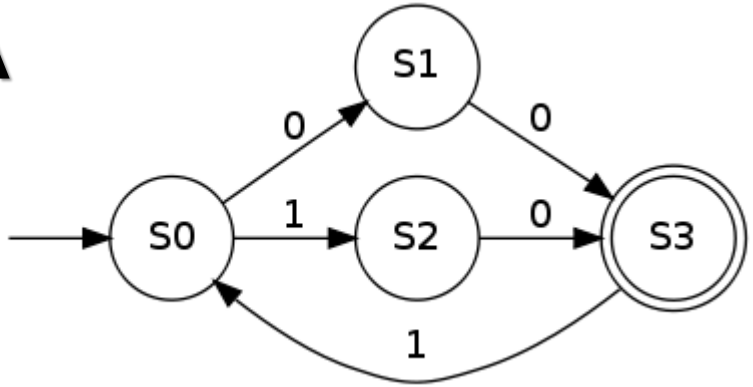
确定的有穷自动机(DFA)

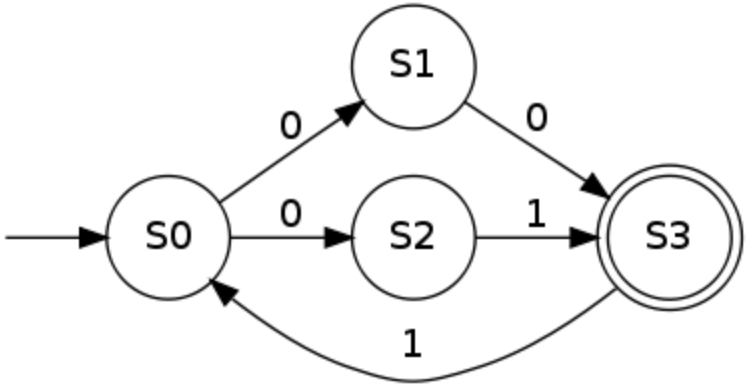
- **DFA(*Deterministic Finite Automata*)是NFA的特例**
  - **没有ε边**
  - **对每个状态s和每个输入符号*a*，有且仅有一条标号为*a*的边离开**

