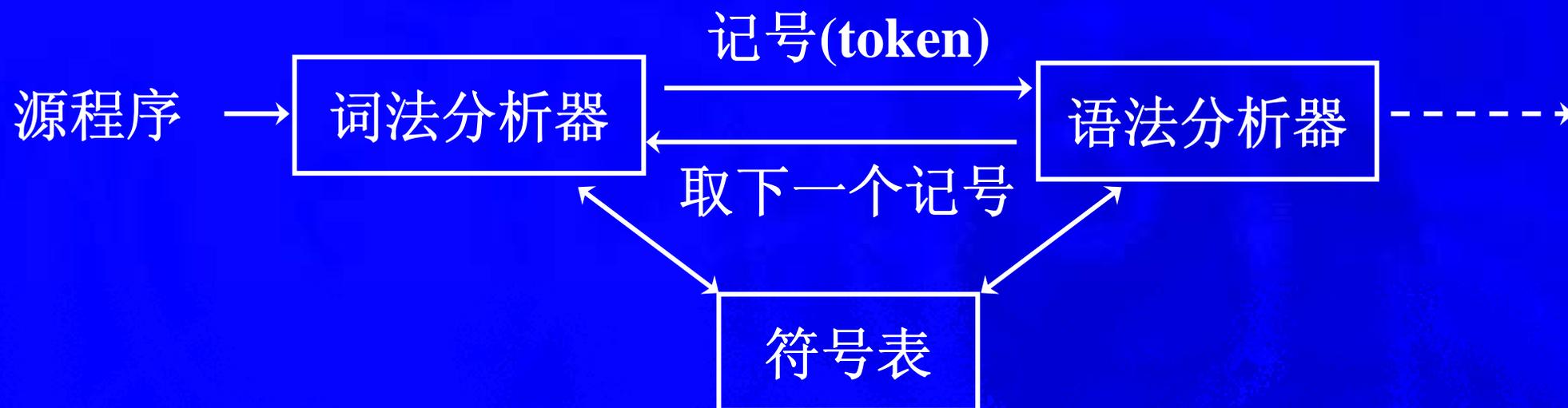


第2章 词法分析



本章内容

- 词法分析器：把构成源程序的字符流翻译成记号流，还完成和用户接口的一些任务
- 围绕词法分析器的自动生成展开
- 介绍正规式、状态转换图和有限自动机概念

2.1 词法记号及属性

2.1.1 词法记号、模式、词法单元

记号名	词法单元例举	模式的非形式描述
if	if	字符 i, f
for	for	字符 f, o, r
relation	<, <=, ==, ...	< 或 <= 或 == 或 ...
id	sum, count, D5	由字母开头的字母数字串
number	3.1, 10, 2.8 E12	任何数值常数
literal	“seg. error”	引号“和”之间的任意字符串，但引号本身除外

2.1 词法记号及属性

- 历史上词法定义中的一些问题
 - 忽略空格带来的困难（如 Fortran语言）
`DO 8 I = 3.75` `DO8I = 3.75`
`DO 8 I = 3,75`
 - 关键字不保留（如 Fortran语言）
`IF THEN THEN THEN=ELSE; ELSE ...`
- 关键字、保留字和标准标识符的区别
 - 保留字是语言预先确定了含义的词法单元
 - 标准标识符也是预先确定了含义的标识符，但程序可以重新声明它的含义

2.1 词法记号及属性

2.1.2 词法记号的属性

$$\text{position} = \text{initial} + \text{rate} * 60$$

的记号和属性值:

⟨**id**, 指向符号表中**position**条目的指针⟩

⟨**assign _ op**⟩

⟨**id**, 指向符号表中**initial**条目的指针⟩

⟨**add_op**⟩

⟨**id**, 指向符号表中**rate**条目的指针⟩

⟨**mul_op**⟩

⟨**number**, 整数值**60**⟩

2.1 词法记号及属性

2.1.3 词法错误

- 词法分析器对源程序采取非常局部的观点
- 例：难以发现下面的错误

fi (a == f (x)) ...

- 在实数是a.b格式下，可以发现下面的错误

123.x

- 紧急方式的错误恢复
删掉当前若干个字符，直至能读出正确的记号
- 错误修补
进行增、删、替换和交换字符的尝试

2.2 词法记号的描述与识别

2.2.1 串和语言

- 字母表：符号的有限集合，例： $\Sigma = \{0, 1\}$
- 串：符号的有穷序列，例： $0110, \varepsilon$
- 语言：字母表上的一个串集
 $\{\varepsilon, 0, 00, 000, \dots\}, \{\varepsilon\}, \emptyset$
- 句子：属于语言的串
- 串的运算
 - 连接（积） $xy, s\varepsilon = \varepsilon s = s$
 - 幂 s^0 为 ε, s^i 为 $s^{i-1}s$ ($i > 0$)

2.2 词法记号的描述与识别

- 语言的运算

- 并: $L \cup M = \{s \mid s \in L \text{ 或 } s \in M\}$

- 连接: $LM = \{st \mid s \in L \text{ 且 } t \in M\}$

- 幂: L^0 是 $\{\varepsilon\}$, L^i 是 $L^{i-1}L$

- 闭包: $L^* = L^0 \cup L^1 \cup L^2 \cup \dots$

- 正闭包: $L^+ = L^1 \cup L^2 \cup \dots$

- 例

$L: \{A, B, \dots, Z, a, b, \dots, z\}, D: \{0, 1, \dots, 9\}$

$L \cup D, LD, L^6, L^*, L(L \cup D)^*, D^+$

2.2 词法记号的描述与识别

2.2.2 正规式

正规式用来表示简单的语言，叫做正规集

正规式	定义的语言	备注
ε	$\{\varepsilon\}$	
a	$\{a\}$	$a \in \Sigma$
$(r) (s)$	$L(r) \cup L(s)$	r 和 s 是正规式
$(r)(s)$	$L(r)L(s)$	r 和 s 是正规式
$(r)^*$	$(L(r))^*$	r 是正规式
(r)	$L(r)$	r 是正规式

$((a) (b)^*) | (c)$ 可以写成 $ab^* | c$

2.2 词法记号的描述与识别

- 正规式的例子 $\Sigma = \{a, b\}$
 - $a \mid b$ $\{a, b\}$
 - $(a \mid b)(a \mid b)$ $\{aa, ab, ba, bb\}$
 - $aa \mid ab \mid ba \mid bb$ $\{aa, ab, ba, bb\}$
 - a^* 由字母 a 构成的所有串集, 含 ε
 - $(a \mid b)^*$ 由 a 和 b 构成的所有串集, 含 ε
- 复杂的例子
($00 \mid 11 \mid ((01 \mid 10)(00 \mid 11)^*(01 \mid 10))$)^{*}
句子: **01001101000010000010111001**

2.2 词法记号的描述与识别

2.2.3 正规定义

- 对正规式命名，使表示简洁

$$d_1 \rightarrow r_1$$

$$d_2 \rightarrow r_2$$

...

$$d_n \rightarrow r_n$$

- 各个 d_i 的名字都不同
- 每个 r_i 都是 $\Sigma \cup \{d_1, d_2, \dots, d_{i-1}\}$ 上的正规式

2.2 词法记号的描述与识别

- 正规定义的例子
 - C语言的标识符是字母、数字和下划线组成的串

letter_ $\rightarrow A | B | \dots | Z | a | b | \dots | z | _$

digit $\rightarrow 0 | 1 | \dots | 9$

id $\rightarrow \text{letter_}(\text{letter_} | \text{digit})^*$

2.2 词法记号的描述与识别

- 正规定义的例子

- 无符号数集合，例1946, 11.28, 63E8, 1.99E-6

digit $\rightarrow 0 \mid 1 \mid \dots \mid 9$

digits $\rightarrow \text{digit digit}^*$

optional_fraction $\rightarrow . \text{digits} \mid \varepsilon$

optional_exponent $\rightarrow (\text{E} (+ \mid - \mid \varepsilon) \text{digits}) \mid \varepsilon$

number $\rightarrow \text{digits optional_fraction optional_exponent}$

- 简化表示

number $\rightarrow \text{digit}^+ (. \text{digit}^+)? (\text{E}(+|-)? \text{digit}^+)?$

2.2 词法记号的描述与识别

- 正规定义的例子（进行下一步讨论的例子）

while \rightarrow while

do \rightarrow do

relop \rightarrow < | <= | == | <> | > | >=

letter \rightarrow *A* | *B* | ... | *Z* | *a* | *b* / ... | *z*

id \rightarrow letter (letter | digit)*

number \rightarrow digit⁺ (.digit⁺)? (E (+ | -)? digit⁺)?

