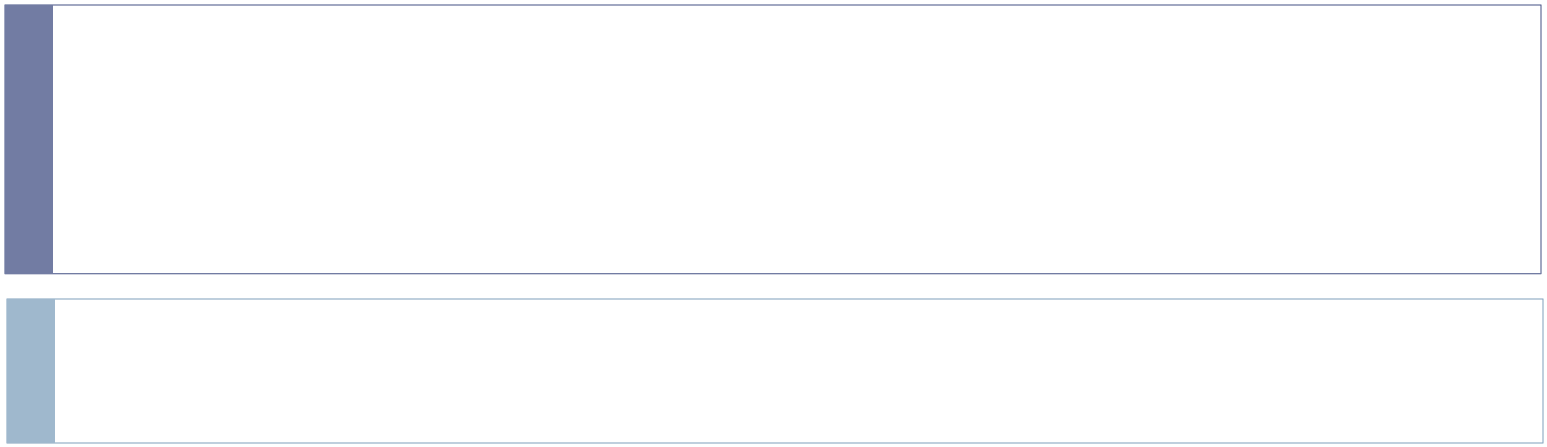
Chapter 3 词法分析



Outlines

- 词法分析器的要求
- 词法分析的设计
- 正规表达式 Regular Expressions
- 有限自动机 Finite Automata
- 词法分析器的自动产生

Regular expression and Finite Automata

- 核心内容
  - 正规式和正规集的递归定义
  - 确定有限自动机（Deterministic FA）
  - 非确定有限自动机（Nondeterministic FA）
  - 正规文法与有限自动机的等价性
  - DFA的化简
- **正规式**
  - 正则表达式（regular expression）
  - 说明单词模式的一种重要的表示法
  - 定义正规集的数学工具
  - 用于描述单词符号
    - 一个字集是正规集当且仅当它能用正规式表示
    - 字也叫字符串

Regular expression

- 正规集：**有穷字母表∑的一些特殊字集**
  - 字符
    - ∑上的元素
  - ∑上的字
    - 由∑中的字符所构成的一个有穷序列
  - 空字
    - 不包含任何字符的序列
    - 记为ε
  - $\sum^*$（∑的闭包）
    - ∑上的所有字的全体
    - 包含ε

设 ∑={a,b}，则

$$\sum^* = \{\epsilon, a, b, aa, ab, ba, \ldots\}$$

Regular expression

- $\sum^*$ 的子集 U , V:
  - 积 $UV = \{\alpha\beta | \alpha \in U \ \& \ \beta \in V\}$
  - n次积 $V^n = VV \ldots V$
  - $V^0 = \{\epsilon\}$
  - V的闭包 $V^* = V^0 \cup V^1 \cup V^2 \cup \cdots$
  - V的正则闭包 $V^+ = VV^*$

Regular expression

- 正规式与正规集的递归定义：
- 1、ε和Φ都是字母表∑上的正规式，它们所表示的正规集分别为{ε}和Φ；
- 2、任何a∈∑，a是∑上的一个正规式，它所表示的正规集为{a}；
- 3、 正规式　　正规集　　正规式　　正规集
- 　　U　　L(U)　　(U | V)　　L(U)∪L(V)
- 　　V　　L(V)　　(U·V)　　L(U)L(V)
- 　　(U)*　　L(U)*（闭包）

Regular expression

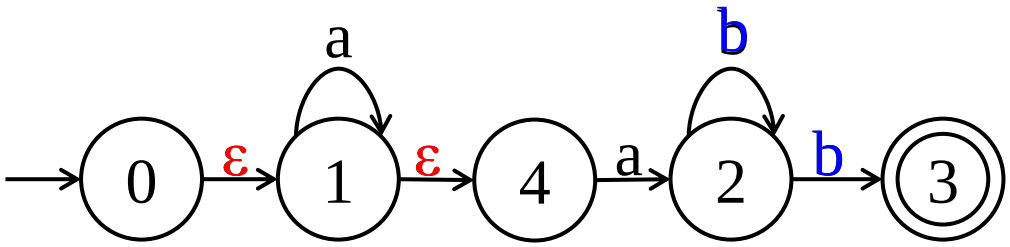
- 仅由有限次使用上述三步骤而得到的表达式才是∑上的正规式。仅由这 些正规式所表示的子集才是∑上的正规集
- 运算符的优先顺序：先“*”，次“·”，最后“|”
- 一个字集合是正规集当且仅当它能用正规式表示

DFA & NFA

- **确定有限自动机(DFA) M是一个五元式**
  - **M = (S, Σ, f, S_0, Z)**
    - 有穷状态集S
    - 有穷的输入字母表Σ，每个元素称为一个输入字符
    - 状态转换函数f，f(s,a) = s’
    - $S_0 \in S$是唯一的初态
    - 终态集Z，$Z \subseteq S$，可空
- NFA非确定有限自动机
  - 状态转换函数f是一个$S \times \Sigma^* \rightarrow 2^S$的映射
    - 一个状态在一个输入下，可以转换到多个状态
    - 输入可以是空串
  - 初始状态是一个集合
    - 可以增加一个新初态，到原来的旧初态用空串连接

NFA

- **NFA识别某个串：**
  - **存在某条路径，路径上所有符号连接之后是这个串**
  - **按输入在图中进行转换，看最后是否停留在终止状态**

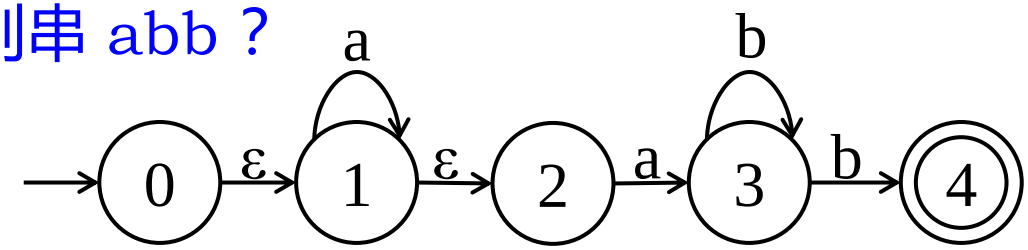


输入字符存在空串

同一个状态在同一个输入
可能转换到两个不同状态

NFA

- 能否识别串 abb？



定义：列出所有路径 $x_0x_1\ldots x_n$，满足

$$(x_0x_1)(x_1x_2)\ldots(x_{n-1}x_n) = abb$$

看这些路径是否有某条路径的x_n为终止态

⇩

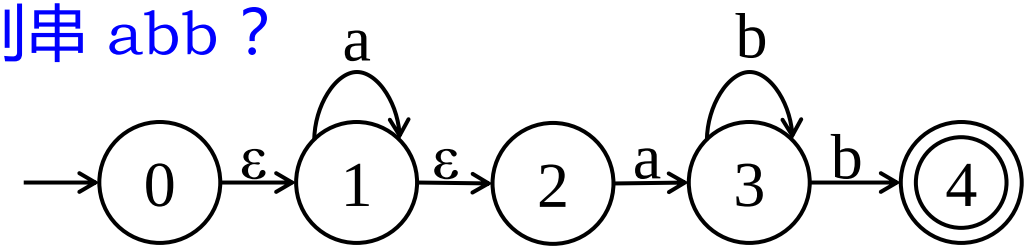
所有满足条件的路径的最后一个状态x_n
组成集合S(abb)
S(abb)是否有终止态？

⇩

如何计算S(abb)？

NFA

- 能否识别串 abb？

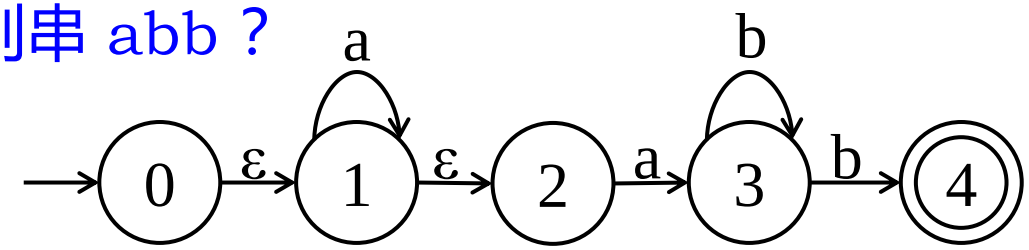


如何计算S(a)?