以下哪些是DFA，哪些是NFA？

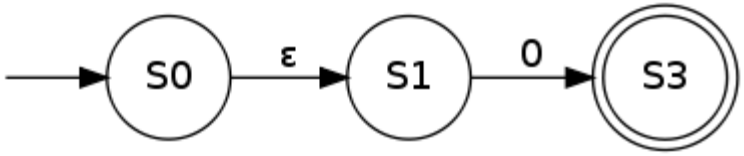
A



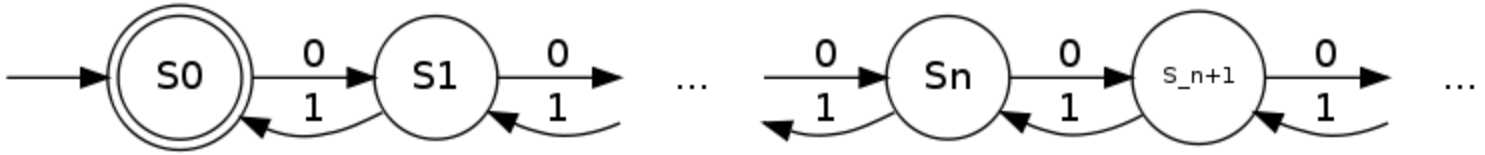
B

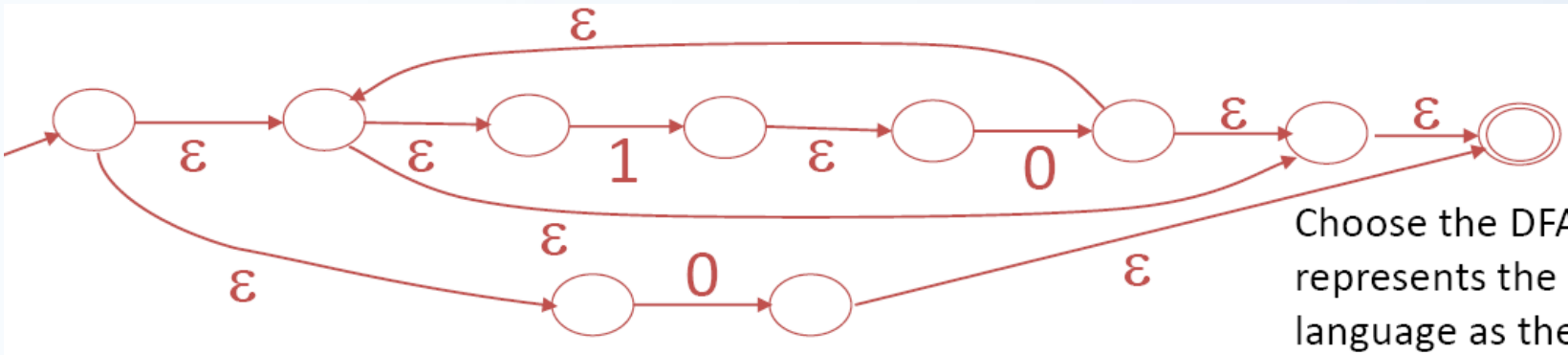


C



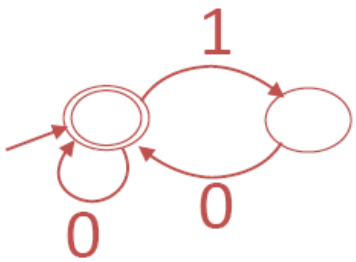
D



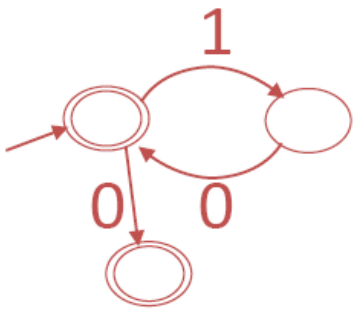


Choose the DFA that represents the same language as the given NFA

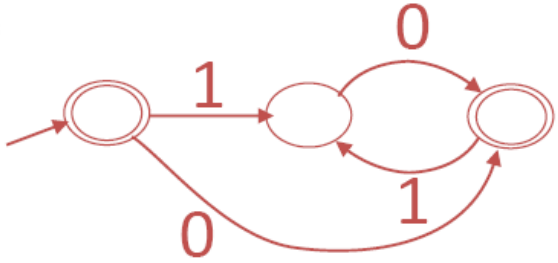
A ○



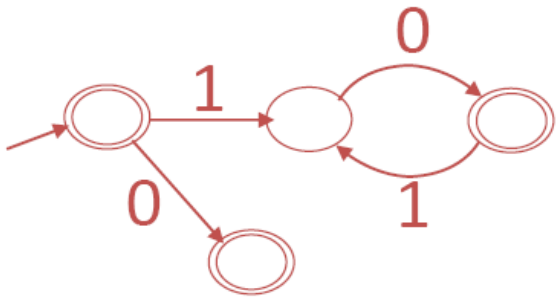
B ○



C ○



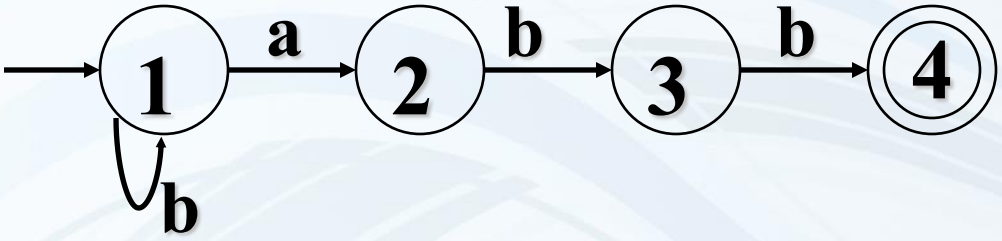
D ○



DFA的两种表示法

状态转换图：包括状态(圆圈)、边、初态、终态

状态转换表：描述各个状态下，对应于各个输入的状态迁移



	a	b
1	2	1
2	----	3
3	----	4
4	----	----

与NFA不同：

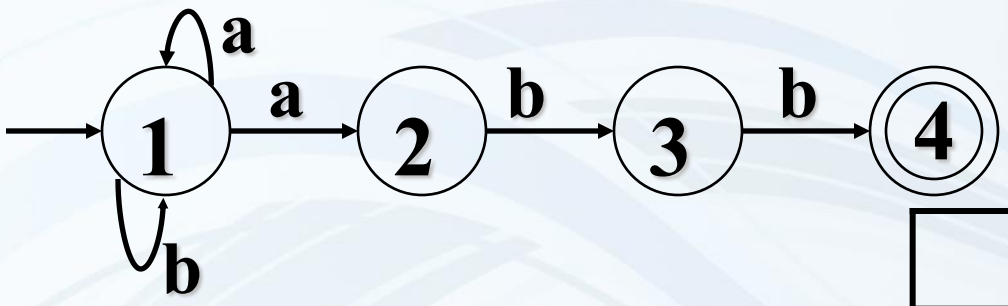
转向目标为单个状态（不是集合），

未定义状态转移表示成横线（不是空集）。

NFA的两种表示法（复习）

状态转换图：包括状态(圆圈)、边、初态、终态

状态转换表：描述各个状态下，对应于各个输入的状态迁移



状态转换图中无 ε 边时，状态转换表中可以忽略 ε 列。

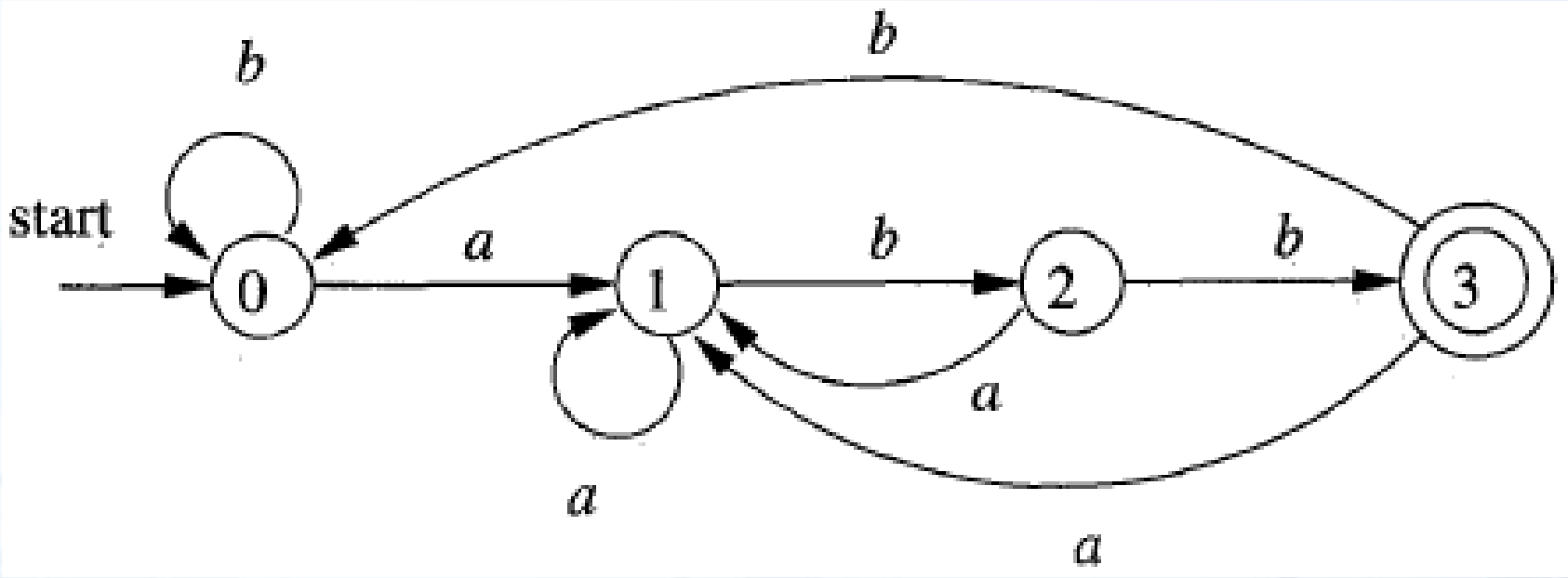
	a	b	ε
1	{1,2}	{1}	Ø
2	Ø	{3}	Ø
3	Ø	{4}	Ø
4	Ø	Ø	Ø

DFA的运行

DFA D: 初态s_0，终态集F，转移函数*move*。
输入: 以eof 结尾的输入串x。
输出: 当D接受x时输出yes ，否则输出no

```
s = s0;
c = nextchar( );
while ( c != eof ) {
    s = move(s, c);
    c = nextchar( );
}
if ( s is in F ) return "yes";  else return "no";
```

接受(a|b)*abb的DFA



NFA状态集上的操作

- ε-closure(s): $s \in S$　　状态s的ε闭包

从状态s出发，仅通过若干条（含0条）ε边就能到达的状态集合

- ε-closure(T): $T \subseteq S$　状态集T的ε闭包

从状态集T中任一状态出发，仅通过若干条（含0条）ε边就能到达的状态集

- *move*(T,a): $T \subseteq S$, $a \in \Sigma$　状态集T的转移函数

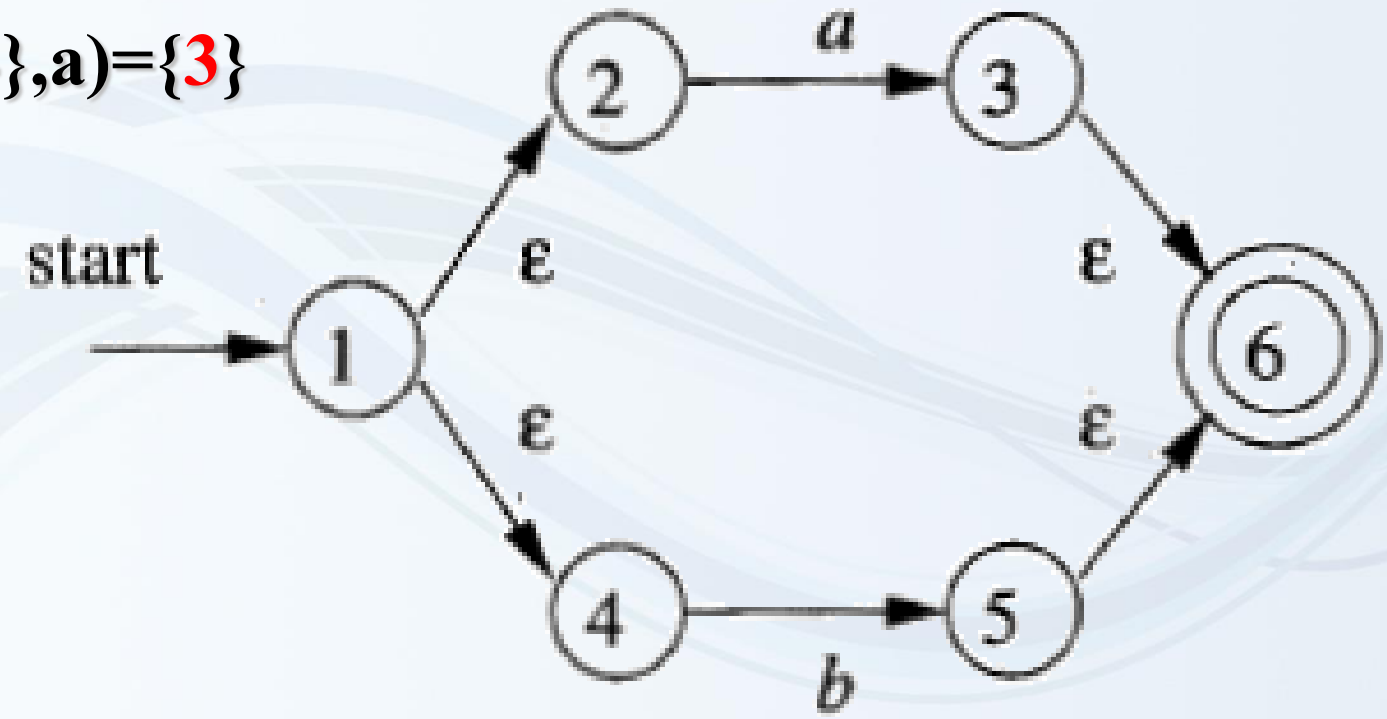
对于状态集T，在输入a时，可以到达的状态集合(不考虑ε边)

