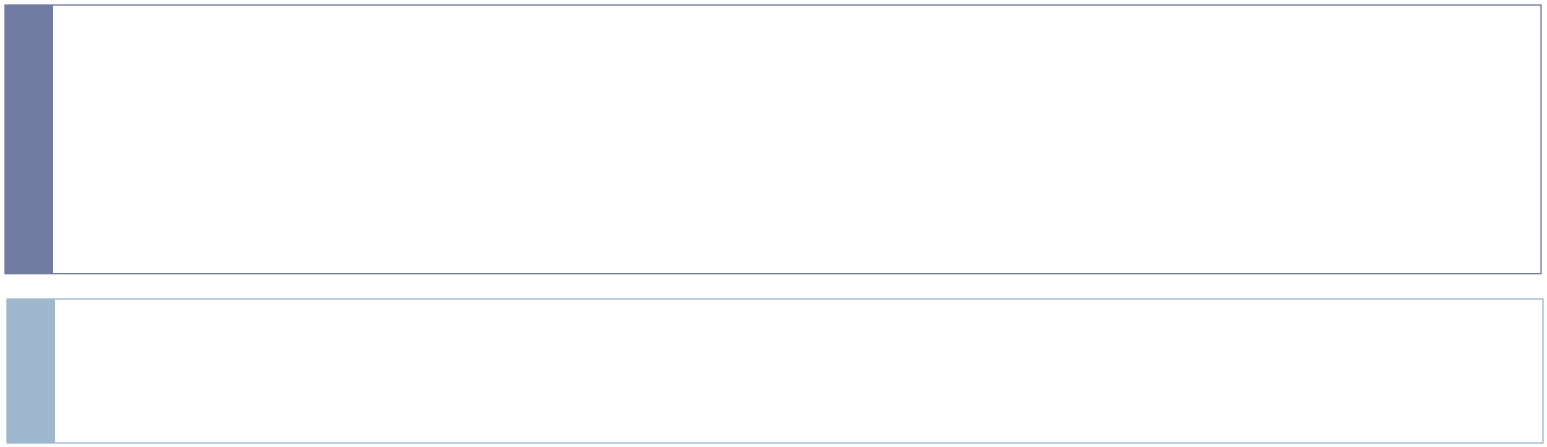
Chapter 3 词法分析

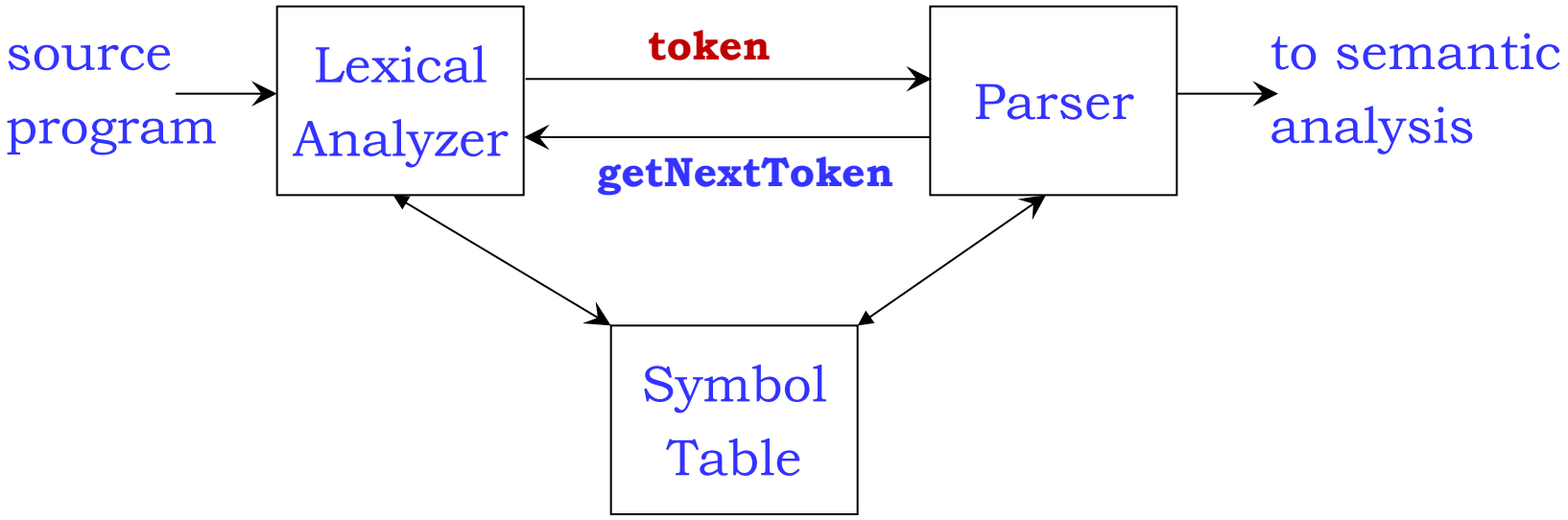


Outlines

- 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

Requirements

- 任务:
  - 从左至右逐个字符地对源程序进行扫描，产生一个个的单词符号，把作为字符串的源程序改造成为单词符号串的中间程序
- 词法分析是编译的基础



Function

- 功能:输入源程序、输出单词符号
  - 单词符号的种类:

关键字	由程序语言定义的具有固定意义的标识符。也可称为保留字： begin，if, while，…
标识符	表示各种名字。如变量名、数组名和过程名
常数	各种类型的常数，一般有整型、实型、布尔型、文字型等。
运算符	+，-，*，/，…
界　符	逗号、分号、括号和空白…

Output

- 输出的单词符号的表示形式:
  - 二元式
  - (单词种别，单词符号的属性值attribute-value)
  - 单词种别通常用整数编码表示。
  - 单词种别可以用以下形式表示：
    - 一类单词统一用一个整数值代表其属性。例如：1代表关键字，2代表标识符等。
    - 每一个单词一个类别。例如：1代表BEGIN，2代表END等。

Notes

- 若一个种别只有一个单词符号，则种别编码就代表该单词符号。假定关键字、运算符和界符都是一符一种。
- 若一个种别有多个单词符号，则对于每个单词符号，给出种别编码和自身的值（属性）。
- 标识符单列一种；标识符自身的值表示成按机器字节划分的内部码。
- 常数按类型分种；常数的值则表示成标准的二进制形式。
- 若是一符一种分种，单词自身值就不需要了。否则，要查符号表。

Example

- 例1 下述C++代码段：**while (i >= j) i - -;** 经词法分析器处理以后，它将被转换为如下的单词符号串

(while , _)
((, _)
(id , 指向i的符号表指针)
(>= , _)
(id , 指向j的符号表指针)
( ) , _)
(id , 指向i的符号表指针)
(- - , _)
(; , _)

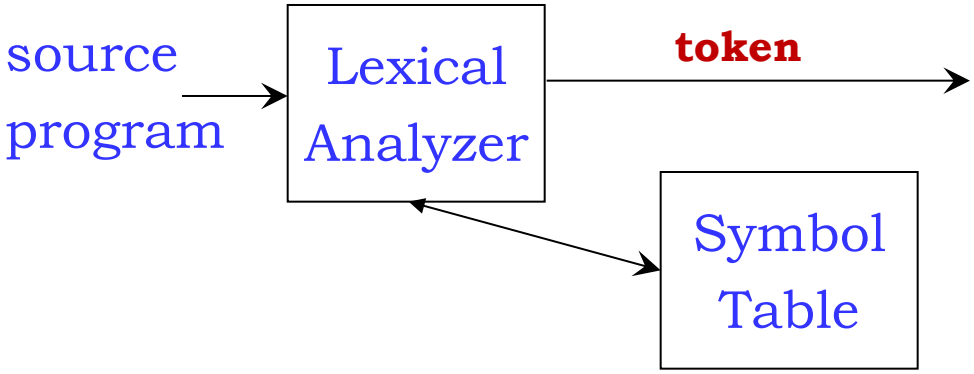
Example

- 例2　C++程序
  - **if (a>100) c++;**
  - 输出单词符号：

<if, ->
<(, ->
<id, a>
<>, ->
<const, 100>
<), ->
<id, c>
<++, ->
<;, ->

One pass

- 词法分析是作为一个独立的阶段，是否应当将其处理为一遍呢?

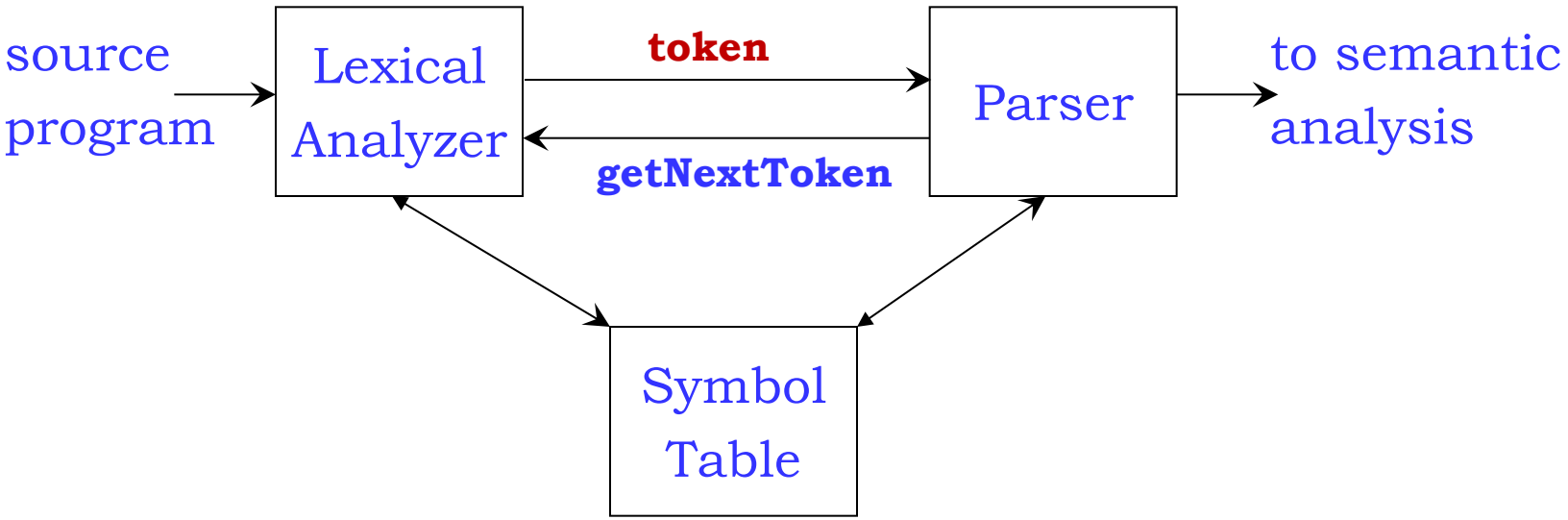


One pass

- 作为独立阶段的优点：结构简洁、清晰和条理化，有利于集中考虑词法分析一些枝节问题。
- 词法分析和语法分析方法不同，词法分析可以使用正则文法自动构造scanner简单。
- 有利于提高语法分析的效率。
- 可以改善词法分析的细节，甚至于一个语法分析配几个scanner，把不同的输入变成一种内部表示。

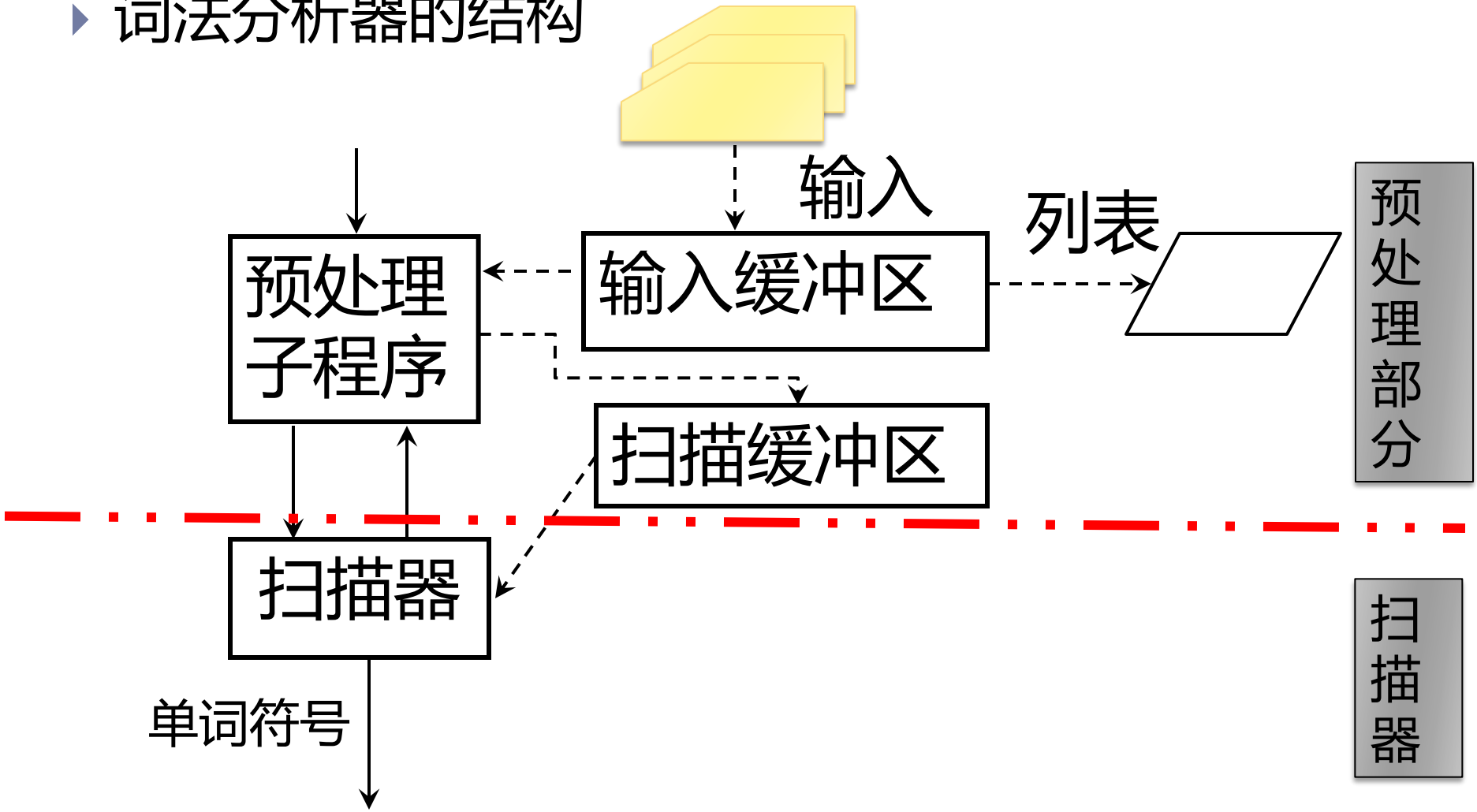
Position

- 词法分析程序和语法分析程序的关系
  - 作为子程序的词法分析器



Design

- 词法分析器的结构



Input and pre-processing

- 输入源程序文本
  - 输入串放在一个缓冲区中
- 预处理子程序
  - 剔除无用地空白、跳格、回车和换行等编辑性字符
  - 区分标号区、连接续行和给出句末符等
- 扫描器
  - 识别单词符号
- 扫描缓冲区

Advanced search

- 关键字识别:
  - 例如:

```
1 DO99K=1，10
2 IF(5.EQ.M)GOTO 55
3 DO99K=1.10
4 IF(5)=55
```

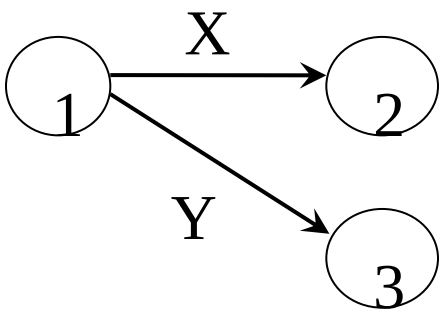
  - 需要超前搜索才能确定哪些是关键字

Advanced search

- 关键字的识别
  - 由程序语言定义的基本字或保留字等
- 标识符识别:
  - 字母开头的字母数字串，后跟界符或算符
- 常数识别:
  - 识别出算术常数并将其转变为二进制内码表示。有些也要超前搜索。如：5.EQ.M 与 5.E08
- 算符和界符的识别
  - 把多个字符符合而成的算符和界符拼合成一个单一单词符号。：=， **， .EQ.