路径	串	最终状态
011	εa	1
0112	εaε	2
0123	εεa	3

NFA

- 能否识别串 abb？



路径为a的串，可以写成**ε*aε***的形式

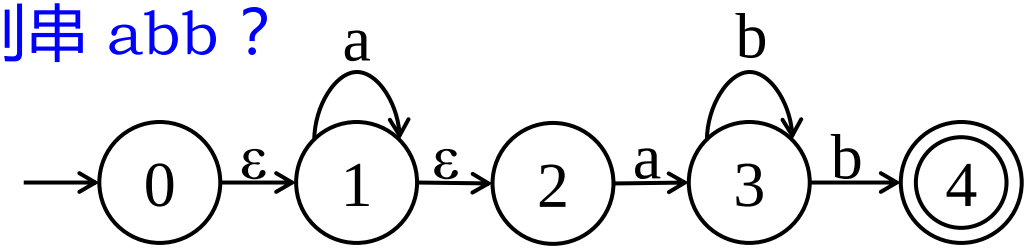
转移	意义	状态集
ε*	从0出发，使用0次或多次ε能到的状态	{0,1,2}
a	从{0,1,2}出发，使用一次a能到的状态	{1,3}
ε*	从{1,3}出发，使用0次或多次ε能到的状态	{1,2,3}

Move({0,1,2}, a)

ε-closure({1,3})

NFA

- 能否识别串 abb？

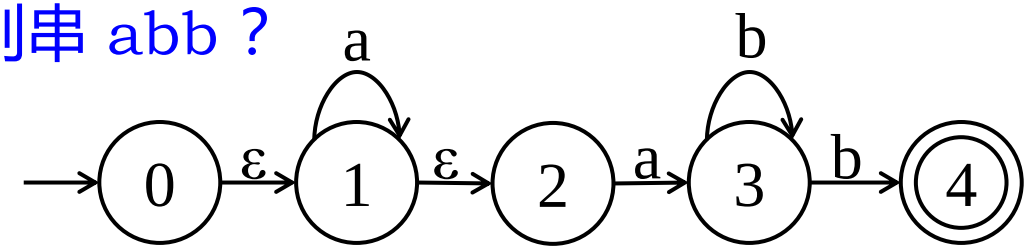


如何计算S(a)？

值	计算方式	状态集
S(ε)	ε-closure({0})	{0,1,2}
	Move({0,1,2},a)	{1,3}
S(a)	ε-closure({1,3})	{1,2,3}

NFA

- 能否识别串 abb？



有一个初始状态　　如何计算S(abb)？

值	计算方式	状态集
S(ε)	ε-closure({0})	{0,1,2}
	Move({0,1,2},a)	{1,3}
S(a)	ε-closure({1,3})	{1,2,3}
	Move({1,2,3},b)	{3,4}
S(ab)	ε-closure({3,4})	{3,4}
	Move({3,4},b)	{3,4}
S(abb)	ε-closure({3,4})	{3,4}

从一个状态集向另一个状态集转移

根据是否包含终止状态来判定是否接收

NFA subset construction

- 算法思路：
  - 从NFA的矩阵表示中可以看出，表项通常是一状态的集合，而在DFA的矩阵表示中，表项是一个状态。
  - NFA到相应的DFA的构造的基本思路是：
    - **DFA的每一个状态对应**NFA的一组状态。
    - DFA使用它的状态去记录在NFA读入一个输入符号后可能达到的所有状态。
  - 操作函数
    - ε-closure(I) ⇨求解连通子图
    - Move(I, a) ⇨对每一个状态，计算a转移后的状态集，然后取并集

NFA subset construction

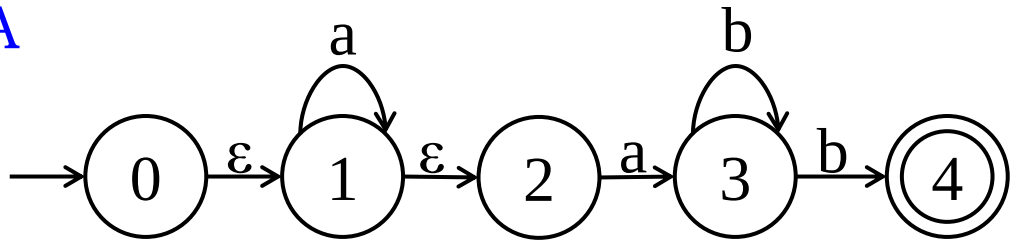
- NFA N=(K，∑，f，K_0，K_t) ->

  DFA M=(S，∑，d，S_0，S_t)，使得L(M)=L(N)：
  - 1. 初始化，S={}，∑相同，S_0和S_t暂不清楚，但S中的任意一个状态均与K的一个子集相对应。
  - 2. 添加第一个状态$S_0 = \varepsilon - closure(K_0)$，并将$S_0$添加到待计算d的列表D中
  - 3. 从D中选取一个需要计算转移的状态S_i，对∑上的每个字母a都计算转移函数和状态

    $$S_j = d(K_i, a) = \varepsilon - closure(Move(S_i, a))$$
  - 如果$d(K_i, a)$没出现过，添加新状态$S_j = d(K_i, a)$到状态集S和待计算列表D中，如果S_j中有终止状态，添加到S_t
  - 4. 重复3，直到D为空列表

NFA

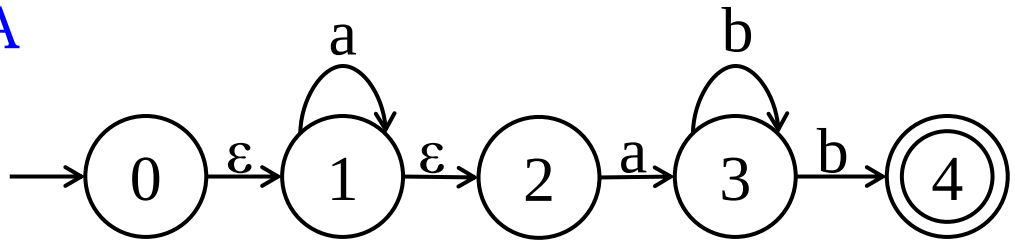
- 构造DFA



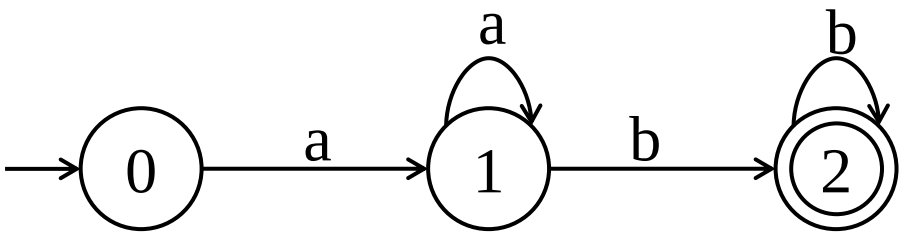
D	next	计算	备注
{}	初始化	$S_0 = \{0,1,2\}$	S_0为初始态
$\{S_0\}$	S_0	$S_1 = d(S_0, a) = \{1,2,3\}$ $d(S_0, b) = \emptyset$	
$\{S_1\}$	S_1	$d(S_1, a) = \{1,2,3\}$ $S_2 = d(S_1, b) = \{3,4\}$	S_2为终止态
$\{S_2\}$	S_2	$d(S_2, a) = \emptyset$ $d(S_2, b) = \{3,4\}$	
{}	停止		

NFA

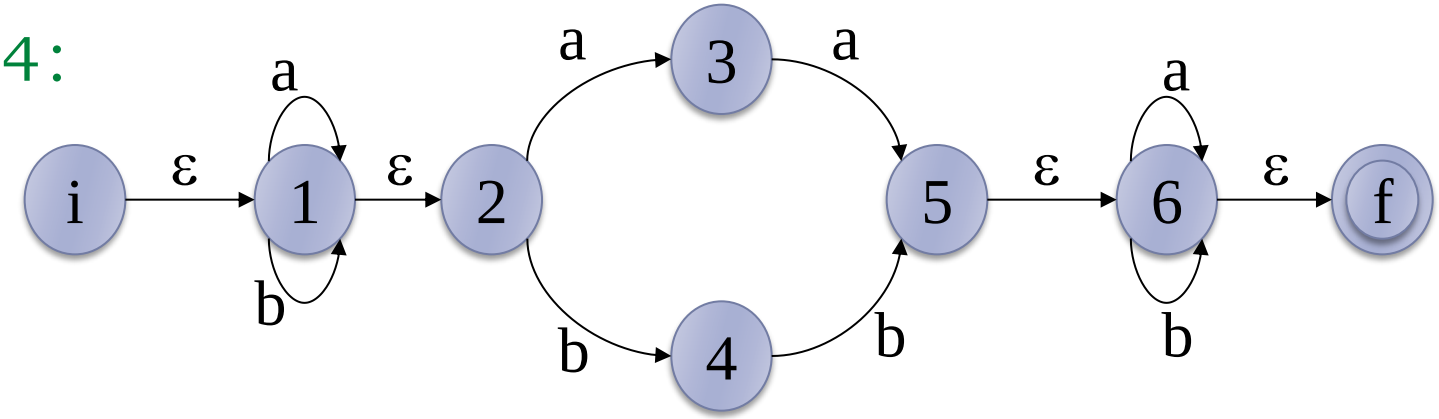
- 构造DFA



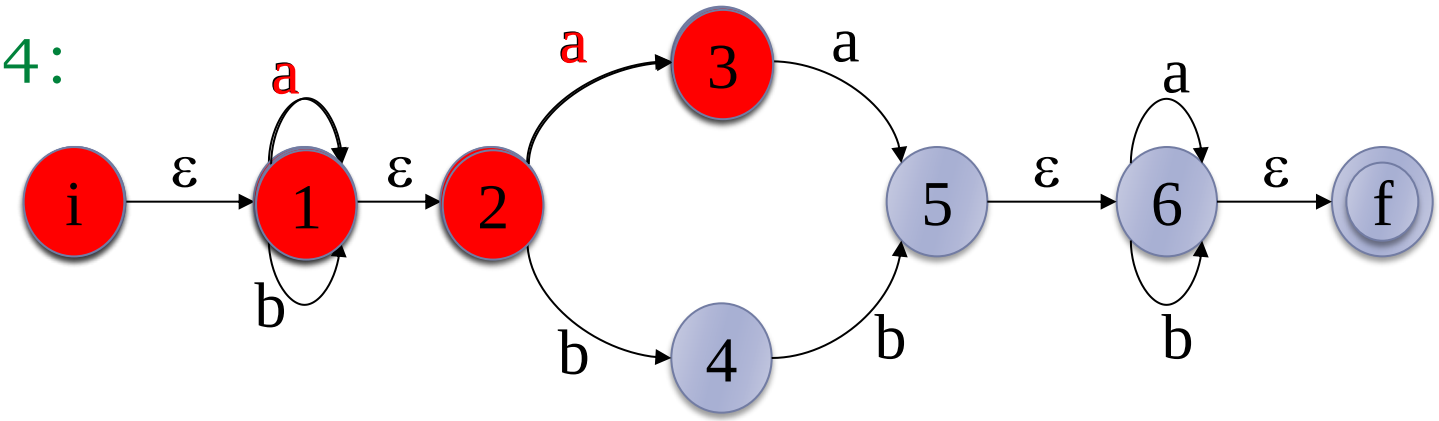
D	d(S$_i$, a)	d(S$_i$, b)
$\{0,1,2\} \rightarrow S_0$	$\{1,2,3\} \rightarrow S_1$	$\emptyset$
S_1	$\{1,2,3\} \rightarrow S_1$	$\{3,4\} \rightarrow S_2$
S_2	$\emptyset$	$\{3,4\} \rightarrow S_2$