NFA状态集上的操作（例）

ε-closure(1)={1,2,4}

ε-closure({1,2,3})={1,2,3,4,6}

move({2,4},a)={3}



ε-closure(T)的计算（深度优先）

（1）初始化：将T中所有状态入栈; 同时置ε-closure(T) 为T

（2）循环，当栈非空时：

2a. pop栈顶 *t*;

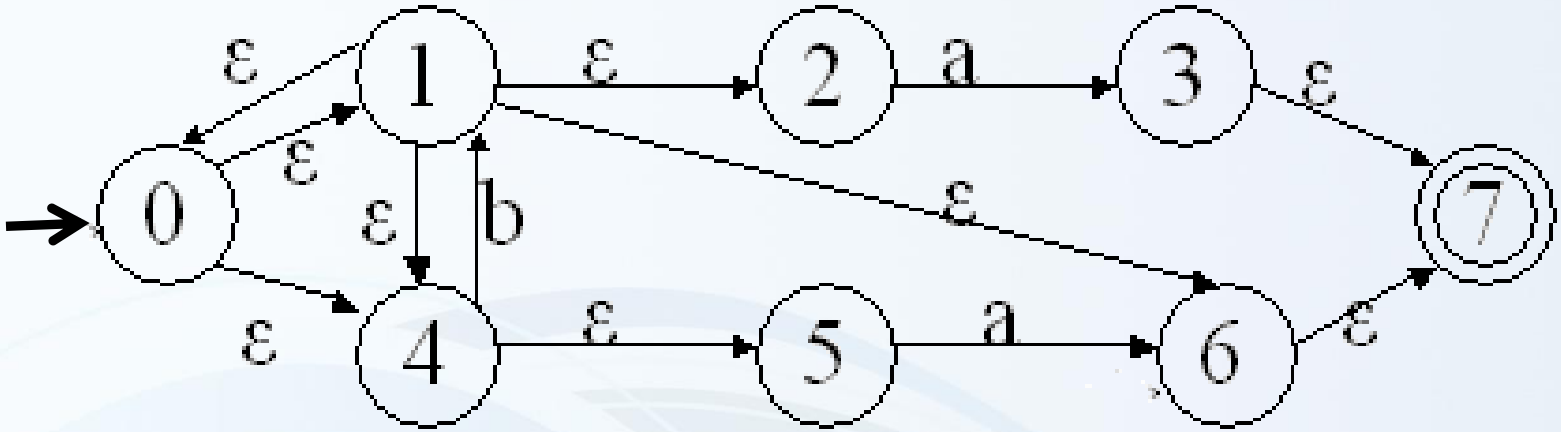
2b. 对于从 *t* 经过一条ε边可达的状态*u*：

如果 *u* 不在 ε-closure(T) 中，则

将 *u* 加入 ε-closure(T); 同时将 *u* 入栈

也可以基于队列，采用宽度优先来计算。

ε-closure(T)的计算(例）



ε-closure(0)={0,1,2,4,5,6,7}

move(ε-closure(0),a)=move({0,1,2,4,5,6,7},a)={3,6} 不需要扩展ε边

ε-closure(move(ε-closure(0),a))= ε-closure({3,6})={3,6,7}

3.7 从正则表达式到自动机

(1) 正则表达式到NFA

(2) NFA到DFA

(3) DFA的简化

由NFA构造DFA的子集构造算法

置ε-closure(s_0)为Dstates中的唯一状态（未标记）

若Dstates中存在未标记状态T，则：

标记T;