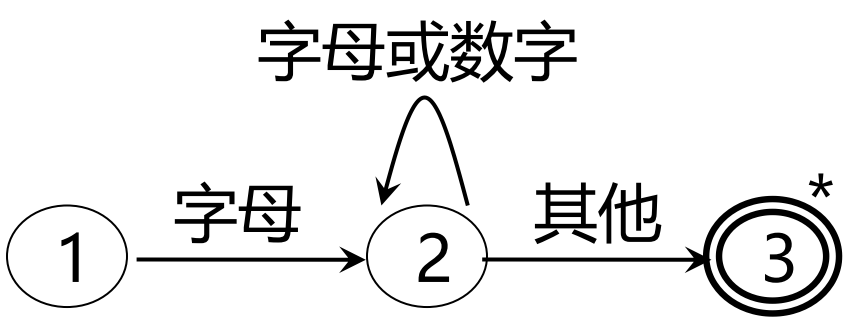
State transition diagram

- 状态转换图是一张有限有向图。
  - 结点代表状态
    - 用圆圈表示。
  - 状态之间用箭弧连结
    - 箭弧上的标记(字符)代表射出结点状态下可能出现的输入字符或字符类。
  - 一张转换图只包含有限个状态，
    - 其中有一个为初态
    - 至少要有一个终态（双圈）

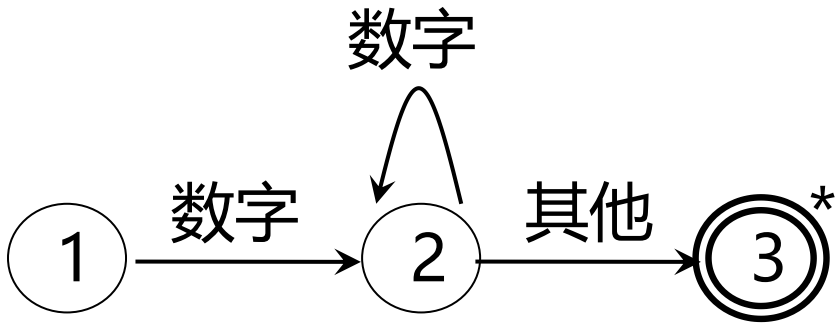


State transition diagram

- 一个状态转换图可用于识别(或接受)一定的字符串。



识别标识符的状态转换图



识别整常数的状态转换图

终结符上的星号 "*"

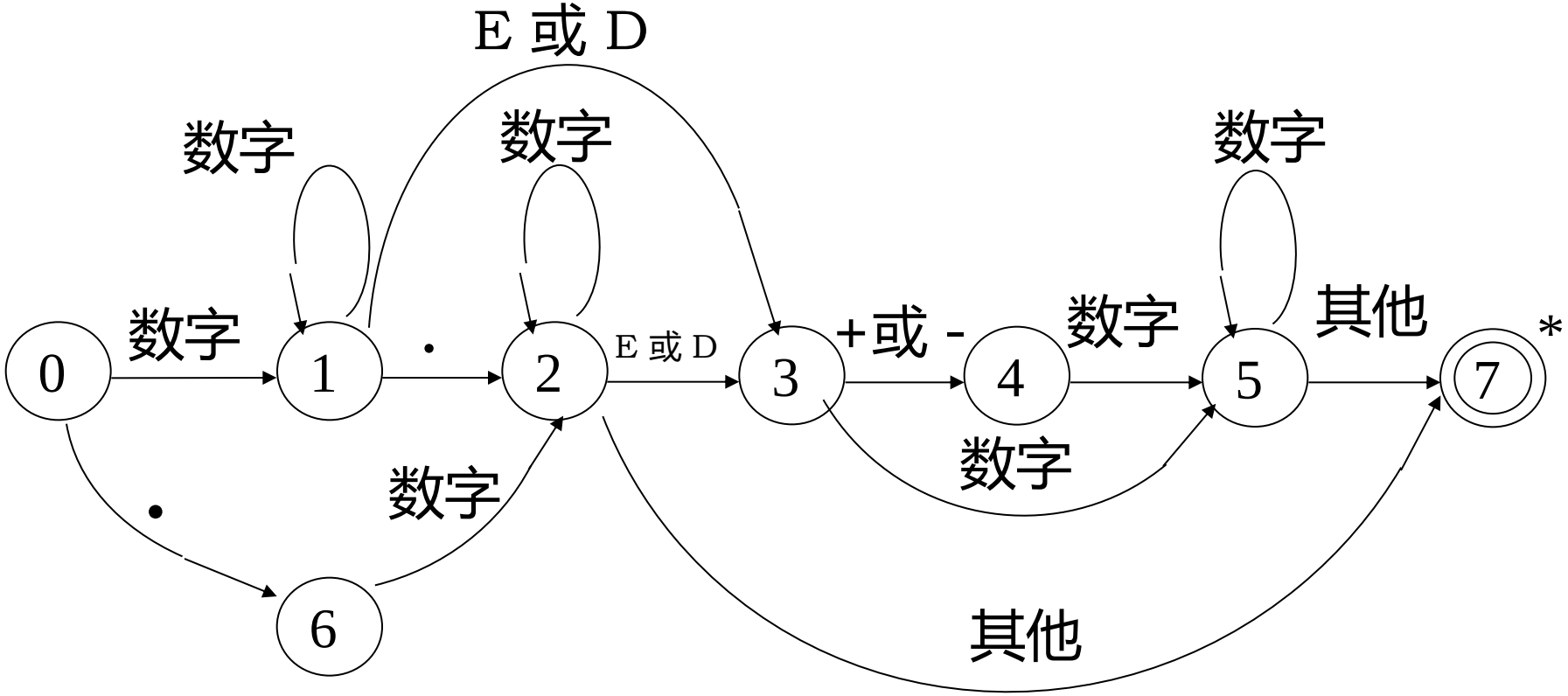
读进不属于标识符部分的字符，则把它退还给输入串

State transition diagram

例如下列实型常数可以被以下转换图识别：

1.23E+4

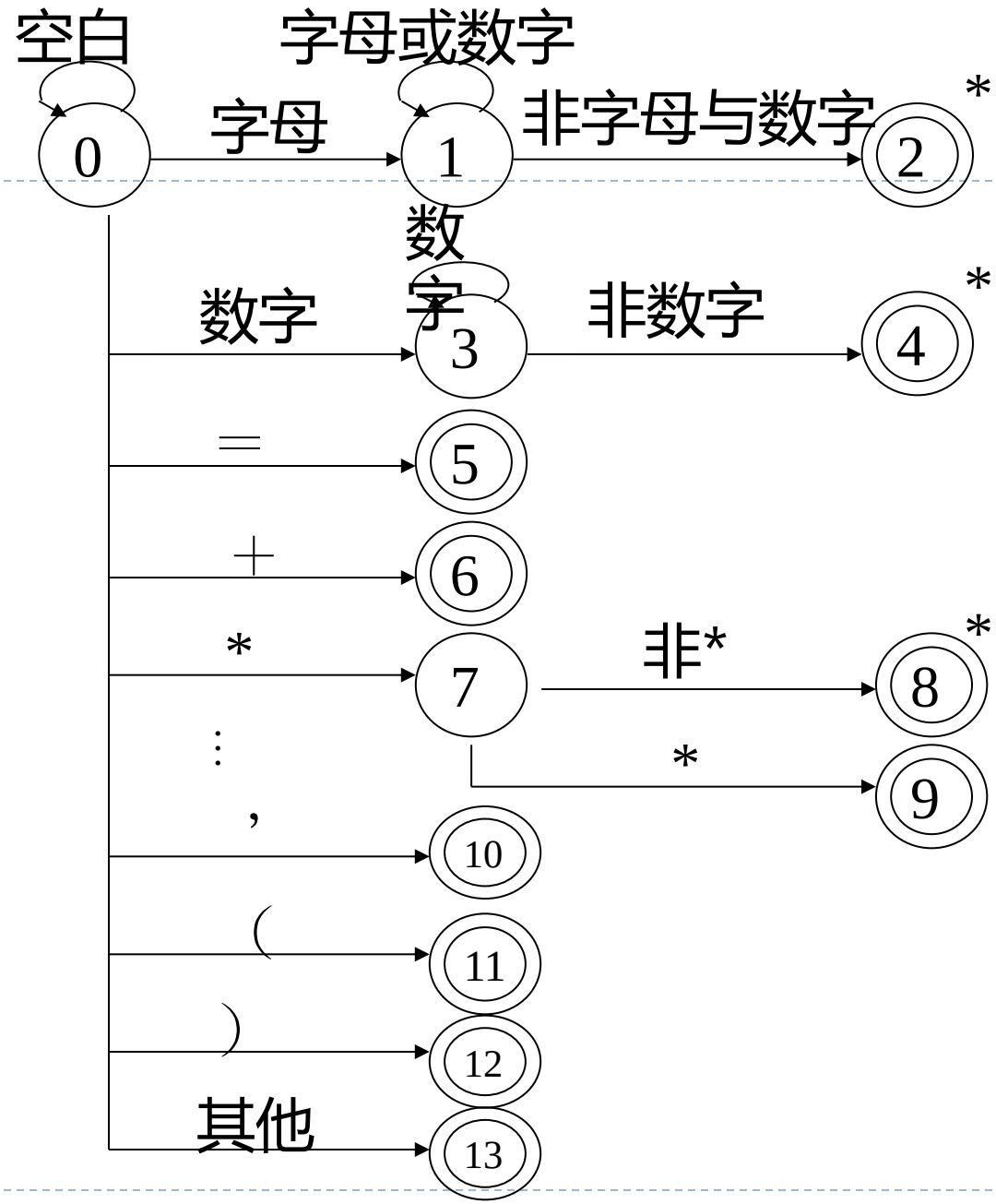
.56E-7



State transition diagram

- 助忆符：直接用编码表示不便于记忆，因此用助忆符来表示编码

单词符号	种别编码	助忆符	内码值
DIM	1	$DIM	-
IF	2	$IF	-
标识符	6	$ID	内部字符串
数值常数	7	$INT	二进制表示

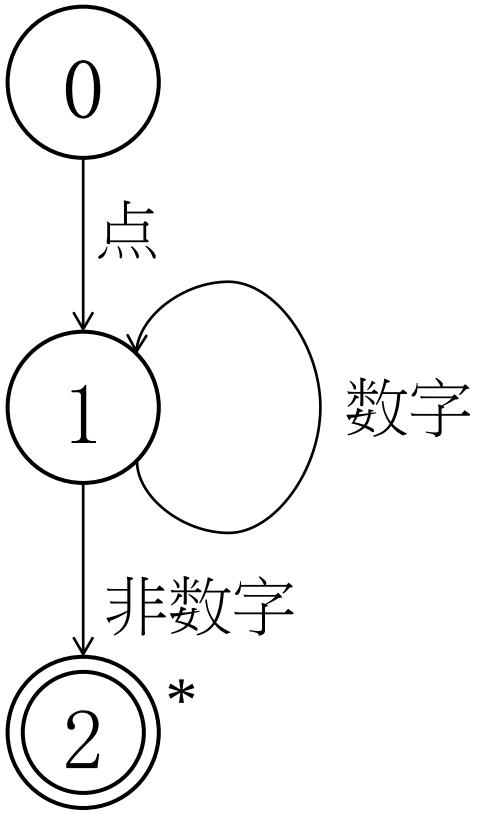

空白
字母或数字
0
字母
1
非字母与数字
2
*
数字
3
非数字
4
*
=
5
+
6
*
7
非*
8
*
*
9
,
10
(
11
)
12
其他
13

State transition diagram

- 几点重要限制——不必使用超前搜索
  - 所有关键字都是保留字；用户不能用它们作自己的标识符
  - 关键字作为特殊的标识符来处理；不用特殊的状态图来识别，只要查保留字表。
  - 如果关键字、标识符和常数(或标号)之间没有确定的运算符或界符作间隔，则必须使用一个空白符作间隔。
    - DO99K=1，10 要写成 DO 99 K=1，10

Implementation

- 例：实现下面的状态图



读取一个字符 c = getc(fp)
回退一个字符 ungetc(c, fp)

```
bool dot_number(FILE *fp) {
    int state = 0;
    while (true) {
        char c = getc(fp);
        if (0 == state) {
            if ('.' == c) {
                state = 1;
            } else {
                return false;
            }
        } else if (1 == state) {
            if ('0' <= c && c <= '9')
                state = 1;
            } else {
                state = 2;
            }
        } else if (2 == state) {
            ungetc(c, fp);
            return true;
        }
    }
    return false;
}
```

Implementation

- 实现方法：
  - 每个状态结对应一小段程序。
- 实现步骤:
  - 对不含回路的分叉结
    - 可用一个CASE语句或一组IF-THEN-ELSE语句实现
  - 对含回路的状态结
    - 可对应一段由WHILE结构和IF语句构成的程序
  - 终态结表示识别出某种单词符号，
    - 对应语句为 RETURN (C，VAL) 其中，C为单词种别，VAL为单词自身值

Implementation

- 全局变量与过程
  - 1) CHAR 字符变量、存放最新读入的源程序字符
  - 2) TOKEN 字符数组，存放构成单词符号的字符串
  - 3) GETCHAR 子程序过程，把下一个字符读入到CHAR中
  - 4) GETNBC 子程序过程，跳过空白符，直至CHAR中读入一非空白符
  - 5) CONCAT 子程序，把CHAR中的字符连接到TOKEN

Implementation

- 全局变量与过程
  - 6) LETTER 布尔函数，判断CHAR中字符是否为字母
  - 7) DIGIT 布尔函数，判断CHAR中字符是否为数字
  - 8) RESERVE 整型函数，对于TOKEN中的字符串查找保留字表，若它是保留字则给出它的编码，否则回送0
  - 9) RETRACT 子程序，把搜索指针回调一个字符位置
  - 10) DTB 函数，把TOKEN中的字符串翻译成二进制码

Implementation

- 为了把状态转换图转化成程序，每个状态要建立一段程序，它要做的工作如下：
- 第一步：从输入缓冲区中取一个字符。为此，我们使用函数GETCHAR，每次调用它，推进先行指针，送回一个字符。
- 第二步：确定在本状态下，哪一条箭弧是用刚刚来的输入字符标识的。如果找到，控制就转到该弧所指向的状态；若找不到，那么寻找该单词的企图就失败了。
- 失 败：先行指针必须重新回到开始指针处，并用另一状态图来搜索另一单词。如果所有的状态转换图都试过之后，还没有匹配的，就表明这是一个词法错误，此时，调用错误校正程序。

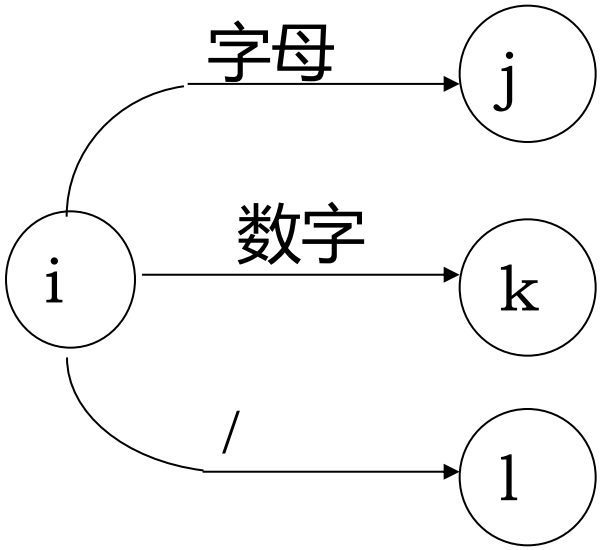
Implementation

例3： 以下CASE语句段对应的状态图:

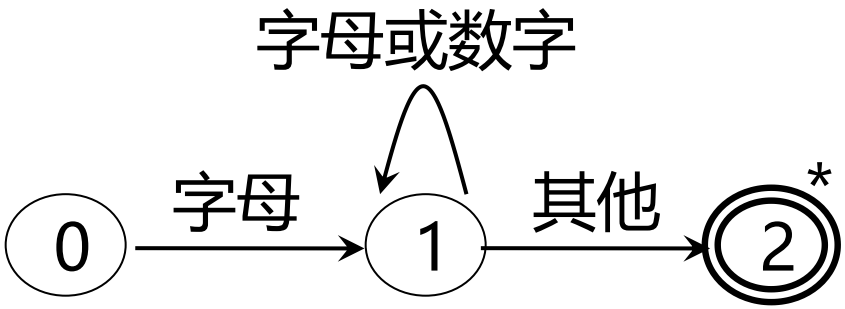
```
state i:   GETCHAR;
           CASE CHAR OF
          'A'..'Z':  ..  state  j ... ;
            '0'..'9':    . state  k ... ;
            '/'      :    . state  l ... ;
           END;
           FAIL
```

过程，将下一输入字符读入CHAR，搜索指示器前移一个字符。

字符变量，存放最新读进的源程序字符。



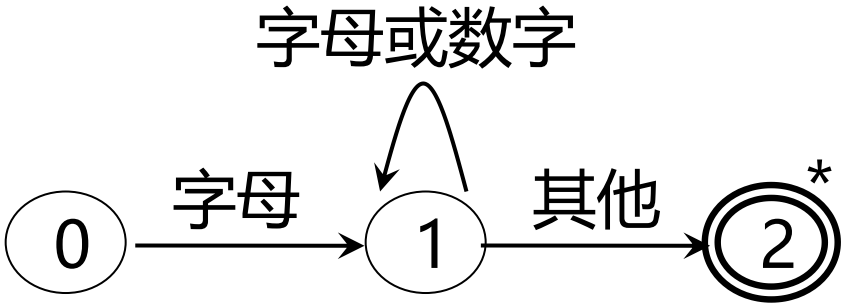
Implementation



识别标识符的状态转换图

- 例4：对于如上的状态转换图，状态0的代码如下所示：
- state 0:　　C := GETCHAR ;
- 　　if LETTER(C) then goto state 1
- 　　else FAIL( )

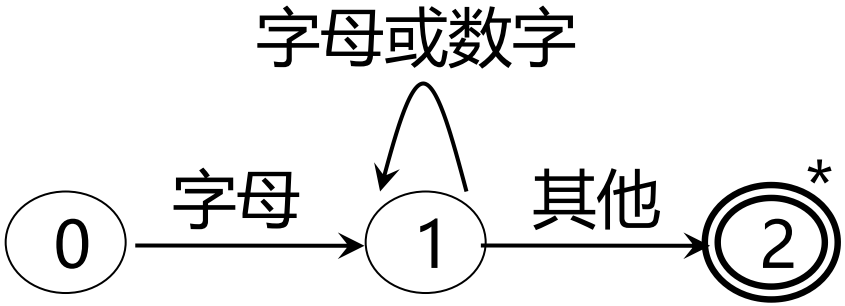
Implementation



识别标识符的状态转换图

- 对于如上的状态转换图，状态1的代码如下所示：
- state 1： C := GETCHAR ;
- if LETTER(C) or DIGIT(C)
- then goto state 1
- else if DELIMITER(C)
- then goto state 2
- else FAIL( )