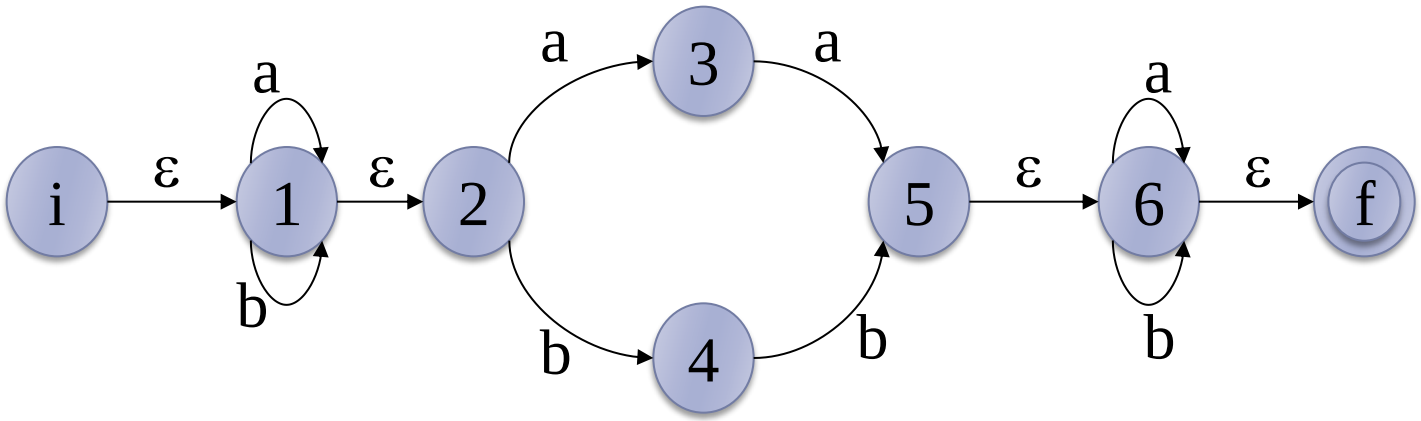
例14:



例14：

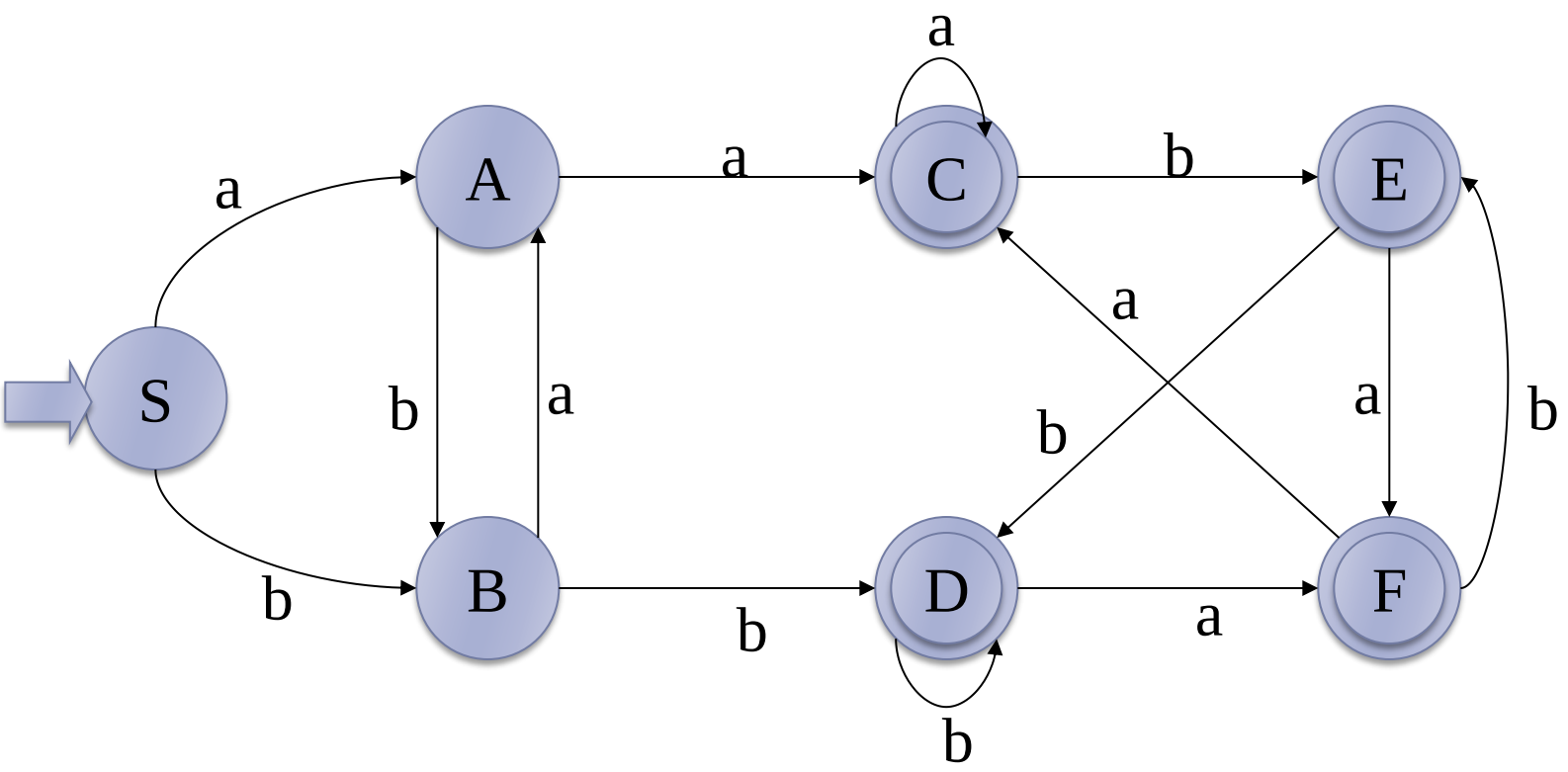


I	{i}	
$J = \epsilon\_closure(I)$	{i, 1, 2}	新状态1
$K = move(J, a)$	{1, 3}	
$L = \epsilon\_closure(K)$	{1, 2, 3}	新状态2



	I_a	I_b
{i,1,2} S	{1,2,3} A	{1,2,4} B
{1,2,3} A	{1,2,3,5,6,f} C	{1,2,4} B
{1,2,4} B	{1,2,3} A	{1,2,4,5,6,f} D
{1,2,3,5,6,f} C	{1,2,3,5,6,f} C	{1,2,4,6,f} E
{1,2,4,5,6,f} D	{1,2,3,6,f} F	{1,2,4,5,6,f} D
{1,2,4,6,f} E	{1,2,3,6,f} F	{1,2,4,5,6,f} D
{1,2,3,6,f} F	{1,2,3,5,6,f} C	{1,2,4,6,f} E

NFA subset construction



Excercise

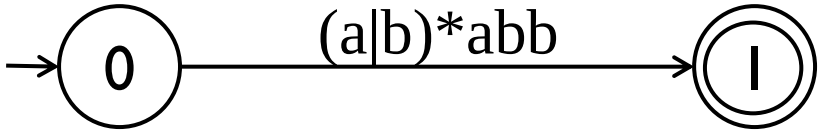
- 例17:

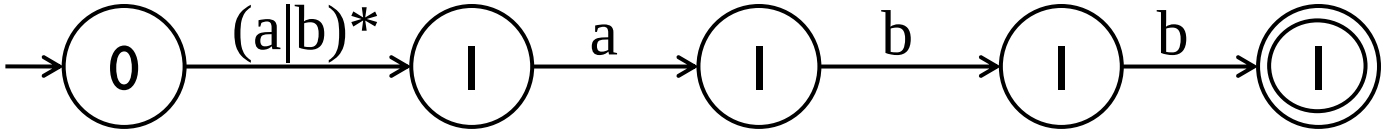
- （1）L(R) =(a|b)*abb，构造DFA M使L(M)=L(R)

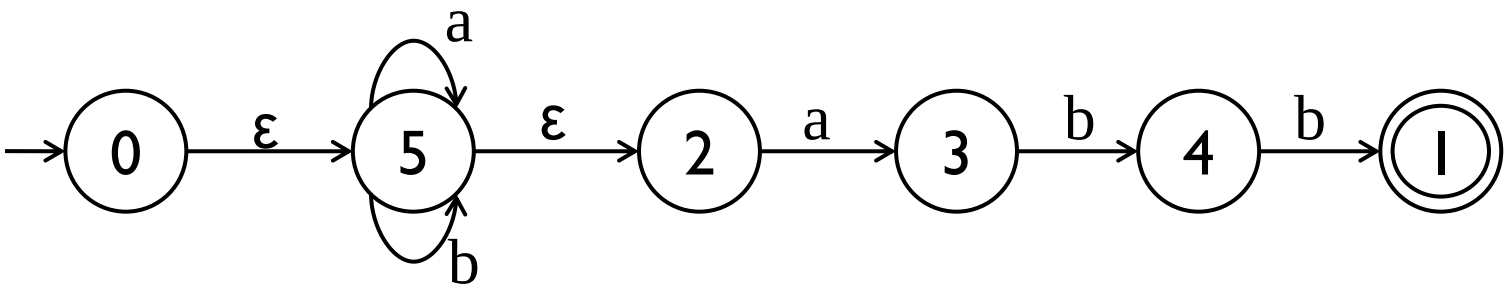
- （2）L(R) =a*b*abb，构造DFA M使L(M)=L(R)