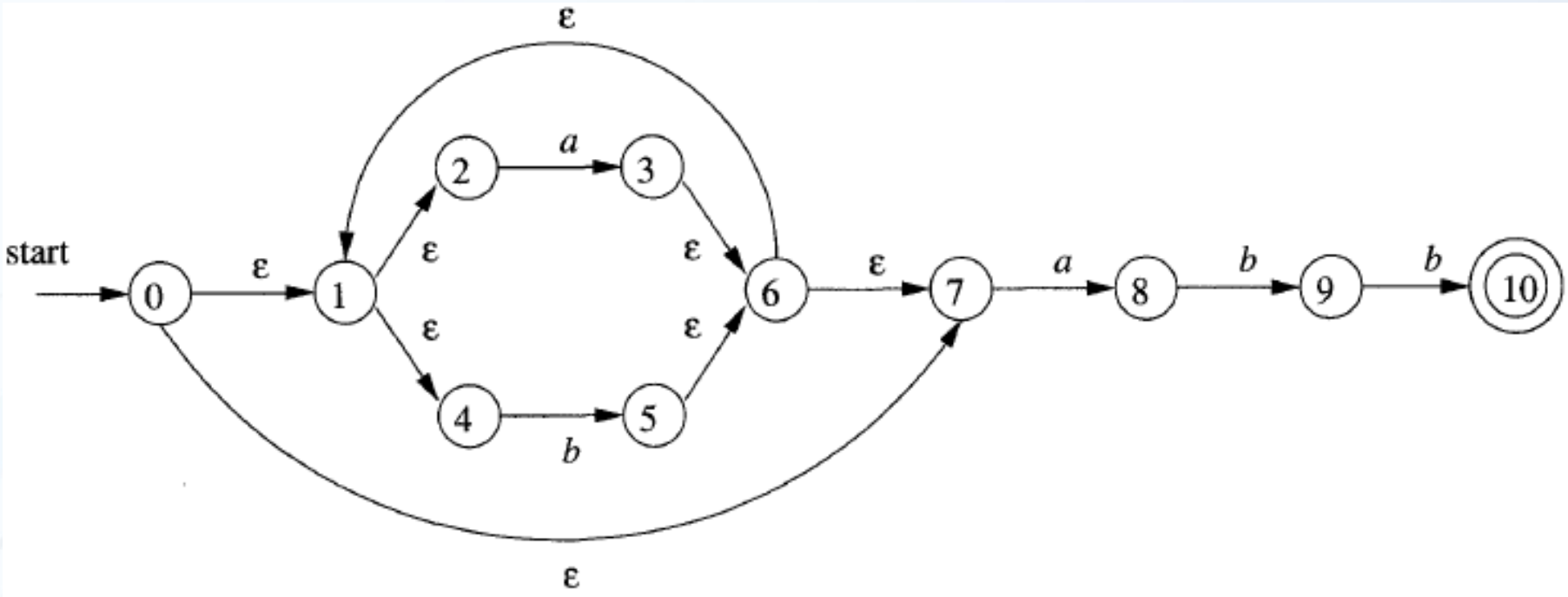
对于每个输入符号a

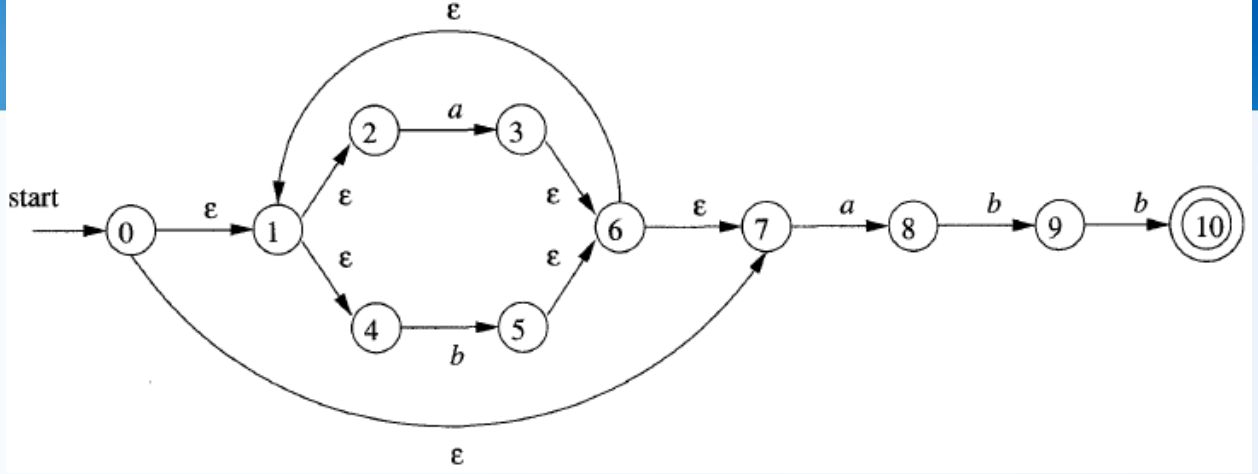
计算*U* =ε-closure(move(T,a));

若*U* 不在 Dstates中，则作为未标记状态加入

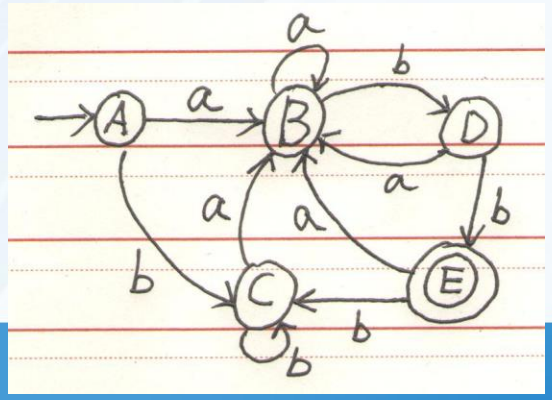
填入转换表对应项目：Dtran[T, a] = U;

由NFA构造DFA的子集构造算法(例)



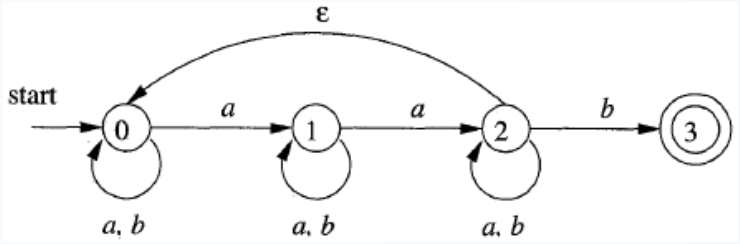


DFA	NFA	a	b
A	ε_closure(0)={0,1,2,4,7}	B	C
B	ε_closure({3,8})={1,2,3,4,6,7,8}	B	D
C	ε_closure(5)={1,2,4,5,6,7}	B	C
D	ε_closure({5,9})={1,2,4,5,6,7,9}	B	E
E*	ε_closure({5,10})={1,2,4,5,6,7,10}	B	C



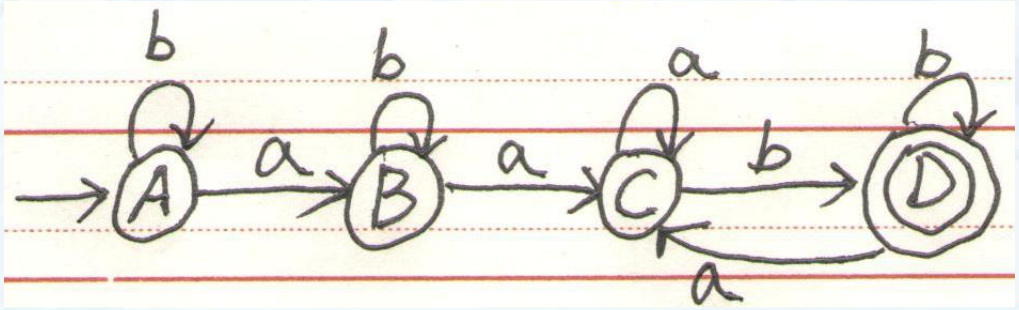
存在大量ε边时
（1）构建非ε边索引（稿纸）
（2）NFA给出ε闭包表示

由NFA构造DFA的子集构造算法(例2)



不存在大量ε边时的解法

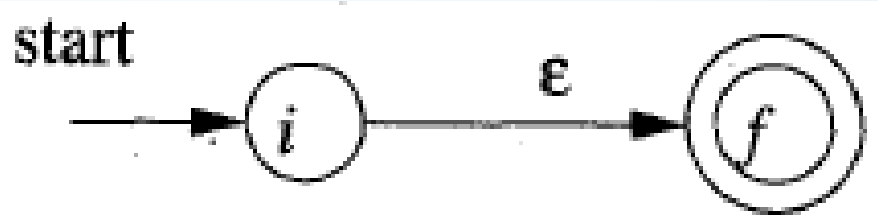
DFA	NFA	a	b
A	{0}	B	A
B	{0,1}	C	B
C	{0,1,2}	C	D
D*	{0,1,2,3}	C	D



由正则表达式构建NFA

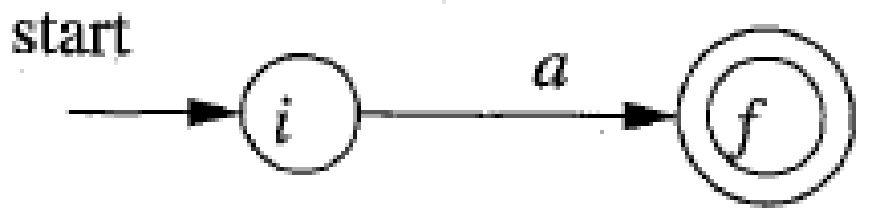
McNaughton-Yamada-Thompson算法

- 正则式ε对应 NFA



- 正则式a∈Σ, 对应NFA

增加2个状态，1条边

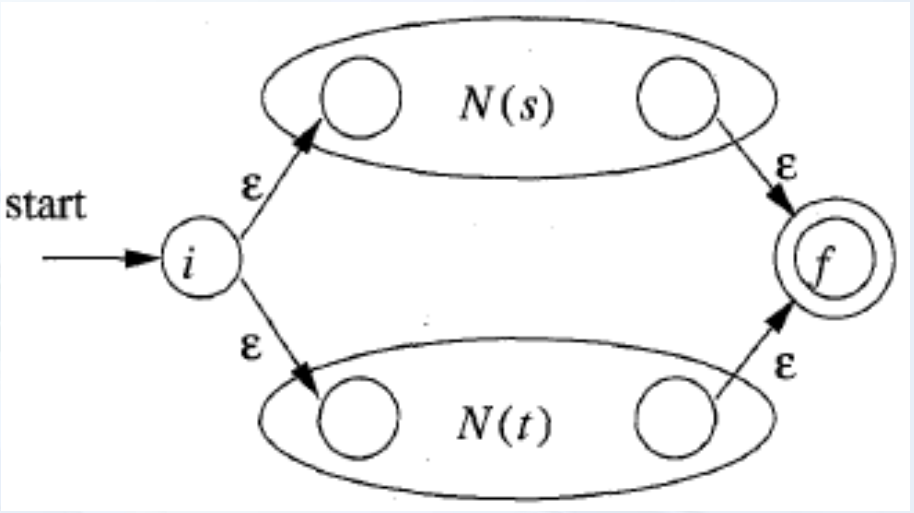


延伸阅读： Regular Expressions and State Graphs for Automata
https://ieeexplore.ieee.org/document/5221603
Programming Techniques: Regular expression search algorithm
https://dl.acm.org/doi/10.1145/363347.363387