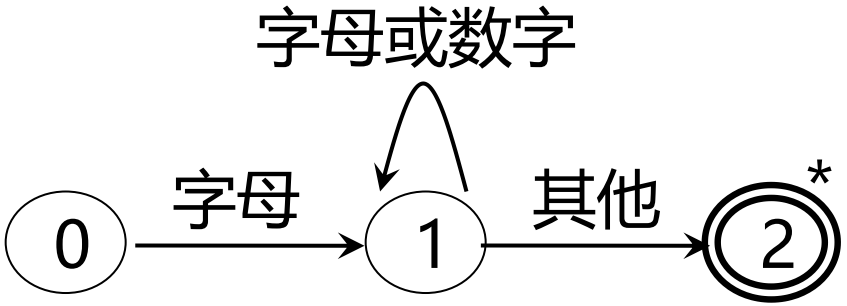
Implementation



识别标识符的状态转换图

- 对于如上的状态转换图，终态——状态2的代码如下所示：
- state 2：　RETRACT( ) ;
- 　　RETURN($id , INSTALL( ))

Implementation

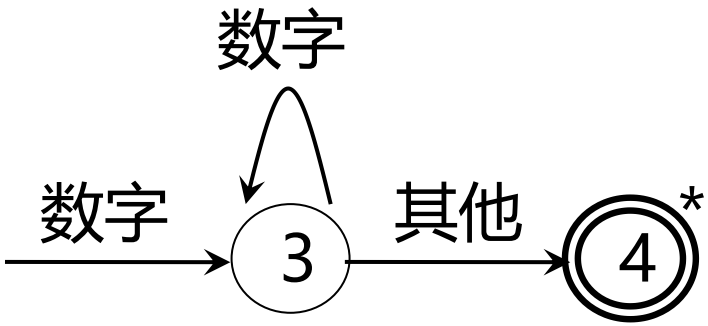


识别标识符的状态转换图

- 修改状态2的代码如下所示:

```
state 2:    RETRACT( ) ;
            c := RESERVE( );
            if  c = 0
                then  RETURN($id , INSTALL )
                else  RETURN(C , _ )
```

Implementation



识别整常数的状态转换图

- 例5：　‘0’… ‘9’：BEGIN
- WHILE DIGIT DO
- BEGIN CONCAT；GETCHAR END;
- RETRACT;
- RETURN($INT,DTB) ;
- END;

Implementation

```
'=':    RETURN ($ASSIGN, -);
'+':    RETURN ($PLUS, -);
'*':
   BEGIN
      GETCHAR;
      IF CHAR='*' THEN RETURN ($POWER, -);
      RETRACT; RETURN ($STAR, -);
   END;
', ':   RETURN ($COMMA, -);
'(':            RETURN ($LPAR, -);
')':            RETURN ($RPAR, -);
```

Regular expression and Finite Automata

- 正规表达式与有限自动机
  - 核心内容
    - 正规式和正规集的递归定义
    - 确定有限自动机（DFA）
    - 非确定有限自动机（NFA）
    - 子集法=NFA确定化算法
    - 正规文法与有限自动机的等价性
    - 正规式与有限自动机的等价性
    - DFA的化简

正规文法、正规式、DFA和NFA在接收语言的能力上是互相等价的

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

Regular expression

- 设U、V和W均为正规式
  - 交换律
    - U|V = V|U
  - 结合律
    - U | (V|W) = (U|V) | W
  - 分配律
    - U (V|W) = UV | UW
    - (V|W) U = VU | WU
  - εU = Uε= U

Example

- 例6：令Σ={a,b}，请写出Σ上的正规式和正规集

正规式	正规集
a	{a}
a|b	{a,b}
ab	{ab}
(a|b)(a|b)	{aa, ab, ba, bb}
a *	{ε ,a, aa, ……, 任意个a的串}
(a|b)*	{ε , a, b, aa, ab, ……, 所有由a和b 组成的串}
(a|b)*(aa|bb)(a|b)*	{Σ*上所有含有两个相继的a或两个相继的b组成的串}

Example

- 例7： 令Σ={a,d}，Σ上的正规式a(a | d) *
  - a代表字母
  - d代表数字

- 定义的正规集为：{a,aa,ad,add,…}
- 正规式包括
  - 字母(字母 | 数字)
  - 表示正规集中的每个元素的模式
  - “字母打头的字母数字串”
- 这是Pascal和多数程序设计语言允许的标识符的词法规则

Regular Expression

- 若两个正规式所表示的正规集相同，则称这**两个正规式等价**。如

$b(ab)^*=(ba)^*b$　　$(a^*b^*)^*=(a|b)^*$

Regular Expression

- 正则表达式应用：匹配字符串
- python的正则表达式模块
  - import re
  - p = re.compile(“pattern”)
  - #从头开始匹配字符串
  - p.match(“target”).group()
  - #在字符串中查找
  - p.search(“target”).group()
  - #找到所有结果
  - p.findall(“target”)

https://www.cnblogs.com/huxi/archive/2010/07/04/1771073.html

Regular Expression

正则表达式	python中	字符串例子
a	a	a
ab	ab	ab
a|b	a|b	a,b
a*	a*	a,aaa
a|b|c|d	[abcd]	b,c
(a|b|c)+	[a-c]+	c,cba
a+b|d	a+b|d	ab,d
?	\?	?

Regular Expression

- 例子
  - url get 请求中参数解析
- url get请求
  - http://www.test.com/example?lang=ch&type=1
  - url: http://www.test.com/example
  - 参数：顺序不影响，lang或type都可以在前

参数	值
lang	ch
type	1

如何用正则表达式来匹配某个参数？

Regular Expression

- 思路
  - 首个字母为?或者&
  - 之后为lang=
  - 之后跟至少一个字母或者数字

(?|&)lang=(a|...|z|A|...|Z|0|...|9)+

(\?|&)lang=[a-zA-Z0-9]+

```
>>> import re
>>> str1 = r"http://www.test.com/example?lang=ch&type=1"
>>> str2 = r"http://www.test.com/example?type=1&lang=ch"
>>> p = re.compile(r"(\?|&)lang=[a-zA-Z0-9]*")
>>> p.search(str1).group()
'?lang=ch'
>>> p.search(str2).group()
'&lang=ch'
```

Finite Automata

- **有限自动机**(也称有穷自动机)作为一种识别装置，它能准确地识别正规集，即识别正规文法所定义的语言和正规式所表示的集合，引入有限自动机这个理论，正是为词法分析程序的自动构造寻找特殊的方法和工具。
- 有限自动机分为两类：**确定的有限自动机**(Deterministic Finite Automata)和**不确定的有限自动机**(Nondeterministic Finite Automata)。

DFA

- **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
- DFA的确定性
  - 状态转换函数f是一个$S \times \Sigma \rightarrow S$的单值部分映射
    - 若f是一个多值函数,则M是非确定有限自动机
  - f(s,a) = s’
    - 表示：输入字符a，则现行状态s将转换到下一状态s’
    - s’称为s的一个后继状态
    - f(s,a)唯一地确定了下一状态

DFA

- **例8：DFA M=({0，1，2，3}，{a，b}，f，0，{3})，其中：f定义如下：**

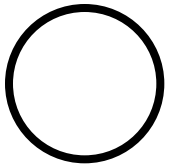
f(0，a)=1　　f(0，b)=2
f(1，a)=3　　f(1，b)=2
f(2，a)=1　　f(2，b)=3
f(3，a)=3　　f(3，b)=3

DFA matrix

- 一个DFA可以用一个矩阵表示（**状态转换矩阵**），该矩阵的**行表示状态，列表示输入字符**，矩阵元素表示相应状态行和输入字符列下的新状态，即k行a列为f(k,a)的值。

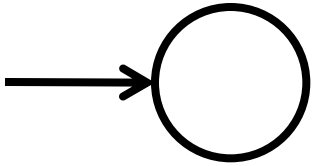
状态\字符	a	b
0	1	2
1	3	2
2	1	3
3	3	3

DFA diagram

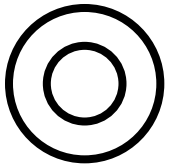


is a state

is a transition



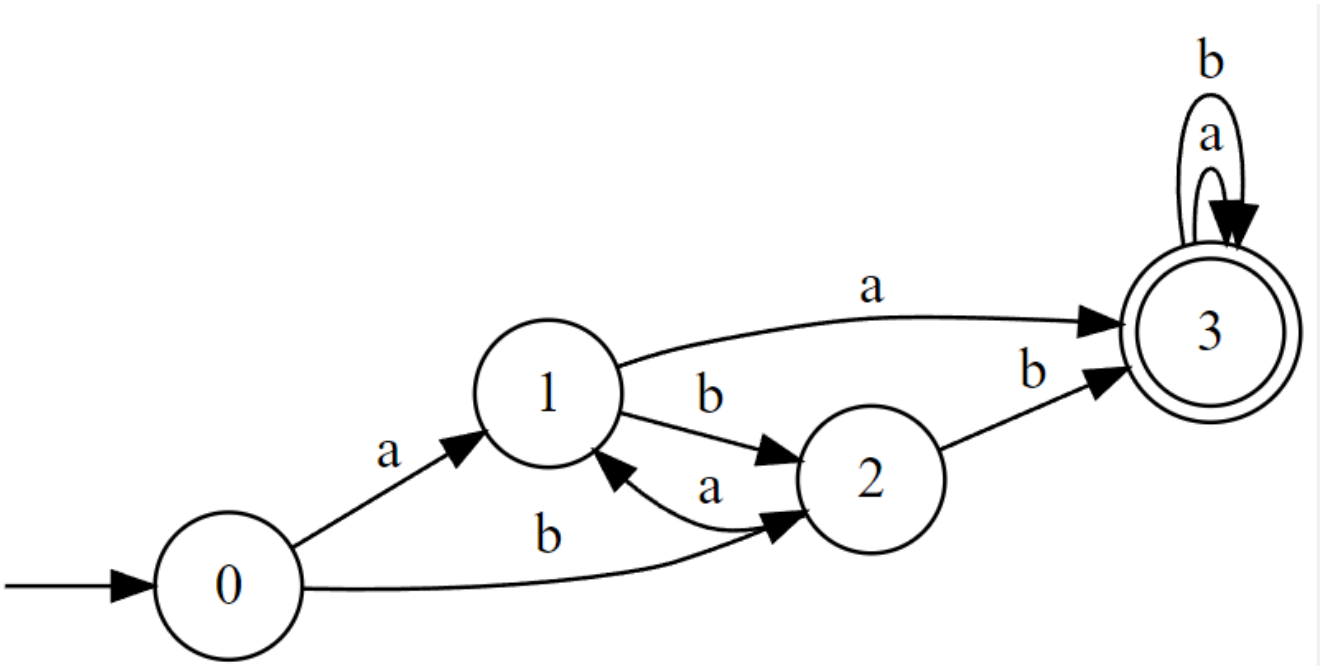
is the start state



is a final state

DFA digram

- DFA也可以表示成一个状态图(或称**状态转换图**)。假定DFA M含有m个状态，n个输入字符，那么这个状态图含有m个结点，每个结点最多有n个弧射出，整个图含有唯一一个初态结点和若干个终态结点。



DFA

- 对于Σ^*中的任何字α，若存在一条从初态到某一终态的道路，且这条路上所有弧上的标记符连接成的字等于α，则称α为DFA M所识别(接收)
- DFA M所识别的字的全体记为L(M)。若M的初态结点同时又是终态结点，则空字ε可为M所识别

DFA acceptance

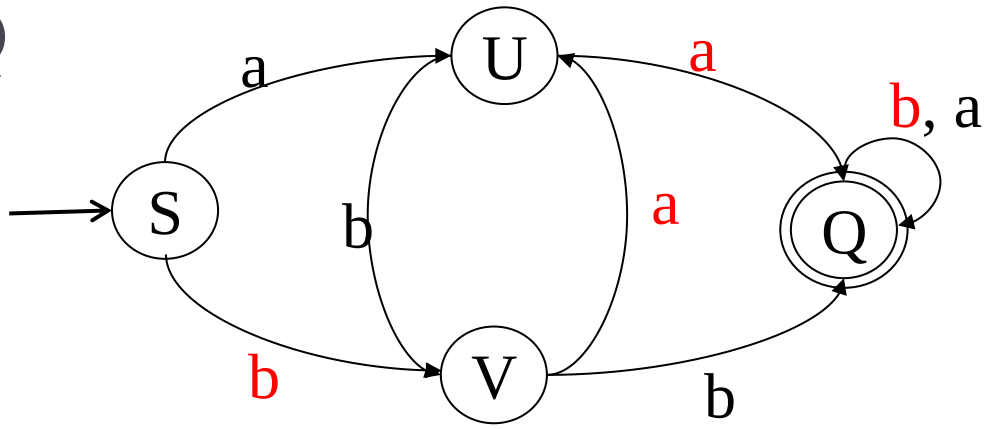
- **例9：证明t=baab被下图的DFA所接受。**
  - ∵a,b∈∑　　∴baab∈∑*
  - f (S, baab) =f (f (S, b) , aab)

    = f (V, aab) = f (f (V, a) , ab)

    =f (U, ab) =f (f (U, a) , b)

    =f (Q, b) =Q

  Q属于终态。
  - 得证。



DFA acceptance

- 证法二：
  - 根据上述状态转换图，可以构造确定有限自动机DFA M = 〈{S,U,V,Q}，{a,b}，F_M，S，{Q}〉
  - 其中，F_M是DFA M的状态转换函数，定义如下：
    - $F_M(S,a) = U$　　$F_M(S,b) = V$
    - $F_M(U,a) = Q$　　$F_M(U,b) = V$
    - $F_M(V,a) = U$　　$F_M(V,b) = Q$
    - $F_M(Q,a) = Q$　　$F_M(Q,b) = Q$
  - ∵a,b∈∑　∴baab∈∑*
  - 对于t = baab，　DFA M存在一条经过S、V、U、Q和Q的通路，使得该通路上所有弧的标记符连接成的字等于t
  - 因此，t 被DFA M所接受

DFA

- 对于任何两个有限自动机M和M′，如果 $L(M)=L(M')$，则称M与M′是等价的.
- 可以证明：Σ上的子集$V\subseteq\Sigma^*$是正规集，当且仅当存在Σ上的DFA M，使得$V = L(M)$。
  - L(M)：语言是句子的集合；句子中的元素是终结符
  - 子集V：正规集是正规式的集合
  - 即：这些字(符串) 都能够被DFA M识别
  - 则：句子中的终结符都是正规式

DFA

- **DFA的确定性**表现在转换函数**f：K×Σ→K是一个单值函数**，也就是说，对任何状态k∈K，和输入符号a∈Σ，f(k,a)唯一地确定了下一个状态。从状态转换图来看，若字母表Σ含有n个输入字符，那末**任何一个状态结点最多有n条弧射出，而且每条弧以一个不同的输入字符标记。**

DFA program

- DFA的行为很容易用程序来模拟
  - DFA M = (K, Σ, f, S, Z) 的行为的模拟程序

```
state = S
c = getchar()
while(c != EOF):
    state = f(state, c)
    c = getchar()
return (c in Z)
```

NFA

- **定义：一个非确定有限自动机(NFA) M是一个五元式M=(S, Σ, f, S_0, Z)，其中：**
  - 1. S： 有穷状态集；
  - 2. Σ：输入字母表(有穷)；
  - 3. f：状态转换函数，为$S\times\Sigma^*\rightarrow 2^S$的部分映射(非单值)；
  - 4. $S_0\subseteq S$是非空的初态集；
  - 5. Z：终态集(可空)，$Z\subseteq S$ 。

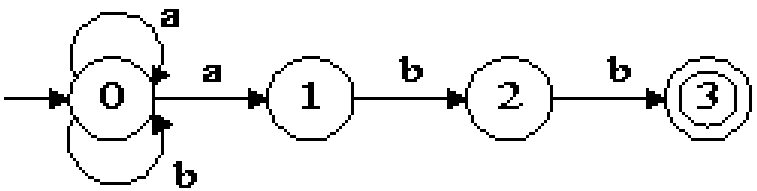
NFA

- **NFA的不确定性**
  - 状态转换函数为$S\times\Sigma^{*}\rightarrow 2^{S}$的部分映射(非单值)

S	2^S	数量
{0}	{}, {0}	2
{0, 1}	{}, {0}, {1}, {0,1}	4
{0, 1, 2}	{}, {0}, {1}, {2}, {0, 1}, {0, 2}, {1, 2}, {0, 1, 2}	8

NFA

- 例10：识别由正规式(a|b)*abb说明的记号的NFA定义如下：
  - S={0,1,2,3}, Σ={a,b}，s_0 = 0，F={3}
  - f = { f(0,a)=0，f(0,a)=1，f(0,b)=0，
    f(1,b)=2，f(2,b)=3 }
  - 它的转换图和转换矩阵表示如图所示。在转换矩阵中，需指出0是初态，3是终态。



(a)