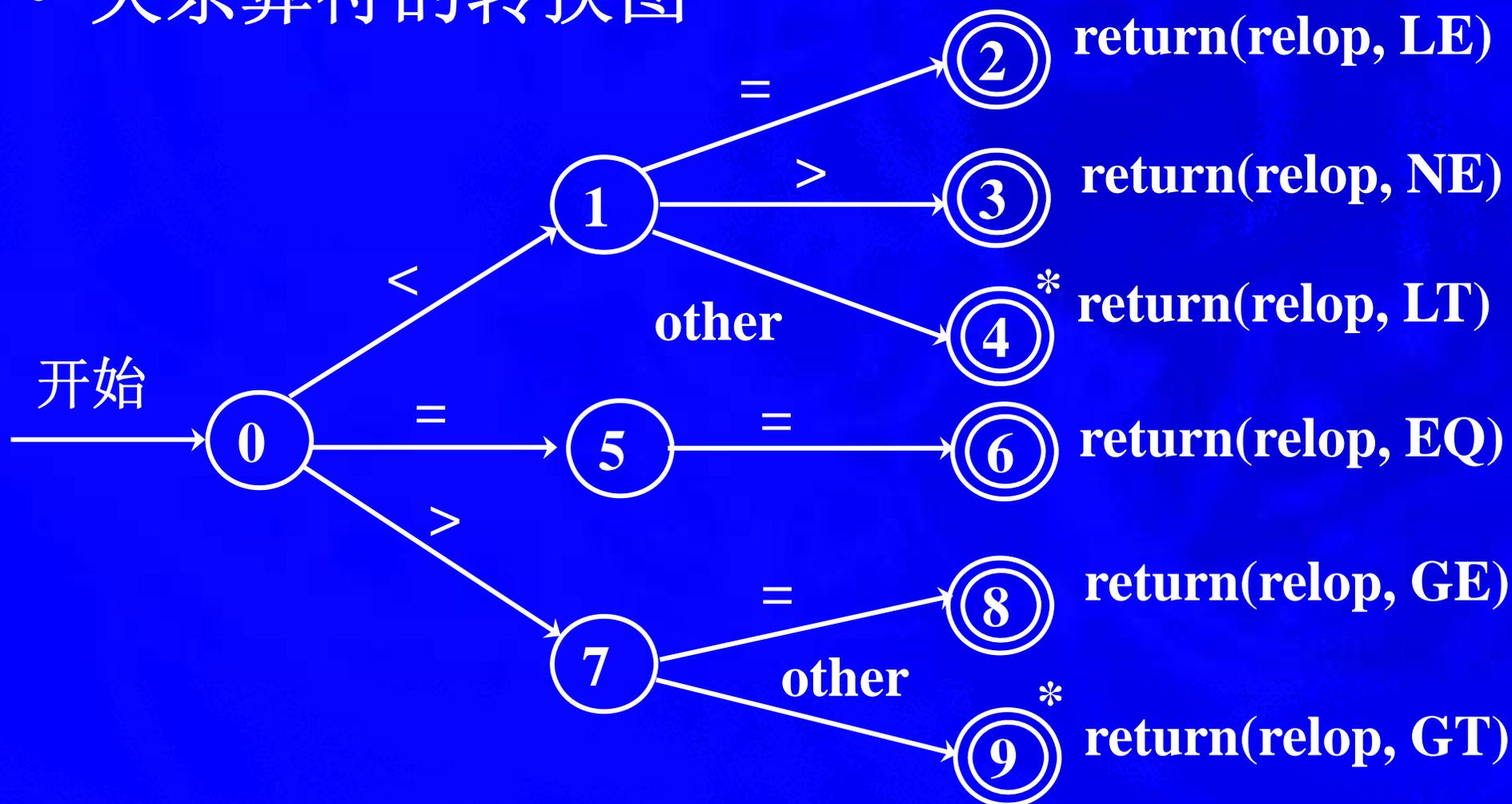
delim \rightarrow blank | tab | newline

ws \rightarrow delim⁺

2.2 词法记号的描述与识别

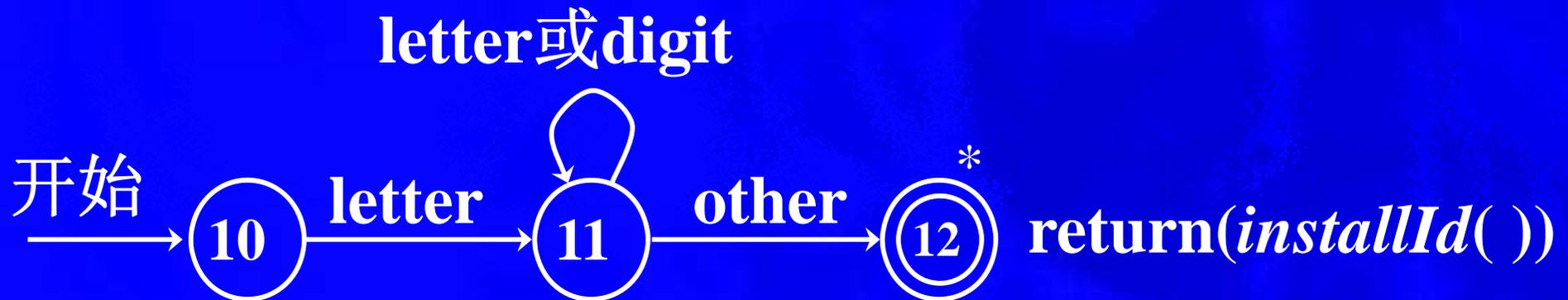
2.2.4 转换图

- 关系算符的转换图



2.2 词法记号的描述与识别

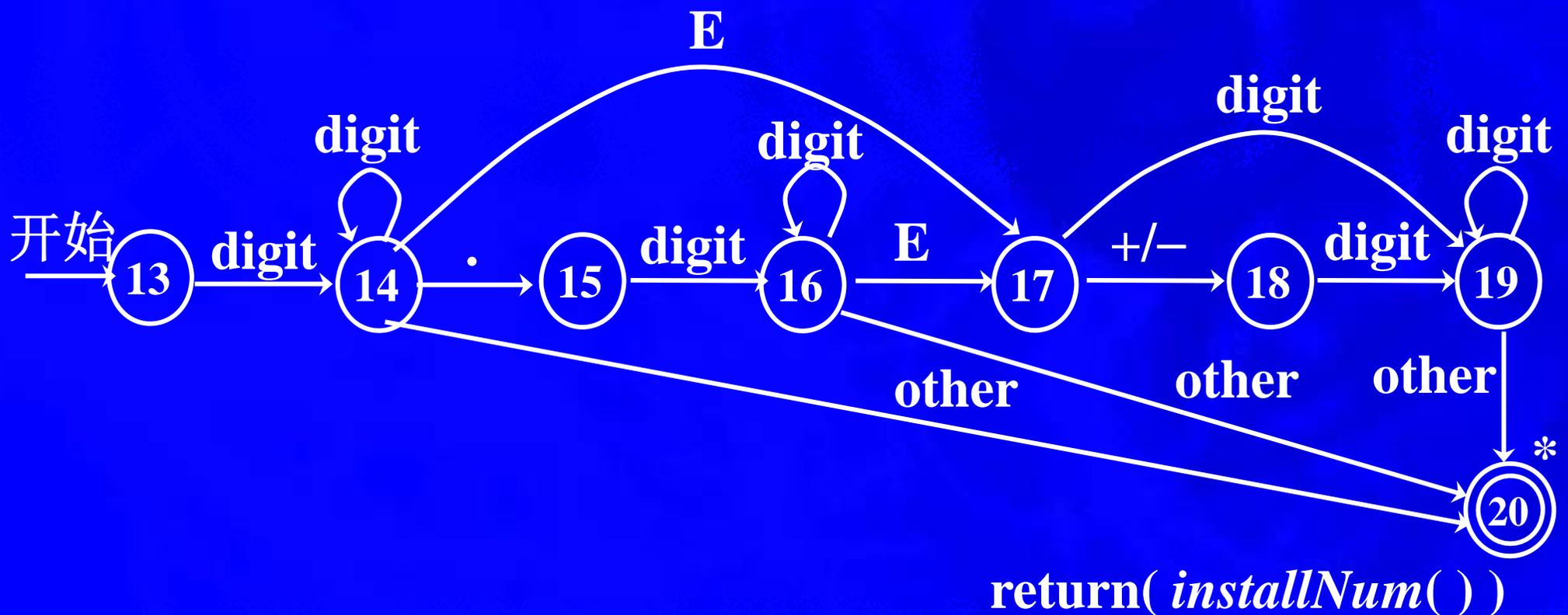
- 标识符和保留字的转换图



2.2 词法记号的描述与识别

- 无符号数的转换图

$\text{number} \rightarrow \text{digit}^+ (. \text{digit}^+)? (\text{E} (+ | -)? \text{digit}^+)?$

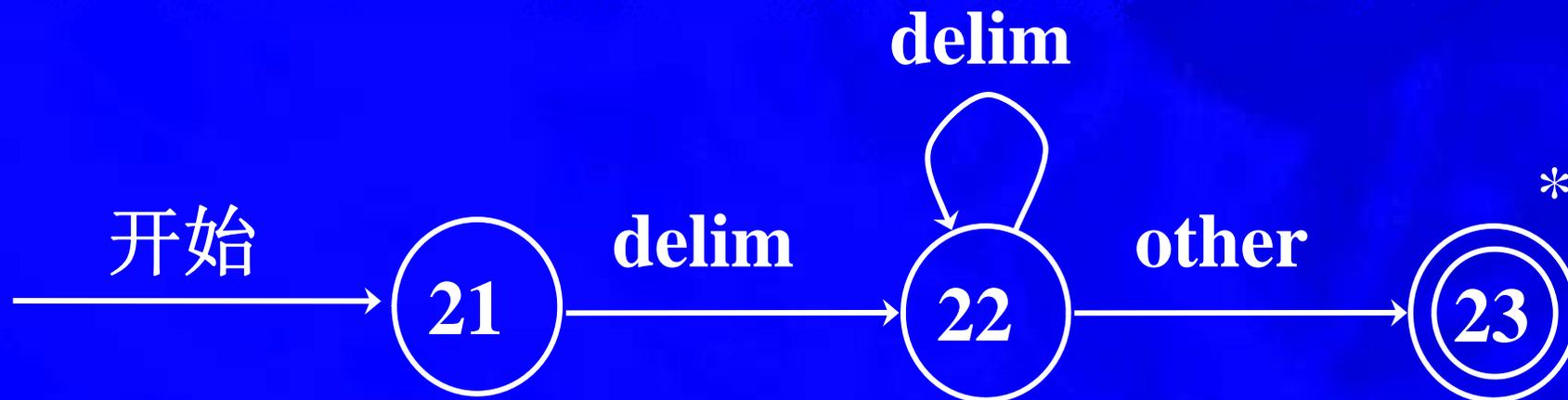


2.2 词法记号的描述与识别

- 空白的转换图

delim → blank | tab | newline

ws → **delim**⁺



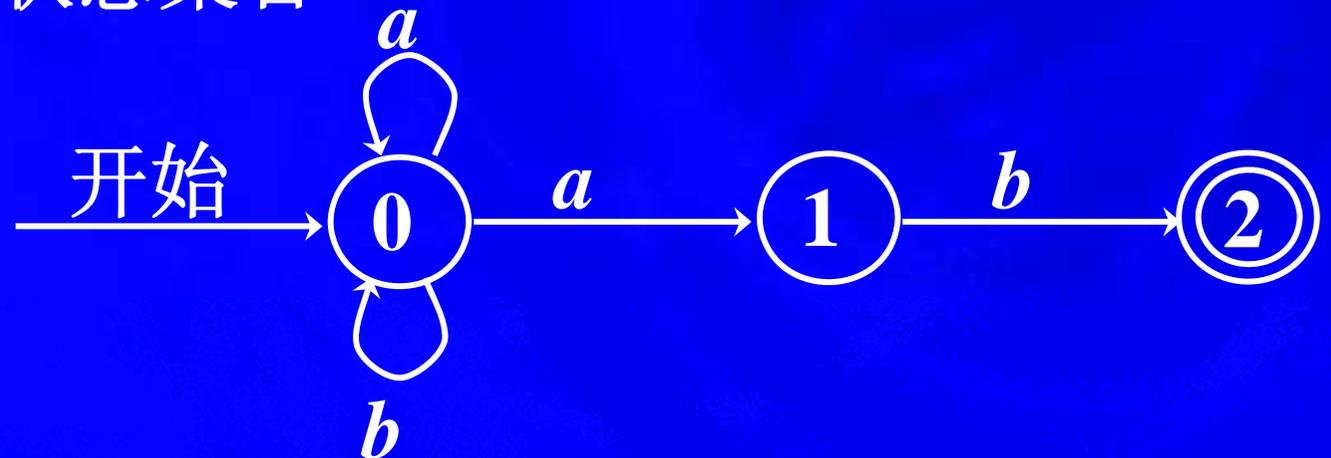
2.3 有限自动机

2.3.1 不确定的有限自动机（简称NFA）

一个数学模型，它包括：

- 1、有限的状态集合 S
- 2、输入符号集合 Σ
- 3、转换函数 $move : S \times (\Sigma \cup \{\varepsilon\}) \rightarrow P(S)$
- 4、状态 s_0 是唯一的开始状态
- 5、 $F \subseteq S$ 是接受状态集合

识别语言
 $(a|b)^*ab$
的NFA

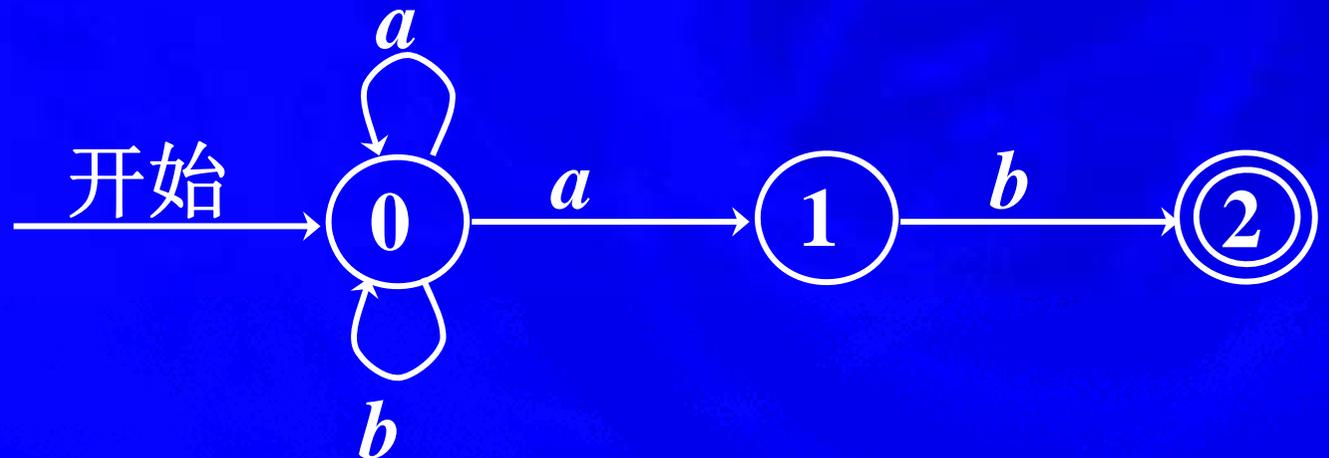


2.3 有限自动机

- NFA的转换表

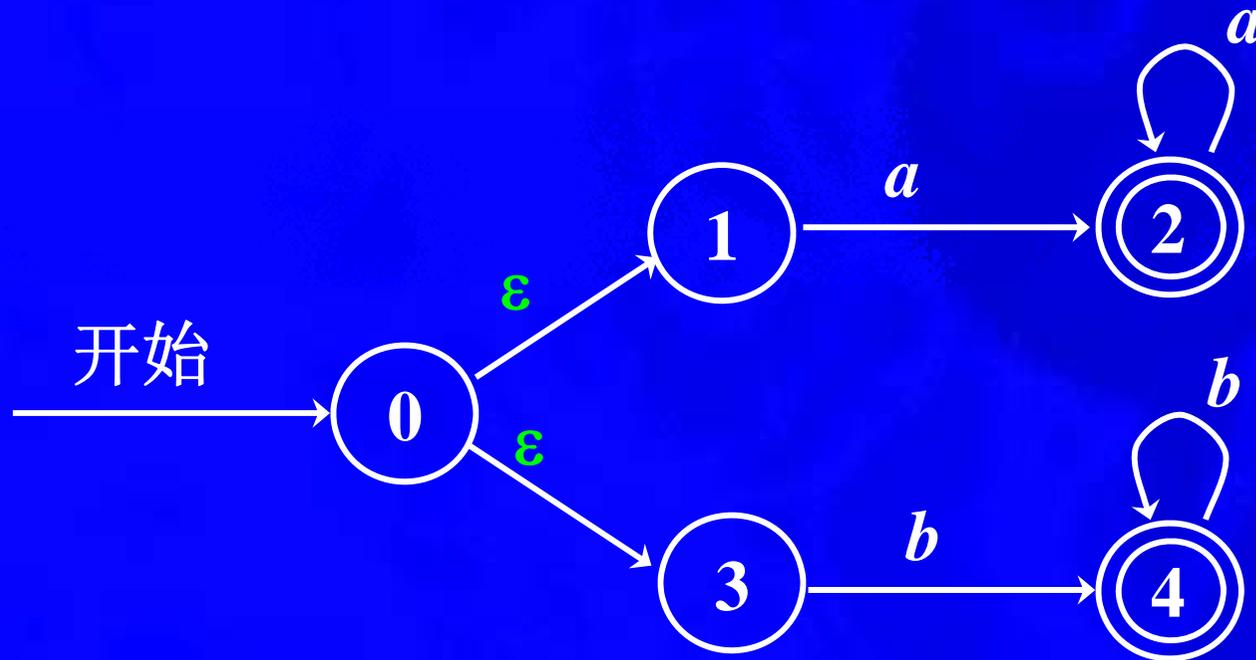
状态	输入符号	
	<i>a</i>	<i>b</i>
0	{0, 1}	{0}
1	∅	{2}
2	∅	∅

