c|d

正则表达式：另一种在Σ*上识别语言的方法

用正则表达式表达语言特征

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

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：$b\{ab\} = \{ba\}b$

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

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

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

正则表达式与3型文法等价（给定文法，可构造正则表达式，反之亦然）

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	

所有包含00的字符串， Σ={0,1}

(0 | 1)*00(0 | 1)*

11011100101
0000
111110111110011111

Source: Stanford CS143 (2012)

所有长度为4的字符串， $\Sigma=\{0,1\}$

(0|1){4}

0000

1010

1111

1000

Source: Stanford CS143 (2012)

最多一个0的字符串， Σ={0,1}

$$1^*(0 \mid \varepsilon)1^*$$

$$1^*0?1^*$$

11110111
111111
0111
0

Source: Stanford CS143 (2012)

合法的Email地址： $\Sigma=\{a, @, .\}$

$$aa^* \ (.aa^*)^* \ @ \ aa^*.aa^* \ (.aa^*)^*$$

$$a^+ \ (.a^+)^* \ @ \ a^+.a^+ \ (.a^+)^*$$

$$a^+(.a^+)^*@a^+(.a^+)^+$$

Source: Stanford CS143 (2012)

偶数： Σ={+, -, 0, 1, 2, 3, 4, 5, 6, 7, 8, 9}

(+|-)?(0|1|2|3|4|5|6|7|8|9)*(0|2|4|6|8)

(+|-)?[0123456789]*[02468]

(+|-)?[0-9]*[02468]

42
+1370
-3248
-9999912

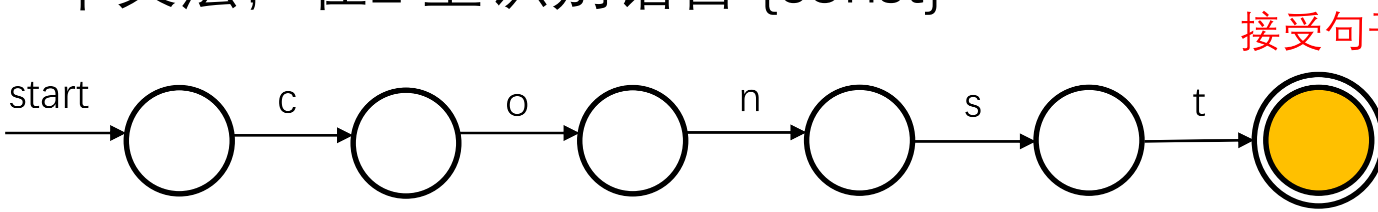
Source: Stanford CS143 (2012)

从“状态图”到“有穷状态自动机”

从状态图到自动机的形式化定义

♦ 例：保留字const识别

构造一个文法，在Σ^*上识别语言 {const}



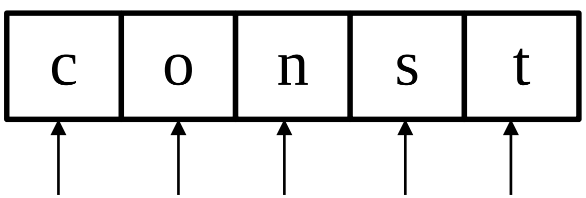
G(Z): Z::->'c'A

A::->'o'B

B::->'n'C

C::->'s'D

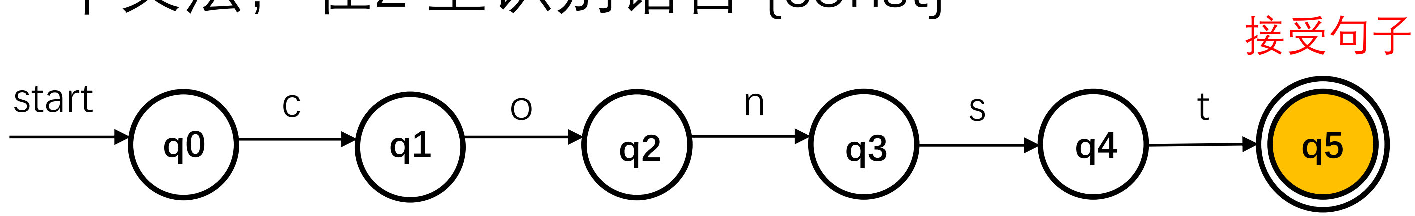
D::->'t'



从状态图到自动机的形式化定义

♦ 例：保留字const识别

构造一个文法，在Σ^*上识别语言 {const}



（有穷状态）自动机：是另一种抽象 Σ^*上的语言 L 的方法

五元组：$(S, \Sigma, \delta, s_0, Z)$

状态集S：{q0, q1, q2, q3, q4, q5}

字母表Σ：{a..z, A..Z, 0..9}

初始状态　s = q0 ∈S

状态转移函数δ: $S\times \Sigma \rightarrow S$ – 描述每一条边

$\delta(q0, c)=q1$, $\delta(q1, o)=q2$

$\delta(q2, n)=q3$, $\delta(q3, s)=q4$

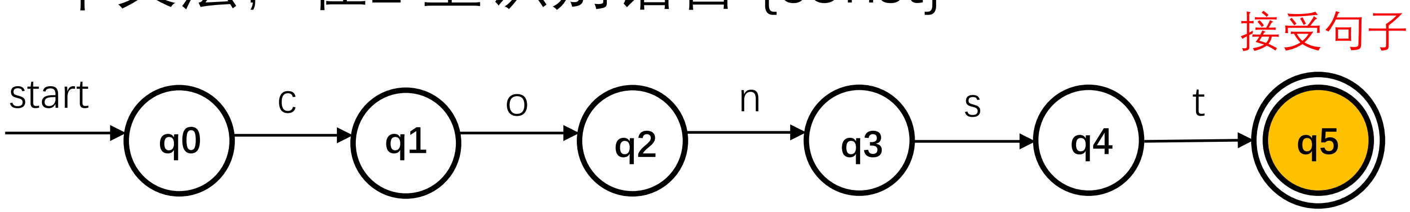
$\delta(q4, 5)=q5$

接受状态 Z = {q5}

从状态图到自动机的形式化定义

♦ 例：保留字const识别

构造一个文法，在Σ^*上识别语言 {const}



(有穷状态）自动机：是另一种抽象 Σ^*上的语言 L 的方法

五元组：$(S, \Sigma, \delta, s_0, Z)$

状态集S：$\boxed{\{q0, q1, q2, q3, q4, q5\}}$

字母表Σ：{a..z, A..Z, 0..9}　　有穷状态

初始状态　s = q0 ∈S

状态转移函数δ: S× Σ→S – 描述每一条边

δ(q0, c)=q1, δ(q1, o)=q2

δ(q2, n)=q3, δ(q3, s)=q4

δ(q4, 5)=q5　　确定

接受状态 Z = {q5}

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)

   $S\times\Sigma\to S$

   $\delta(s,a)=s'$, $s, s' \in S$, $a\in\Sigma$
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}, δ, 0, {3})

δ (0, a) =1 δ (0, b) =2

δ (1, a) =3 δ (1, b) =2

δ (2, a) =1 δ (2, b) =3

δ (3, a) =3 δ (3, b) =3

状态转换函数δ可用一矩阵来表示：

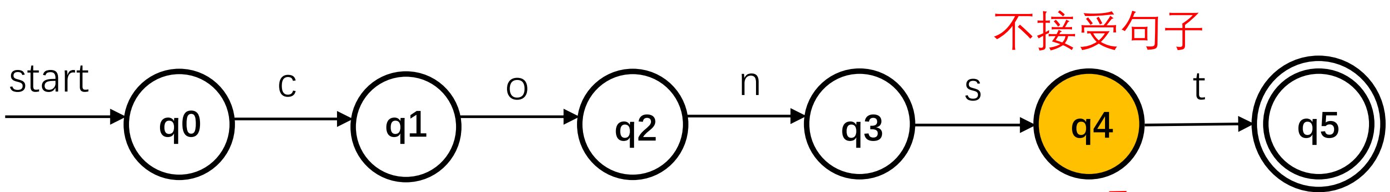
状态 \ 输入字符	a	b
0	1	2
1	3	2
2	1	3
3	3	3

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！

识别一个“单词（token, symbol）”的文法：

- ♦ **自动机接受字符x （从而识别语言L）**

构造一个文法，在Σ^*上识别语言 {const}



G(Z):

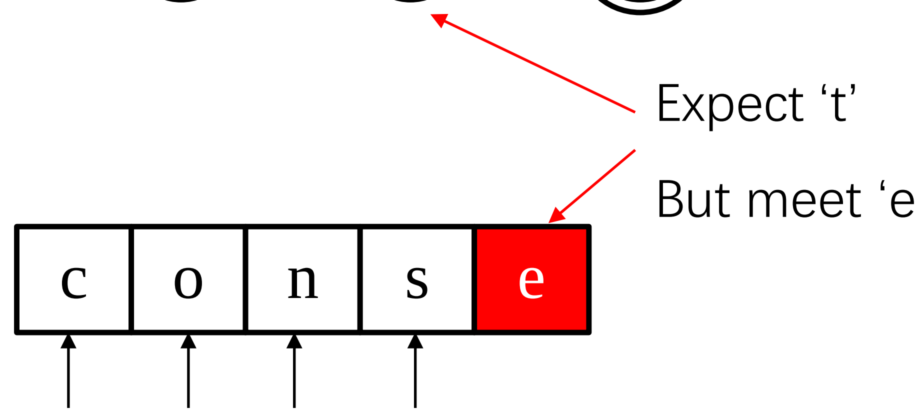
Z::->’c’A

A::->’o’B

B::->’n’C

C::->’s’D

D::->’t’



DFA M所接受的符号串：
令 $\alpha = a_1a_2\cdots\cdots a_n$， $\alpha \in \Sigma$，**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。**