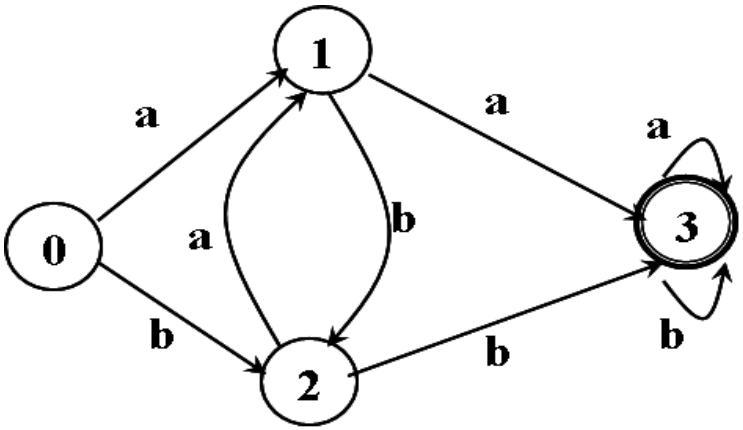
	a	b
0	{0,1}	{0}
1	—	{2}
2	—	{3}
3	—	—

(b)

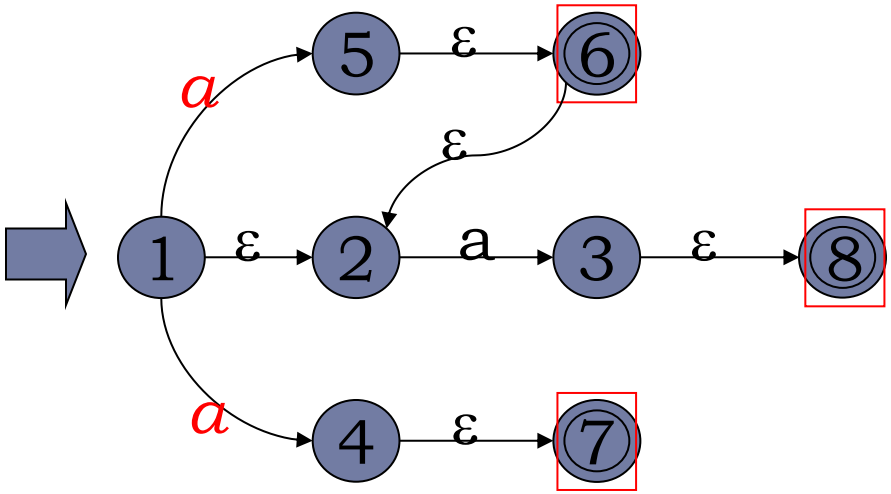
NFA

- NFA的特点是它的不确定性，即在当前状态下，对同一个输入字符，可能有多于一个的下一状态转移。
  - 不确定性反映在NFA的定义中，就是f函数是一对多的；
  - 反映在转换图中，就是从一个节点可通过多于一条标记相同字符的边转移到不同的状态；
  - 反映在转换矩阵中，就是$f[s_i, s_j]$中不是一个单一状态，而是一个状态的集合。
- 从状态图中看NFA 和DFA的区别：
  - 1．弧上的标记可以是Σ^*中的一个字，而不一定是单个字符
  - 2. 同一个字可能出现在同状态射出的多条弧上。

DFA & NFA



状态转换图



DFA

NFA

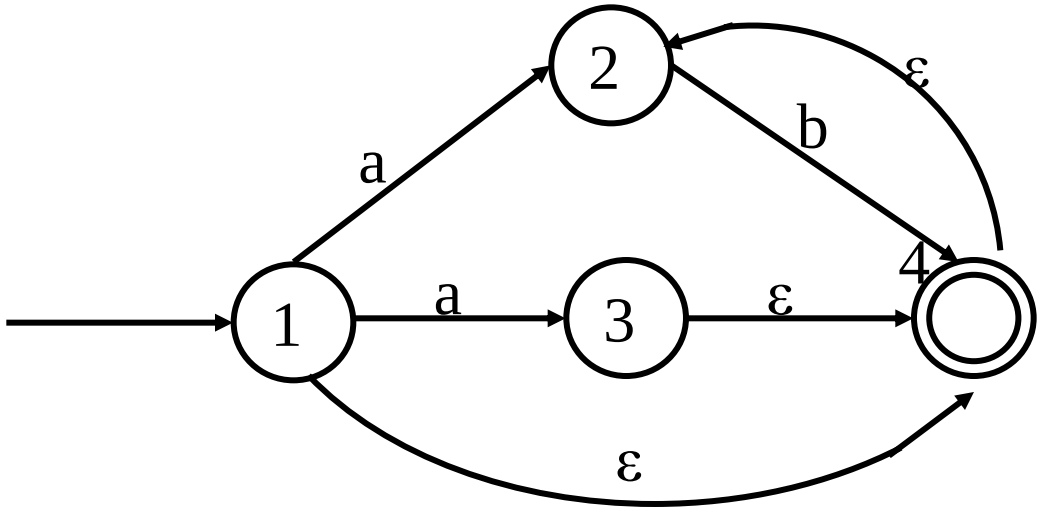
DFA=<状态集,字母表,单值映射,初态,终态集>

NFA=<状态集,字母表,多值映射,初态集,终态集>

NFA

- 对于Σ^*中的任何串t，若存在一条从某一初态结到某一终态结的道路，且这条道路上所有弧的标记字依序连接成的串(不理睬那些标记为ε的弧)等于t，则称t可为NFA M所识别(读出或接受)。若M的某些结既是初态结又是终态结，或者存在一条从某个初态结到某个终态结的道路，其上所有弧的标记均为ε，那么空字可为M所接受。
- NFA M所能接受的符号串的全体记为L(M)。

NFA



- 上图所示的NFA的以下两个转换序列都可以接受串abb：

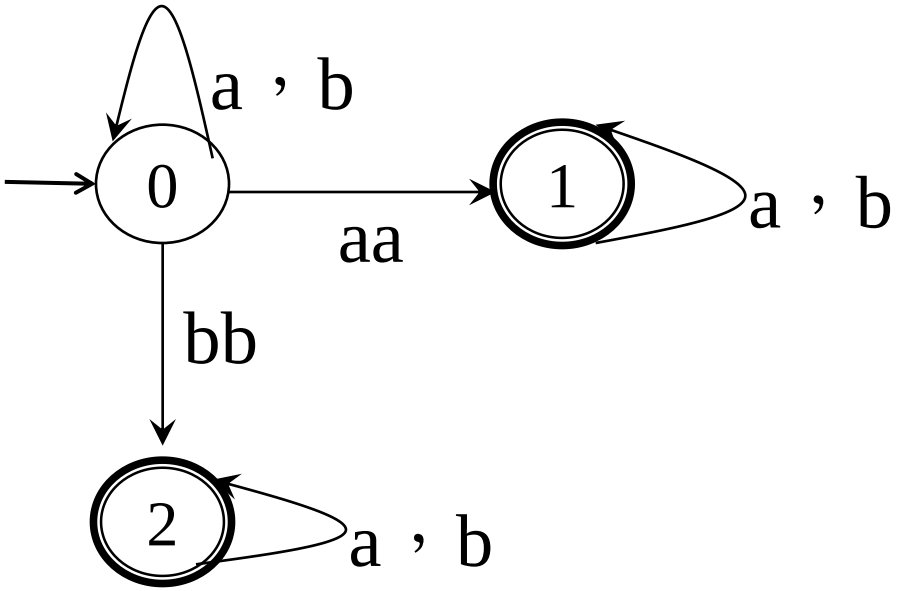
$$\rightarrow 1 \xrightarrow{a} 2 \xrightarrow{b} 4 \xrightarrow{\varepsilon} 2 \xrightarrow{b} 4$$

$$\rightarrow 1 \xrightarrow{a} 3 \xrightarrow{\varepsilon} 4 \xrightarrow{\varepsilon} 2 \xrightarrow{b} 4 \xrightarrow{\varepsilon} 2 \xrightarrow{b} 4$$

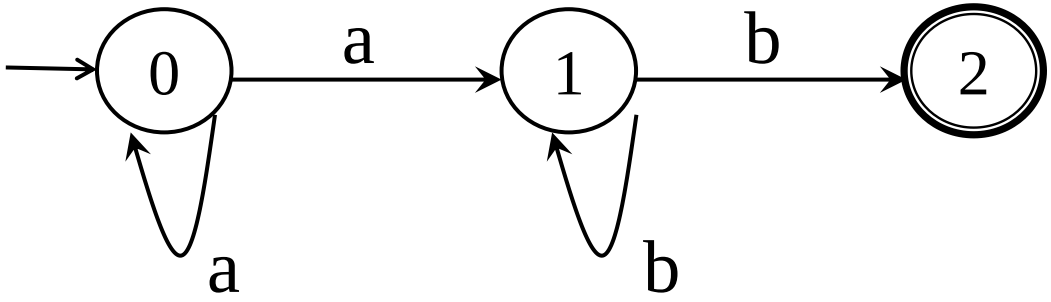
NFA

- 定义：对于任何两个有限自动机M和M’，如果L(M)=L(M’)，则称M与M’等价。
- 自动机理论中一个重要的结论：判定两个自动机等价性的算法是存在的。
- 对于每个NFA M存在一个DFA M’，使得L(M)=L(M’)。亦即DFA与NFA描述能力相同。
- DFA是NFA的特例。

NFA – 例12



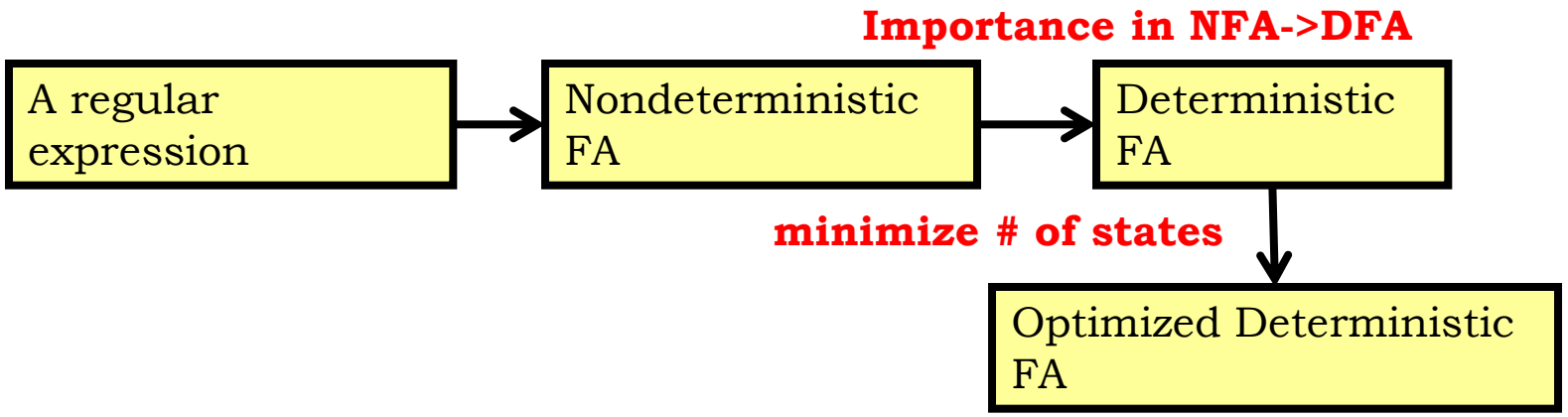
识别所有含相继两个a
或相继两个b的字



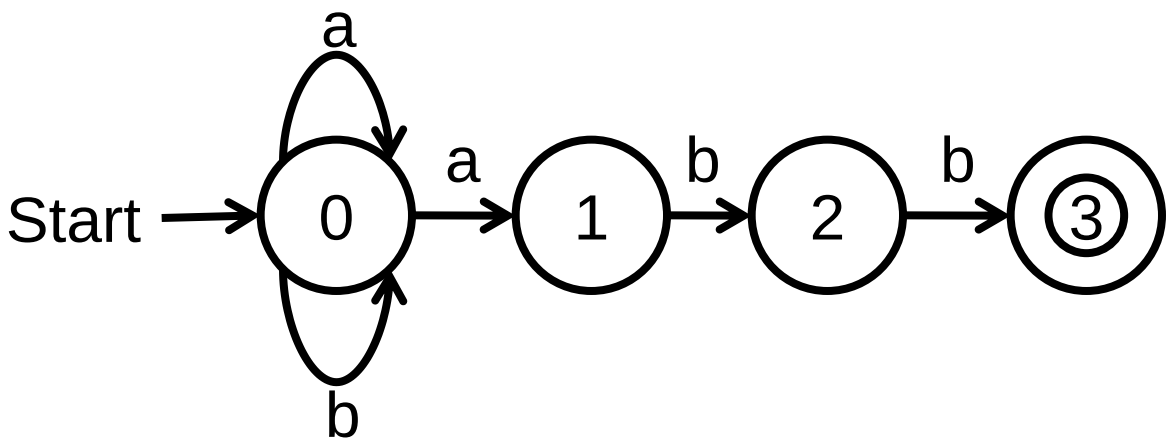
$\{a^m b^n \mid m，n \geq 1\}$

NFA

- DFA是NFA的特例
- 对每个NFA N一定存在一个DFA M，使得 L(M)=L(N)。对每个NFA N存在着与之等价的DFA M。
- 有一种算法，将NFA转换成接受同样语言的DFA。这种算法称为**子集法 (subset construction)**。
- **与某一NFA等价的DFA不唯一**。



What's the problem with NFA?

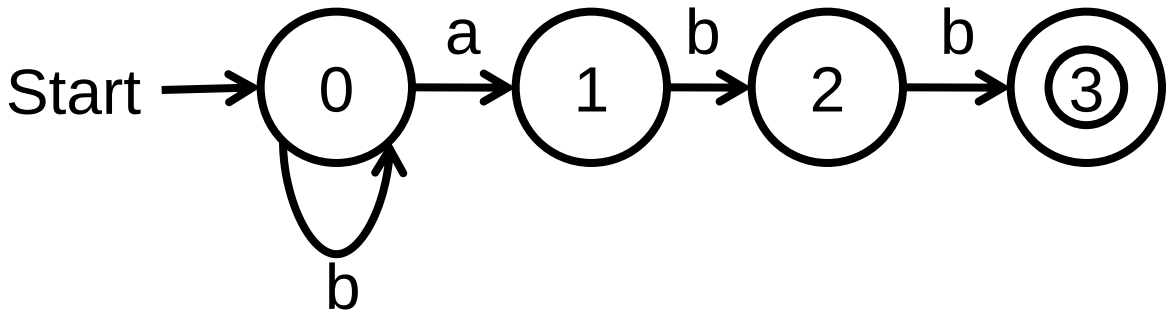


Ambiguous!!!

Which one should we select?

Input : babb Processing Token b a

What's the problem with NFA?



No Ambiguous!!!

It have unique path!

Input : babb Processing Token b a b b

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。

NFA subset construction

- 定义对状态集合I的几个有关运算
  - 1. 状态集合I的ε-闭包，表示为ε-closure(I)，定义为一状态集，是状态集I中的任何状态s经任意条ε弧而能到达的状态的集合。
  - 状态集合I的任何状态s都属于ε-closure(I)。
  - 2. 状态集合I的a弧转换，表示为move(I,a)定义为状态集合J，其中J是所有那些可从I中的某一状态经过一条a弧而到达的状态的全体。

NFA subset construction

- 构造ε-closure(I)的算法：
  - 将I的所有状态压入栈中
  - 将ε-closure(I)初始化为I
  - while （栈不为空）do {
    将栈顶元素t弹出栈中；
    for（每个满足如下条件的u：从t出发有一个标号为ε的转换到达状态u）
    if （u不在ε-closure(I) 中）then {
    将u加入到ε-closure(I) 中；
    将u压入栈中；
    }
    }

NFA subset construction

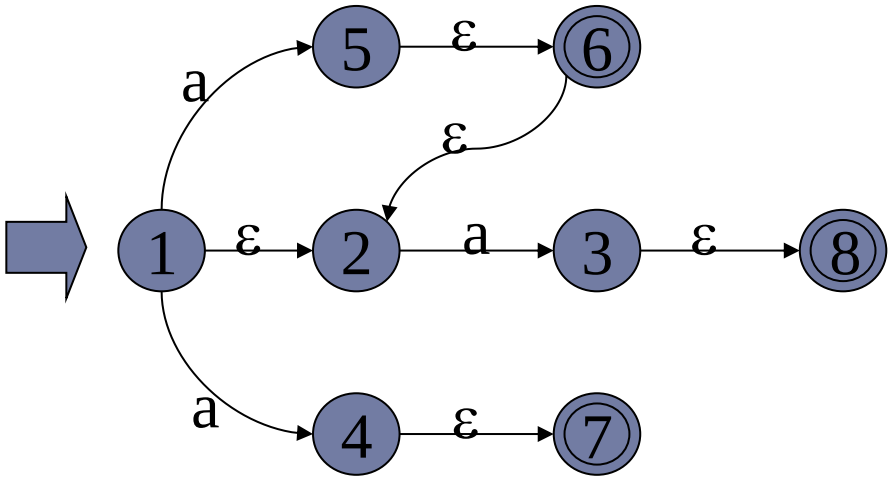
- 状态集合I的有关运算的示例

  I={1}，ε-closure(I)={1,2};

  I={5}，ε-closure(I)={5,6,2};

  move({1,2},a)={5,3,4}

  ε-closure({5,3,4})={2,3,4,5,6,7,8};



NFA subset construction

- 假设NFA N=(K，∑，f，K_0，K_t)按如下办法构造一个DFA M=(S，∑，d，S_0，S_t)，使得L(M)=L(N)：
  - 1. M的状态集S由K的一些子集组成。用[S_1 S_2… S_j]表示S的元素，其中S_1，S_2,… S_j是K的状态。并且约定，状态S_1，S_2,… S_j是按某种规则排列的，即对于子集{S_1，S_2}={ S_2，S_1}来说，S的状态就是[S_1 S_2]；