识别语言
 $(a|b)^*ab$
的NFA



2.3 有限自动机

- 例 识别 $aa^* | bb^*$ 的NFA



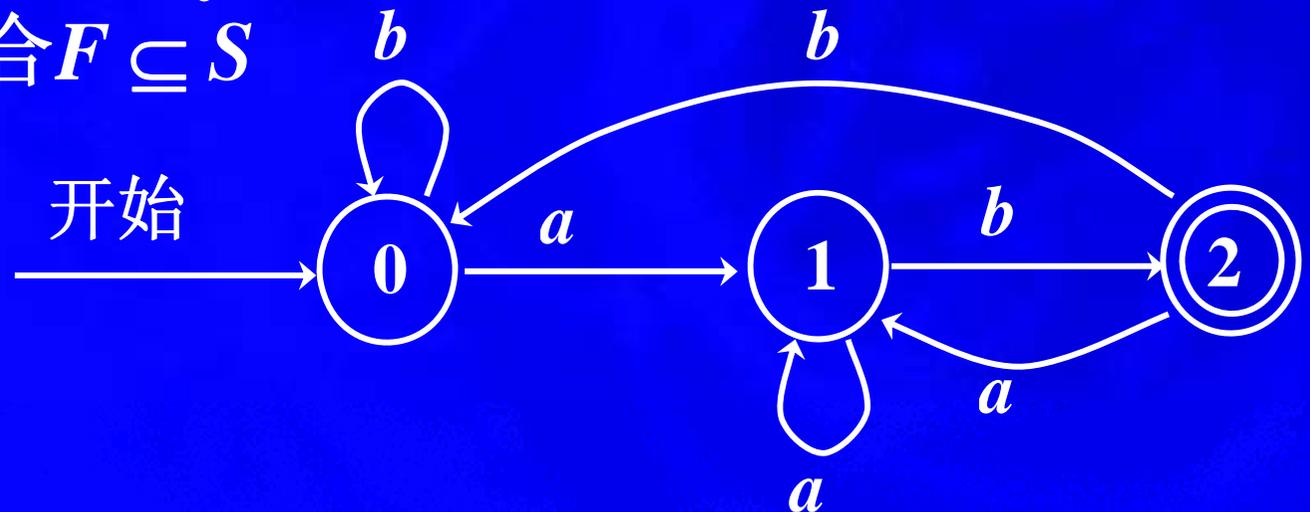
2.3 有限自动机

2.3.2 确定的有限自动机（简称DFA）

一个数学模型，包括：

- 1、有限的状态集合 S
- 2、输入符号集合 Σ
- 3、转换函数 $move : S \times \Sigma \rightarrow S$ ，且可以是部分函数
- 4、唯一的开始状态 s_0
- 5、接受状态集合 $F \subseteq S$

识别语言
 $(a|b)^*ab$
的DFA



2.3 有限自动机

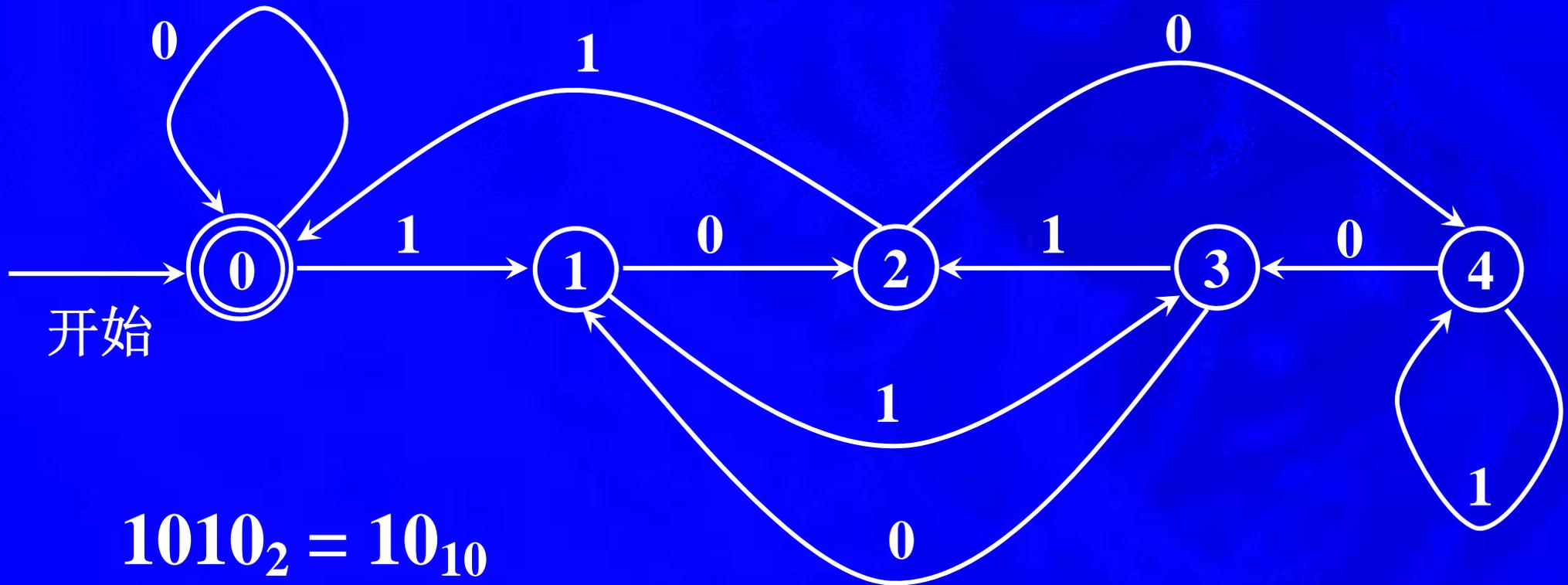
- 例 DFA, 识别{0,1}上能被5整除的二进制数

	已读过	尚未读	已读部分的值
某时刻	101	0111000	5
读进0	1010	111000	$5 \times 2 = 10$
读进1	10101	11000	$10 \times 2 + 1 = 21$