$\delta(s_0, a_1) = s_1$

$\delta(s_1, a_2) = s_2$

………

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

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

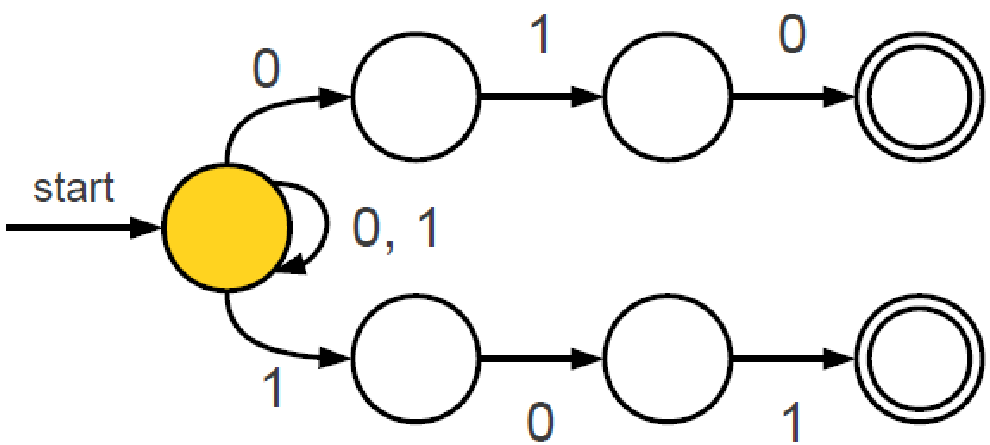
换言之：若存在一条初始状态到某一终止状态的路径，且这条路径上能有弧的标记符号连接成符号串α，则称α为DFA M（接受）识别。

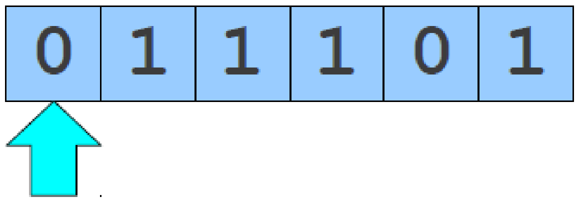
DFA M所接受的语言为： $L(M) = \{ \alpha \mid \delta(s_0, \alpha) = s_n, s_n \in Z \}$

确定型与非确定型（有穷状态）自动机

（有穷状态）自动机：是另一种抽象 Σ*上的语言 L 的方法

更复杂一些的情况：



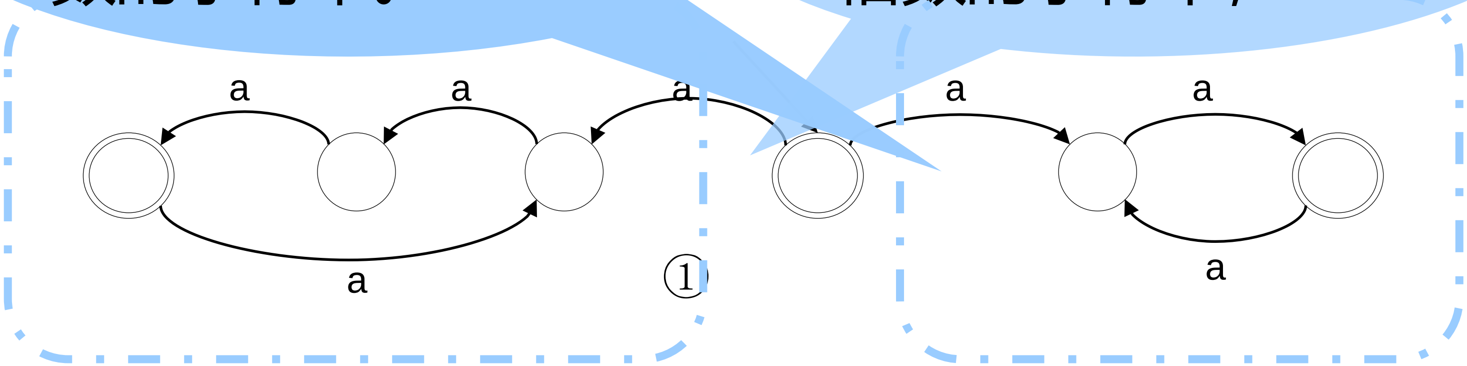


Source: Stanford CS143 (2012)

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；

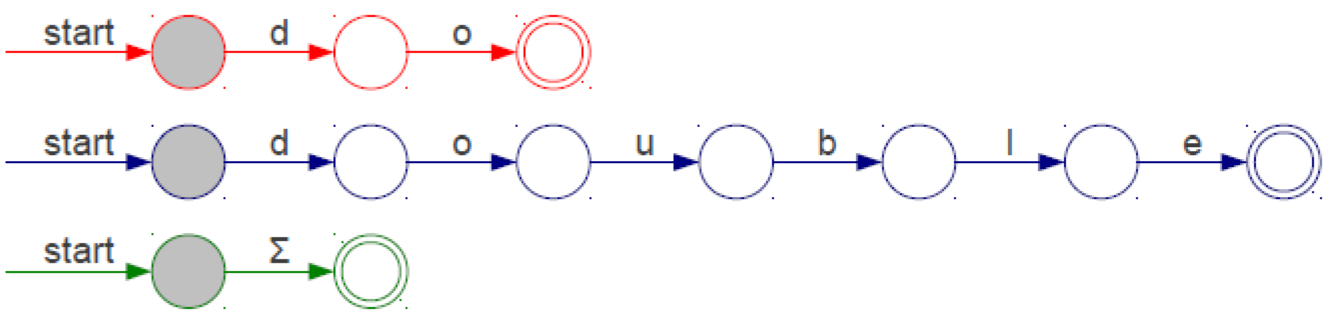


因此，该NFA所能接受的语言是所有由a组成，长度为2和3的倍数的字符串的集合。在第一次状态转换中，自动机需要选择要走的路径。只要有任何路径可匹配输入字符串，该串就必须被接受，因此NFA必须正确“猜测”所需的路径。

同时有好几个单词的可能性?

通常我们选择 最长的匹配，或者显示指定优先级（如：保留字优先)

```
T_Do          do
T_Double      double
T_Mystery     [A-Za-z]
```



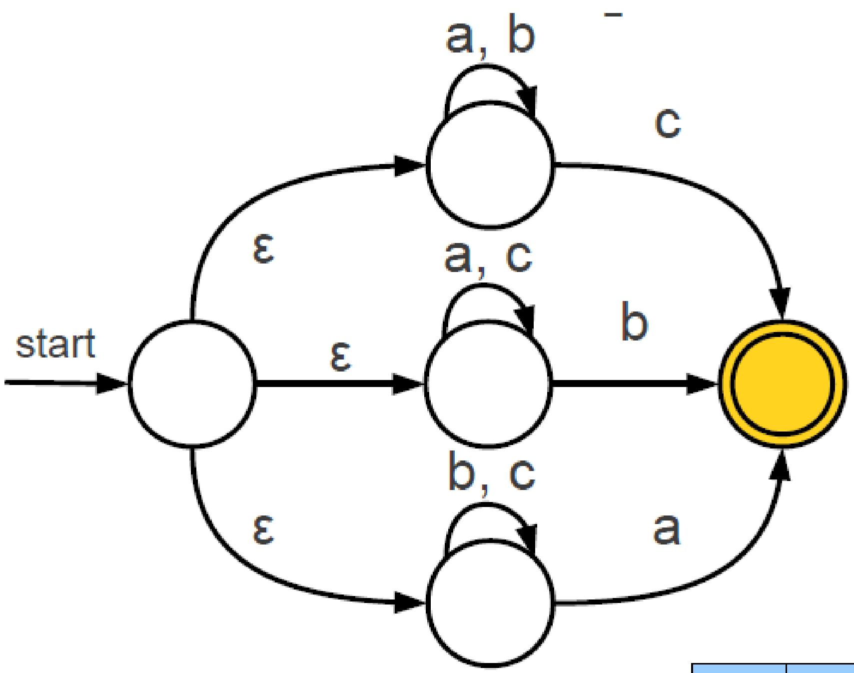
D	O	U	B	D	O	U	B	L	E

Source: Stanford CS143 (2012)

非确定型（有穷状态）自动机：引入空字符

（有穷状态）自动机：是另一种抽象 Σ*上的语言 L 的方法

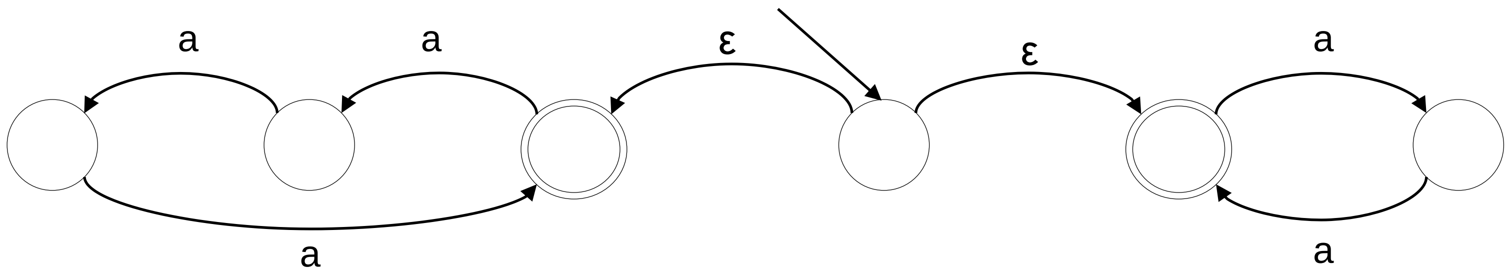
更复杂一些的情况：



b	c	b	a

Source: Stanford CS143 (2012)

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。

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。

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:

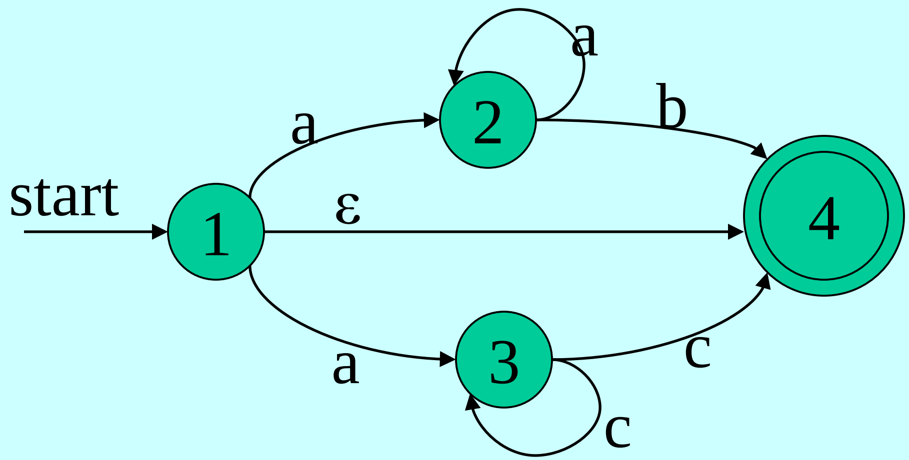
$$L(M')=\{\alpha|\delta(S_0,\alpha)=S' \quad S'\cap Z\neq\Phi\}$$