Ex

- (a | b)*abb -> NFA -> DFA

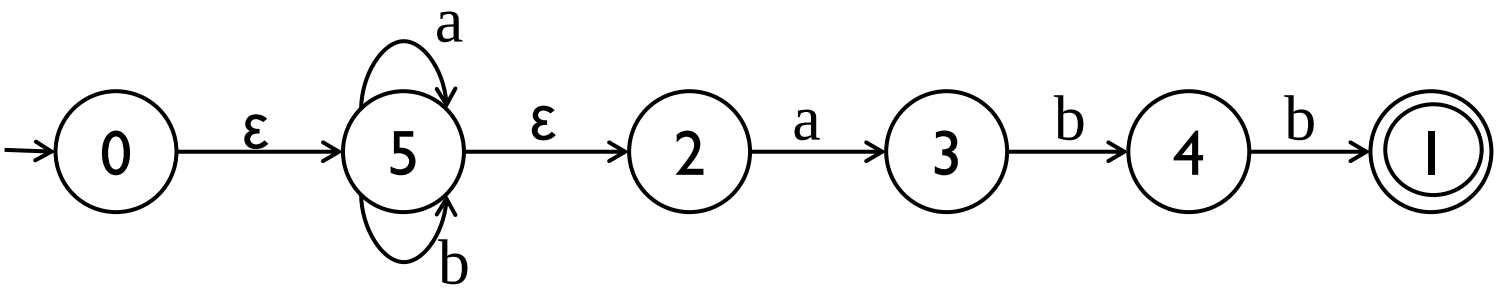






Ex

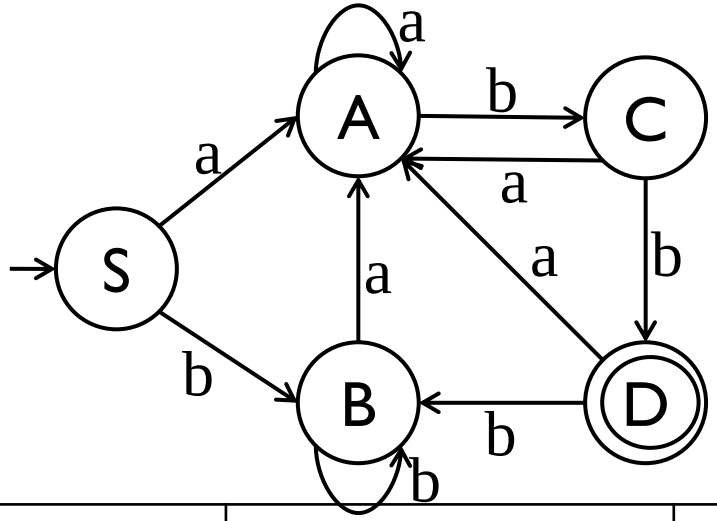
- (a|b)*abb -> NFA -> DFA



I	I_a	I_b
{0, 5, 2}S	{5, 2, 3}A	{5, 2}B
{5, 2, 3}A	{5, 2, 3}A	{5, 2, 4}C
{5, 2}B	{5, 2, 3}A	{5, 2}B
{5, 2, 4}C	{5, 2, 3}A	{5, 2, 1}D
{5, 2, 1}D	{5, 2, 3}A	{5, 2}B

Ex

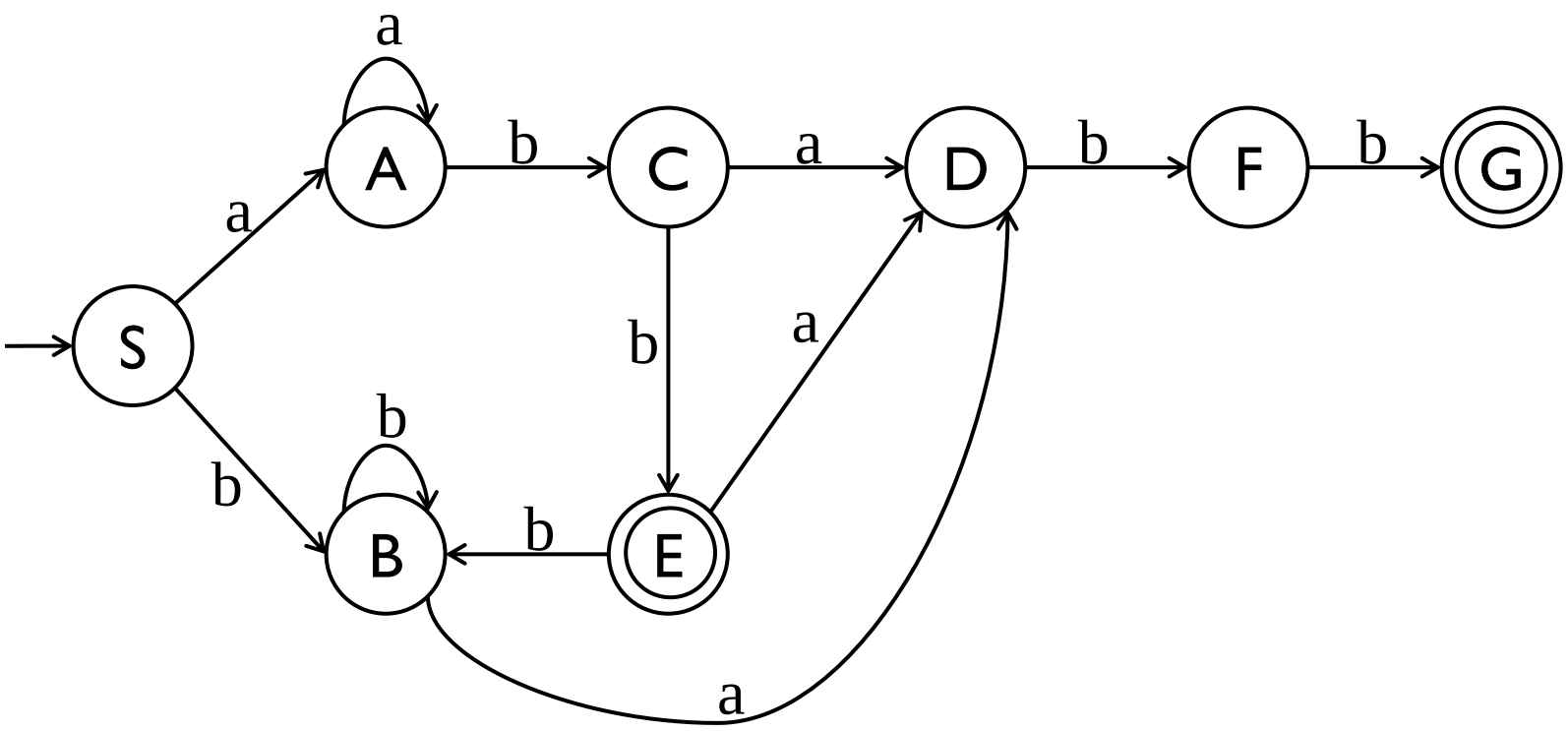
- $(a|b)^*abb$ -> NFA -> DFA



I	I_a	I_b
{0, 5, 2}S	{5, 2, 3}A	{5, 2}B
{5, 2, 3}A	{5, 2, 3}A	{5, 2, 4}C
{5, 2}B	{5, 2, 3}A	{5, 2}B
{5, 2, 4}C	{5, 2, 3}A	{5, 2, 1}D
{5, 2, 1}D	{5, 2, 3}A	{5, 2}B

Ex

- a*b*abb -> NFA -> DFA

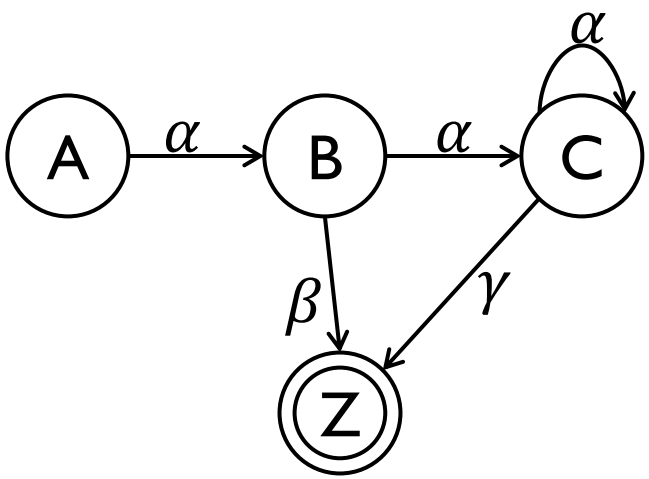


Regular grammar and FA

- 对于正规文法G和有限自动机M，如果L(G)=L(M)，则称G和M是等价的。
- **结论：**
  - （1）对于每一个右线性正规文法或左线性正规文法G，都存在一个FA M，使L(M)=L(G)。
  - （2）对于每一个DFA M，都存在一个右线性正规文法G和一个左线性正规文法G’，使L(M)=L(G)=L(G’)。

Regular grammar and FA

- 正规文法：
  - $A \rightarrow \alpha B$
  - $B \rightarrow \alpha C|\beta$
  - $C \rightarrow \alpha C|\gamma$



Minimize number of states

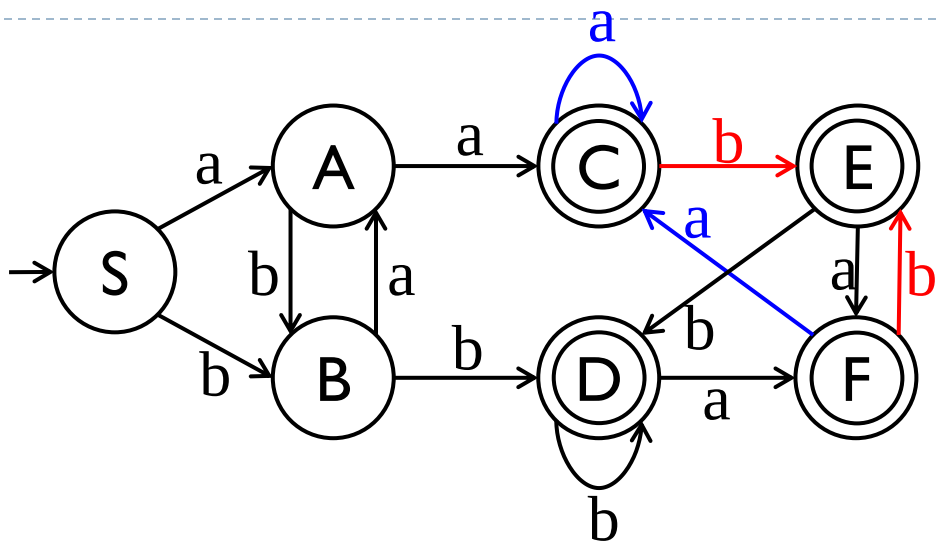
- 说一个有限自动机是化简了的，即是说，它没有多余状态并且它的状态中没有两个是互相等价的。一个有限自动机可以通过消除多余状态和合并等价状态而转换成一个最小的与之等价的有限自动机。