NFA subset construction

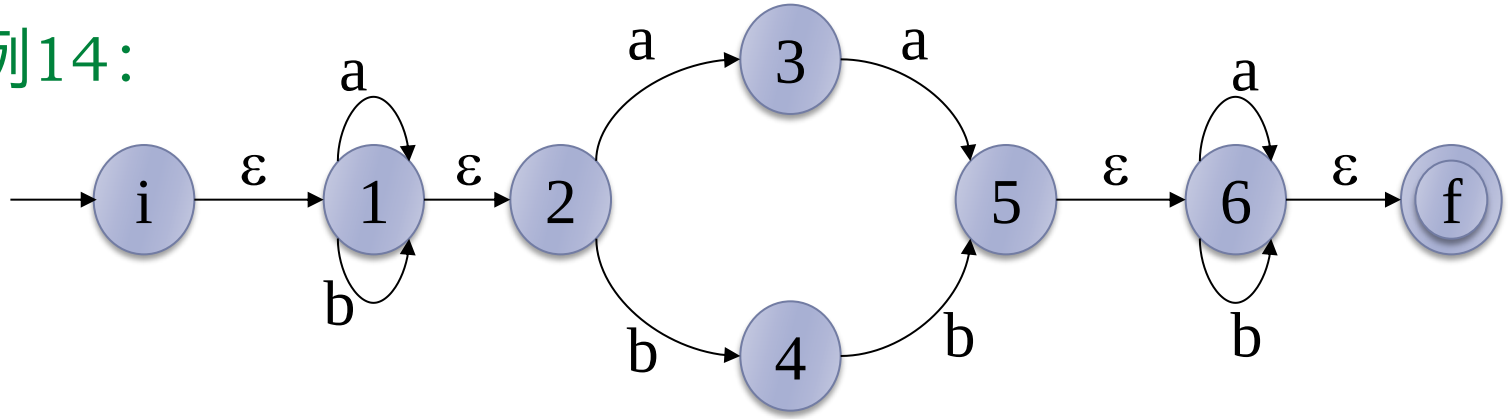
- 2. M和N的输入字母表是相同的，即是Σ;
- 3. 转换函数是这样定义的:
  - $d([S_1\ S_2,\ldots S_j], a) = [R_1R_2\ldots R_t]$
  - 其中
  - $\{R_1, R_2,\ldots, R_t\} = \varepsilon\text{-closure}(\ move(\{S_1, S_2,\ldots S_j\}, a)\ )$
- 4. $S_0 = \varepsilon\text{-closure}(K_0)$为M的开始状态;
- 5. $S_t=\{[S_i\ S_k\ldots S_e]$，其中$[S_i\ \ S_k\ldots S_e]\in S$且$\{S_i\ ,\ S_k,\ldots S_e\}\cap K_t\neq\Phi\}$

NFA subset construction

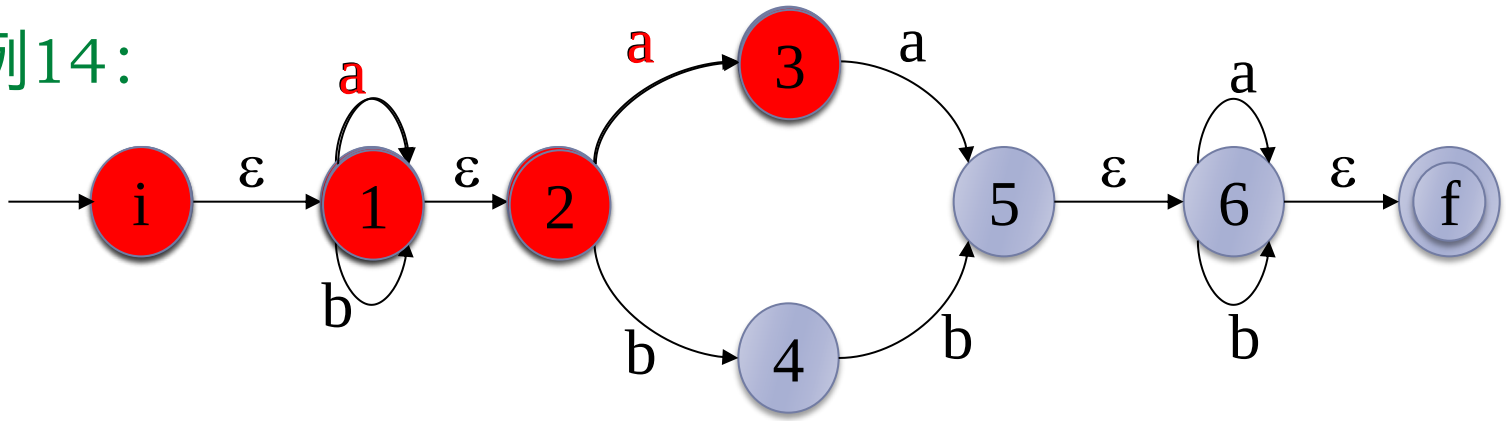
- 构造NFA N的**状态K的子集**的算法：
  - 假定所构造的子集族为C，即C= (T_1, $T_{2,}$,… T_I),其中T_1, $T_{2,}$,… T_I为状态K的子集。
  - 1 开始，令ε-closure(K_0)为C中唯一成员，并且它是未被标记的。
  - 2 while （C中存在尚未被标记的子集T）do

```
{
        标记T;
                for 每个输入字母a  do
        {
                U:= ε-closure(move(T,a)) ;
                 if  U不在C中  then
                        将U作为未标记的子集加在C中
         }
}
```

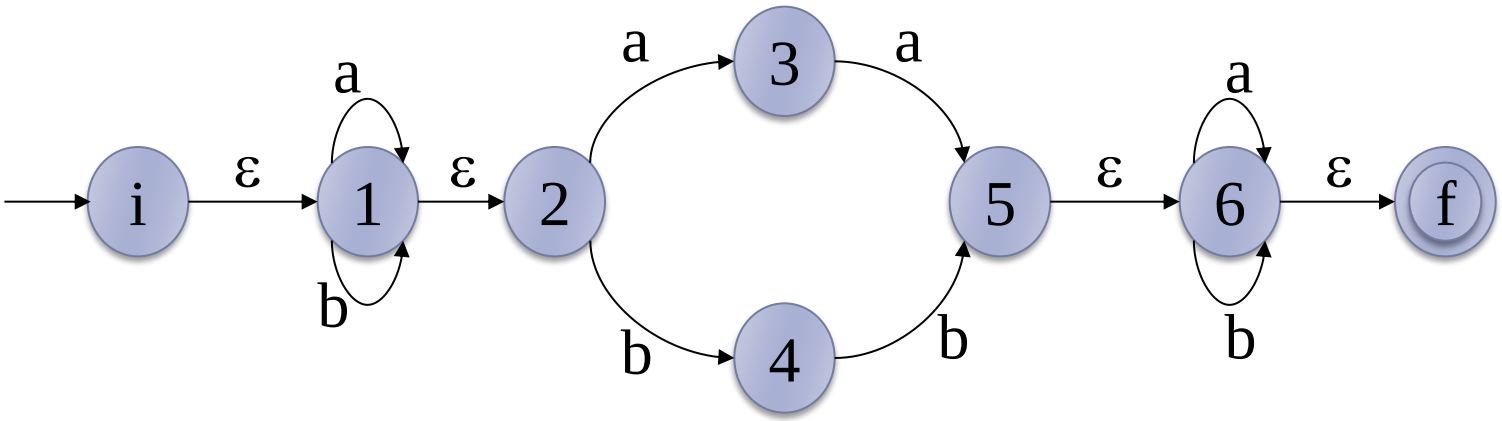
例14:



例14：

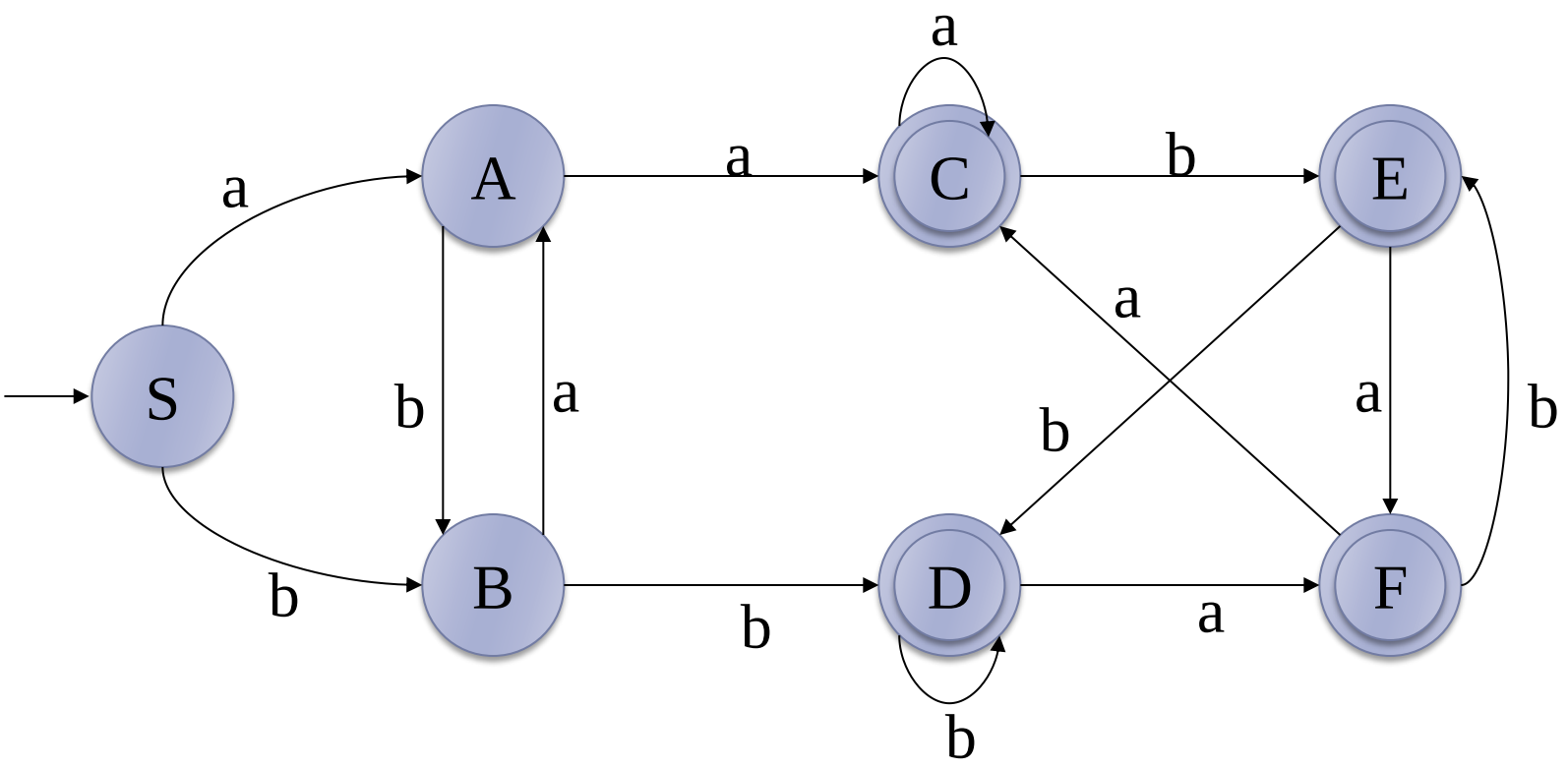


I	{i}	
$J = \epsilon\_closure(I)$	{i, 1, 2}	新状态1
$K = move(J, a)$	{1, 3}	
$L = \epsilon\_closure(K)$	{1, 2, 3}	新状态2



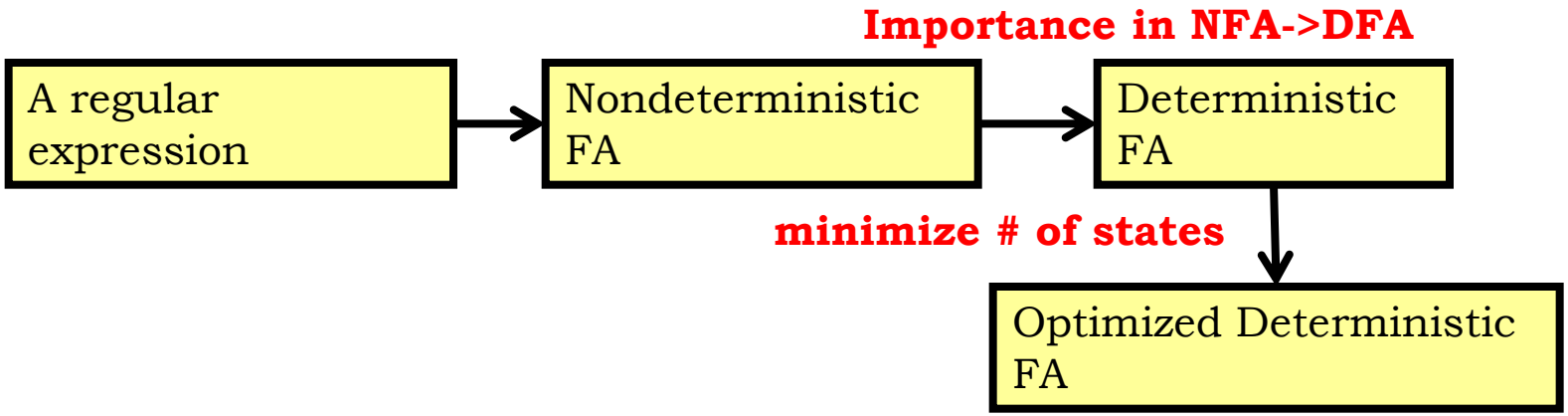
	I_a	I_b
{i,1,2} S	{1,2,3} A	{1,2,4} B
{1,2,3} A	{1,2,3,5,6,f} C	{1,2,4} B
{1,2,4} B	{1,2,3} A	{1,2,4,5,6,f} D
{1,2,3,5,6,f} C	{1,2,3,5,6,f} C	{1,2,4,6,f} E
{1,2,4,5,6,f} D	{1,2,3,6,f} F	{1,2,4,5,6,f} D
{1,2,4,6,f} E	{1,2,3,6,f} F	{1,2,4,5,6,f} D
{1,2,3,6,f} F	{1,2,3,5,6,f} C	{1,2,4,6,f} E

NFA subset construction



RE and FA equivalence

- **定理：**
  - 1. Σ上的每个非确定有限自动机M所能识别的字的全体L(M)是Σ上的一个正规集。
  - 2. 对于Σ上的每个正规集S，存在一个Σ上的确定有限自动机M，使得S=L(M)。
- **对转换图概念拓广，令每条弧可用一个正规式作标记。(对一类输入符号)**

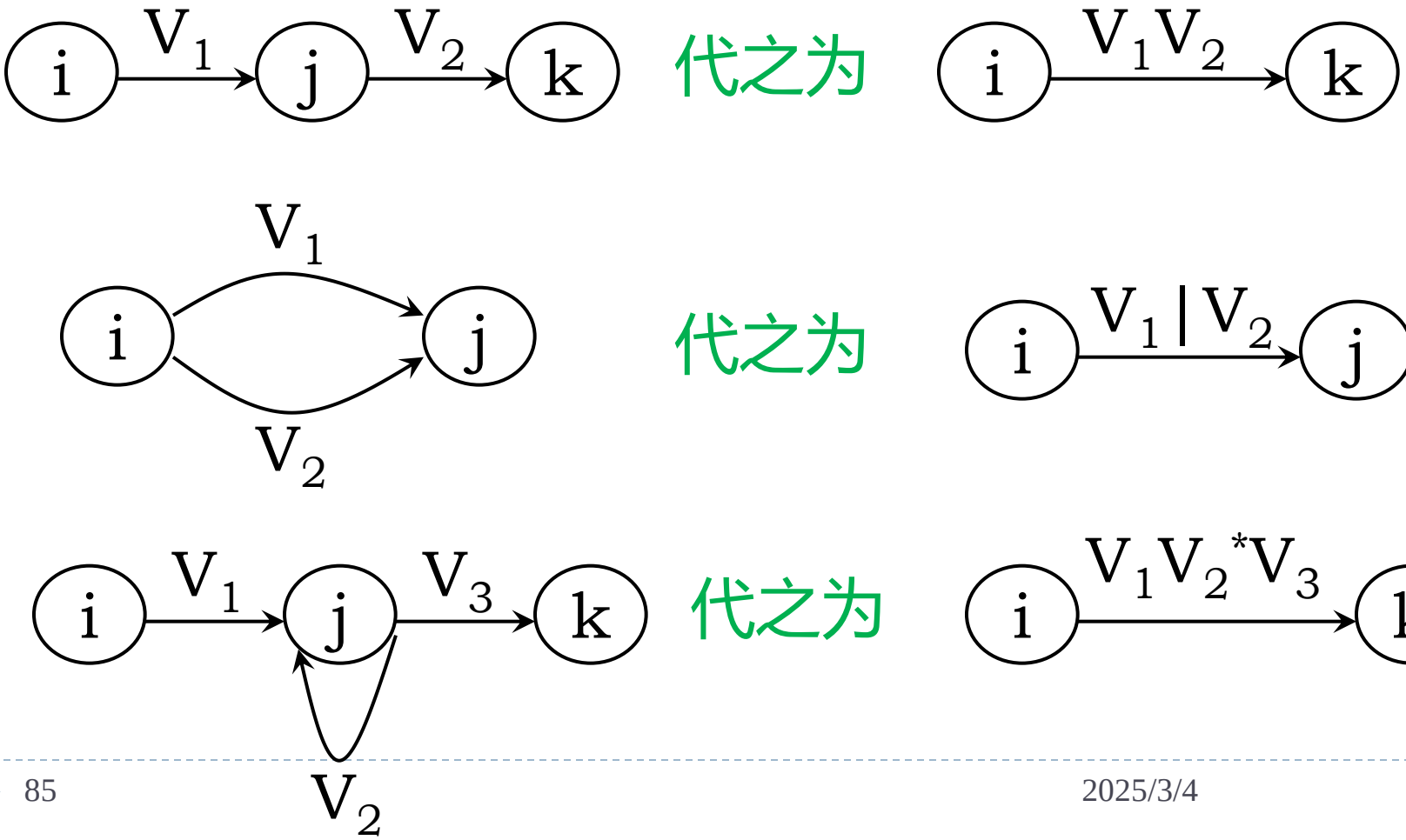


RE and FA equivalence

- **证明：**
- **1 对Σ上任一NFA M，构造一个Σ上的正规式V，使得L(V)=L(M)。**
  - 首先，在M的转换图上加进两个状态X和Y，从X用ε弧连接到M的所有初态结点，从M的所有终态结点用ε弧连接到Y，从而形成一个新的NFA，记为M’，它只有一个初态X和一个终态Y，显然L(M)=L(M’)。