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

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

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

NFA的确定化

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。

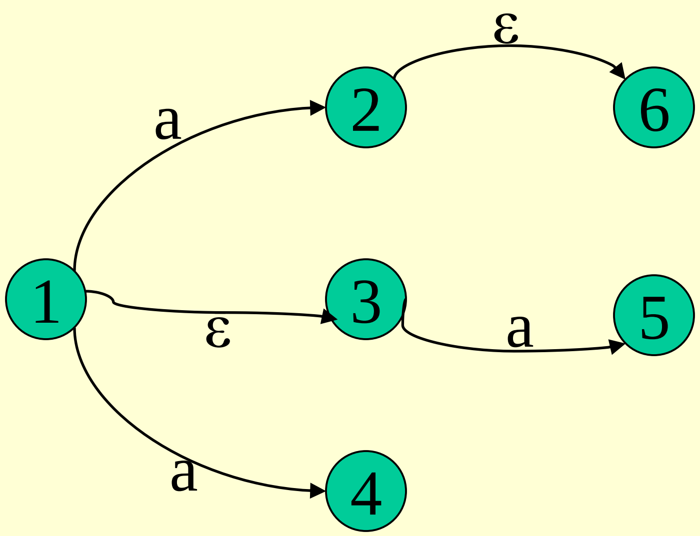
可以通过一例子来说明状态子集的ε-闭包的构造方法

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，$a \in \Sigma$

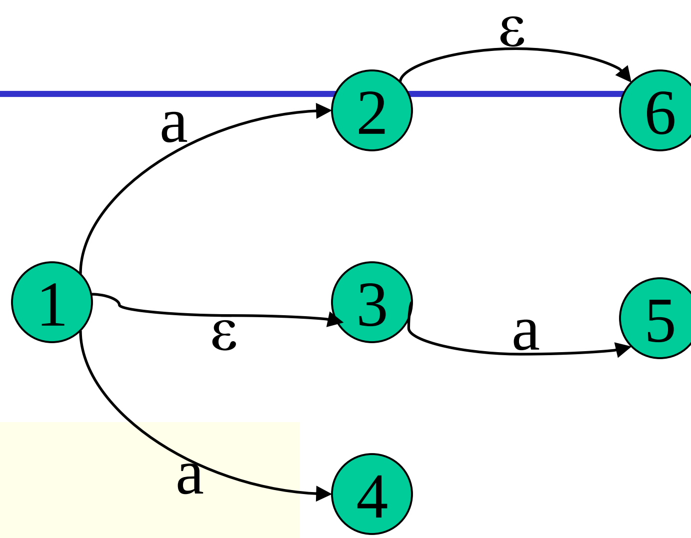
定义：$I_a = \varepsilon\text{-closure}(J)$

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

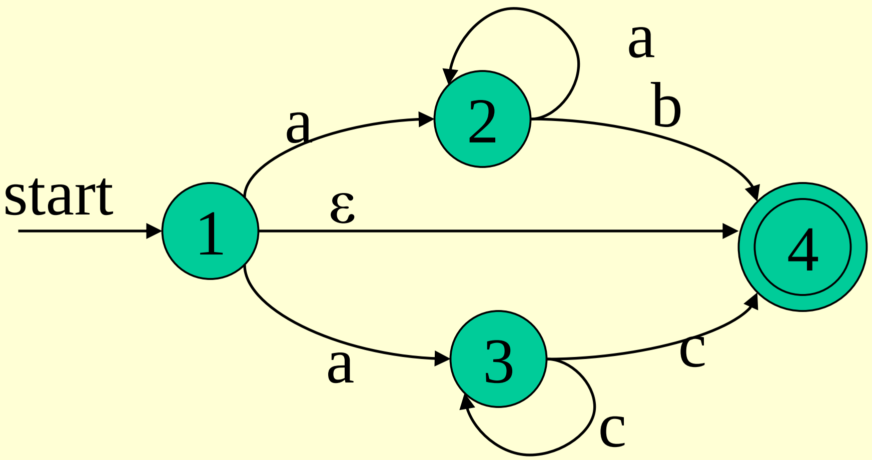


例：令 $I=\{1\}$

$$
\begin{aligned}
I_a &= \varepsilon\text{-closure}(J) \\
&= \varepsilon\text{-closure}(\delta(1,\ a)) \\
&= \varepsilon\text{-closure}(\{2,\ 4\}) \\
&= \{2,\ 4,\ 6\}
\end{aligned}
$$

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’



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

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

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

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

$=\{2,3\}$

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

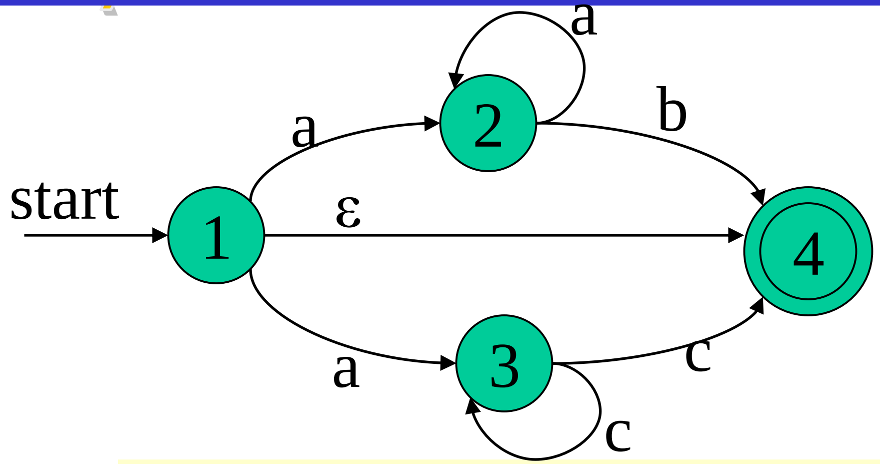
$=\varepsilon\text{-closure}(\varphi)$

$=\varphi$

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

$=\varphi$

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



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}

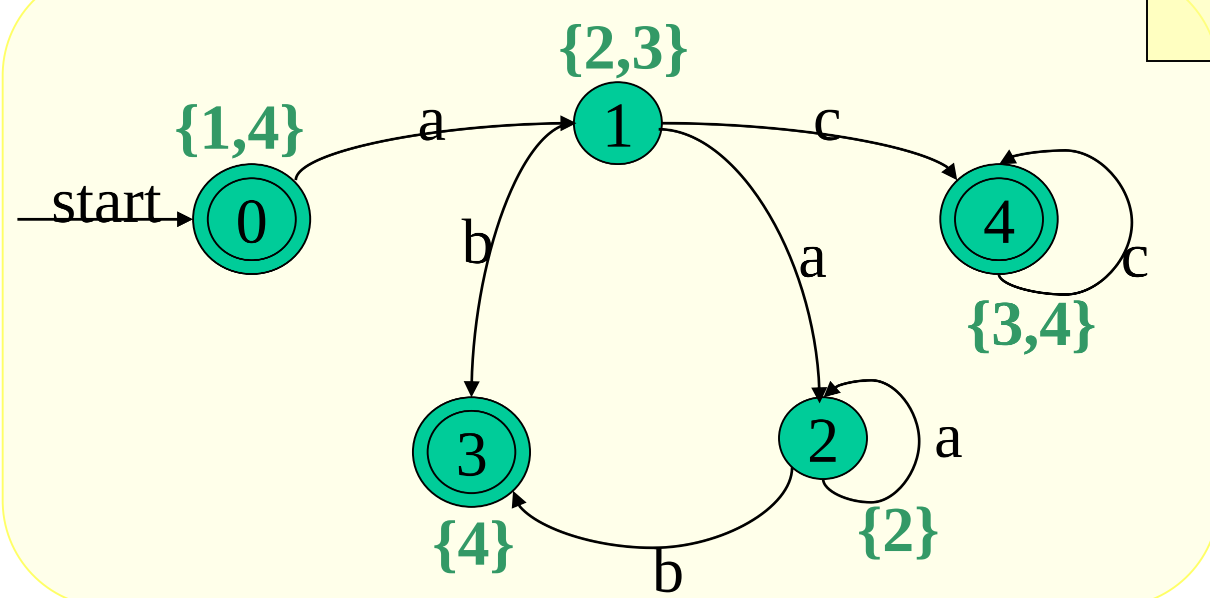
将求得的状态转换矩阵重新编号

DFA M状态转换矩阵：

状态 \ 符号	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：

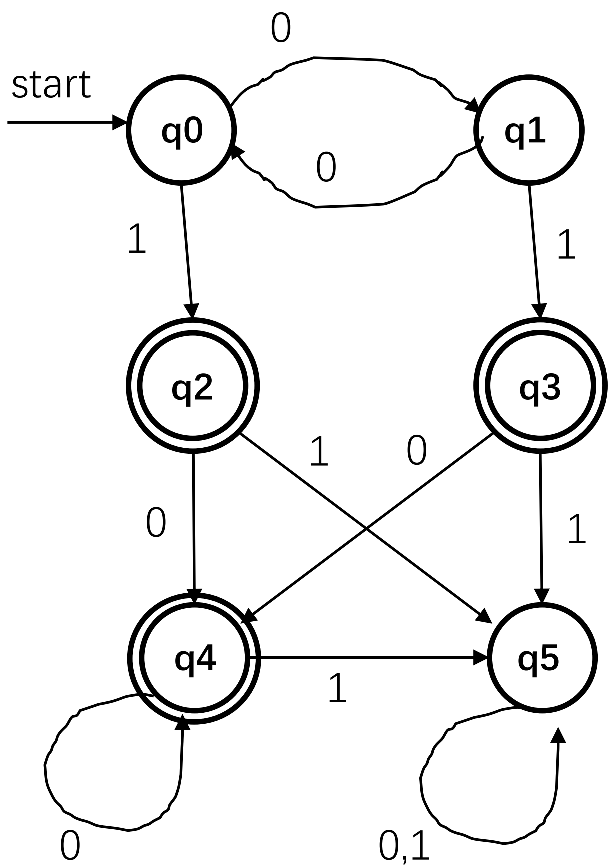
1.正则表达式与有穷自动机，给出了两者的定义。
用3型文法所定义的语言都可以用正则表达式描述，
用正则表达式描述单词是为了自动生成词法分析程序。
有一个正则表达式则对应一个正则集合。