由正则表达式构建NFA(II)

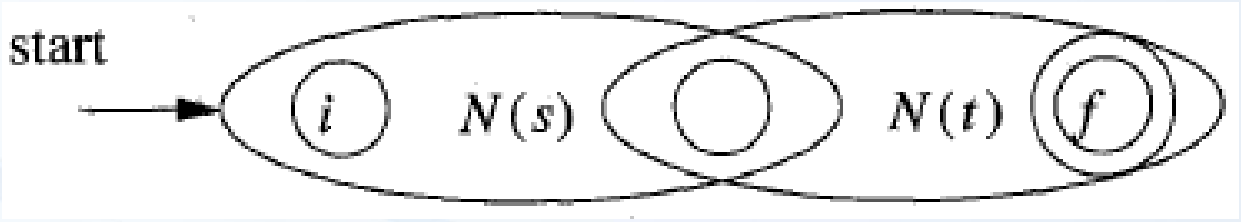
- 若s,t是正则式, N(s)和N(t)是对应的NFA，则s|t对应之NFA为:

增加2个状态，4条边



由正则表达式构建NFA(III)

- 若s, t为正则式，N(s)和N(t)是对应的NFA，则st对应之NFA为：

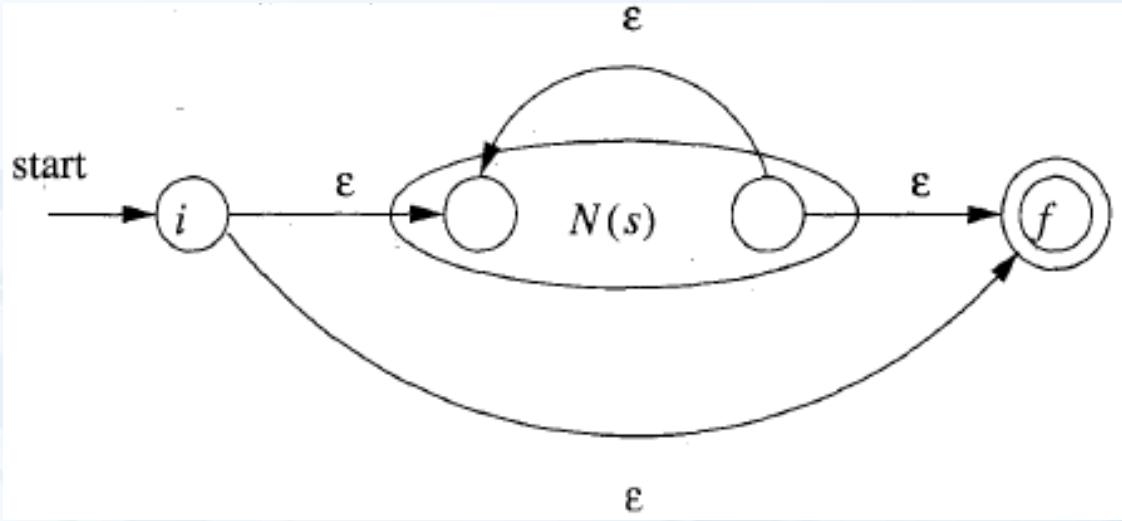


减少1个状态

由正则表达式构建NFA(IV)

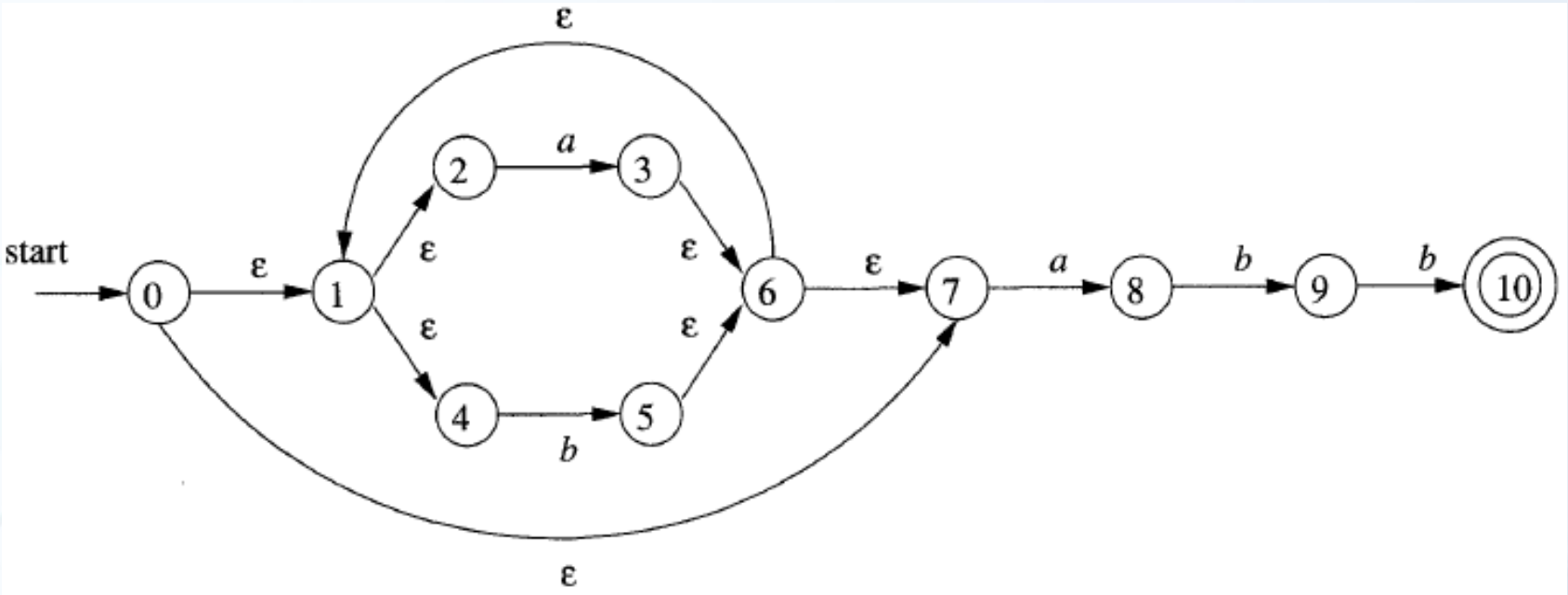
- 如果s是正则式, N(s)为对应NFA，则s*对应之NFA为:

增加2个状态，4条边



为(a|b)*abb构造一个NFA(II)