5个状态即可, 分别代表已读部分的值除以5的余数

2.3 有限自动机

- 例 DFA, 识别 $\{0,1\}$ 上能被5整除的二进制数



$$1010_2 = 10_{10}$$

$$111_2 = 7_{10}$$

2.3 有限自动机

2.3.3 NFA到DFA的变换

子集构造法

1、DFA的一个状态是NFA的一个状态集合

2、读了输入 $a_1 a_2 \dots a_n$ 后，

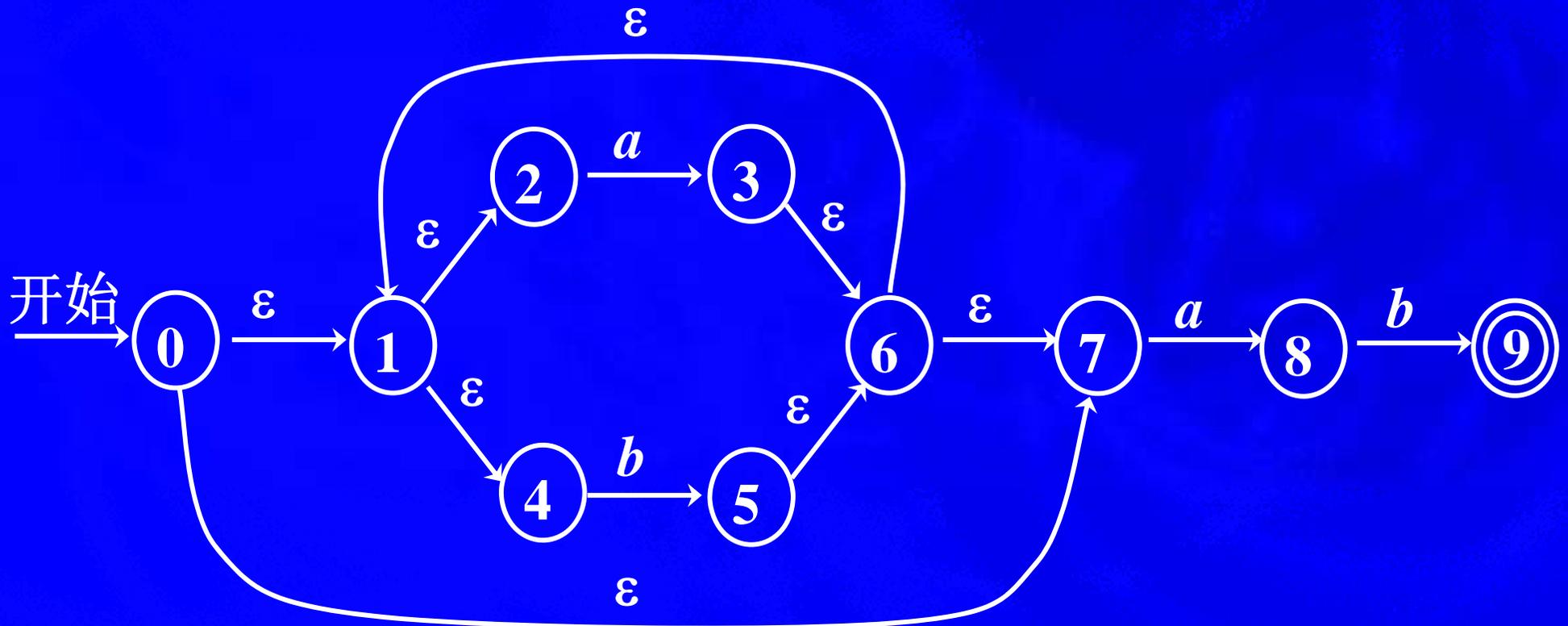
NFA能到达的所有状态： s_1, s_2, \dots, s_k ，则

DFA到达状态 $\{s_1, s_2, \dots, s_k\}$

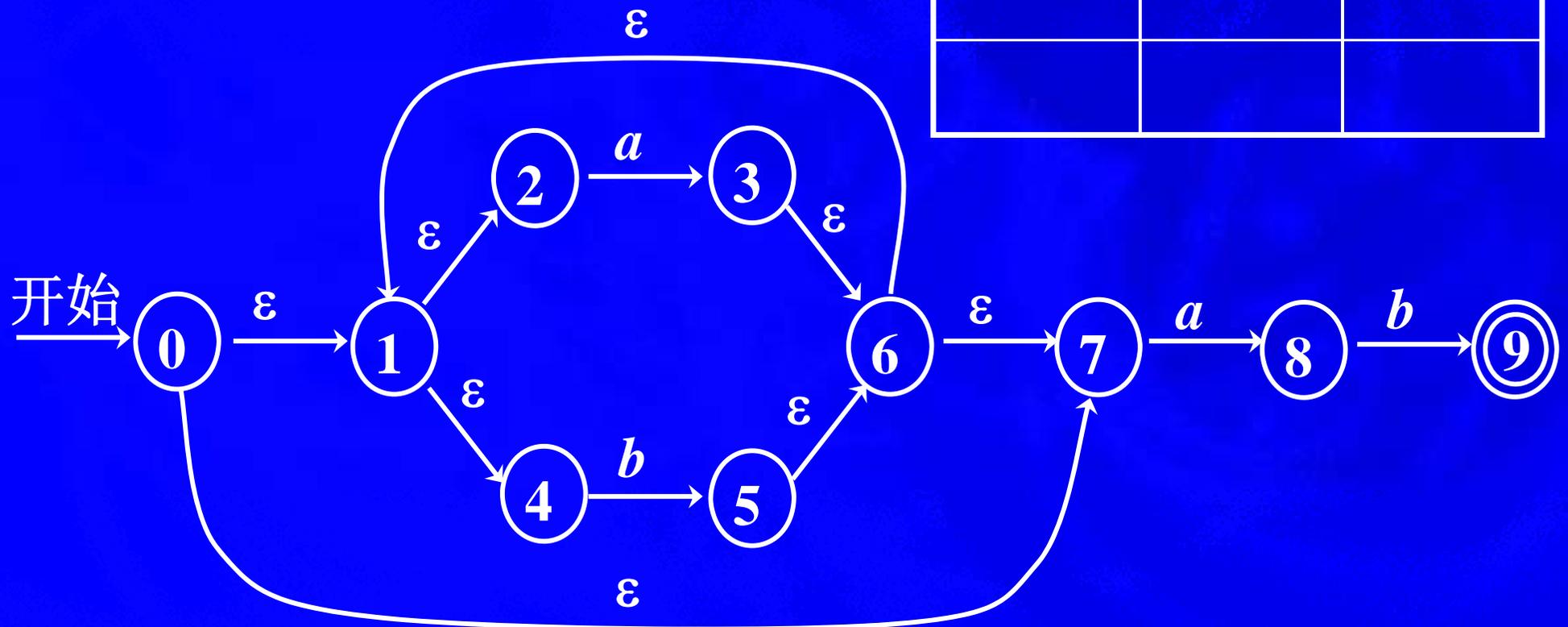


2.3 有限自动机

- 例 $(a|b)^*ab$, NFA如下, 把它变换为DFA



2.3 有限自动机

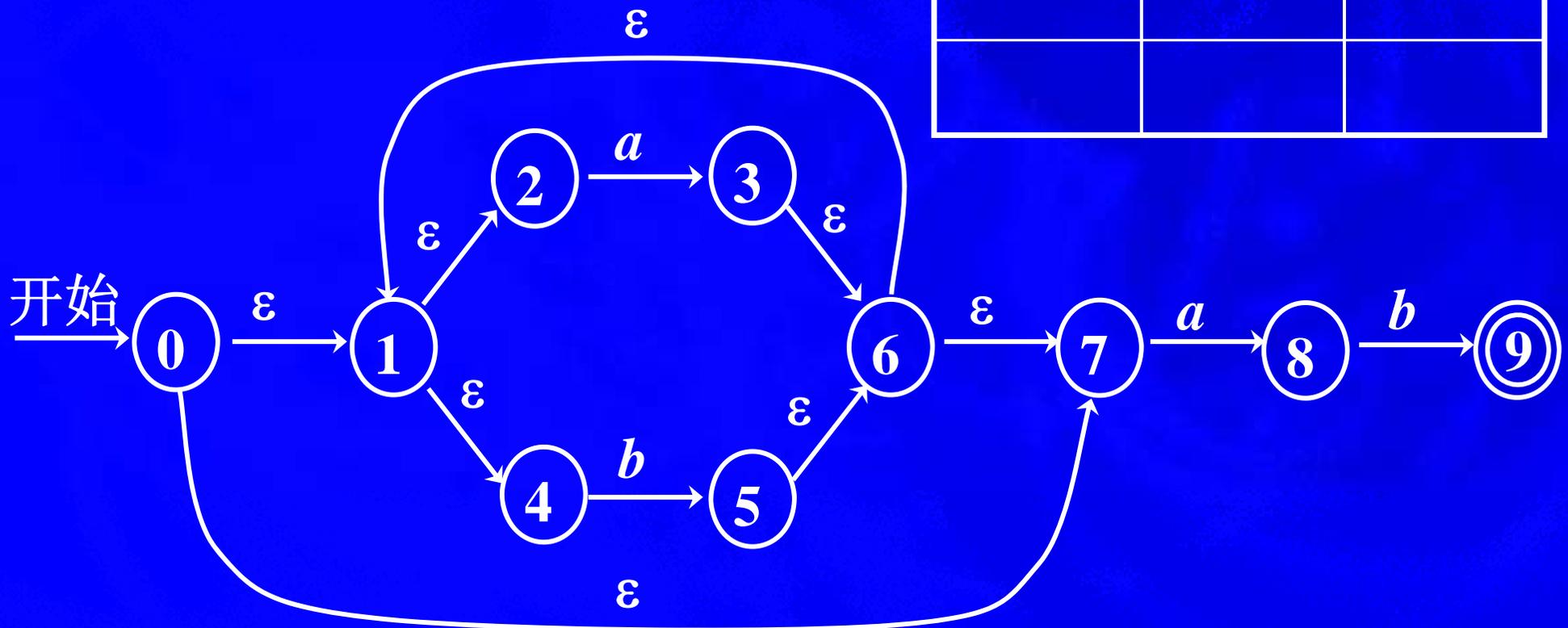


状态	输入符号	
	<i>a</i>	<i>b</i>

2.3 有限自动机

$A = \{0, 1, 2, 4, 7\}$

状态	输入符号	
	a	b
A		

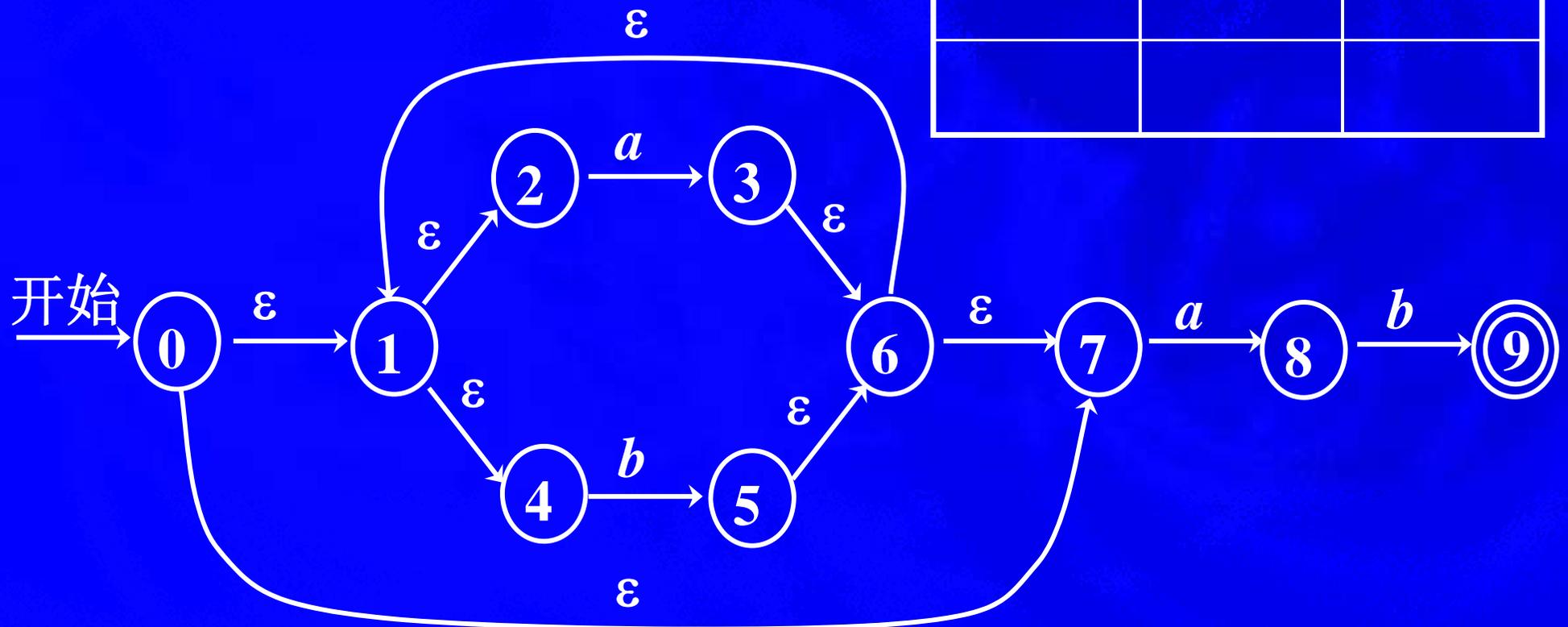


2.3 有限自动机