- 所谓有限自动机的多余状态，是指这样的状态：从自动机的开始状态出发，任何输入串也不能到达的那个状态；或者从这个状态没有通路到达终态。

Minimize number of states

- 对DFA M的化简：寻找一个状态数比M少的DFA M’，使得L(M)=L(M’)
- 假设s和t为M的两个状态，称s和t等价:如果从状态s出发能读出某个字α而停止于终态，那么同样，从t出发也能读出α而停止于终态；反之亦然。
- 两个状态不等价，则称它们是可区别的。

Minimize number of states

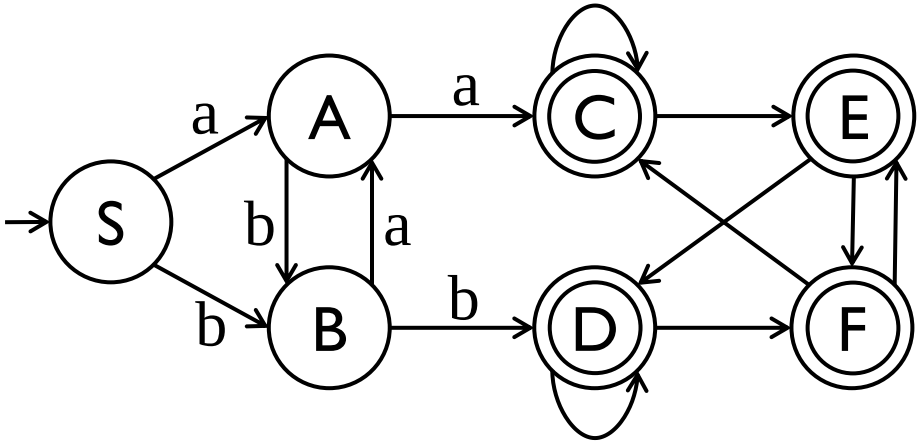


等价状态

f(C,a)=f(F,a)=C

f(C,b)=f(F,b)=E

Minimize number of states

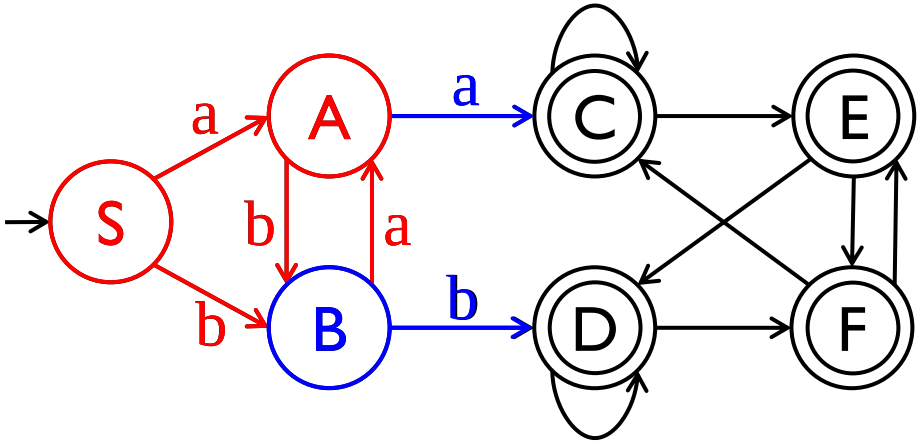


初始划分：按是否终止态划分为两个集合

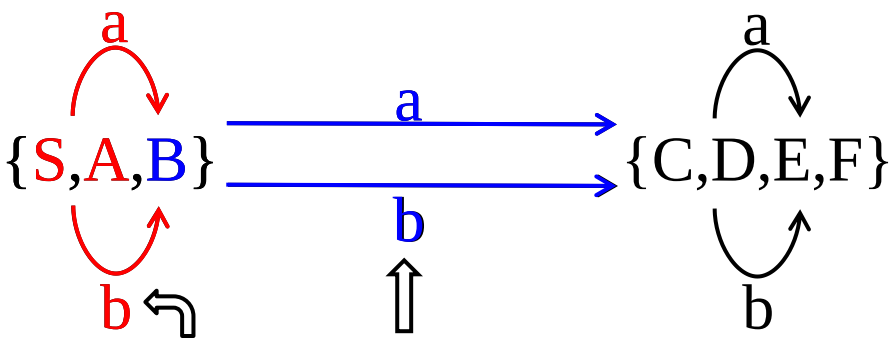
非终止态
{S,A,B}

终止态
{C,D,E,F}

Minimize number of states

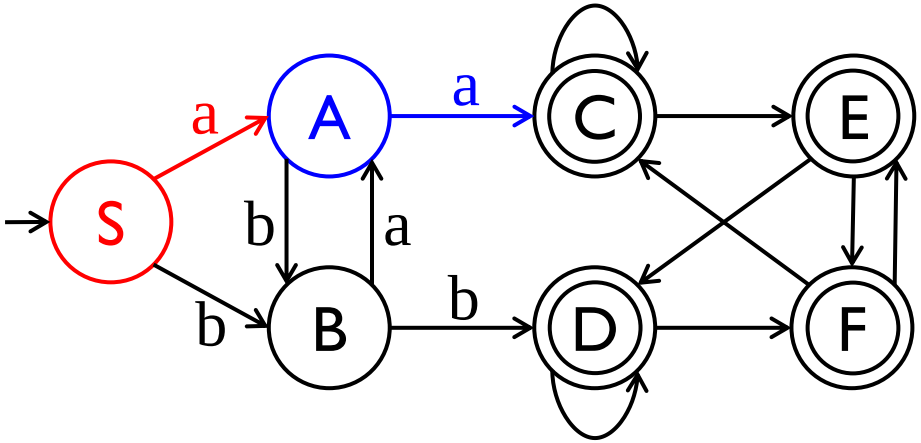


计算每个状态集合上的转移

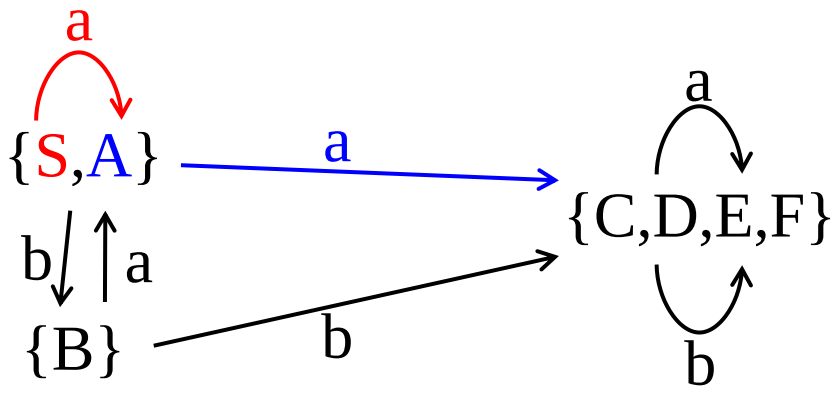


按两条路径进行拆分

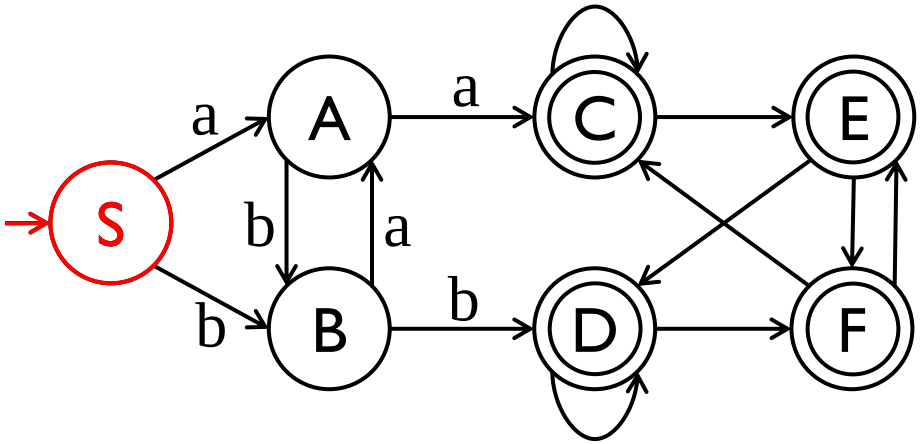
Minimize number of states



计算每个状态集合上的转移

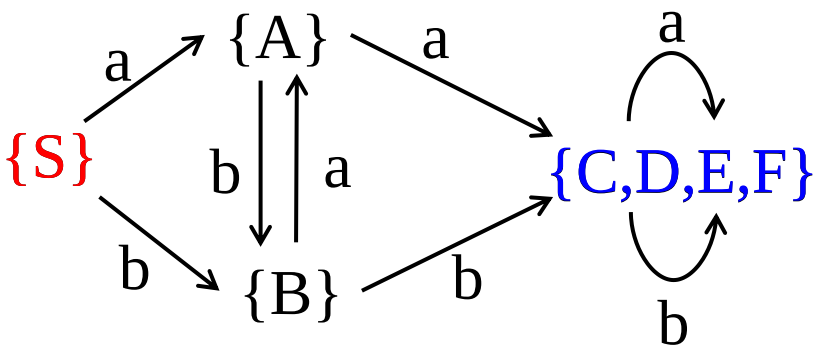


Minimize number of states



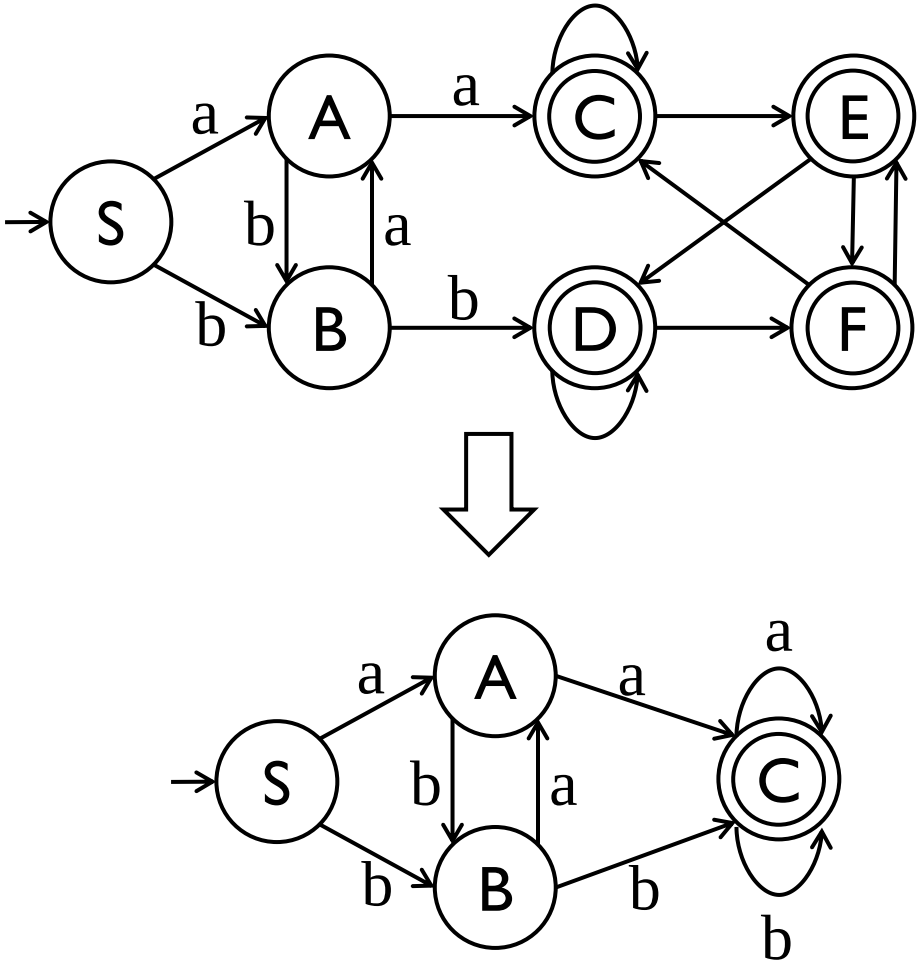
最终划分

包含初始状态的集合为初态



包含多个状态的集合简化为一个状态

Minimize number of states



Minimize number of states

- **DFA的最小化**就是寻求最小状态DFA
- 最小状态DFA的含义:
  - 没有多余状态(死状态)
  - 没有两个状态是互相等价（不可区别）
- 过程:
  - 把一个DFA的状态分成一些不相交的子集，使得任何不同的两子集的状态都是可区别的，而同一子集中的任何两个状态都是等价的。
  - 算法假定每个状态射出的弧都是完全的,否则,引入一个新状态,叫死状态,该状态是非状态，将不完全的输入弧都射向该状态，对所有输入，该状态射出的弧还回到自己.

Minimize number of states

- **DFA的最小化算法：**
  - DFA M = （K,∑,f, k_0, k_t),最小状态DFA M’
  - 1.构造状态的一初始划分∏：终态k_t和非终态K- k_t两组(group)。
  - 2.对∏施用过程PP 构造新划分$\prod_{new}$
    - for G in Π:

      （1）对G进行划分，G中的两个状态s和t被划分在同一个组中的充要条件是: 任何输入字符a，move(s,a)和move(t,a)在Π的同一个组中；

      （2）用新划分的组替代G，形成新的划分Π_{new}；

Minimize number of states

- **DFA的最小化算法：**