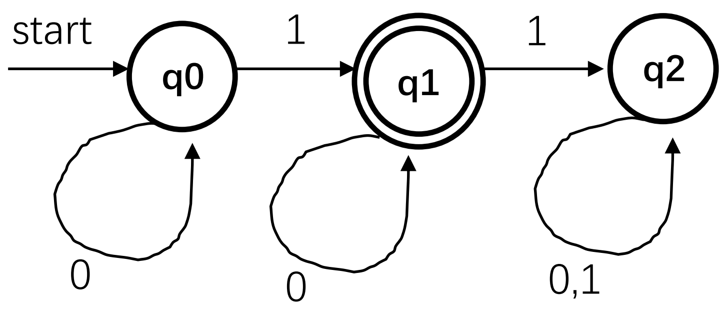
2. NFA M’ 的定义、确定化 ➔ **对任何一个NFA M’, 都可以构造出一个DFA M, 使得 L(M) = L(M’)**

构造出来的DFA M唯一吗？

DFA的极简化

构造一个自动机，识别语言 0*10*





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

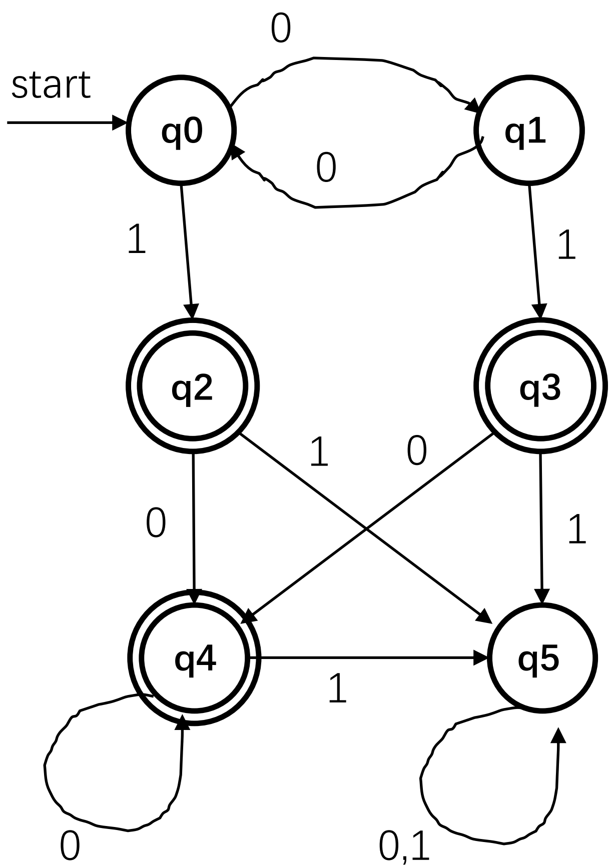
(2)等价状态⟺状态s和t的等价条件是:

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。

构造一个自动机，识别语言 0*10*

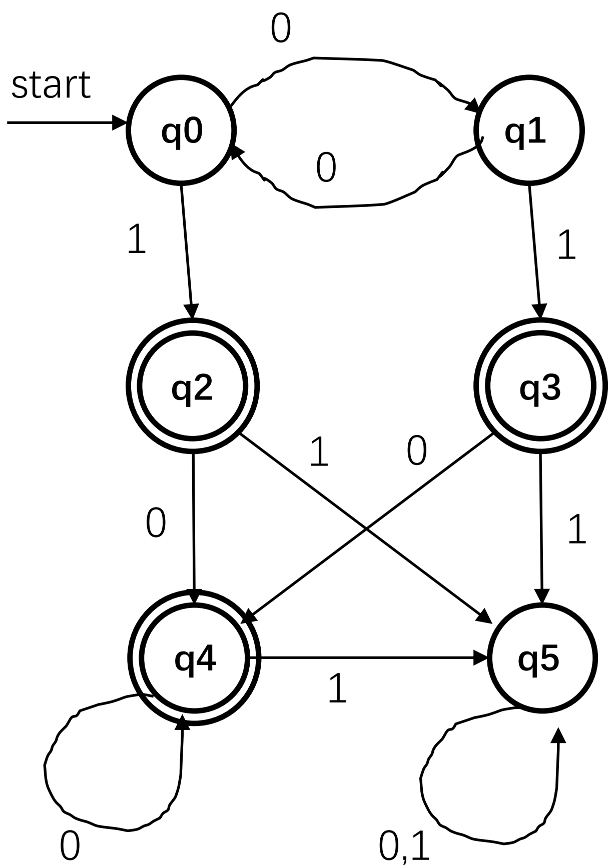


没有不可达状态

“终态”和“非终态”显然可区分

q_0	-					
q_1		-				
$\mathbf{q_2}$	×	×	-			
$\mathbf{q_3}$	×	×		-		
$\mathbf{q_4}$	×	×			-	
q_5			×	×	×	-
	q_0	q_1	$\mathbf{q_2}$	$\mathbf{q_3}$	$\mathbf{q_4}$	q_5

构造一个自动机，识别语言 0*10*



没有不可达状态

"终态"和"非终态"显然可区分

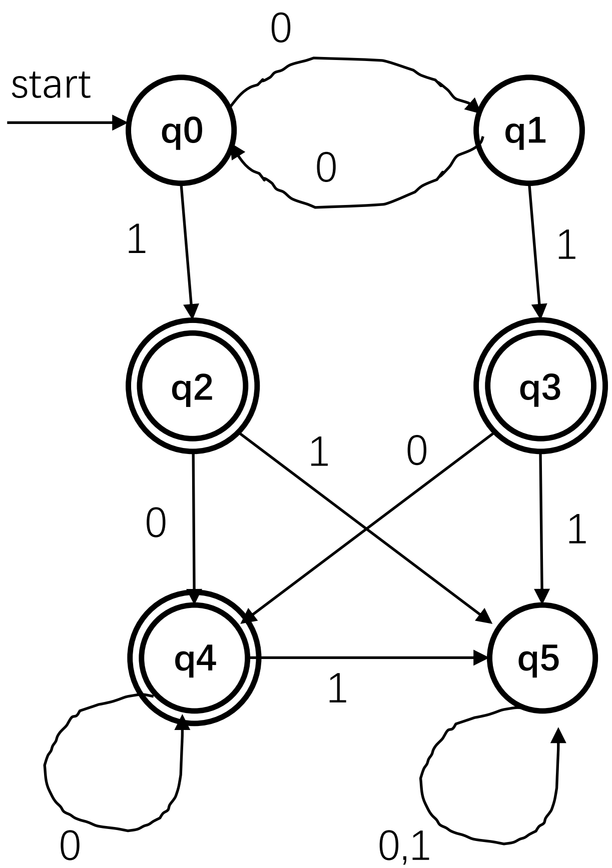
需考察的未标记对：(q0, q1), (q0, q5), (q2, q3), (q2, q4), (q3, q4)

Delta(q0, 1) = q2, Delta(q1, 1) = q3

所以，如果q2 ≡ q3，则有 q0 ≡ q1

q_0	-					
q_1	?	-				
q_2	×	×	-			
q_3	×	×		-		
q_4	×	×			-	
q_5			×	×	×	-
	q_0	q_1	q_2	q_3	q_4	q_5

构造一个自动机，识别语言 0*10*



没有不可达状态

“终态”和“非终态”显然可区分

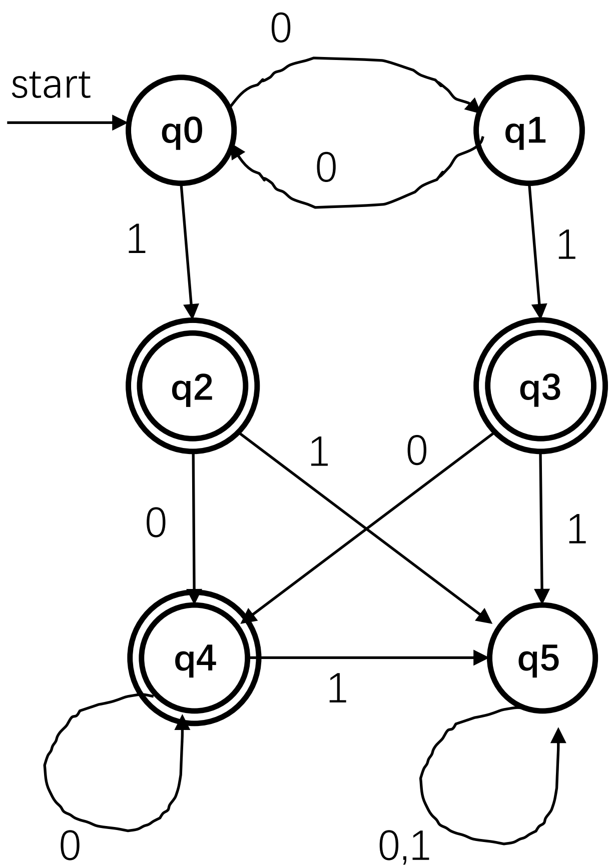
需考察的未标记对：**(q0, q1)**, **(q0, q5)**, **(q2, q3)**, **(q2, q4)**, **(q3, q4)**

Delta(q2, 0) = Delta(q3, 0) = q4

Delta(q2, 1) = Delta(q3, 1) = q5

q_0	-					
q_1	**2?3**	-				
$\mathbf{q_2}$	×	×	-			
$\mathbf{q_3}$	×	×	**?**	-		
$\mathbf{q_4}$	×	×			-	
q_5			×	×	×	-
	q_0	q_1	$\mathbf{q_2}$	$\mathbf{q_3}$	$\mathbf{q_4}$	q_5