$A = \{0, 1, 2, 4, 7\}$

$B = \{1, 2, 3, 4, 6, 7, 8\}$

状态	输入符号	
	a	b
A	B	

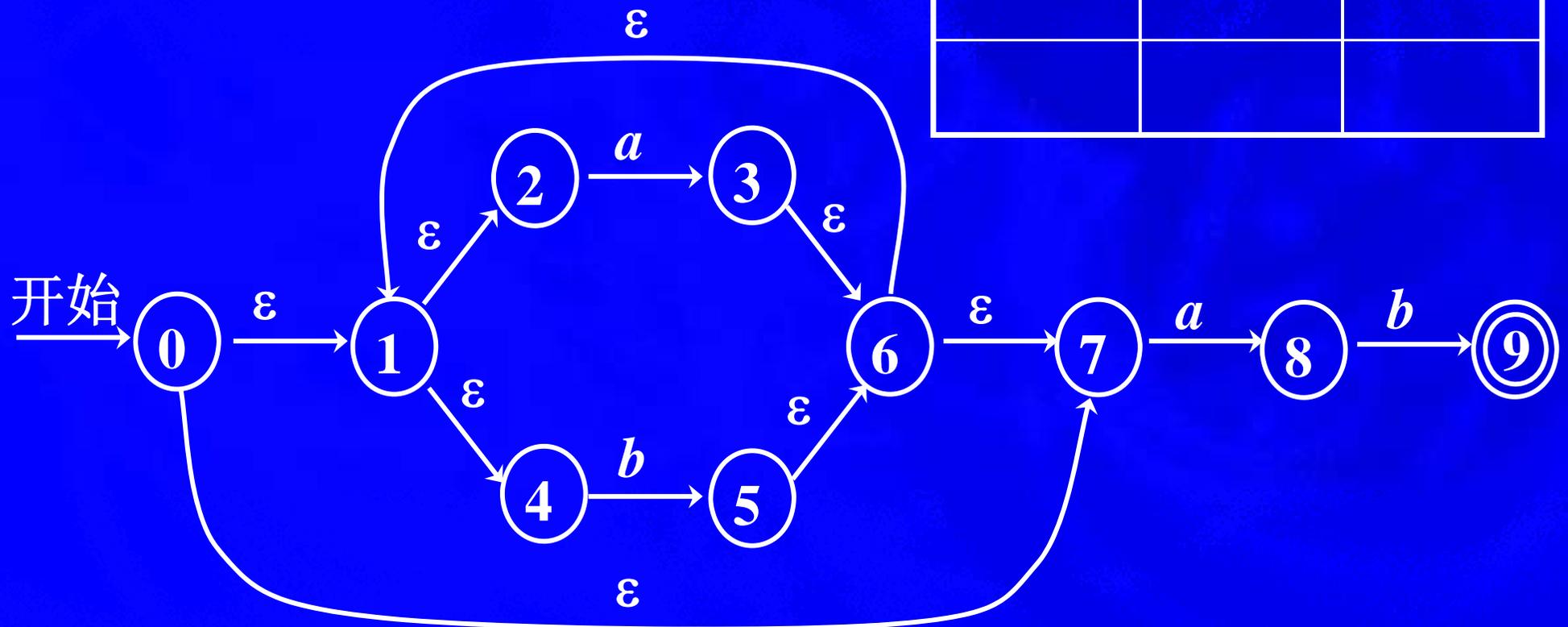


2.3 有限自动机

$A = \{0, 1, 2, 4, 7\}$

$B = \{1, 2, 3, 4, 6, 7, 8\}$

状态	输入符号	
	a	b
A	B	
B		



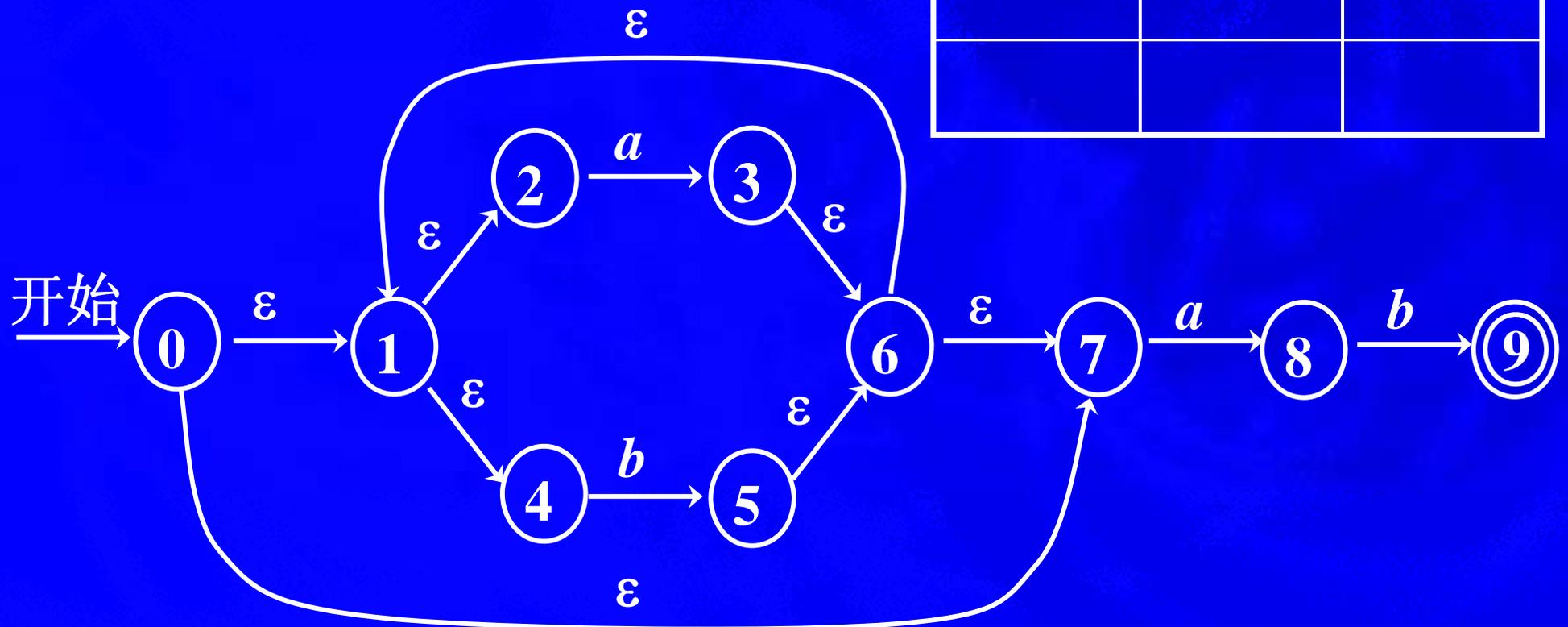
2.3 有限自动机

$A = \{0, 1, 2, 4, 7\}$

$B = \{1, 2, 3, 4, 6, 7, 8\}$

$C = \{1, 2, 4, 5, 6, 7\}$

状态	输入符号	
	a	b
A	B	C
B		



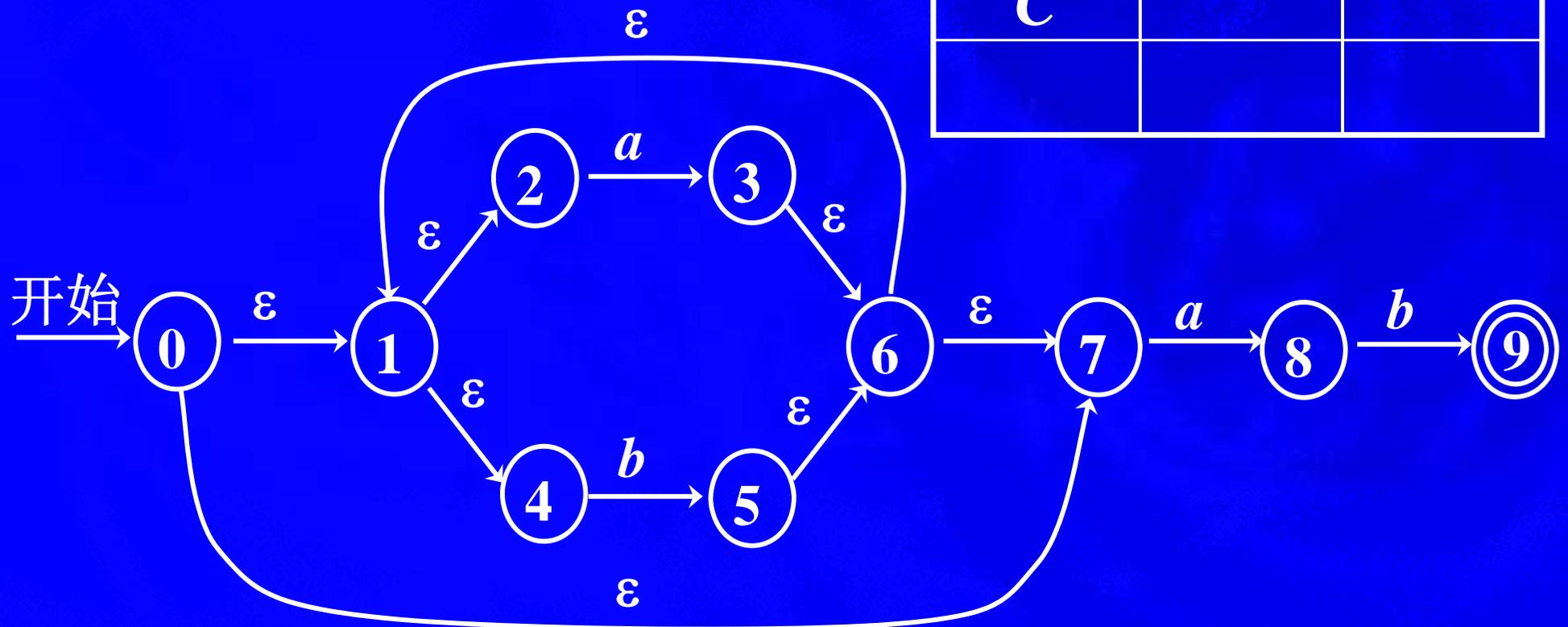
2.3 有限自动机

$A = \{0, 1, 2, 4, 7\}$

$B = \{1, 2, 3, 4, 6, 7, 8\}$

$C = \{1, 2, 4, 5, 6, 7\}$

状态	输入符号	
	a	b
A	B	C
B		
C		



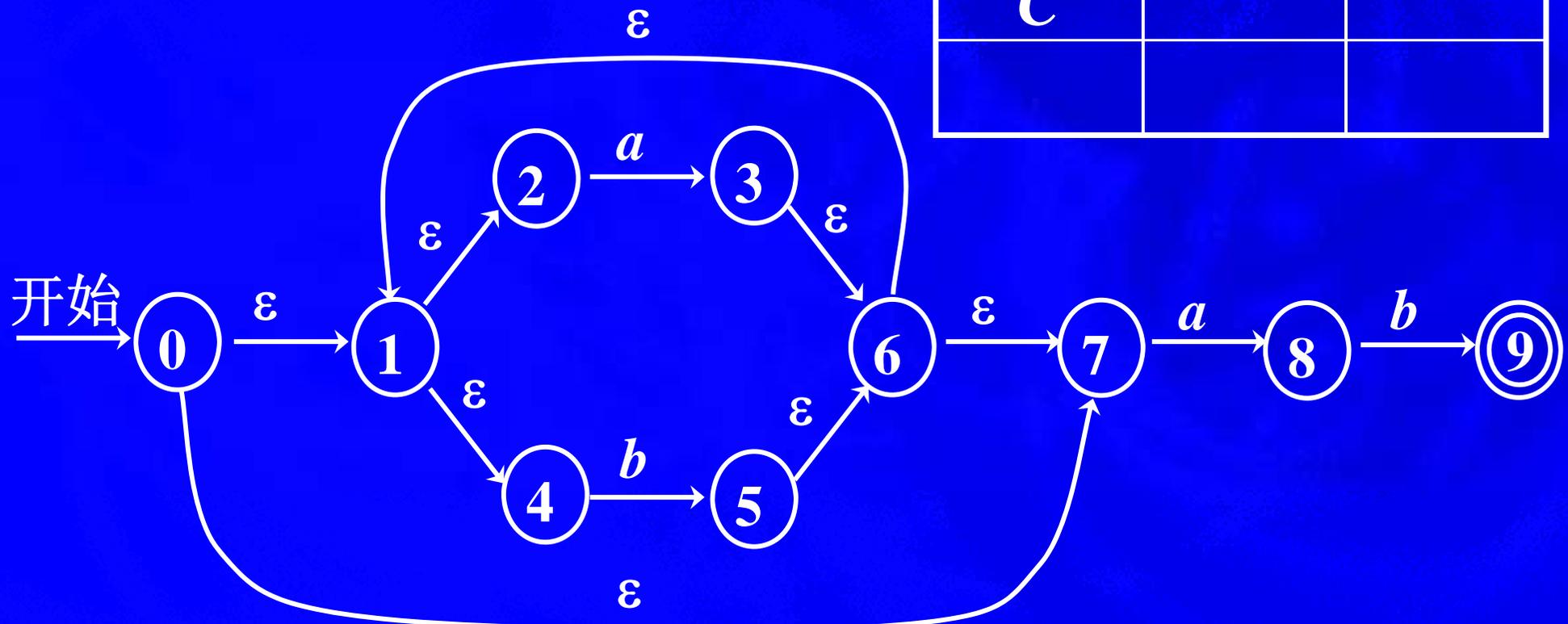
2.3 有限自动机

$A = \{0, 1, 2, 4, 7\}$