  - 3. 如$\prod_{new}=\prod$,则令$\prod_{final}=\prod$,并继续步骤4，否则令$\prod:=\prod_{new}$，重复步骤2。
  - 4.为$\prod_{final}$中的每一组选一代表，这些代表构成M’的状态。若k是一代表且f(k,a)=t,令r是t组的代表，则M’中有一转换f’(k,a)=r，M’的开始状态是含有S_0的那组的代表,M’ 的终态是含有F的那组的代表。
  - 5.去掉M’中的死状态。

Minimize number of states – 例19

$\prod_0$:{S,A,B}

{C,D,E,F}

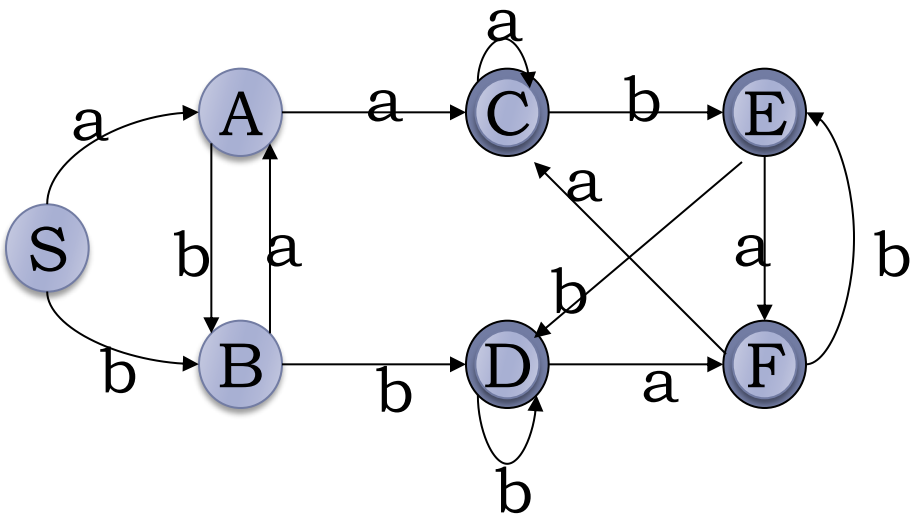
$\prod_1$:{S,A,B}

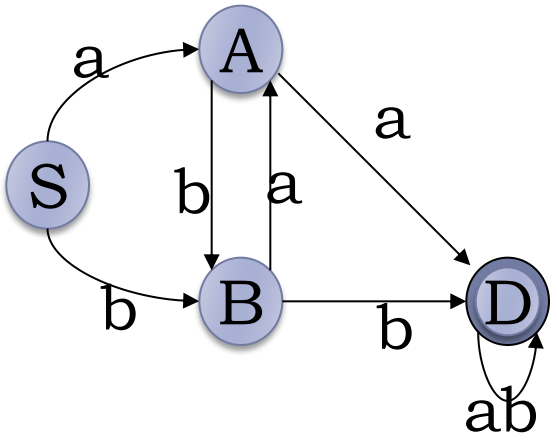
a

$\prod_2$: {A} {S,B}

b

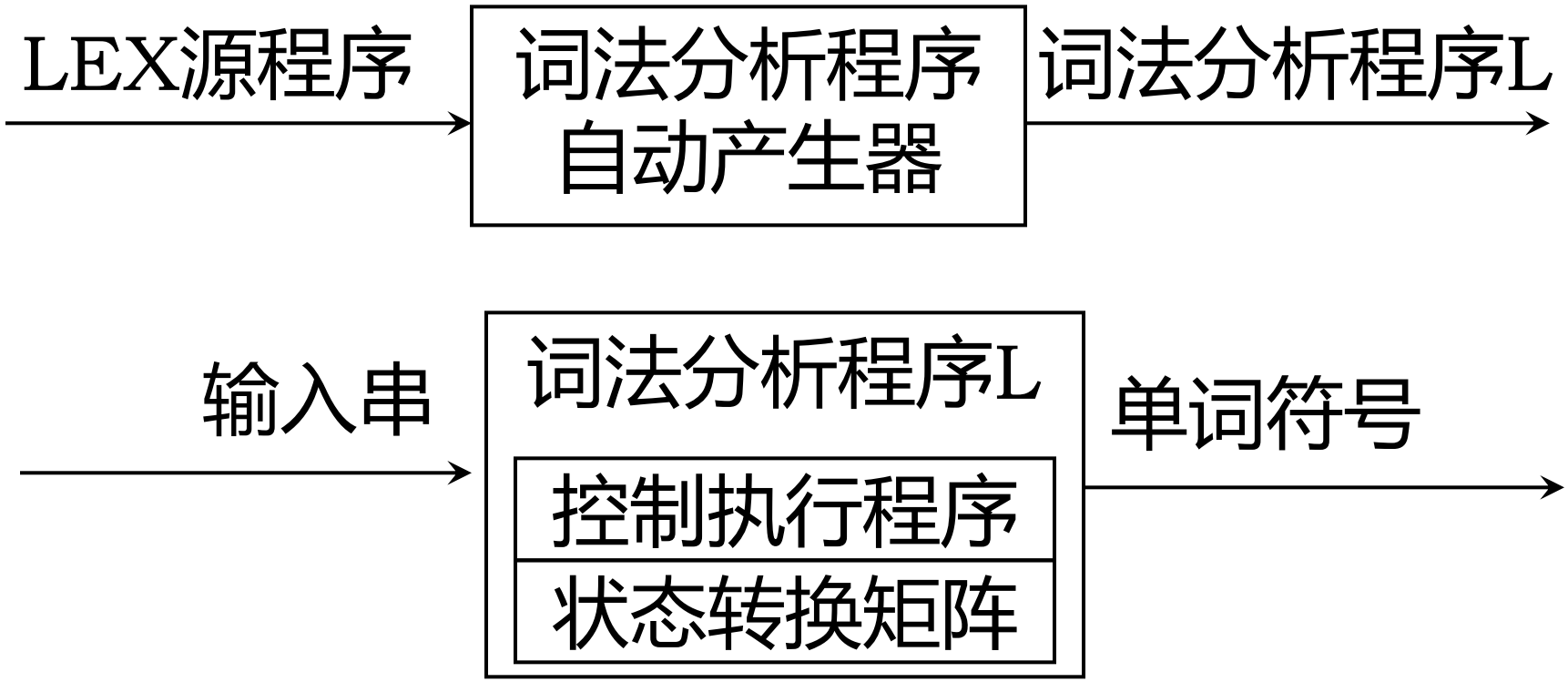
{ S }{B}





LEX

- LEX程序由一组正规式以及相应的动作组成。



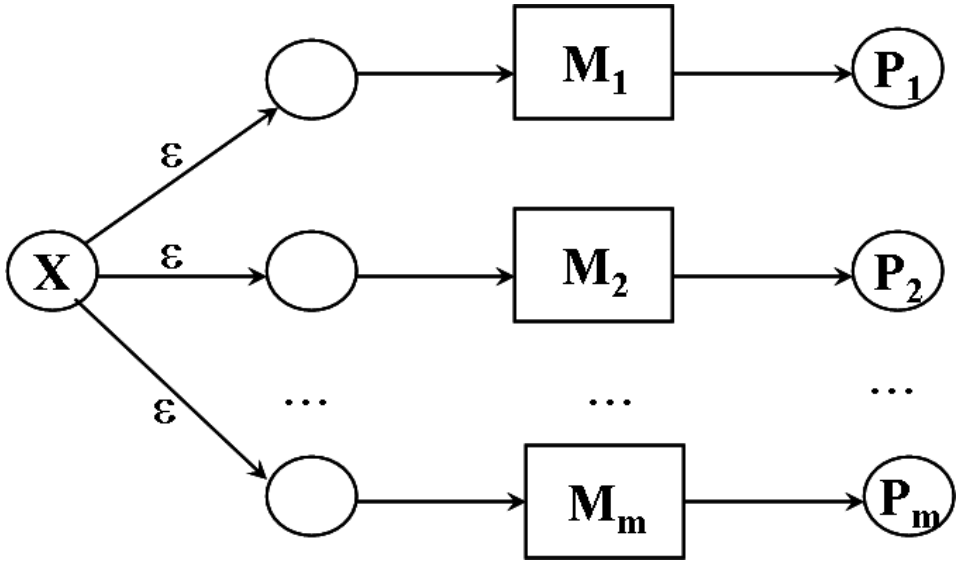
Lex

- **LEX程序：**

```
AUXILIARY DEFINITION     /* 正规定义式 */
   letter→A|B|…|Z              digit →0|1|…|9
RECOGNITION RULES      /*  识别规则  */
1        DIM                          { RETURN (1,-) }
2        IF                           { RETURN (2,-) }
3        DO                           { RETURN (3,-) }
4        STOP                         { RETURN (4,-) }
5        END                          { RETURN (5,-) }
6        letter(letter|digit)         { RETURN (6, TOKEN) }
7        digit(digit)*                { RETURN (7, DTB) }
8        =                            { RETURN (8, -) }
9        +                            { RETURN (9,-) }
10               *                            { RETURN (10,-) }
11               **                           { RETURN (11,-) }
12               ,                            { RETURN (12,-) }
13               (                            { RETURN (13,-) }
14               )                            { RETURN (14,-) }
```