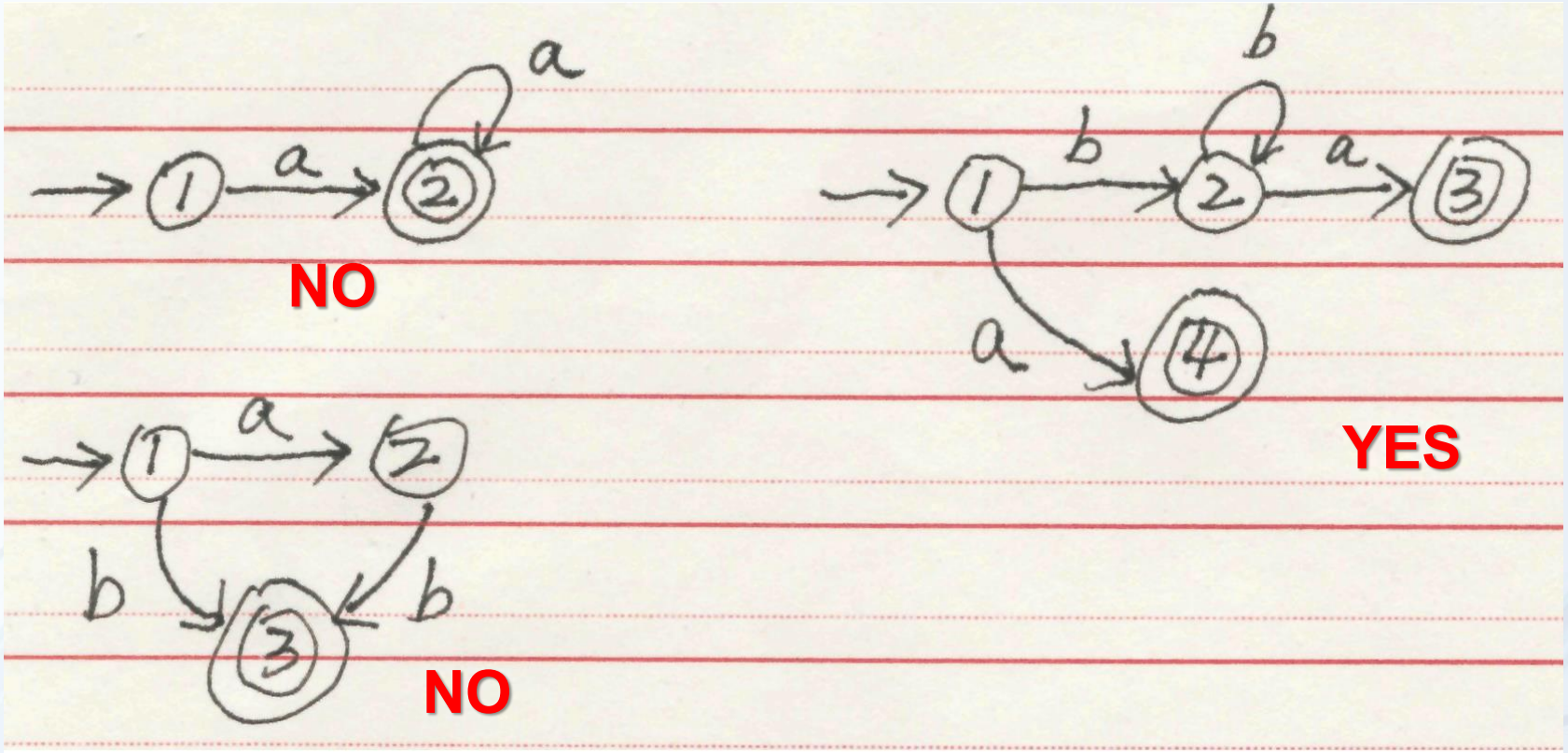
为(a|b)*abb构造一个NFA(II)



最小化DFA

- 自动机的等价：可以存在多个识别同一语言的DFA，这些DFA等价。
  - 同构：如果仅需变换状态的名字就可以将一个DFA变成另一个，则这两个DFA同构。
- 两个状态的等价：同时满足以下条件
  - 同时是或不是接受状态（终态）
  - 对任意的输入，总是转向同一状态或等价状态
- 任何正则语言都有唯一的状态数最少的DFA
- 通过合并等价状态，可以得到状态数最少的DFA

状态1和2是否等价？



最小化DFA（2）

- 工作原理：将状态集分划成多组，每组中的各个状态不可区分。然后，将各组中的状态合并成一个状态。
- 过程：

1.首先划分成两组：终结状态集F和非终结状态集S-F.