$B = \{1, 2, 3, 4, 6, 7, 8\}$

$C = \{1, 2, 4, 5, 6, 7\}$

状态	输入符号	
	a	b
A	B	C
B	B	
C		



2.3 有限自动机

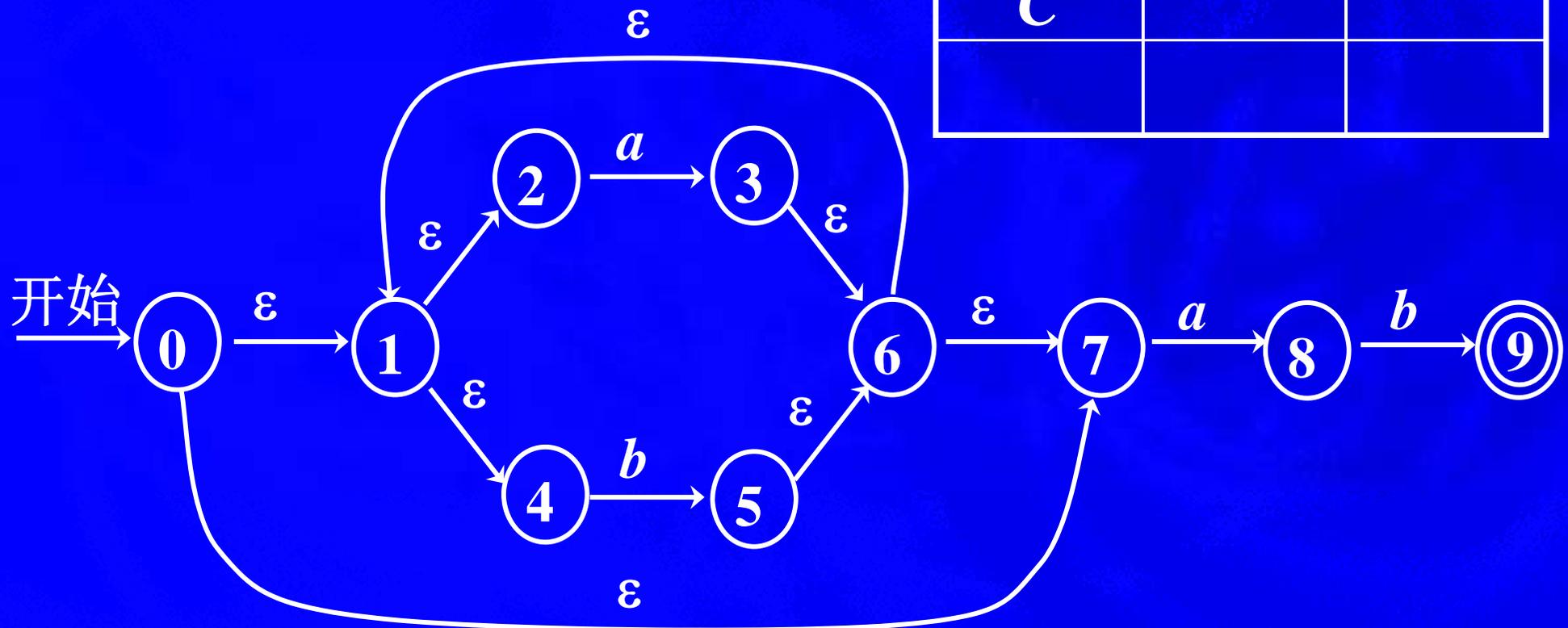
$A = \{0, 1, 2, 4, 7\}$

$B = \{1, 2, 3, 4, 6, 7, 8\}$

$C = \{1, 2, 4, 5, 6, 7\}$

$D = \{1, 2, 4, 5, 6, 7, 9\}$

状态	输入符号	
	a	b
A	B	C
B	B	D
C		



2.3 有限自动机

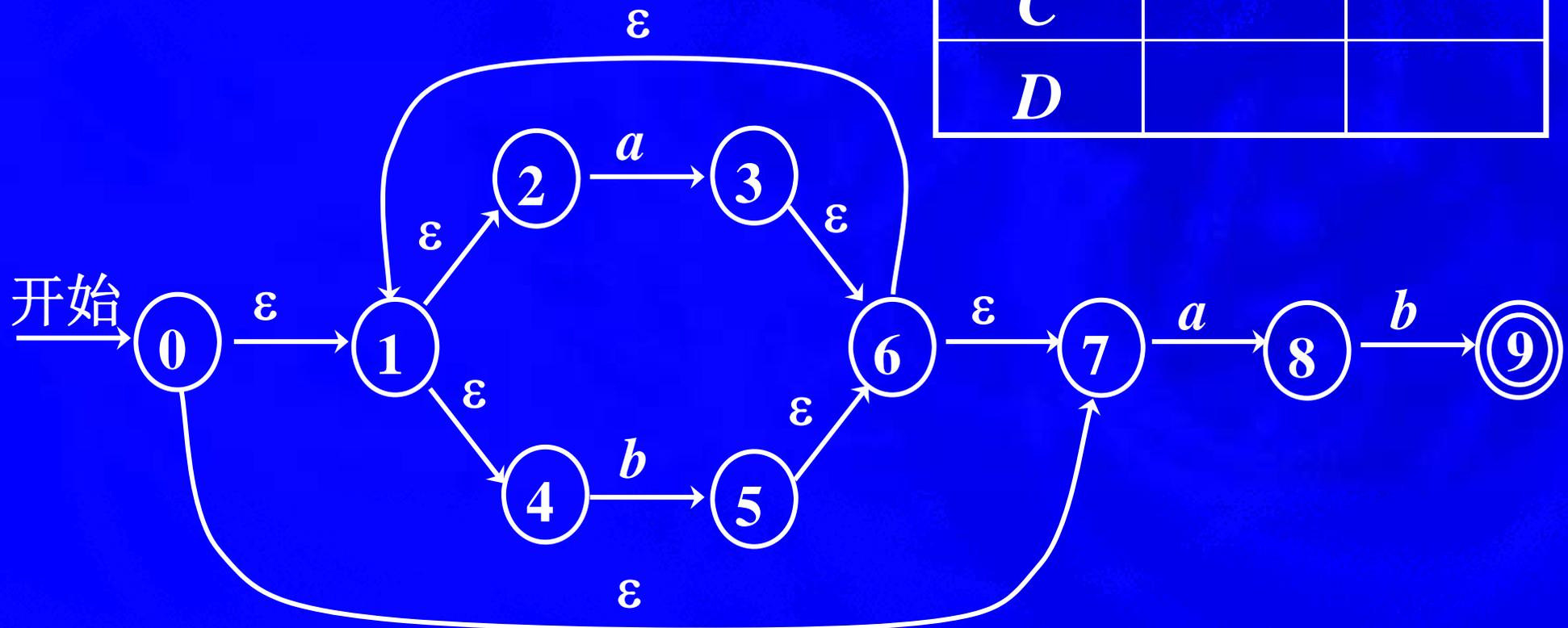
$A = \{0, 1, 2, 4, 7\}$

$B = \{1, 2, 3, 4, 6, 7, 8\}$

$C = \{1, 2, 4, 5, 6, 7\}$

$D = \{1, 2, 4, 5, 6, 7, 9\}$

状态	输入符号	
	a	b
A	B	C
B	B	D
C		
D		



2.3 有限自动机

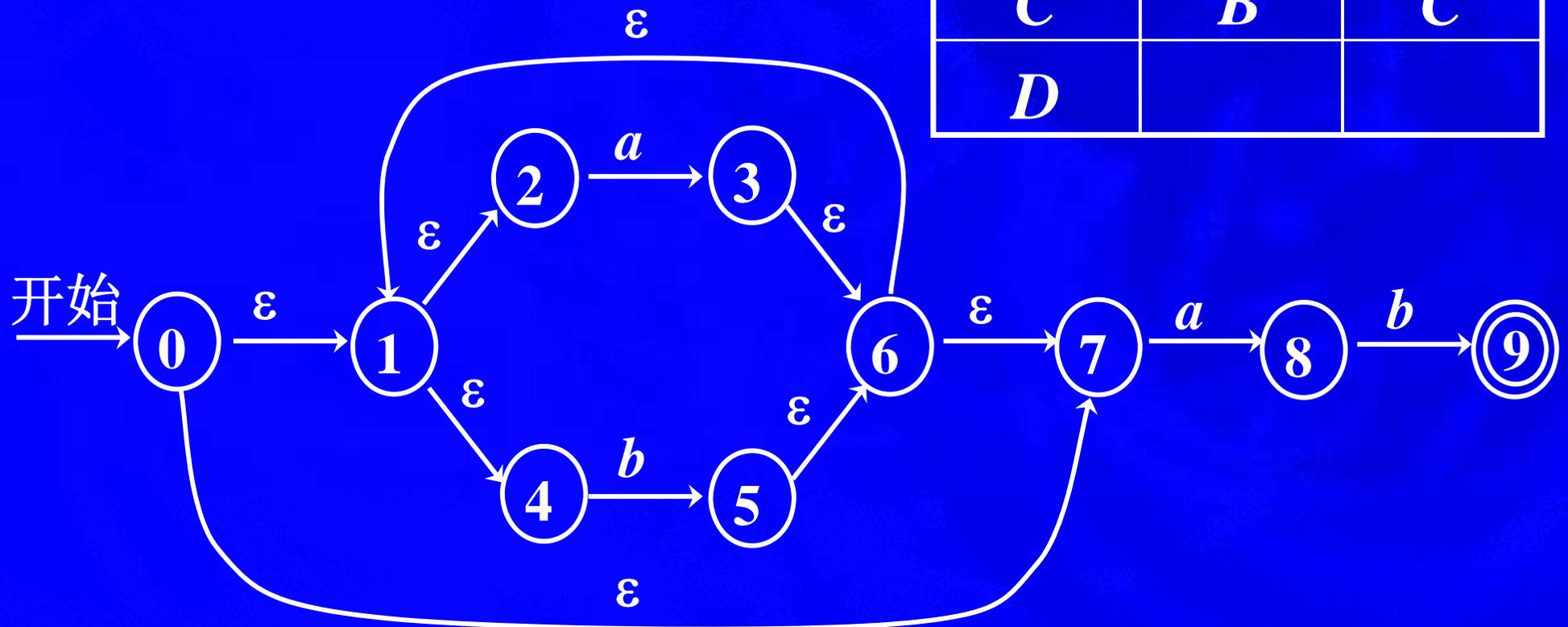
$A = \{0, 1, 2, 4, 7\}$

$B = \{1, 2, 3, 4, 6, 7, 8\}$