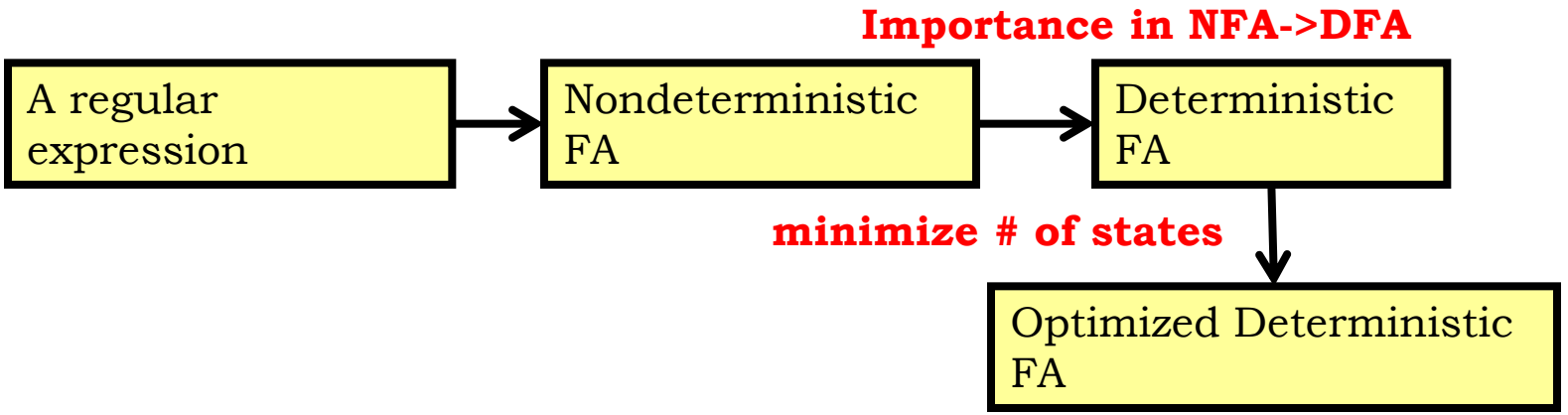
LEX

- **LEX的工作过程**
  - 对每条识别规则P_i构造一个相应的NFA M_i
  - 引进新初态X，通过ε弧将这些自动机连接成一个新的NFA
  - 把M确定化、最小化，生成该DFA的状态转换表和控制执行程序



Summary

- 状态转换图
- 正规表达式与正规集
- DFA与NFA
- NFA的确定化——子集法
- DFA的最小化——分割法
- 词法分析器的构造过程：



Exercise

- 例20：构造一个DFA，它接收∑={a, b}上所有满足下述条件的字符串：字符串中的每个a都有至少一个b直接跟在其右边。

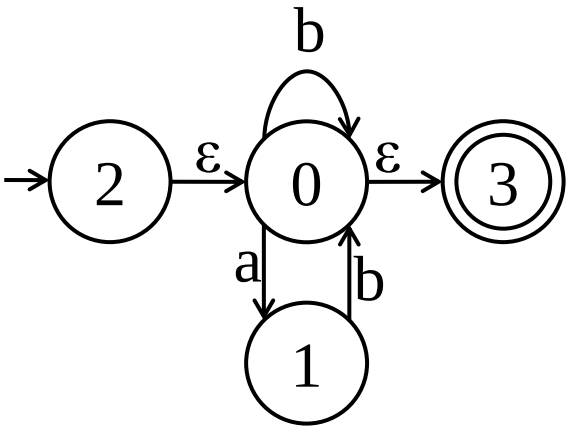
题意→RE→NFA→DFA

$$(b|ab)^*$$

Exercise

(b|ab)*

RE->NFA->DFA



Move函数

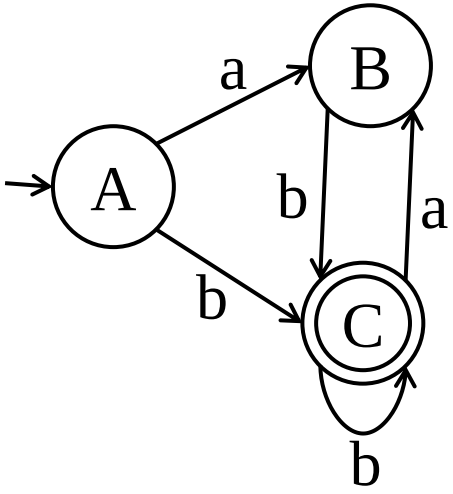
	a	b
{2,0,3}A	{1}	{0}
	{1}B	{0,3}C
{1}B	{}	{0}
		{0,3}C
{0,3}C	{1}	{0}
	{1}B	{0,3}C

ε-closure函数

Exercise

(b|ab)*

RE->NFA->DFA



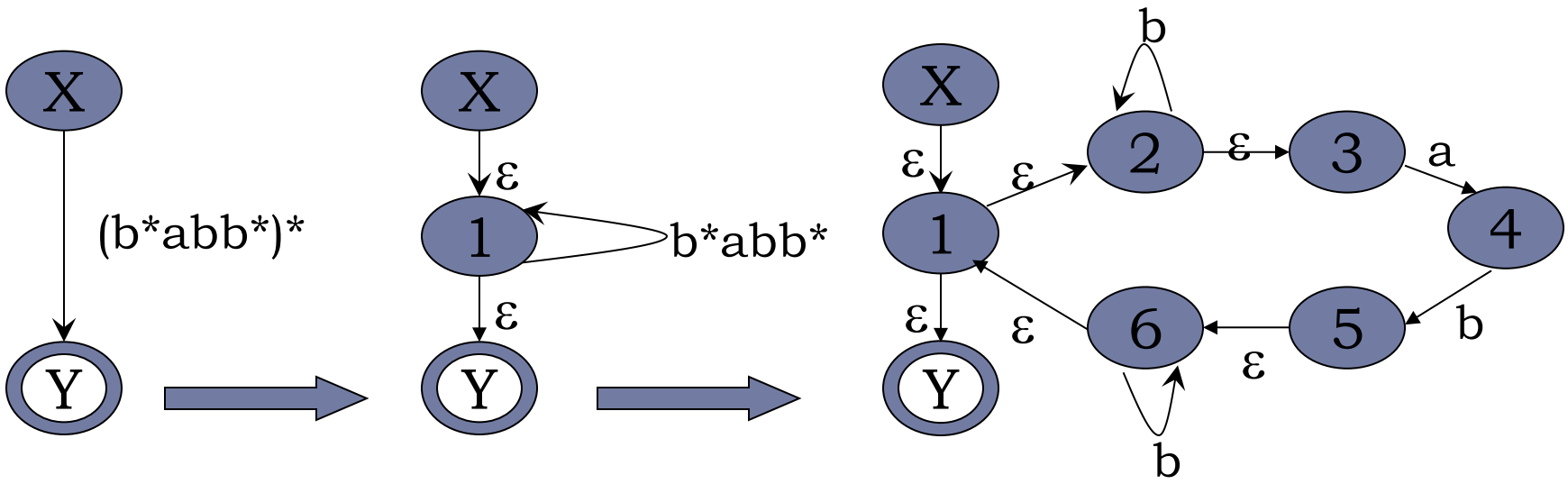
	a	b
{2,0,3}A	{1}	{0}
	{1}B	{0,3}C
{1}B	{}	{0}
		{0,3}C
{0,3}C	{1}	{0}
	{1}B	{0,3}C

Exercise

- 例20：构造一个DFA，它接收∑={a, b}上所有满足下述条件的字符串：字符串中的每个a都有至少一个b直接跟在其右边，并且如果不为空串，则字符串至少有一个a。
- 解：
  - 题意→RE→NFA→DFA
  - 已知∑={a, b}，根据题意得出相应的的正规式为：
  - **$(b^*abb^*)^*$**

Exercise

- 根据正规式画出相应的DFA M，如下图所示
- 用子集法将其确定化



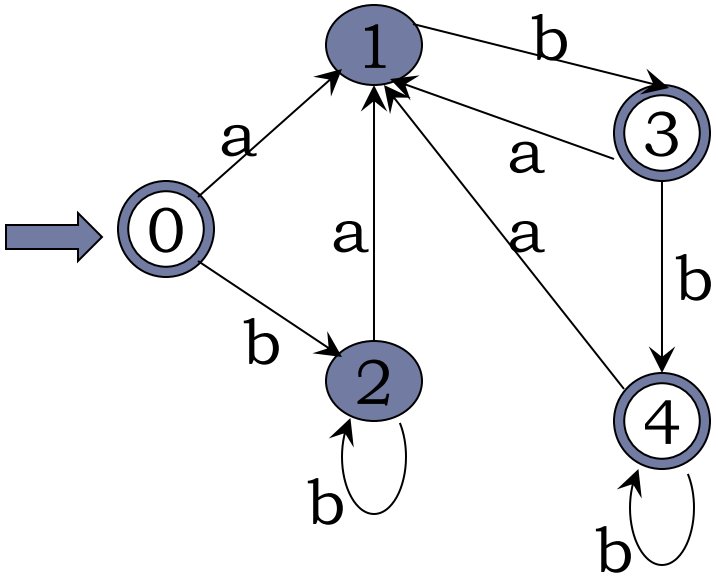
I	I_a	I_b
{X,1,2,3,Y}	{4}	{2,3}
{4}	—	{5,6,1,2,3,Y}
{2,3}	{4}	{2,3}
{5,6,1,2,3,Y}	{4}	{6,1,2,3,Y}
{6,1,2,3,Y}	{4}	{6,1,2,3,Y}