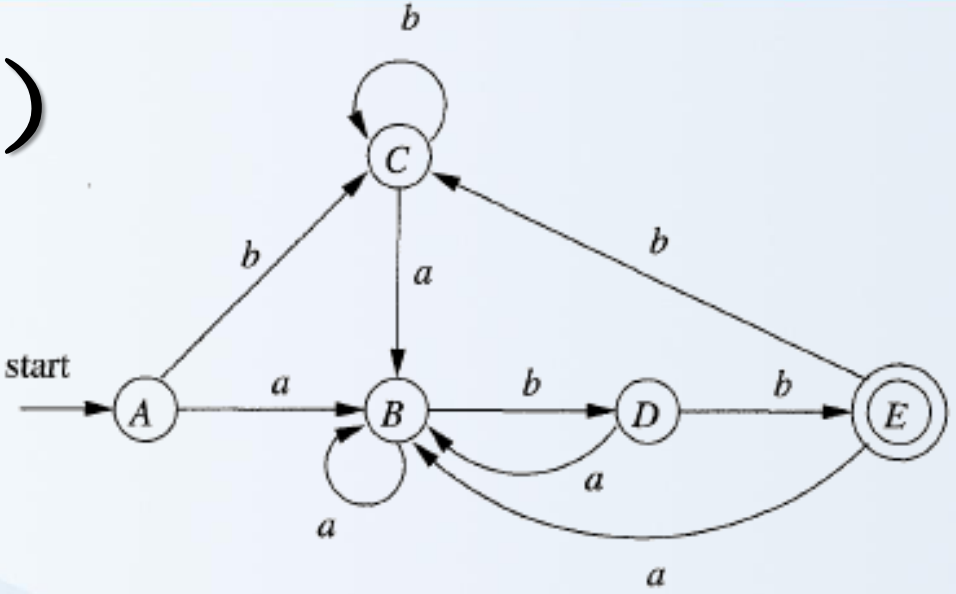
2. 循环直到不可继续细分

考察当前划分中的每一组G：

如果有必要，将G划分成若干子组，使各子组中的任意两个状态s和t对于给定输入均转向当前划分中的同一组

3. 每组状态用一个状态代表，原DFA中所有指向该组中某状态的边，都要与该代表状态关联。

最小化DFA（例）



(1)将状态集划分为g_1={A,B,C,D}和g_2={E}

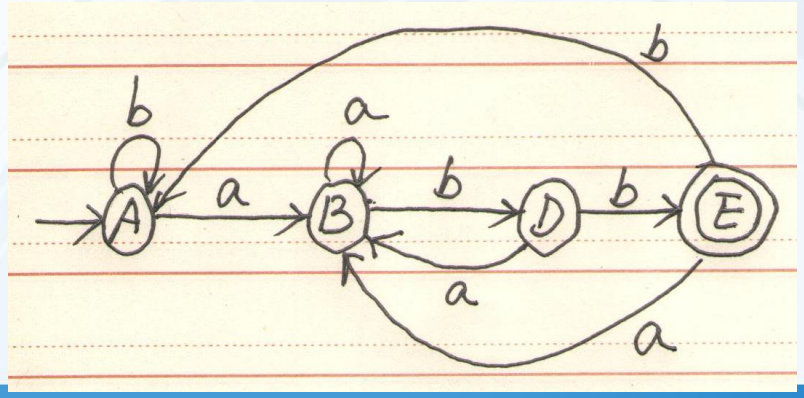
(2)将g_1划分为g_3={A,B,C}和g_4={D}

(3)将g_3划分为g_5={A,C}和g_6={B}

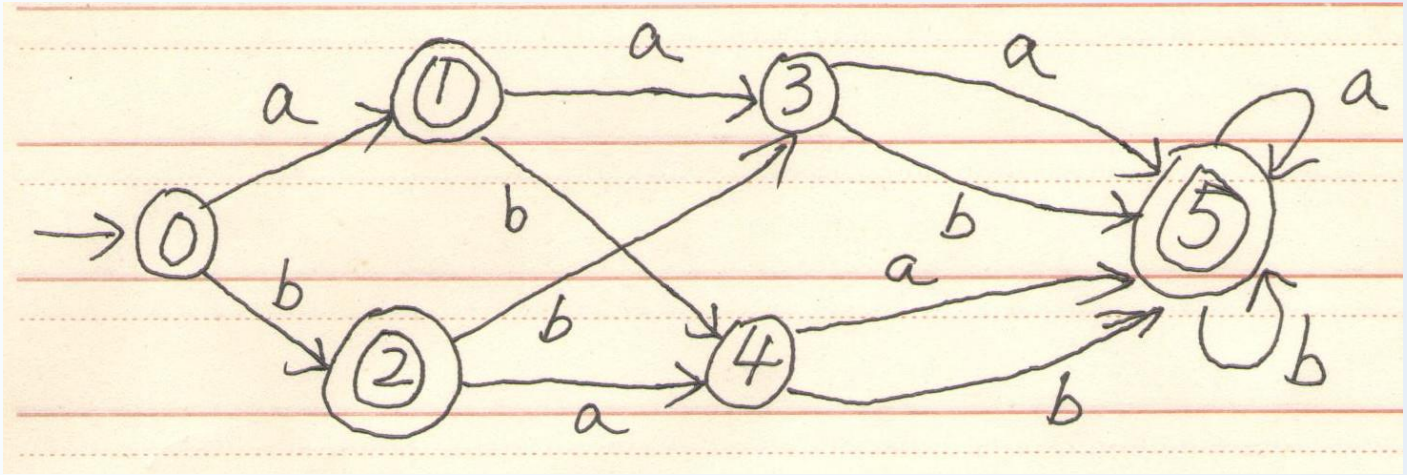
(4)不可继续划分，故所得划分为：

g_2={E}, g_4={D}, g_5={A,C}, g_6={B}

合并等价状态，得最小DFA



例2



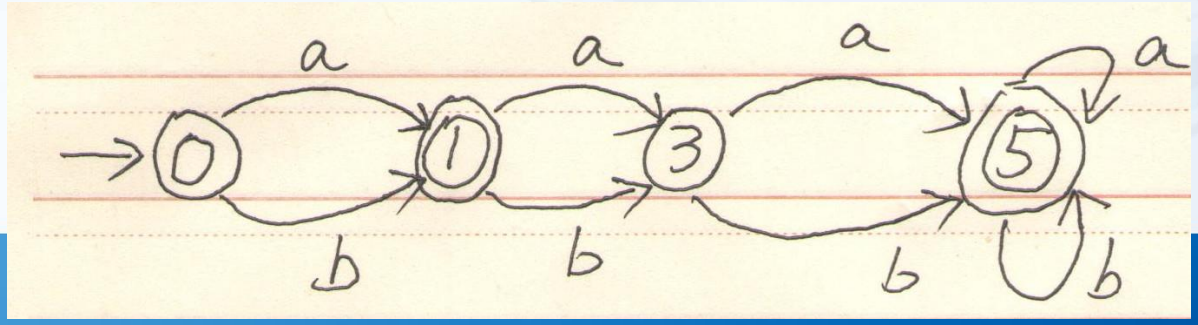
（1）将状态集划分为g_1={0,3,4}, g_2={1,2,5}

（2）将g_2划分为g_3={1,2}, g_4={5}

（3）将g_1划分为g_5={0}，g_6={3,4}

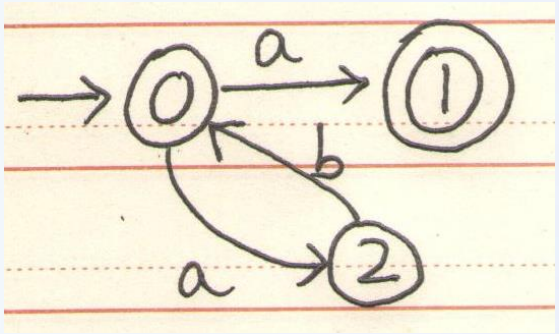
不可继续划分，故所得分划为：g_3={1,2}, g_4={5}, g_5={0}，g_6={3,4}