$C = \{1, 2, 4, 5, 6, 7\}$

$D = \{1, 2, 4, 5, 6, 7, 9\}$

状态	输入符号	
	a	b
A	B	C
B	B	D
C	B	C
D		



2.3 有限自动机

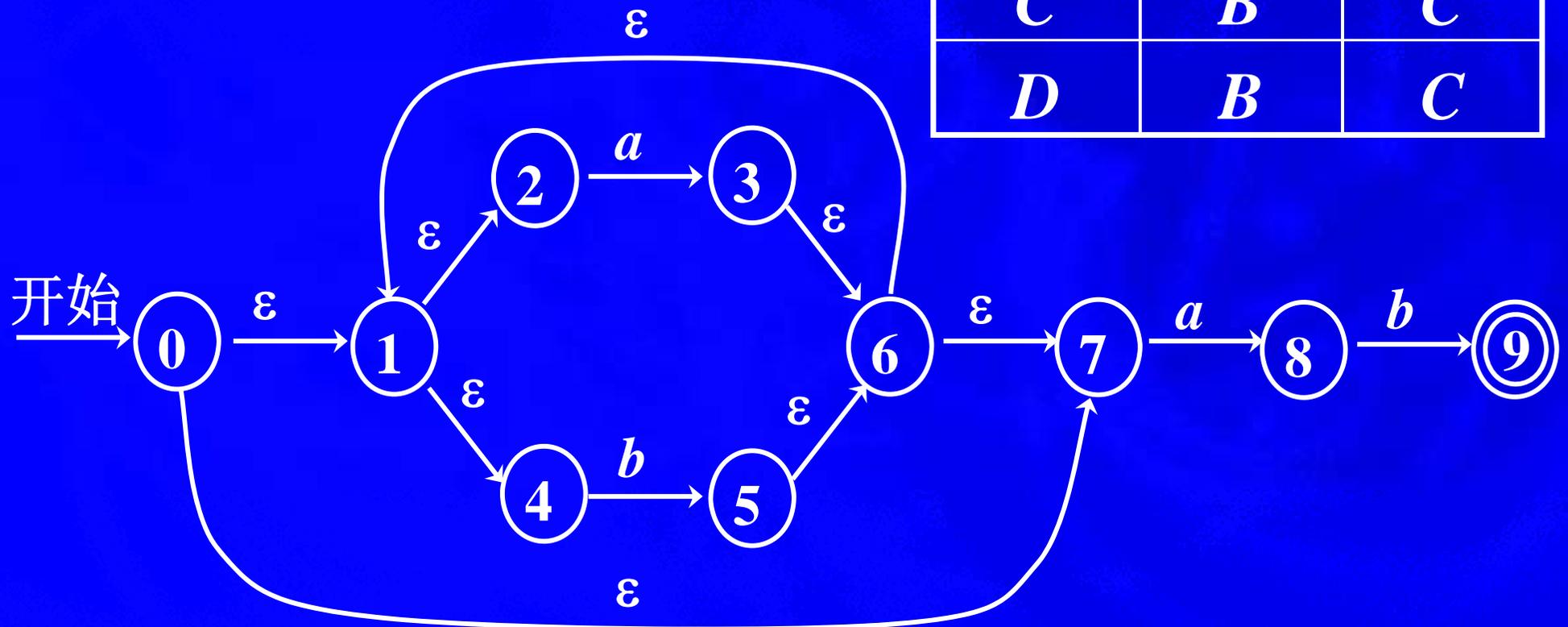
$A = \{0, 1, 2, 4, 7\}$

$B = \{1, 2, 3, 4, 6, 7, 8\}$

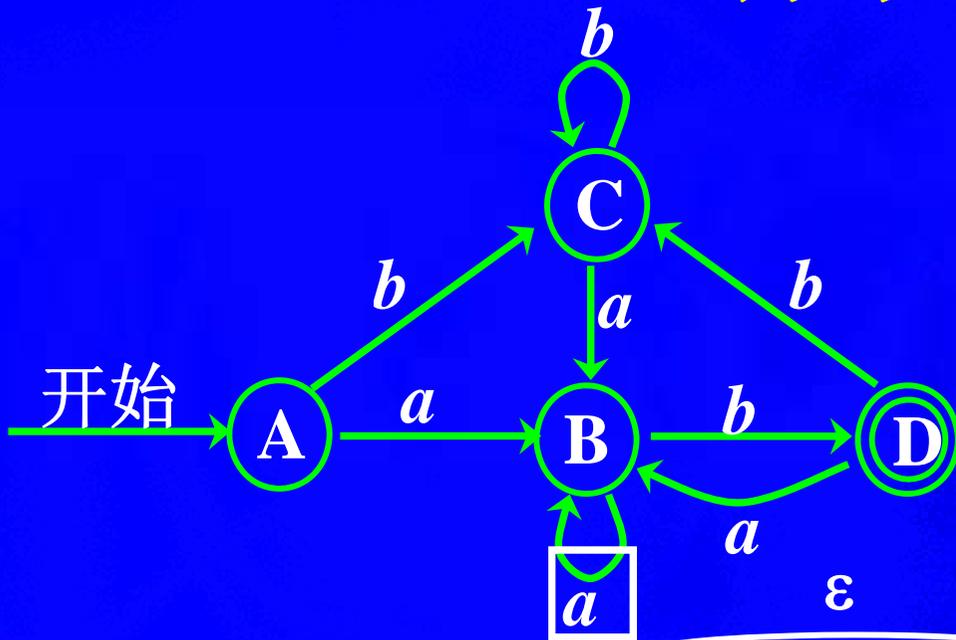
$C = \{1, 2, 4, 5, 6, 7\}$

$D = \{1, 2, 4, 5, 6, 7, 9\}$

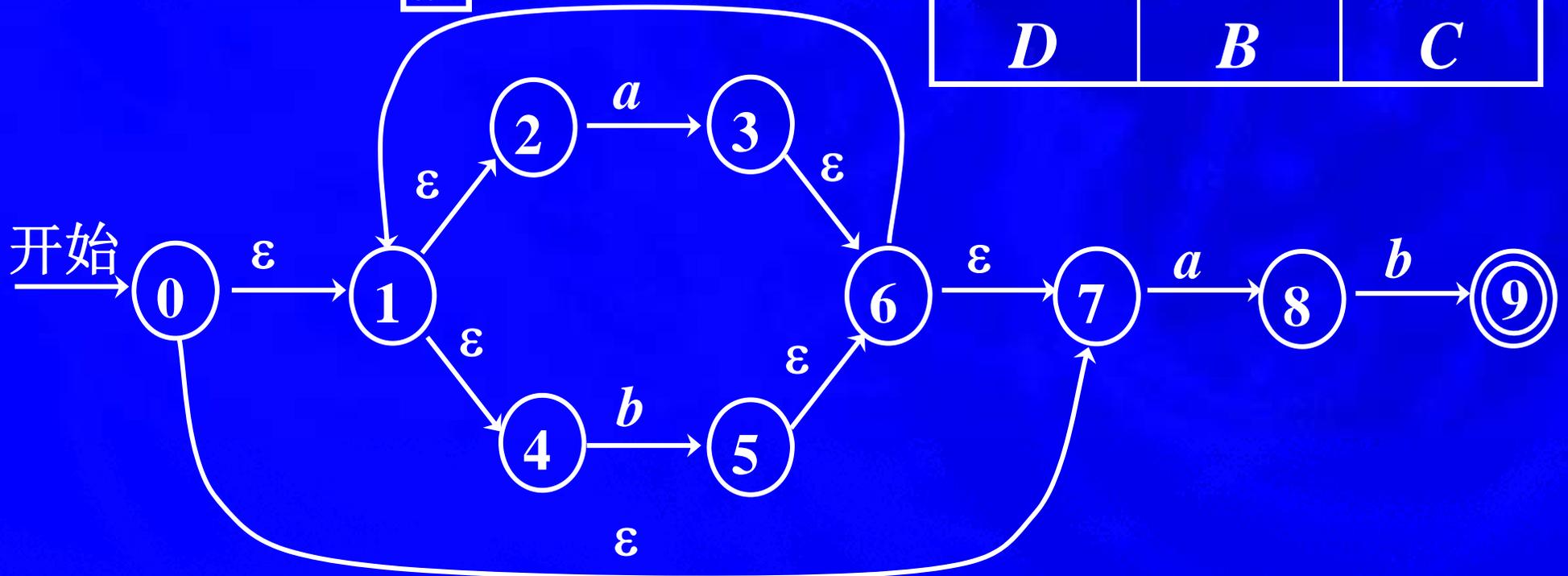
状态	输入符号	
	a	b
A	B	C
B	B	D
C	B	C
D	B	C



2.3 有限自动机

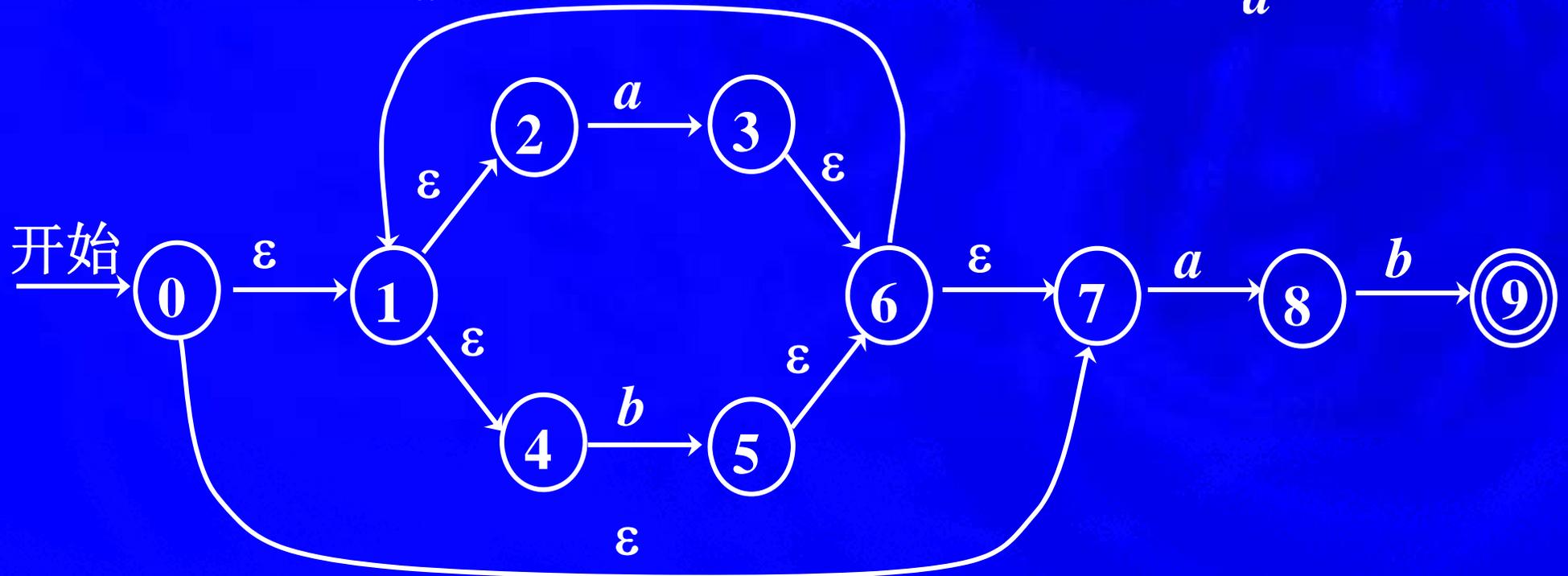
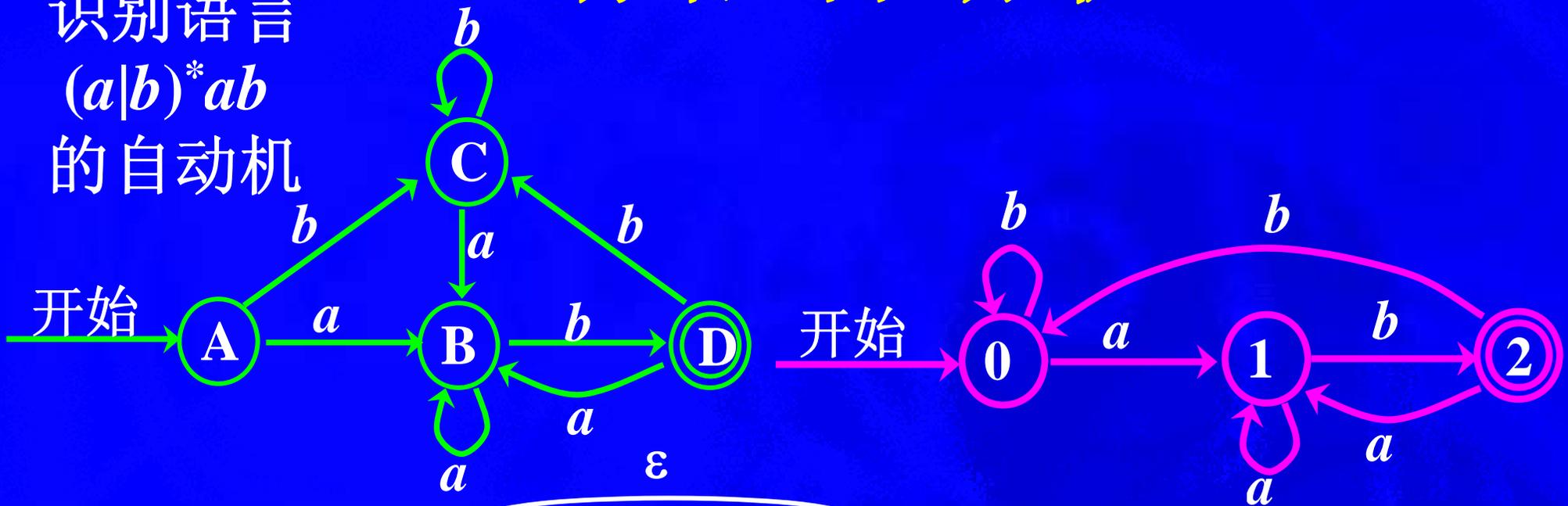


状态	输入符号	
	<i>a</i>	<i>b</i>
<i>A</i>	<i>B</i>	<i>C</i>
<i>B</i>	<i>B</i>	<i>D</i>
<i>C</i>	<i>B</i>	<i>C</i>
<i>D</i>	<i>B</i>	<i>C</i>