构造一个自动机，识别语言 0*10*



没有不可达状态

"终态" 和 "非终态" 显然可区分

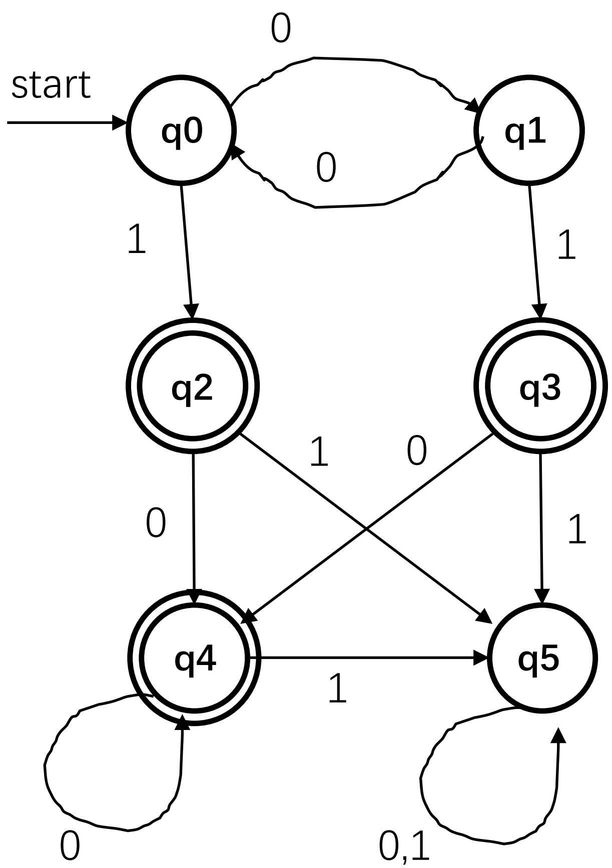
需考察的未标记对：(q0, q1), (q0, q5), (q2, q3), (q2, q4), (q3, q4）

Delta(q2, 0) = Delta(q3, 0) = q4

Delta(q2, 1) = Delta(q3, 1) = q5

q_0	-					
q_1	√	-				
$\mathbf{q_2}$	×	×	-			
$\mathbf{q_3}$	×	×	√	-		
$\mathbf{q_4}$	×	×			-	
q_5			×	×	×	-
	q_0	q_1	$\mathbf{q_2}$	$\mathbf{q_3}$	$\mathbf{q_4}$	q_5

构造一个自动机，识别语言 0*10*



没有不可达状态

“终态”和“非终态”显然可区分

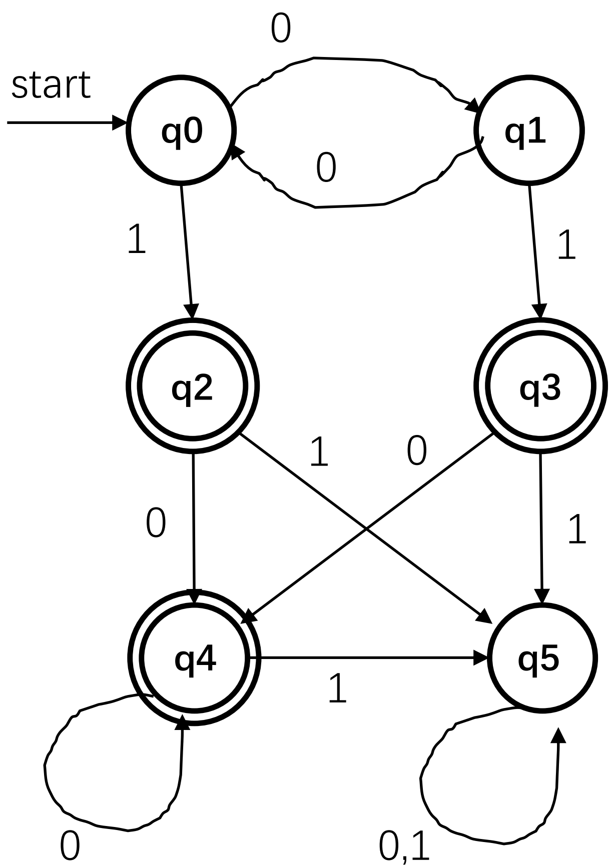
需考察的未标记对：(q0, q1), (q0, q5), (q2, q3), (q2, q4), (q3, q4）

Delta(q0, 0) = q1, Delta(q5,0)=q5

Delta(q0, 1) = q2终态, Delta(q5,1)=q5非终态

q_0	-					
q_1	√	-				
q_2	×	×	-			
q_3	×	×	√	-		
q_4	×	×			-	
q_5	?	?	×	×	×	-
	q_0	q_1	q_2	q_3	q_4	q_5

构造一个自动机，识别语言 0*10*



没有不可达状态

“终态”和“非终态”显然可区分

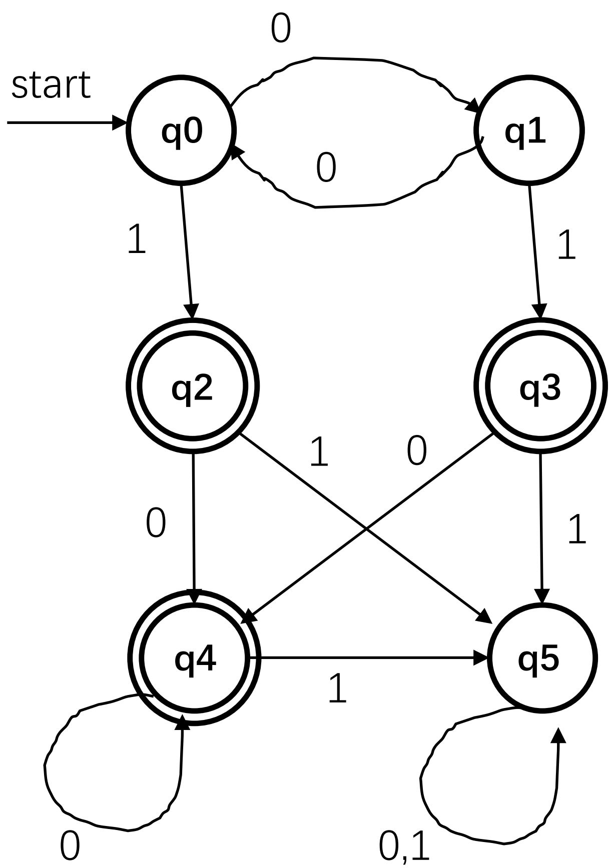
需考察的未标记对：(q0, q1), (q0, q5), (q2, q3), (q2, q4), (q3, q4）

因此，q0, q5可区分，同理q1,q5可区分

Delta(q0, 1) = q2终态, Delta(q5,1)=q5非终态

q_0	-					
q_1	√	-				
q_2	×	×	-			
q_3	×	×	√	-		
q_4	×	×			-	
q_5	×	×	×	×	×	-
	q_0	q_1	q_2	q_3	q_4	q_5

构造一个自动机，识别语言 0*10*



没有不可达状态

“终态”和“非终态”显然可区分

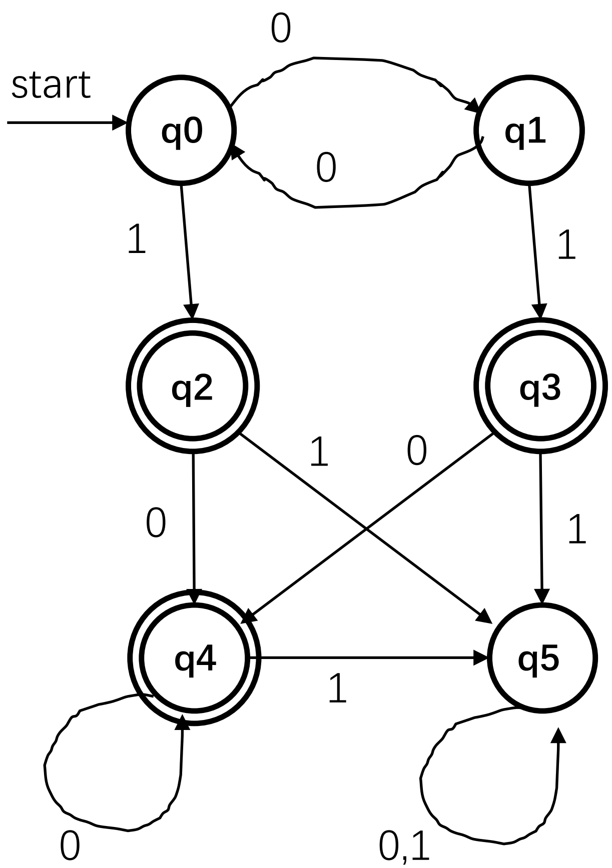
需考察的未标记对：(q0, q1), (q0, q5), (q2, q3), (q2, q4), (q3, q4）

Delta(q2, 0) = Delta(q4, 0) = q4

Delta(q2, 1) = Delta(q4, 1) = q5

q_0	-					
q_1	√	-				
q_2	×	×	-			
q_3	×	×	√	-		
q_4	×	×	?		-	
q_5	×	×	×	×	×	-
	q_0	q_1	q_2	q_3	q_4	q_5

构造一个自动机，识别语言 0*10*



没有不可达状态

“终态”和“非终态”显然可区分

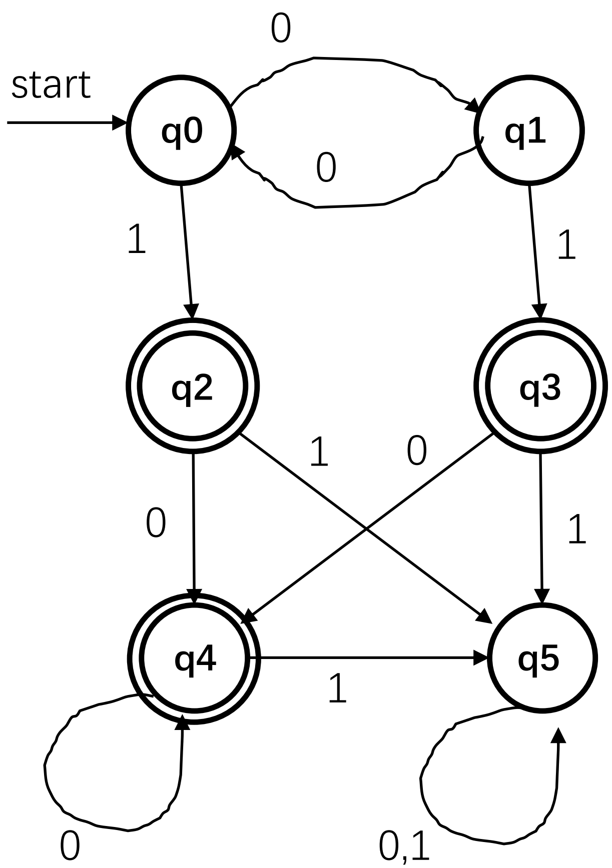
需考察的未标记对：(q0, q1), (q0, q5), (q2, q3), (q2, q4), (q3, q4）