RE and FA equivalence

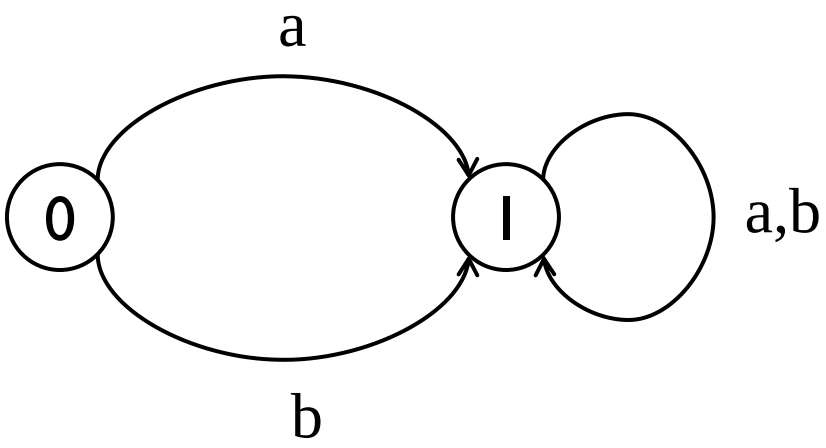
- 然后，反复使用下面的一条规则，逐步消去的所有结点，直到只剩下X和Y为止



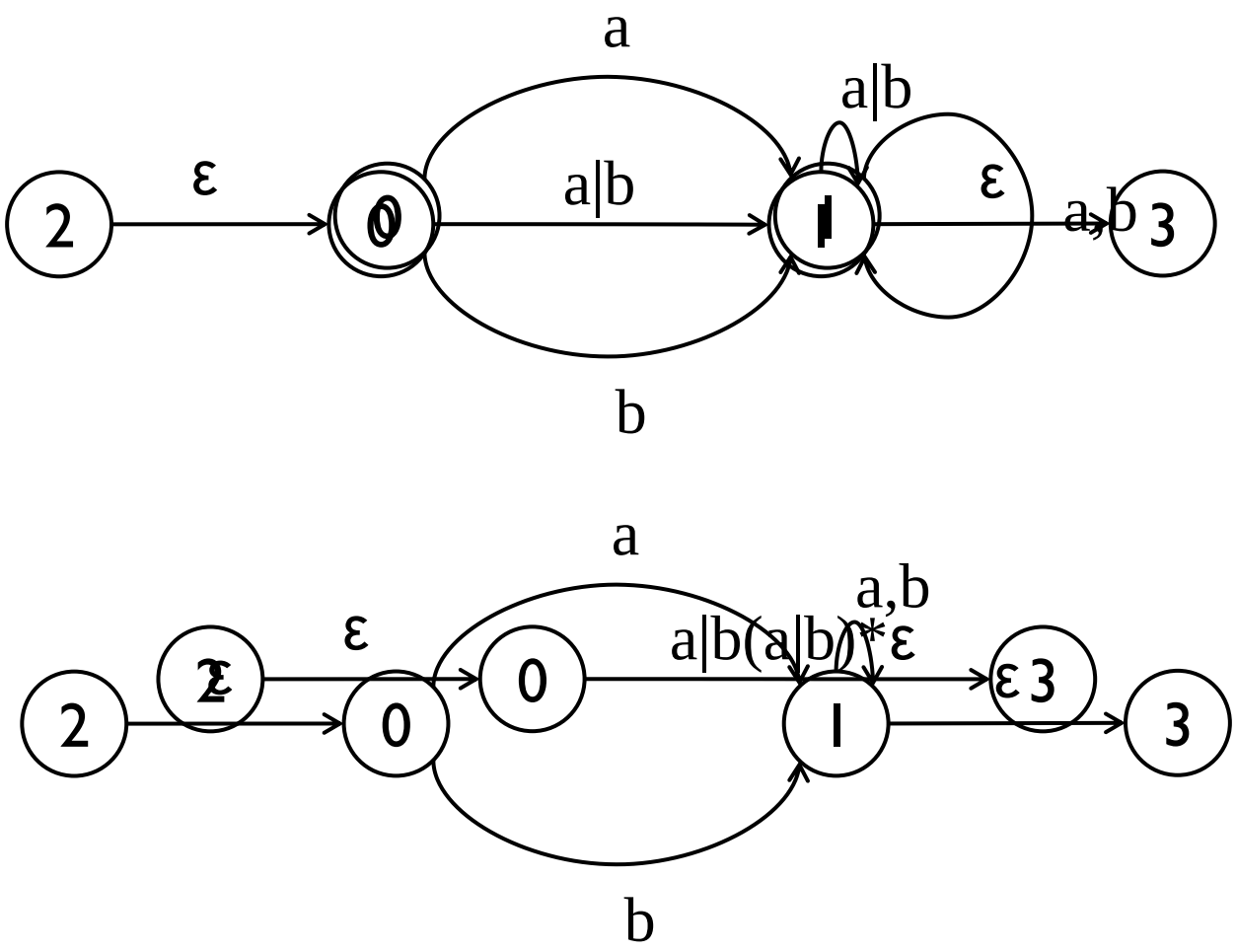
RE and FA equivalence

- 最后，X到Y的弧上标记的正规式即为所构造的正规式V，
- 显然L(V)=L(M)=L(M’)

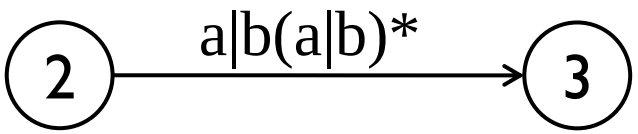
- 例15:

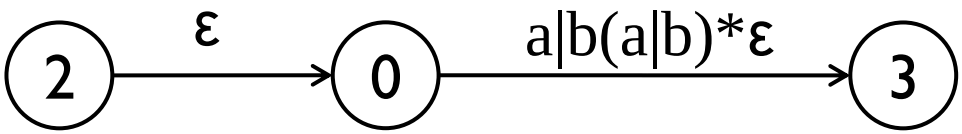


RE and FA equivalence



RE and FA equivalence



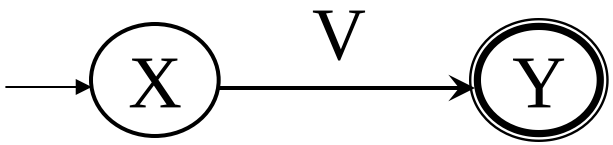


RE and FA equivalence

- **证明：**
- **2　对于Σ上的每个正规式V存在一个Σ上的DFA M使得L(V)=L(M)**
- **分两步证明：**
  - 1) 构造Σ上的NFA M’ 使得 L(V)=L(M’)
  - 2) 把M’确定化为等价的DFA。
    - **NFA确定化——采用子集法**

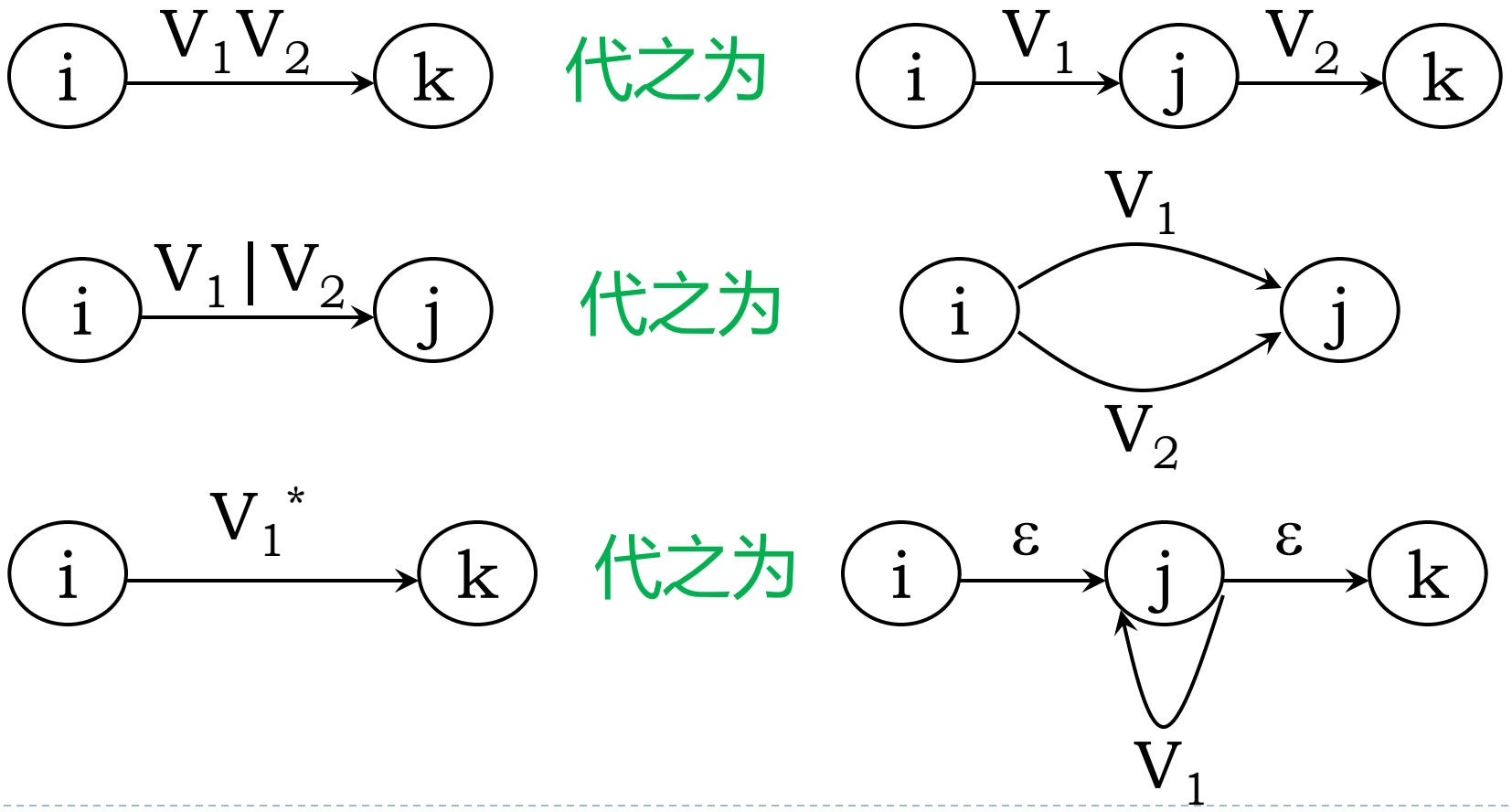
RE and FA equivalence

- 1) 构造Σ上的NFA M’ 使得 L(V)=L(M’)
  - **首先，把V表示成**



RE and FA equivalence

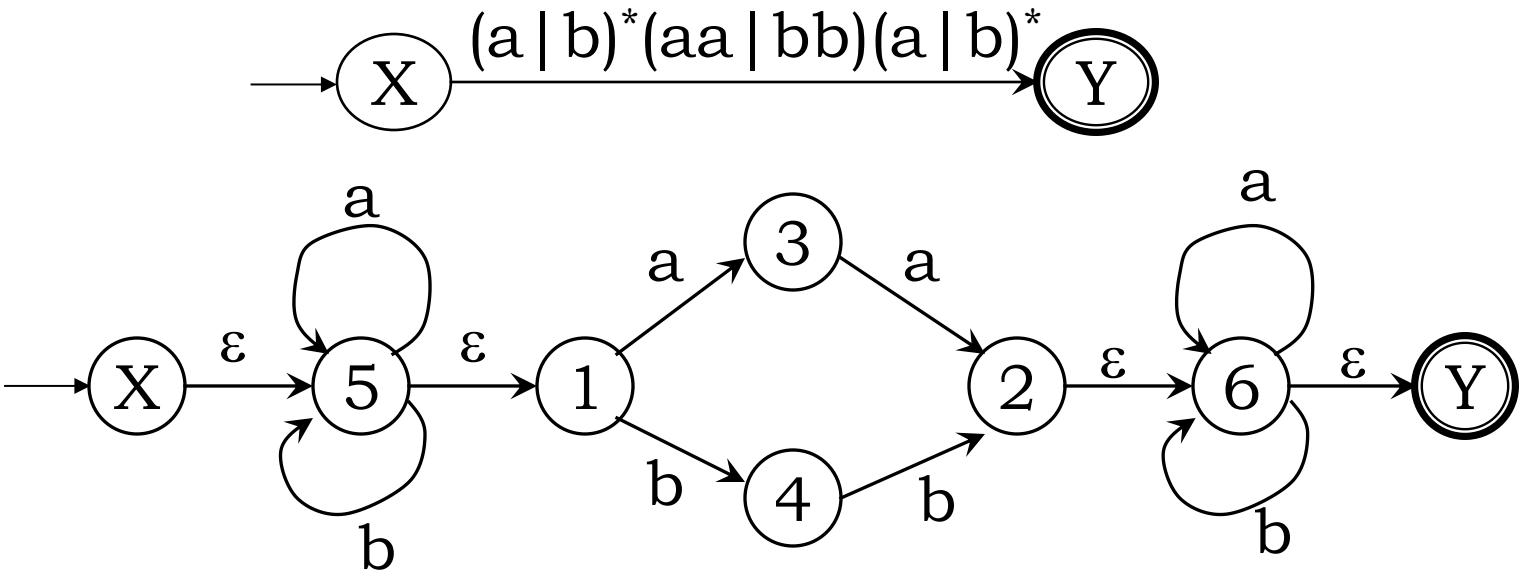
然后，按下面的三条规则对V进行分裂：



RE and FA equivalence

逐步把这个图转变为每条弧只标记为Σ上的一个字符或ε，最后得到一个NFA M’，显然L(M’)=L(V)

- 例16：(a|b)*(aa|bb)(a|b)*



Excercise

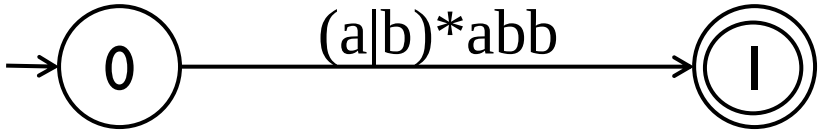
- 例17:

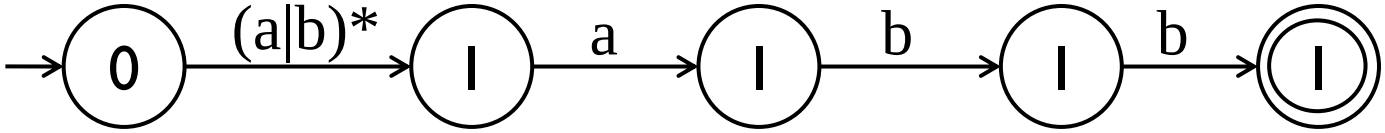
- （1）L(R) =(a|b)*abb，构造DFA M使L(M)=L(R)

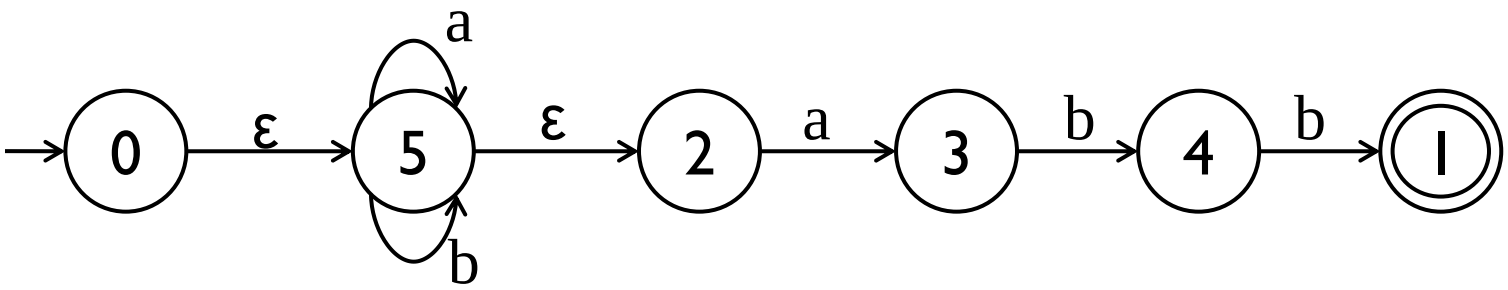
- （2）L(R) =a*b*abb，构造DFA M使L(M)=L(R)