合并等价状态，得最小DFA



例：试说明正则式$(ab)^*a$和$a(ba)^*$是否等价。

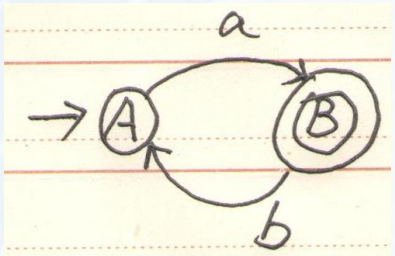
解：（1） $(ab)^*a$对应的NFA如右



确定化此NFA

DFA	NFA	a	b
A	{0}	B	---
B*	{1,2}	---	A

即所得DFA如下，显然DFA已最简化



说明：

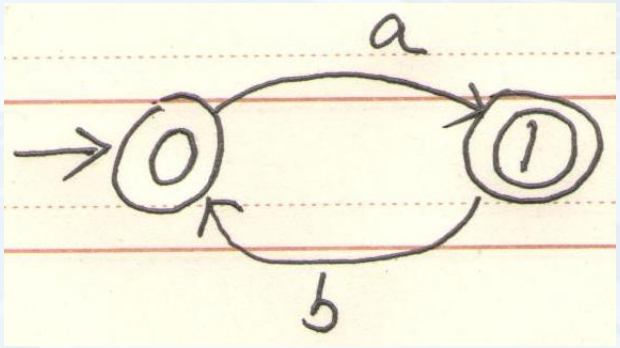
（1）正则式对应最小DFA唯一

（2）可以通过对应最小DFA是否等价来判断两个正则式是否等价

例：试说明正则式$(ab)^*a$和$a(ba)^*$是否等价。

解：（2）$a(ba)^*$对应的NFA如下

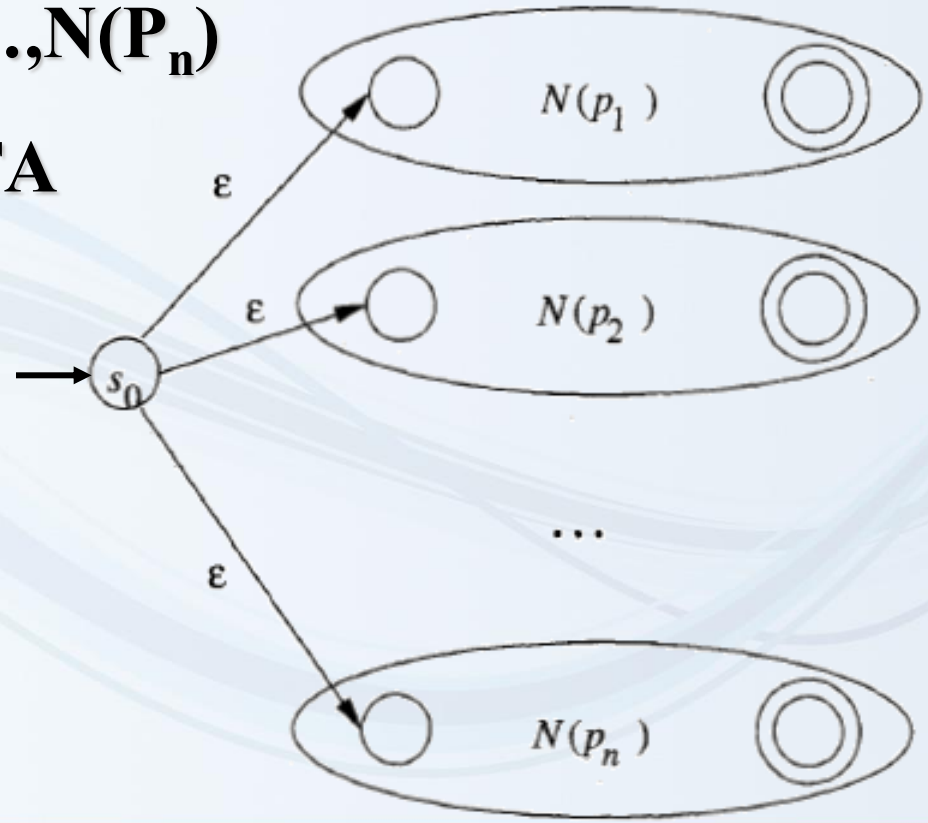
显然此NFA已确定化，状态0和2为等价状态，合并后，得最小化的DFA如下



比较（1）（2），知$(ab)^*a$和$a(ba)^*$对应的最简DFA等价，故两正则式等价。

Lex描述 → 词法分析器

- 给出LEX模式$P_1, P_2,\ldots,P_n$(描述记号的正则式)
- 分别构建 $N(P_1), N(P_2), \ldots,N(P_n)$
- 集成各子NFA得最终 NFA
- 确定化得DFA，并简化



有多个含义不同的终态

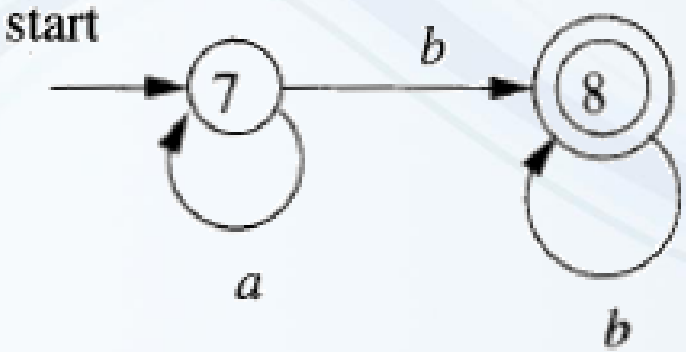
例子

a

abb

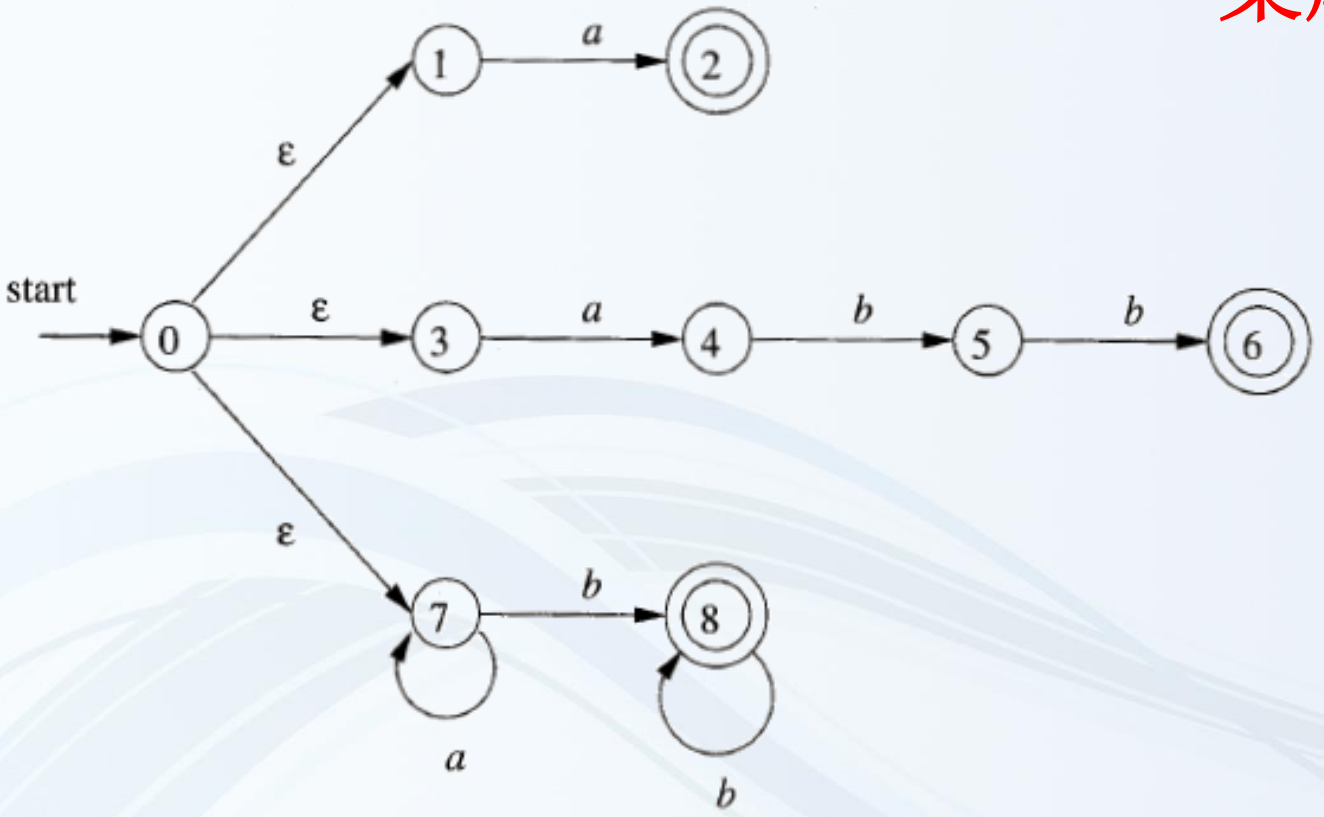
a*b$^+$

分别给出各个正则式对应的**NFA**

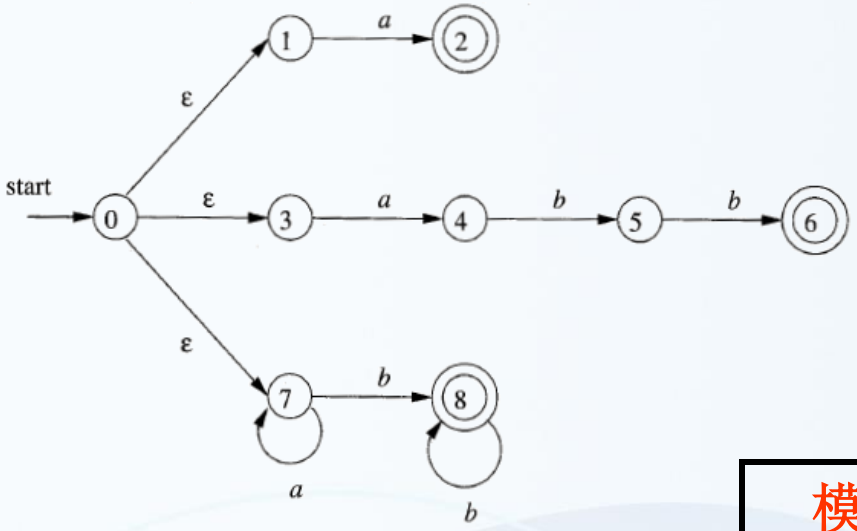


例子（续1）

集成NFA



3 个终态的含义有何不同？



确定化NFA，得DFA

模式	DFA	NFA	a	b
none	A	{0,1,3,7}	B	C
a	B	{2,4,7}	D	E
a^*b^+	C	{8}	---	C
none	D	{7}	D	C
a^*b^+	E	{5,8}	---	F
abb	F	{6,8}	---	C

注意各个终态对应的记号（模式）！