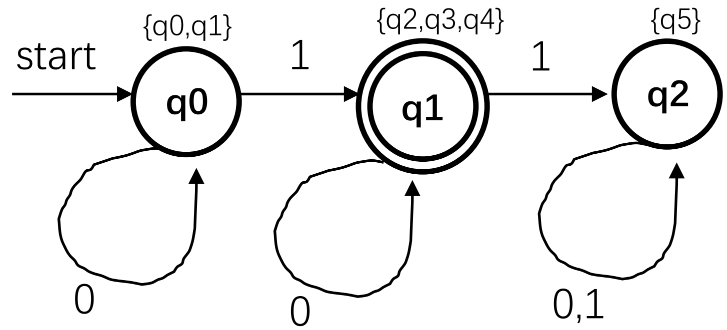
所以，q2, q4可合并

同理，q3, q4可合并

q_0	-					
q_1	√	-				
q_2	×	×	-			
q_3	×	×	√	-		
q_4	×	×	√	√	-	
q_5	×	×	×	×	×	-
	q_0	q_1	q_2	q_3	q_4	q_5

构造一个自动机，识别语言 0*10*

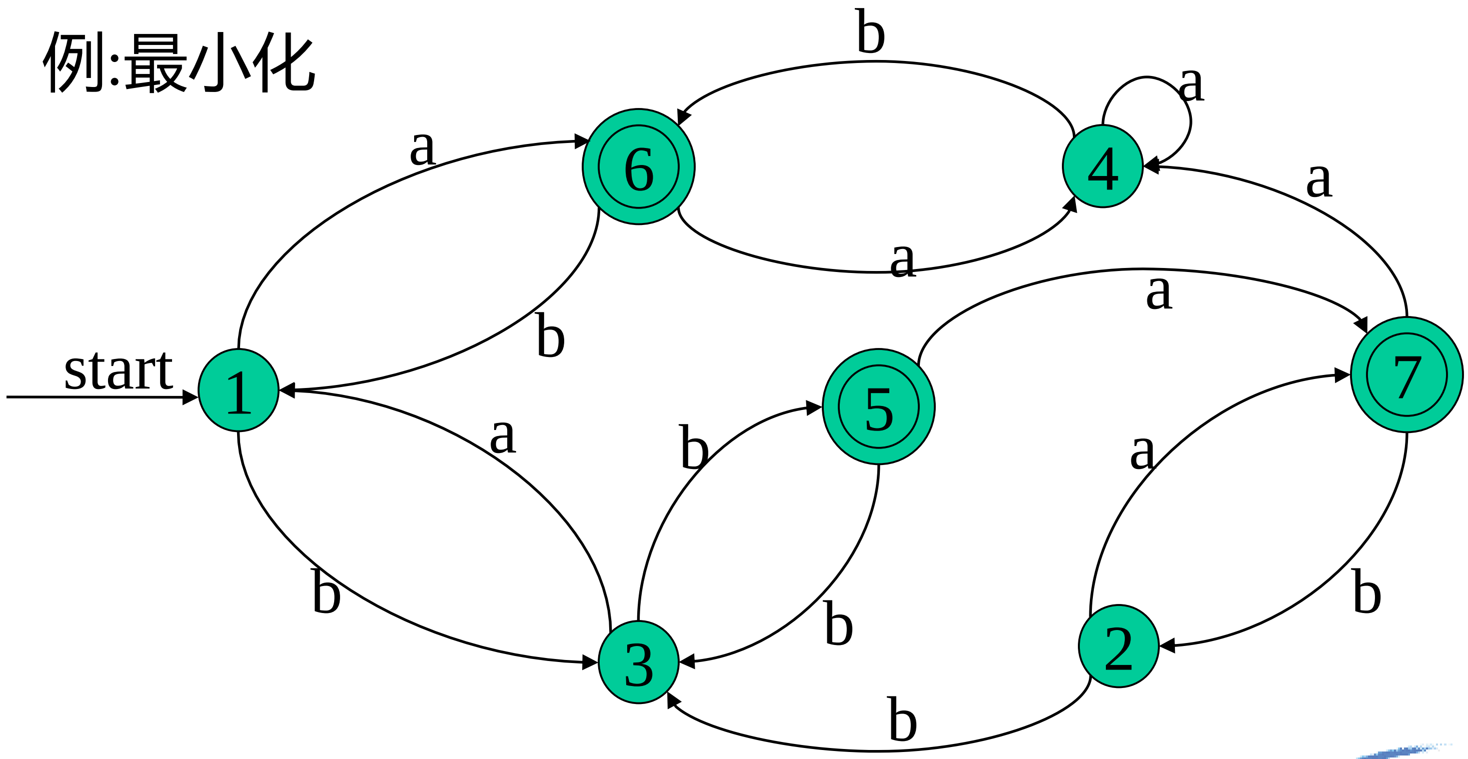


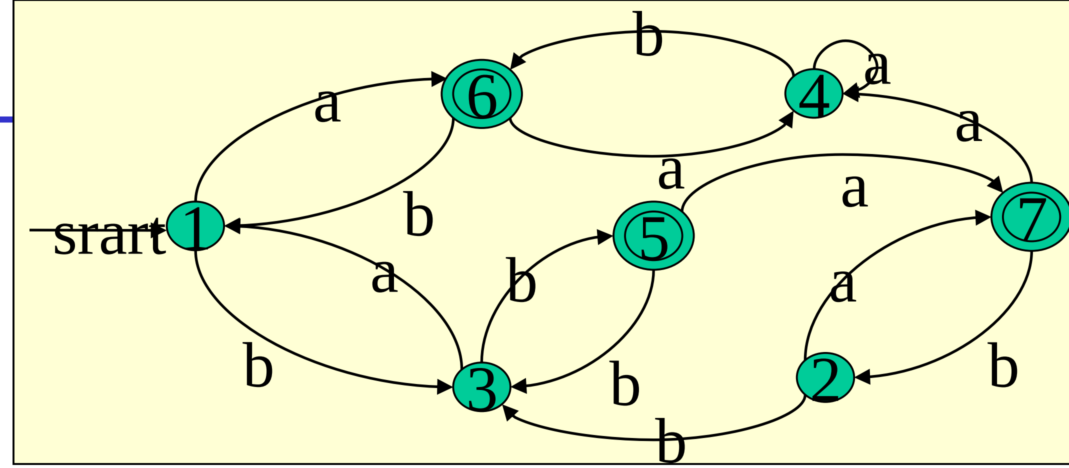


q_0	-					
q_1	√	-				
q_2	×	×	-			
q_3	×	×	√	-		
q_4	×	×	√	√	-	
q_5	×	×	×	×	×	-
	q_0	q_1	q_2	q_3	q_4	q_5

"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



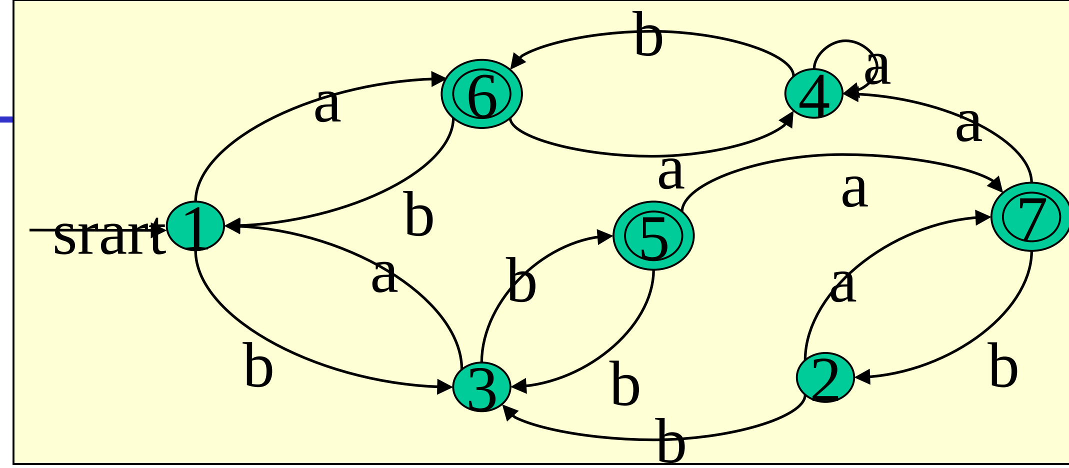


解:

(一)区分终态与非终态

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

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



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

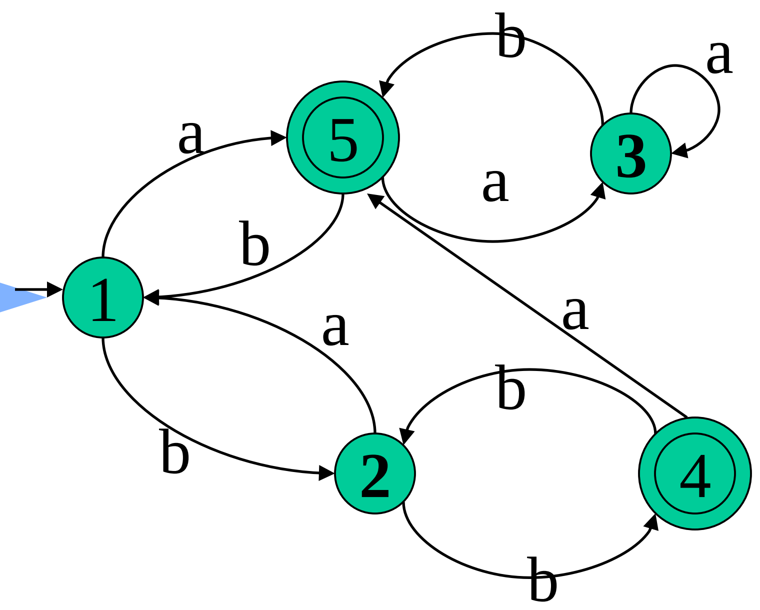
	a	b	区号
1	6	3	1
2	7	3	
3	1	5	2
4	4	6	3
5	7	3	4
6	4	1	5
7	4	2	