Ex

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

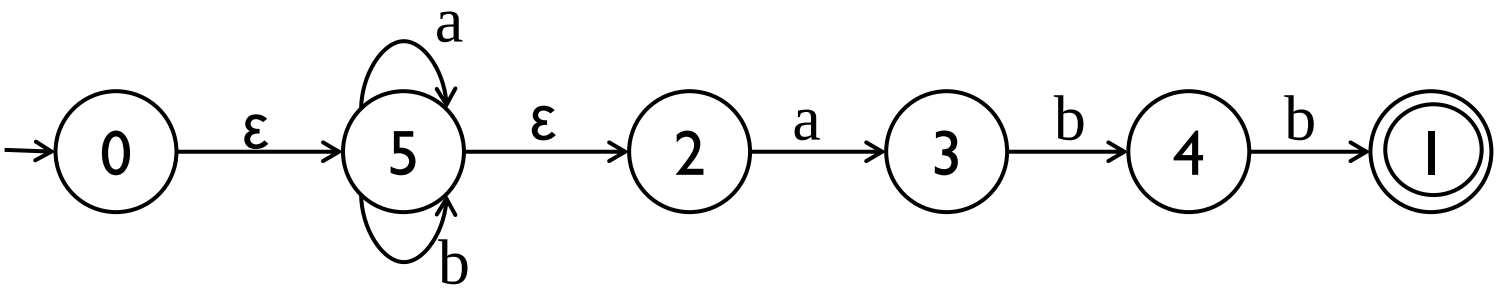






Ex

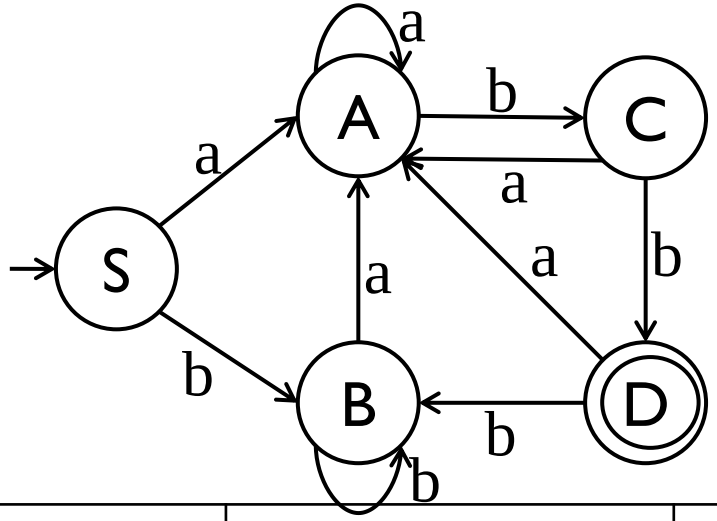
- (a|b)*abb -> NFA -> DFA



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

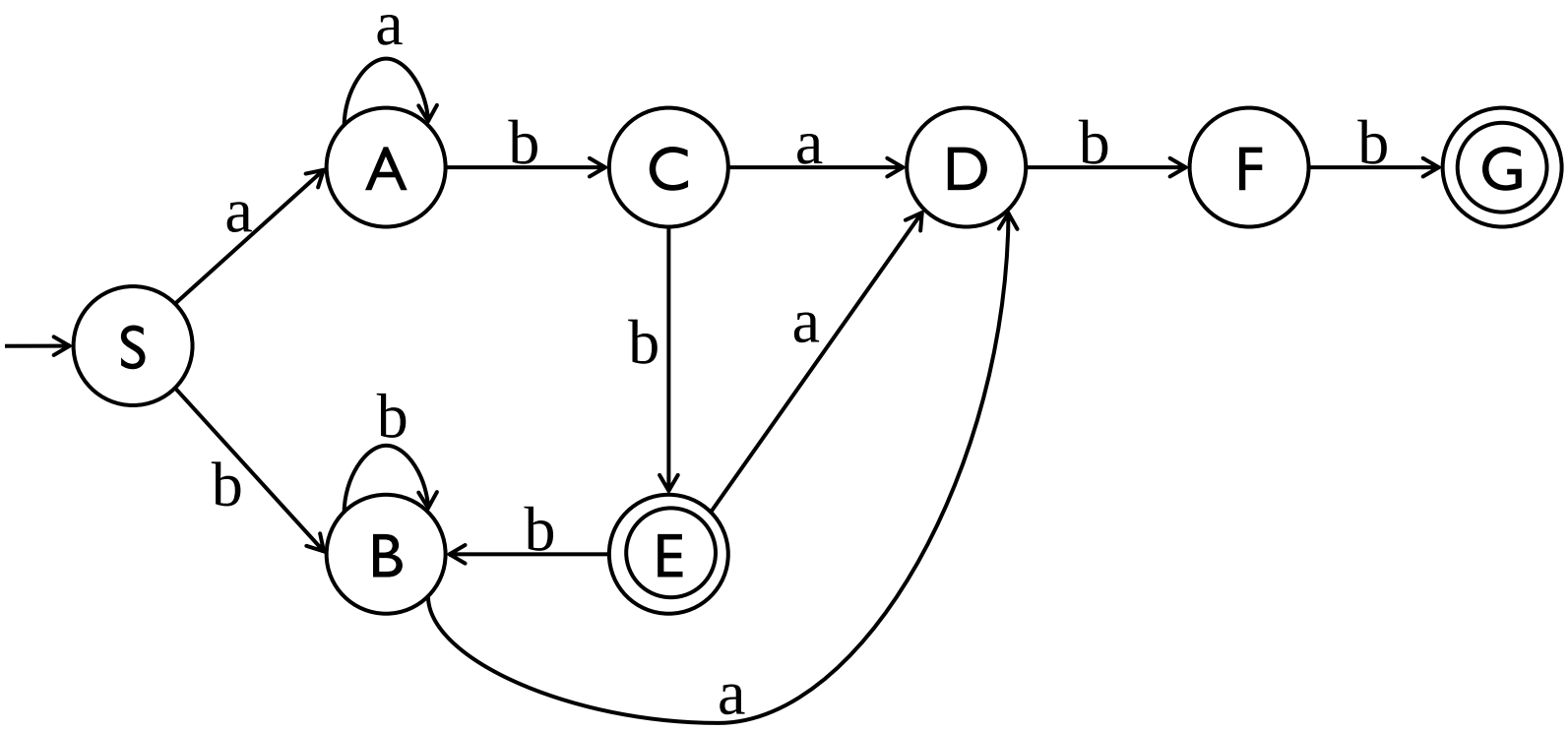
- $(a|b)^*abb$ -> NFA -> DFA



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

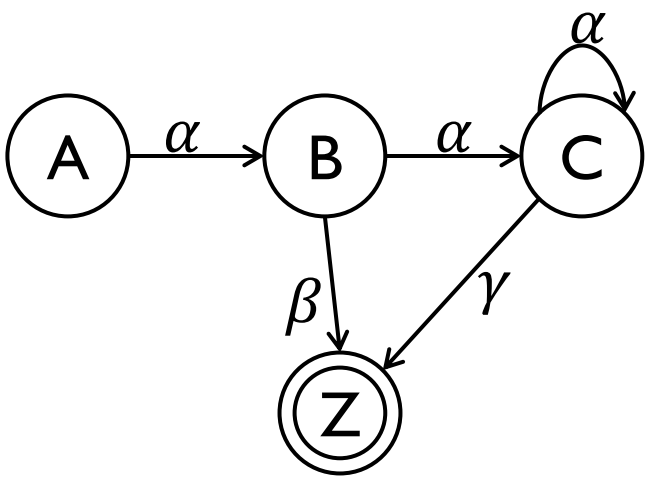


Regular grammar and FA

- 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。

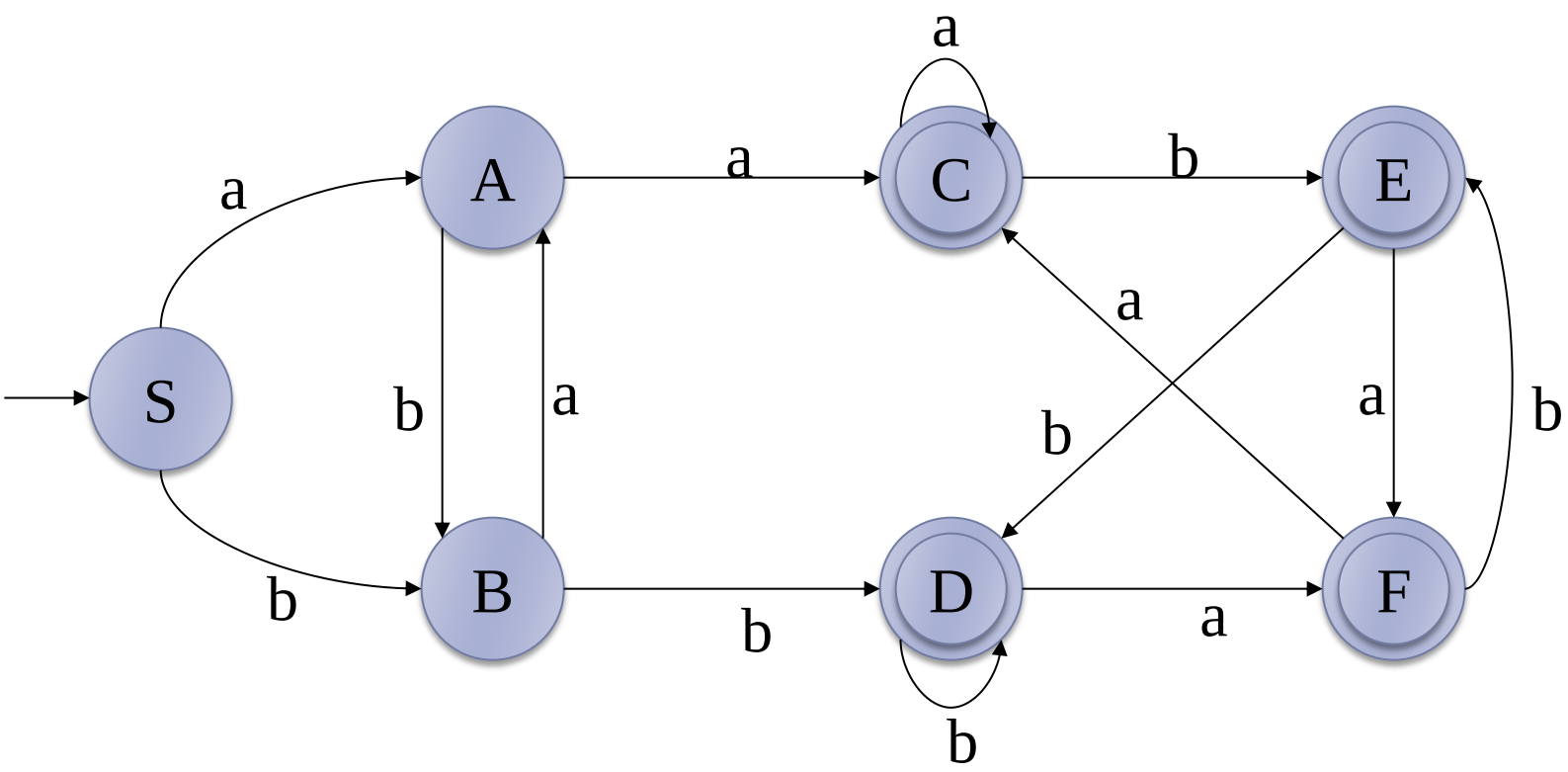
- 所谓有限自动机的多余状态，是指这样的状态：从自动机的开始状态出发，任何输入串也不能到达的那个状态；或者从这个状态没有通路到达终态。