重新命名

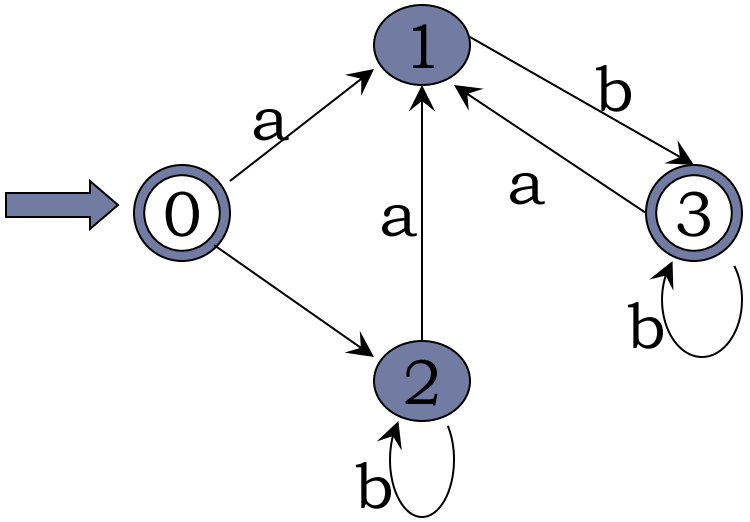
I	I_a	I_b
0	1	2
1	—	3
2	1	2
3	1	4
4	1	4

Exercise

- 由DFA得状态图

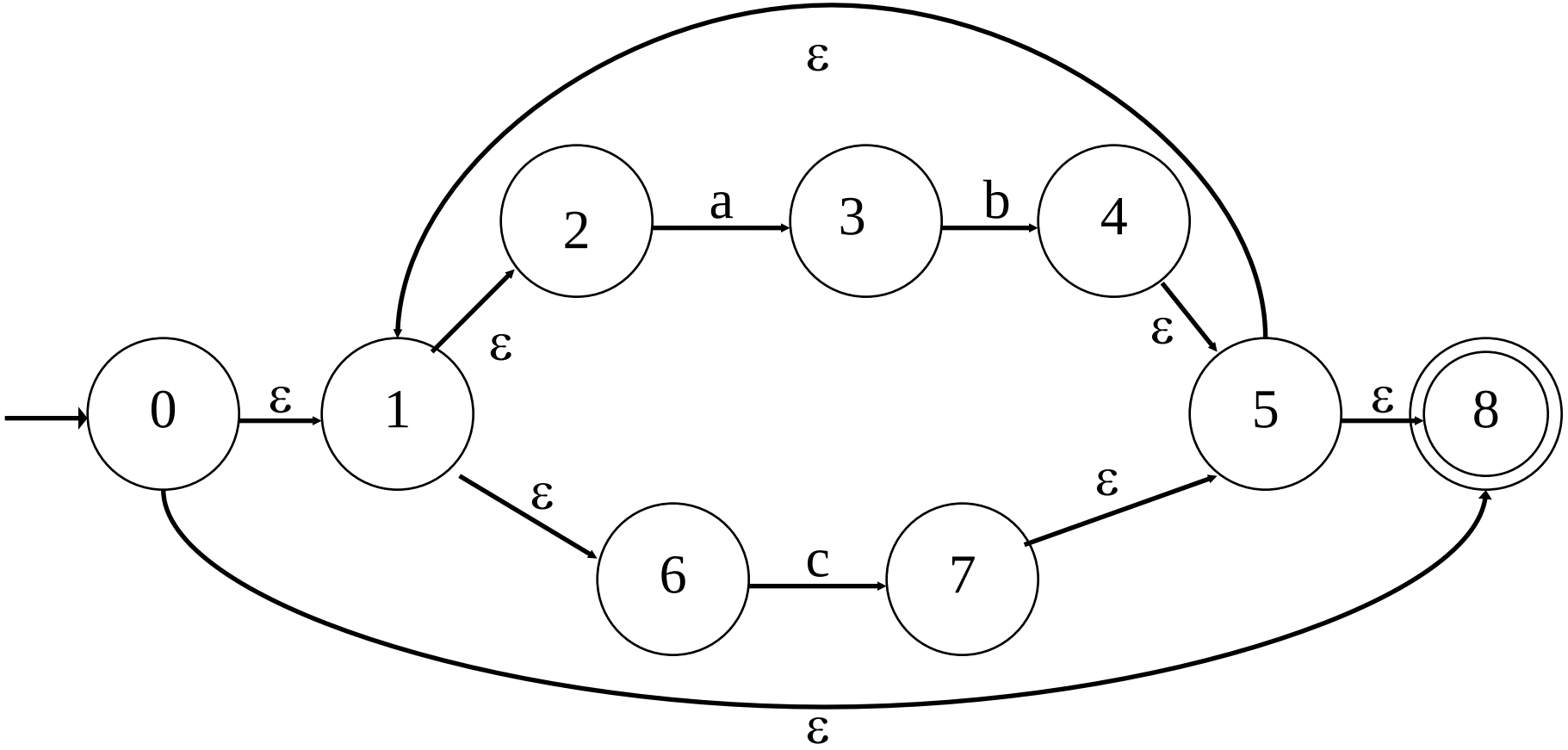


- 用最小化方法化简得：{0}，{1}，{2}，{3,4}，按顺序重新命名DFA M’

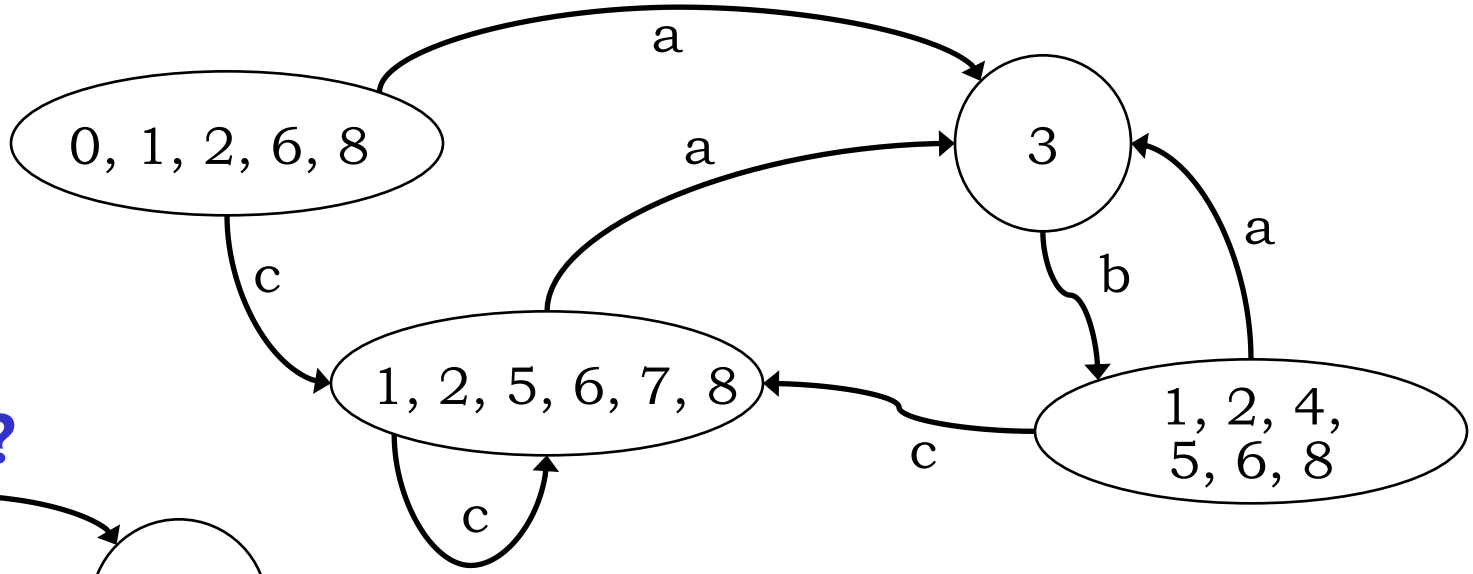


Exercise

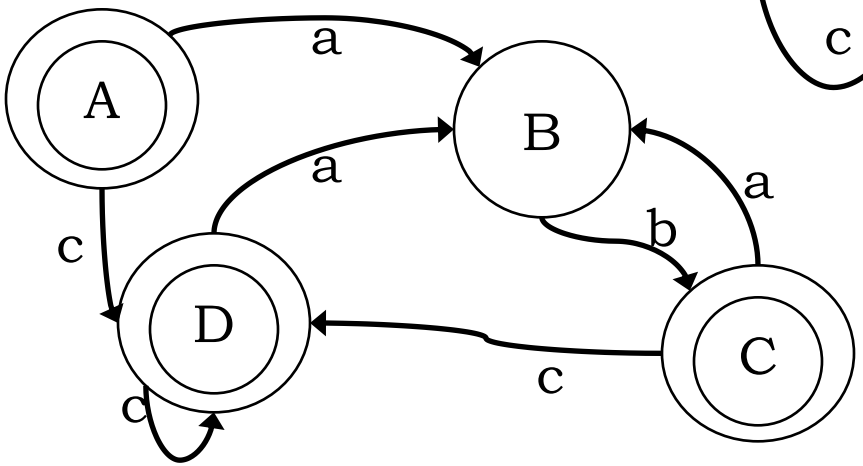
- 例21：NFA->DFA



Exercise



哪些是终态？



练习

构造正规表达式

((a|b)*|(bb)*)*

的最小DFA

练习

- 设M=({x,y}, {a,b}, f, x, {y}) 为一有限自动机，其中f定义如下：

f(x, a)={x, y}　　f(x,b)={y}

f(y, a) =Φ　　f(y,b)={x,y}

请构造相应的确定有限自动机

Exercises

- 课本第63-64页, 下堂课我们将讲解
  - 6 (4)
  - 7 (1)
  - 8 (1)
  - 12 (1)