	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	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



词法分析的自动化

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

给定RE → NFA → DFA → 极小化，从而自动生成词法分析程序

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：

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

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：

$P_1 \quad \{A_1\}$

$P_2 \quad \{A_2\}$

⋮

⋮

$P_m \quad \{A_m\}$

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符号的LEX源程序：

例：LEX 源程序

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

AUXILIARY DEF

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:

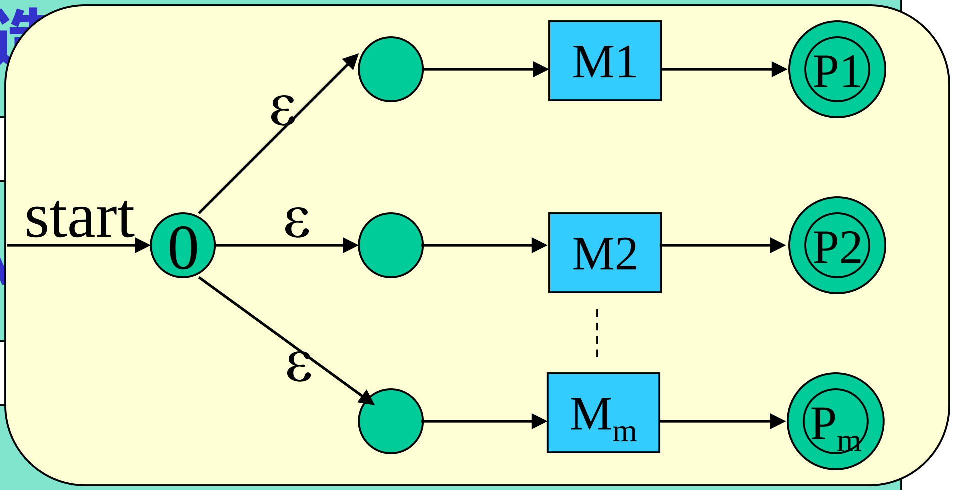
词法分析程序	状态转换矩阵(DFA)
	控制执行程序

∴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

- 将各条规则的有穷自动机M



- 确定化　　NFA⇒DFA

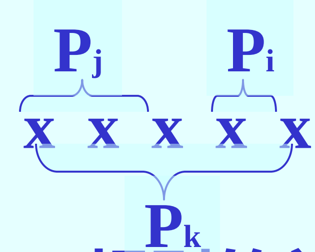
生成该DFA的状态转换矩阵和控制执行程序

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。



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·“$\underbrace{\overset{P_1}{\text{begin}}\cdot}_{P_8}$”