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

- 例18： C和D同是终态,读入a到达C和F, C和F同是终态, C和F读入a都到达C,读入b都到达E. C和F等价



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}

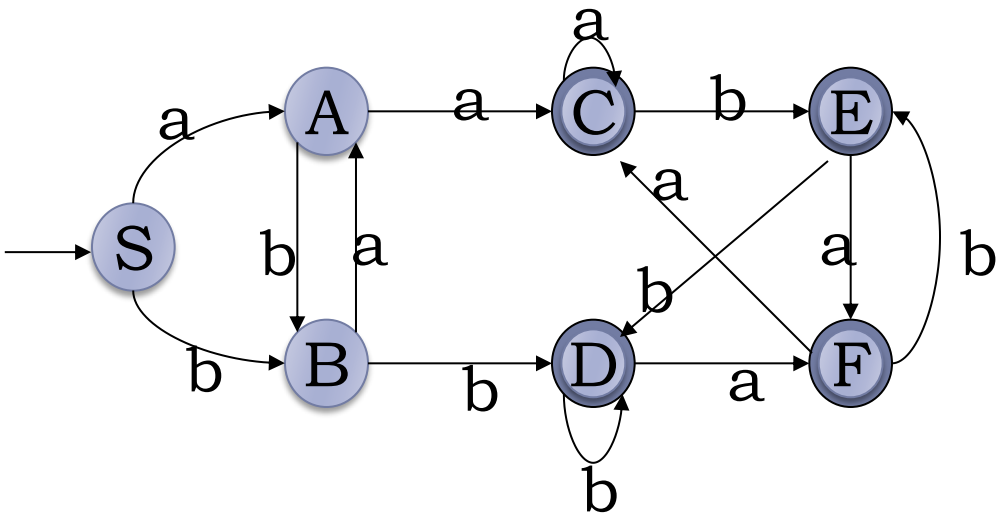
$\prod_1$:{S,A,B}

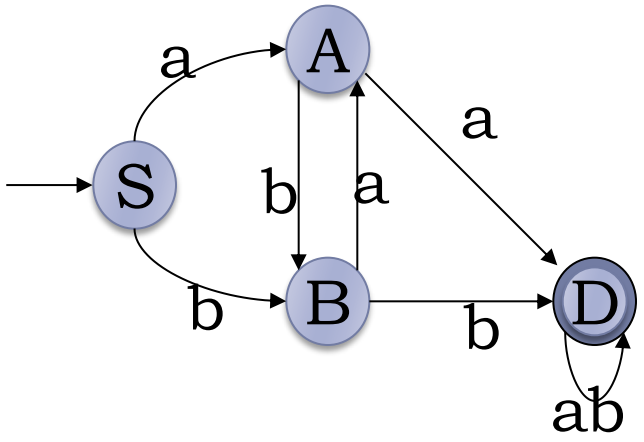
a

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

b

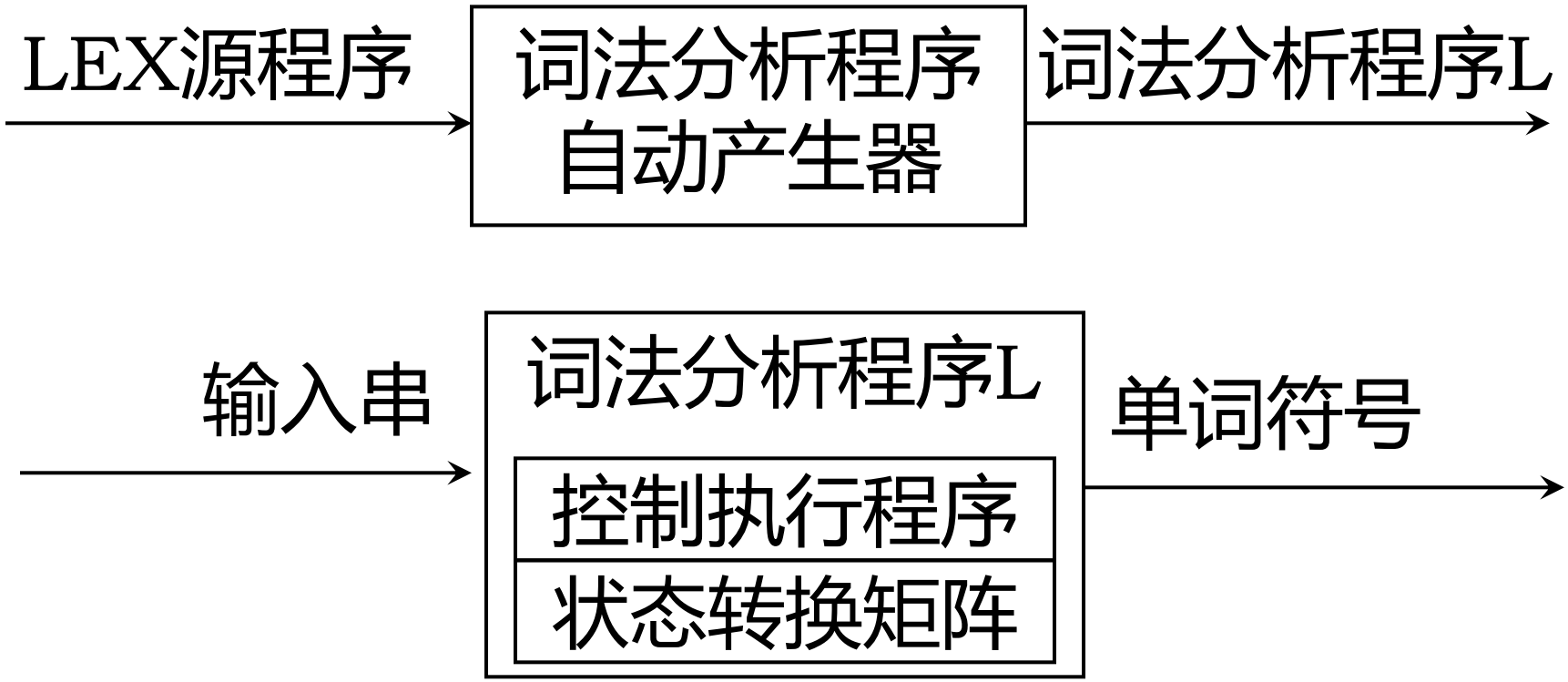
{ S }{B}





LEX

- LEX程序由一组正规式以及相应的动作组成。



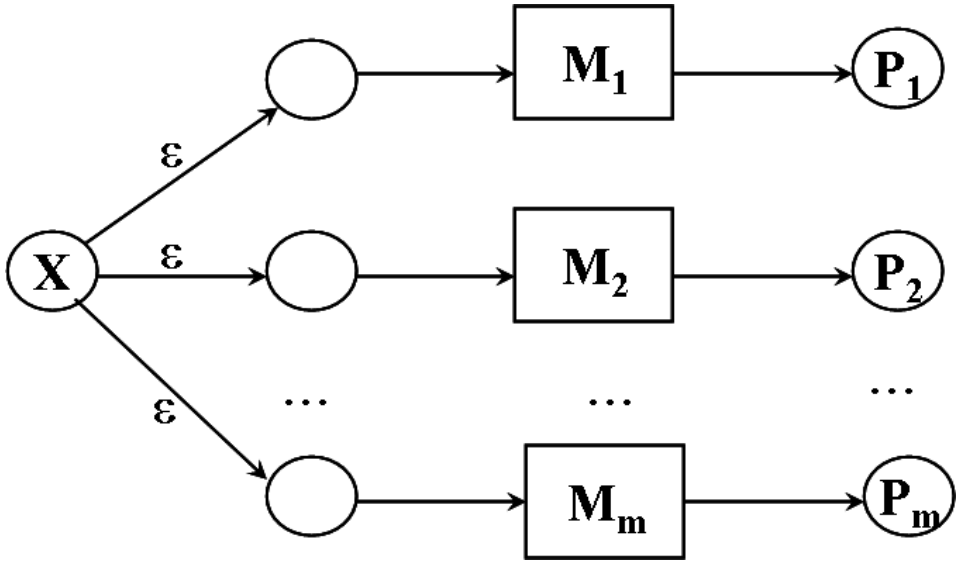
Lex

- **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,-) }
```

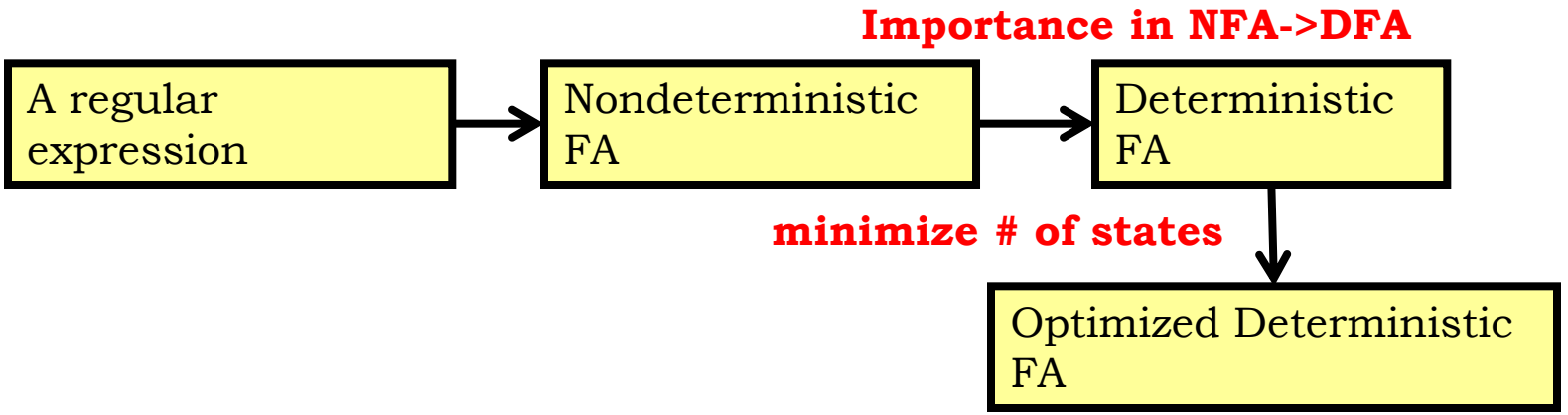
LEX

- **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

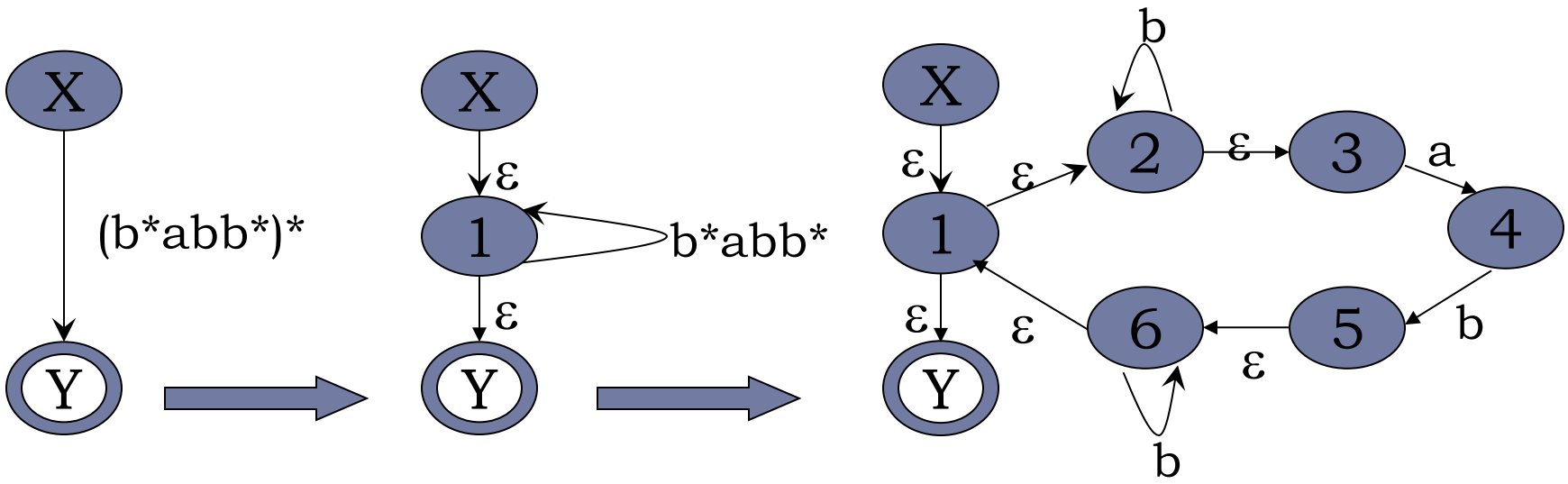
- 例20：构造一个DFA，它接收∑={a, b}上所有满足下述条件的字符串：字符串中的每个a都有至少一个b直接跟在其右边。

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。
- 解：
  - 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



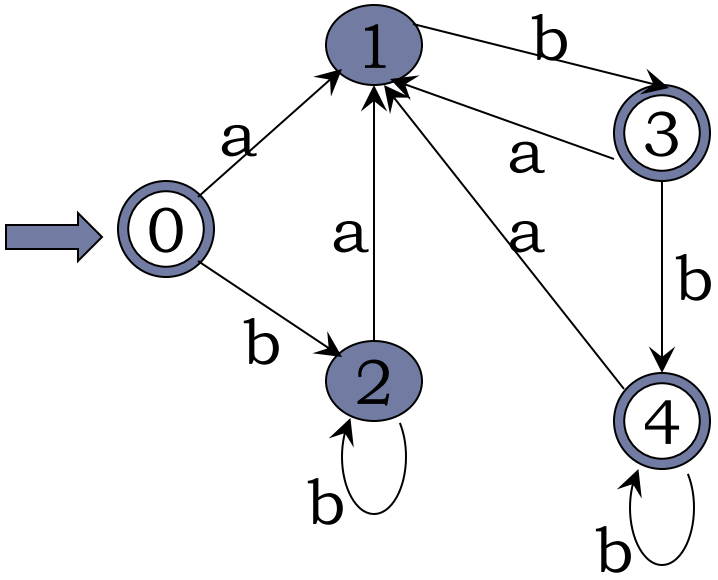
I	I_a	I_b
{X,1,2,3,Y}	{4}	{2,3}
{4}	—	{5,6,1,2,3,Y}
{2,3}	{4}	{2,3}
{5,6,1,2,3,Y}	{4}	{6,1,2,3,Y}
{6,1,2,3,Y}	{4}	{6,1,2,3,Y}

重新命名

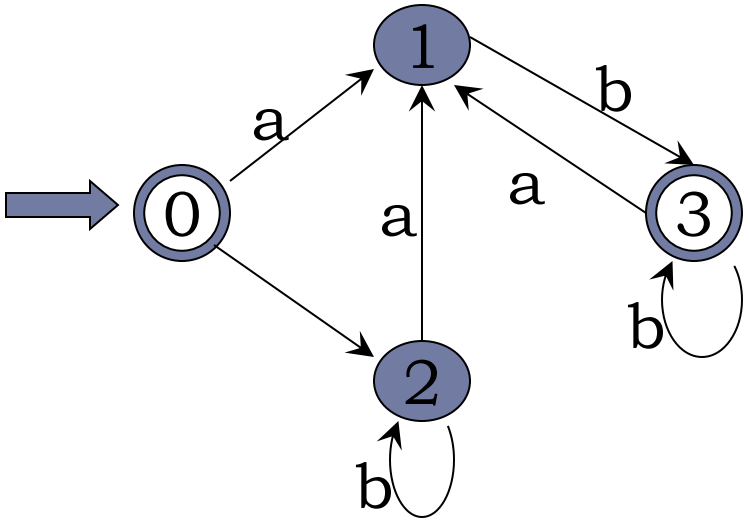
I	I_a	I_b
0	1	2
1	—	3
2	1	2
3	1	4
4	1	4

Exercise

- 由DFA得状态图

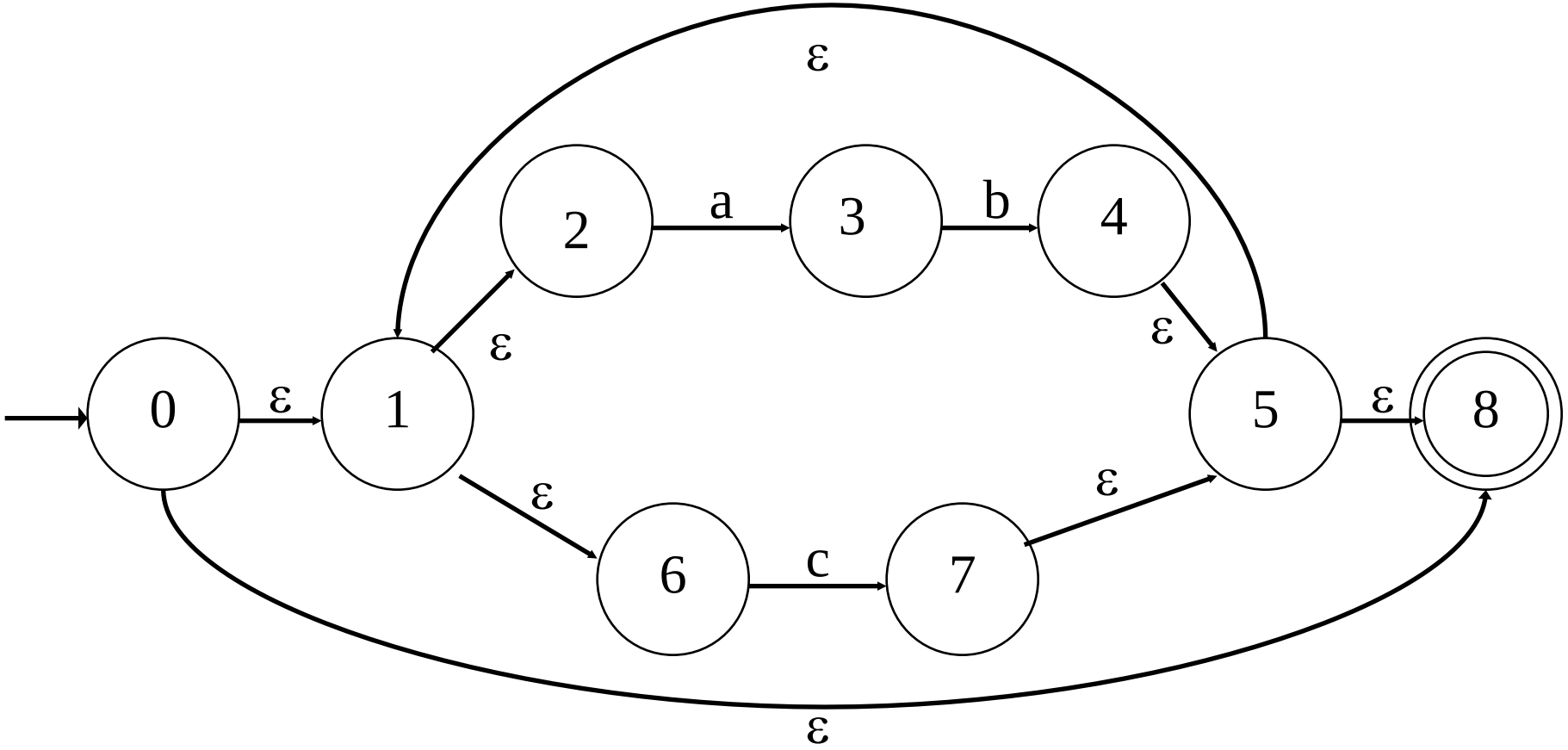


- 用最小化方法化简得：{0}，{1}，{2}，{3,4}，按顺序重新命名DFA M’

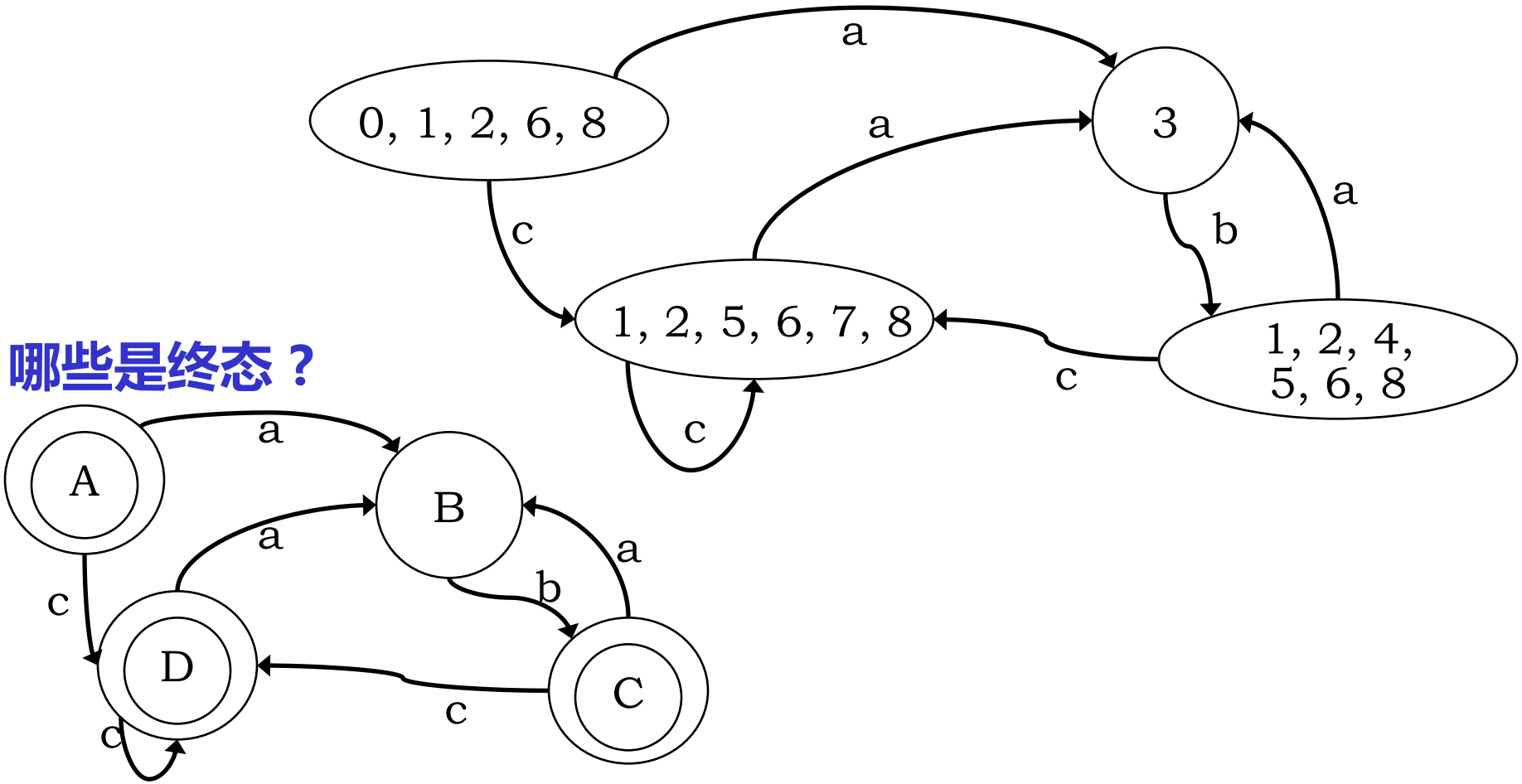


Exercise

- 例21：NFA->DFA



Exercise



哪些是终态？

练习

构造正规表达式

((0|1)*|(11)*)*

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)