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。

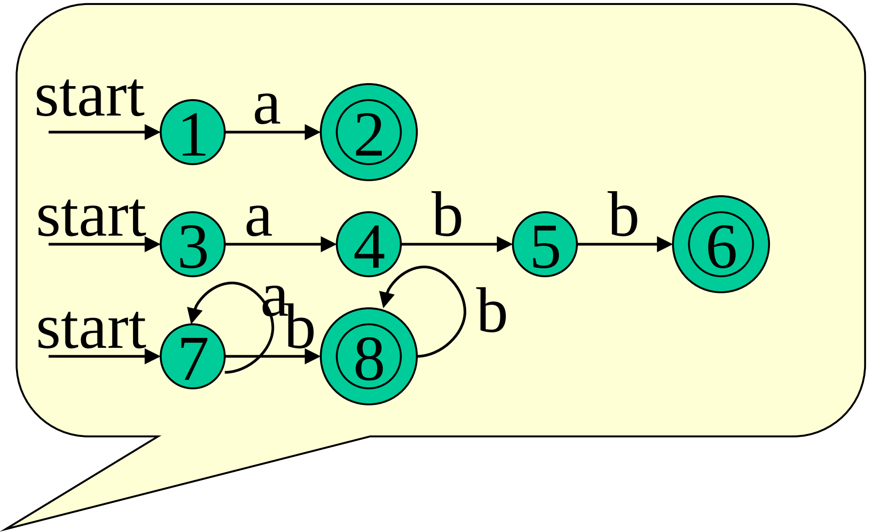
可以通过一个例子来说明这些问题：

例： LEX源程序

a { }

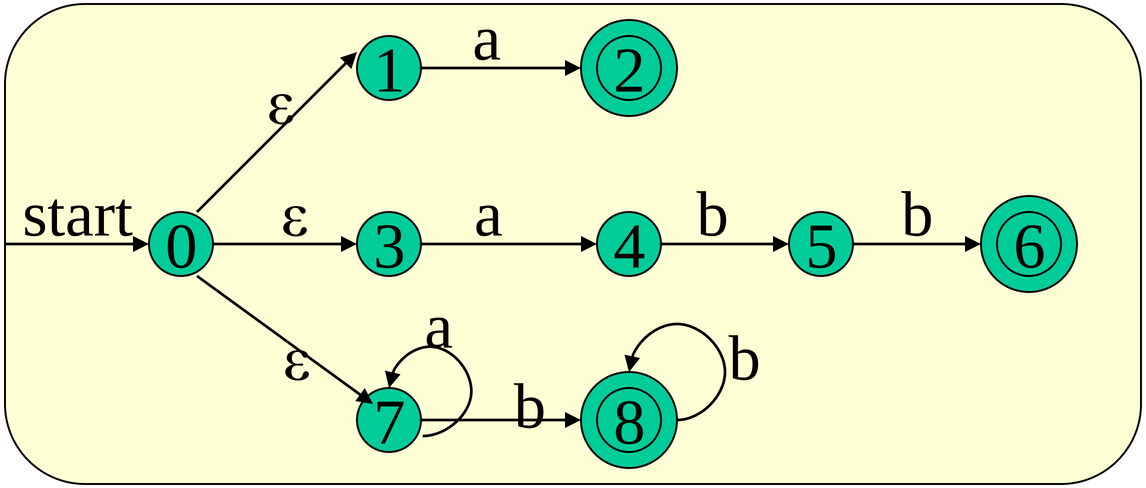
abb { }

a^*bb^* { }

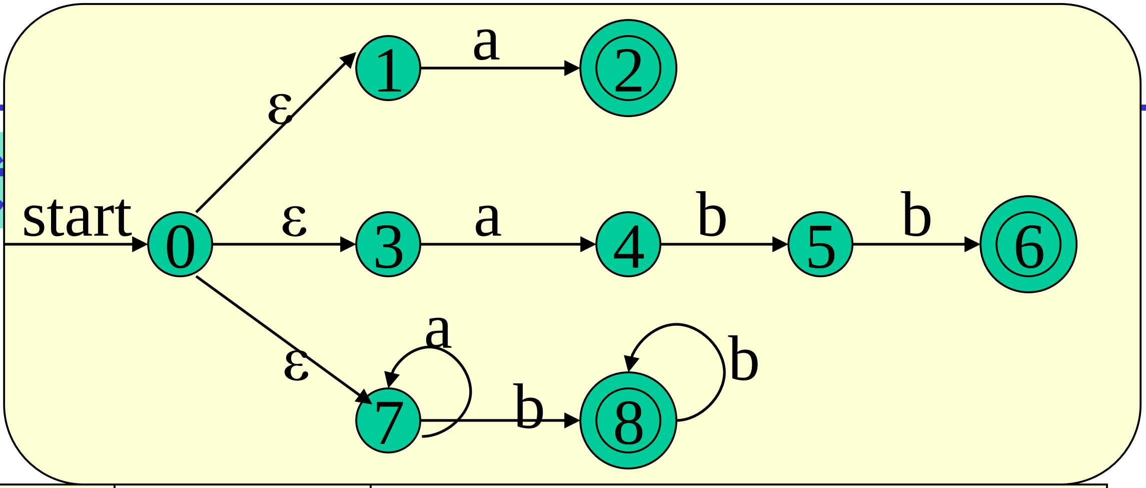


一.读LEX源程序，分别生成NFA，用状态图表示为：

二.合并成一个NFA：



三.确定化　　给出状



	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

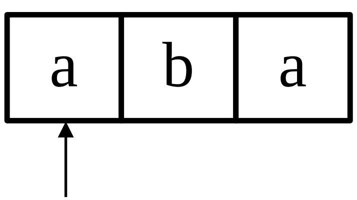
在此DFA中　　**初态为{0,1,3,7}**

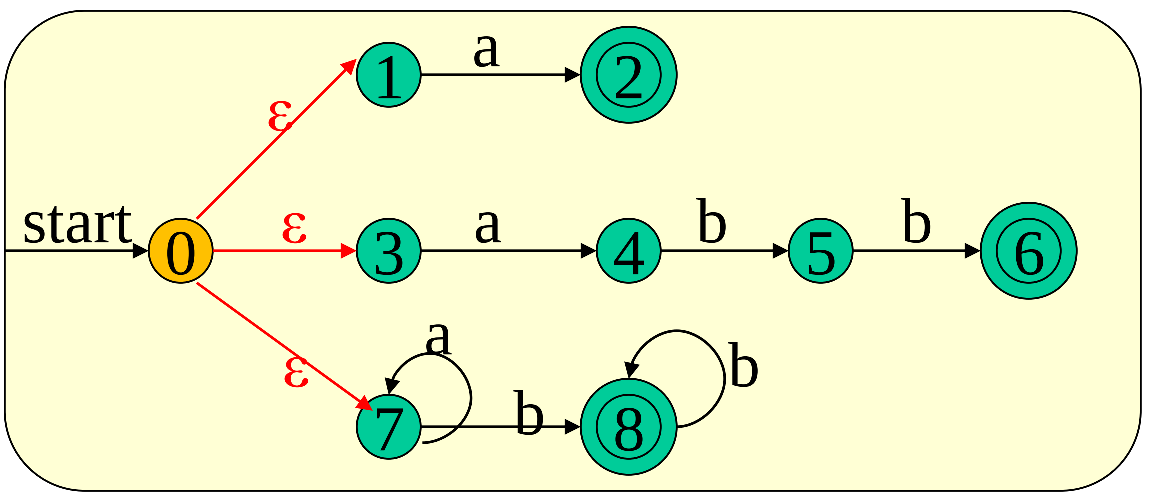
终态为{2,4,7},{8},{5,8},{6,8}

词法分析程序的分析过程

令输入字符串为aba…

读入字符	进入状态
开始	{0,1,3,7}

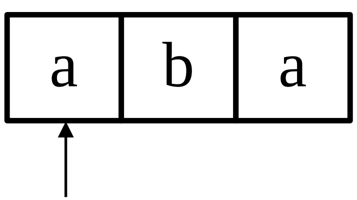


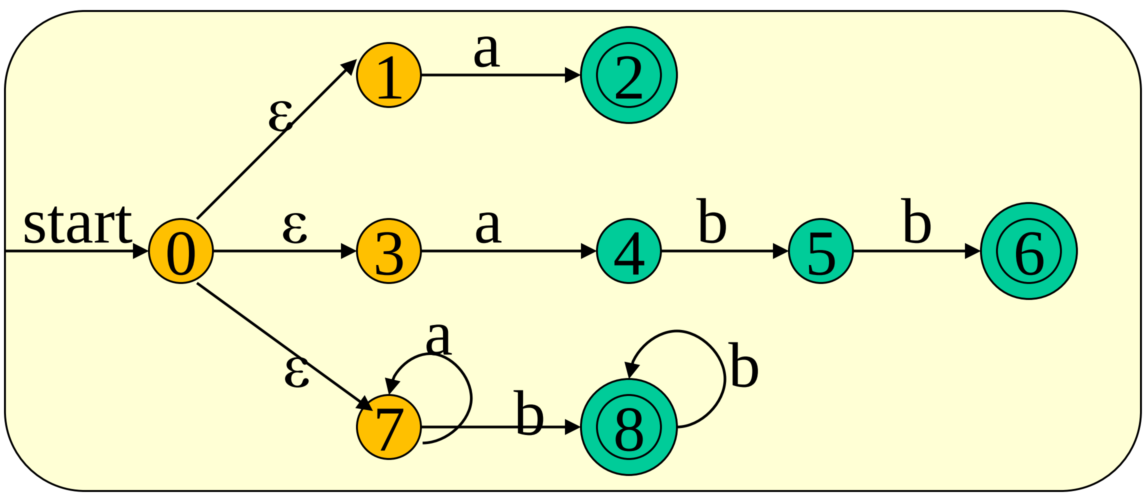


词法分析程序的分析过程

令输入字符串为aba…

读入字符	进入状态
开始	{0,1,3,7}



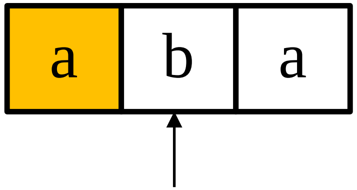


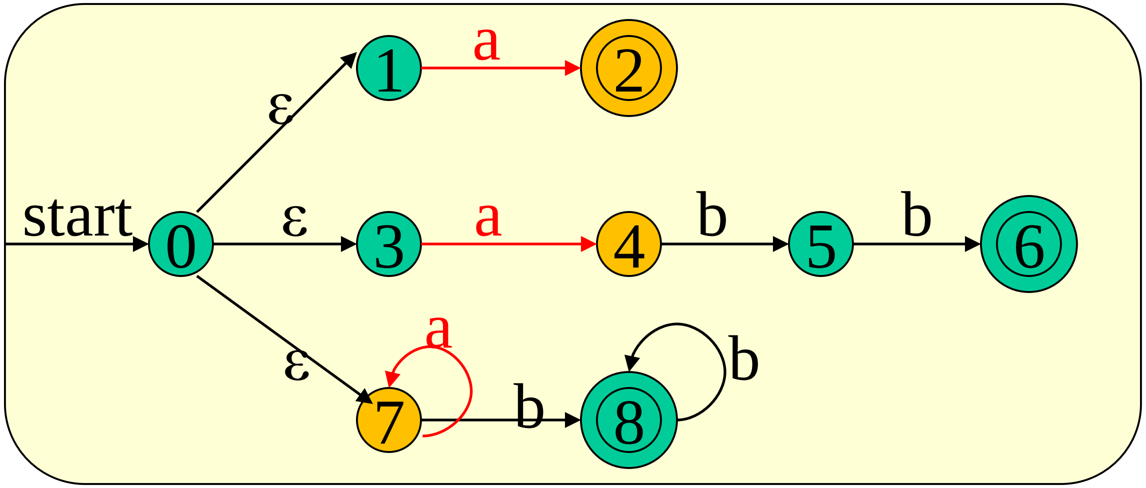
词法分析程序的分析过程

令输入字符串为abb…

(1) 吃进字符a

读入字符	进入状态
开始	{0,1,3,7}
a	{2,4,7}





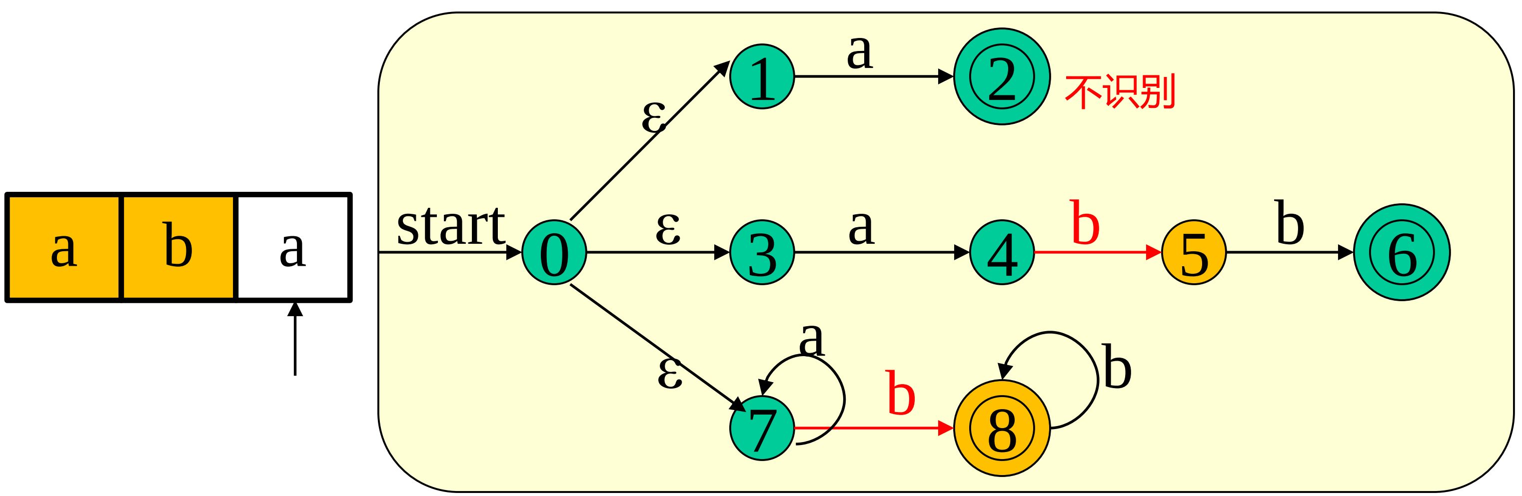
词法分析程序的分析过程

令输入字符串为aba…

(1) 吃进字符a

(2) 吃进字符b

读入字符	进入状态
开始	{0,1,3,7}
a	{2,4,7}
b	{5,8}



词法分析程序的分析过程

令输入字符串为aba…

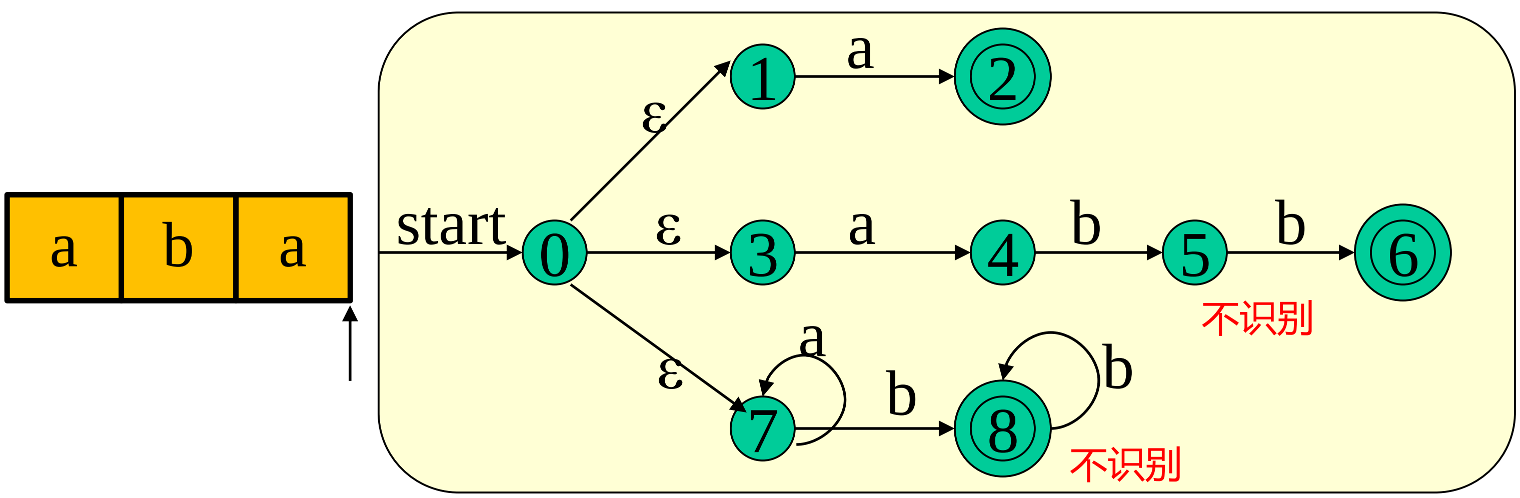
(1) 吃进字符a

(2) 吃进字符b

(3) 吃进字符a，不接受字符串

反序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)



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;

谢谢！