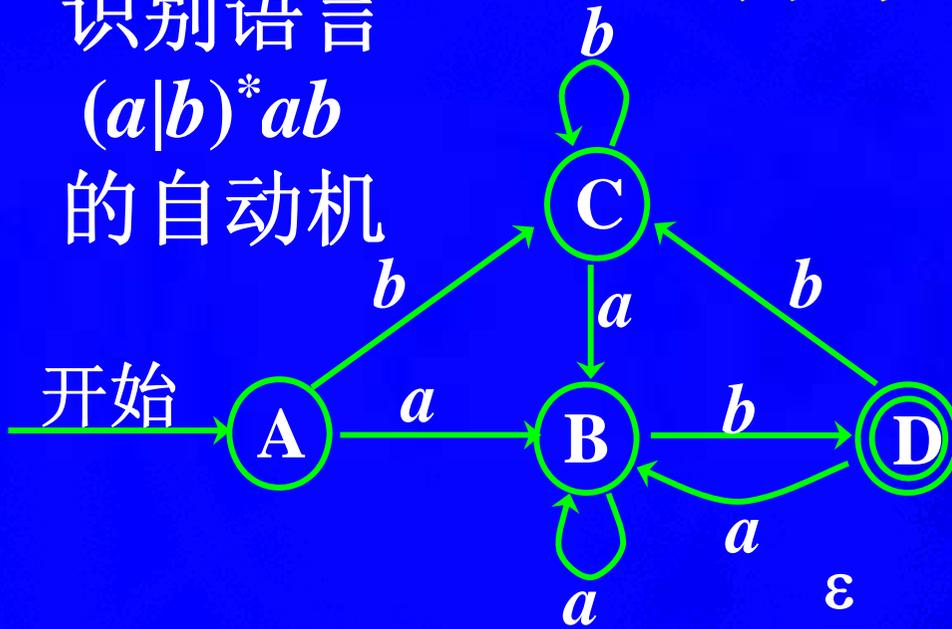
2.3 有限自动机

识别语言
 $(a|b)^*ab$
的自动机

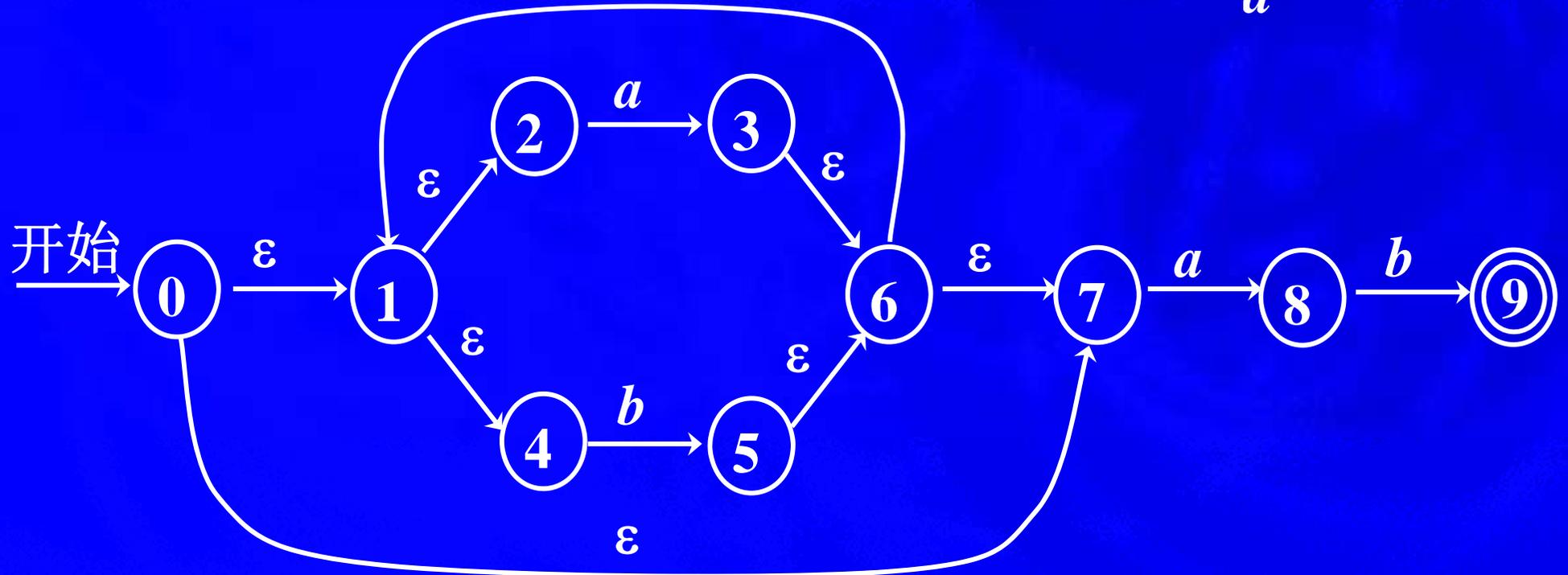
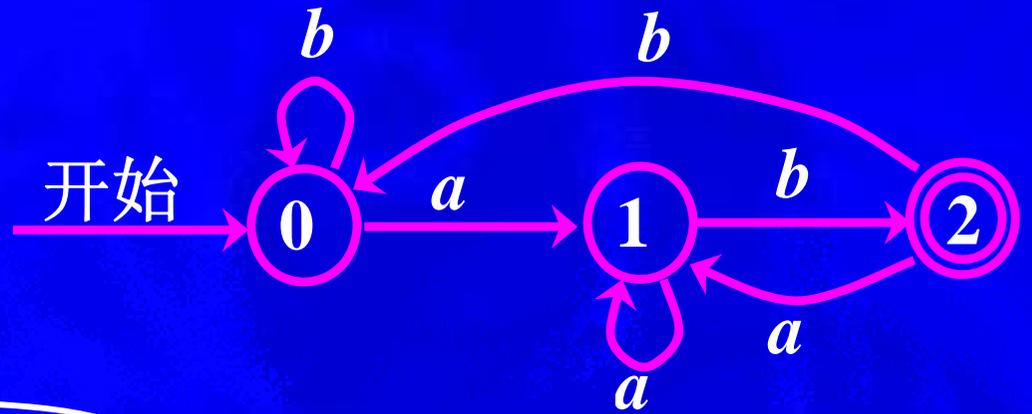


2.3 有限自动机

识别语言
 $(a|b)^*ab$
的自动机



子集构造法不一定得到最简DFA

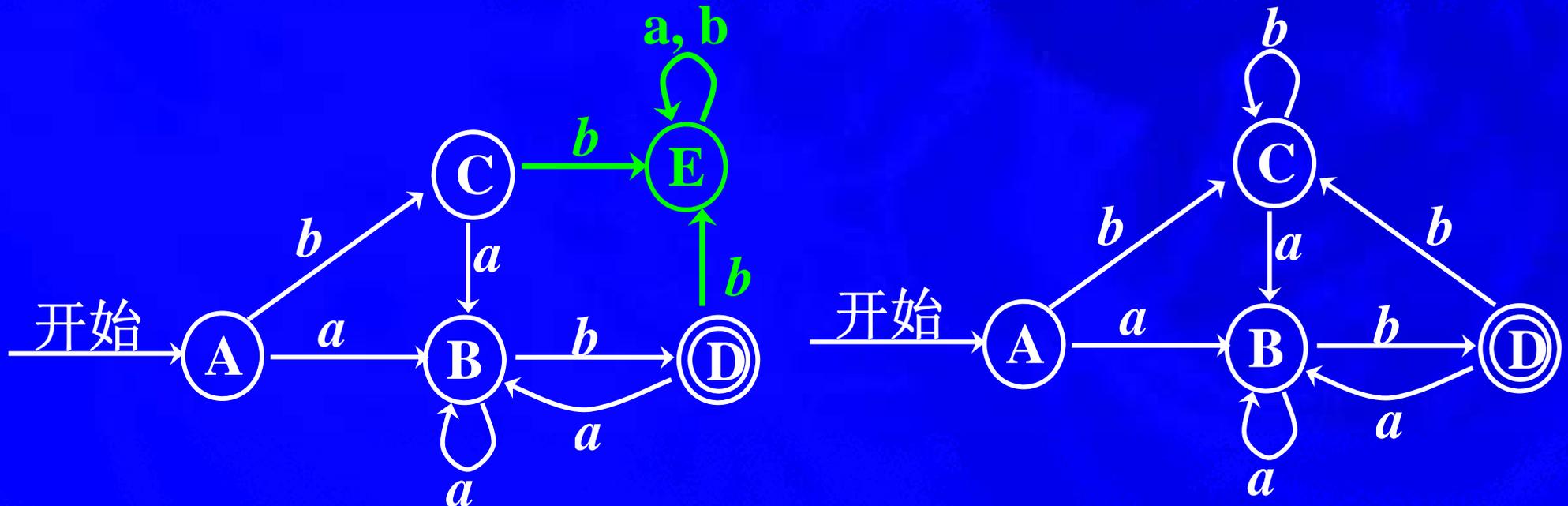


2.3 有限自动机

2.3.4 DFA的化简

- 死状态

- 在转换函数由部分函数改成全函数表示时引入
- 左图需要引入死状态E；右图无须引入死状态



2.3 有限自动机

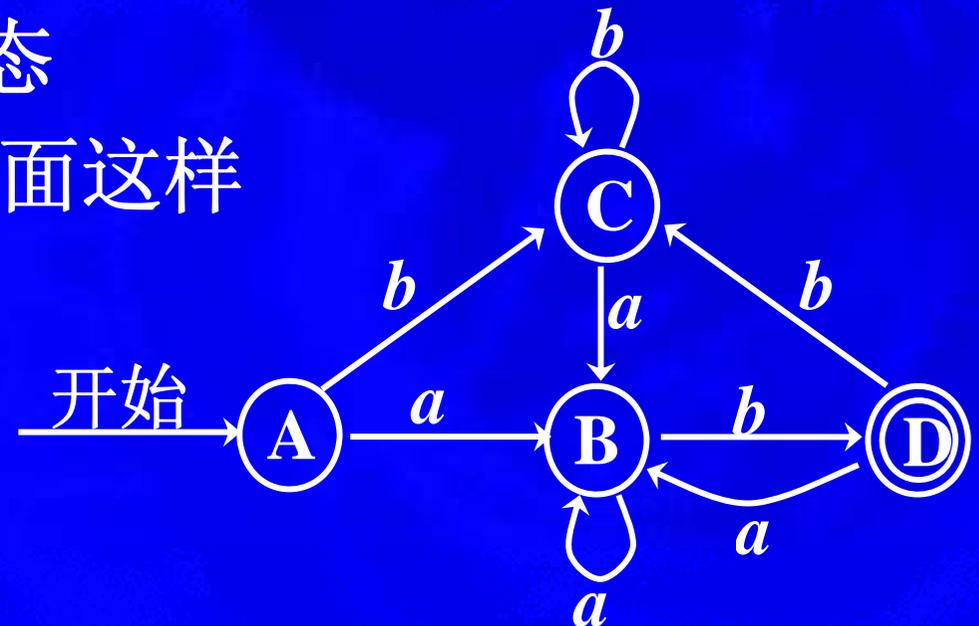
- 可区别的状态

- A和B是可区别的状态

- 从A出发，读过串**b**，到达非接受状态C，而
 - 从B出发，读过串**b**，到达接受状态D

- A和C是不可区别的状态

- 无任何串可用来像上面这样
 - 区别它们



2.3 有限自动机

- 方法

1. $\{A, B, C\}, \{D\}$

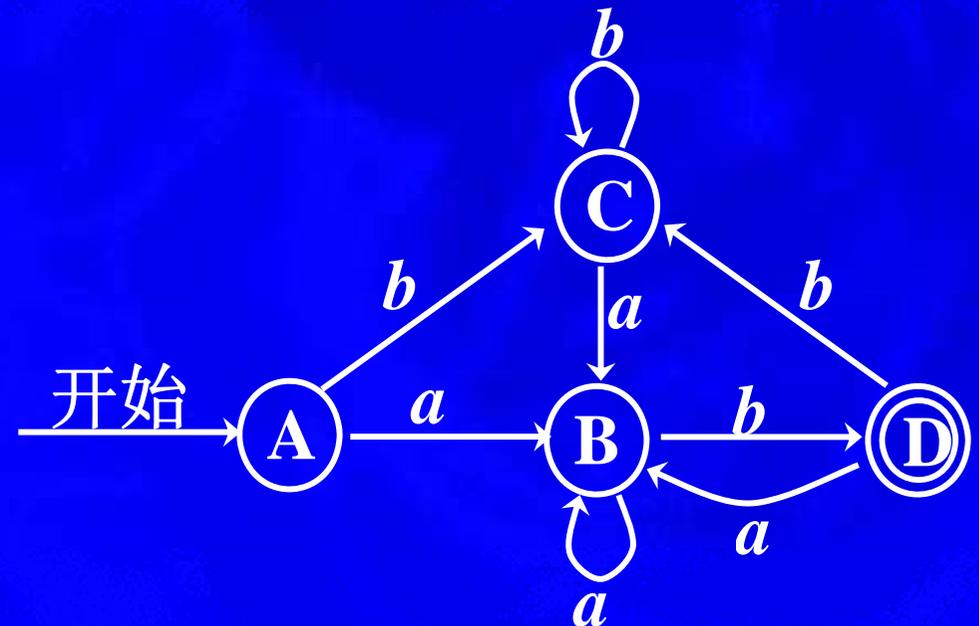
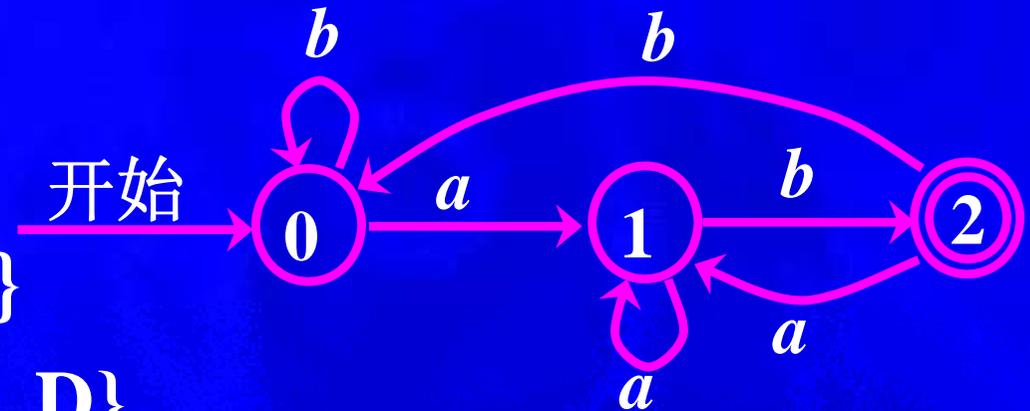
$move(\{A, B, C\}, a) = \{B\}$

$move(\{A, B, C\}, b) = \{C, D\}$

2. $\{A, C\}, \{B\}, \{D\}$

$move(\{A, C\}, a) = \{B\}$

$move(\{A, C\}, b) = \{C\}$

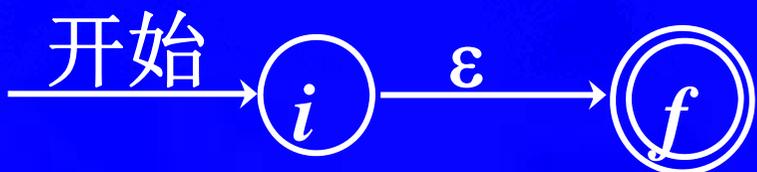


2.4 从正规式到有限自动机

- 从正规式建立识别器的步骤
 - 从正规式构造NFA（本节介绍）
用语法制导的算法，它用正规式语法结构来指导构造过程
 - 把NFA变成DFA（子集构造法，已介绍）
 - 将DFA化简（合并不可区别的状态，也已介绍）

2.4 从正规式到有限自动机

- 首先构造识别 ε 和字母表中一个符号的NFA
重要特点：仅一个接受状态，它没有向外的转换



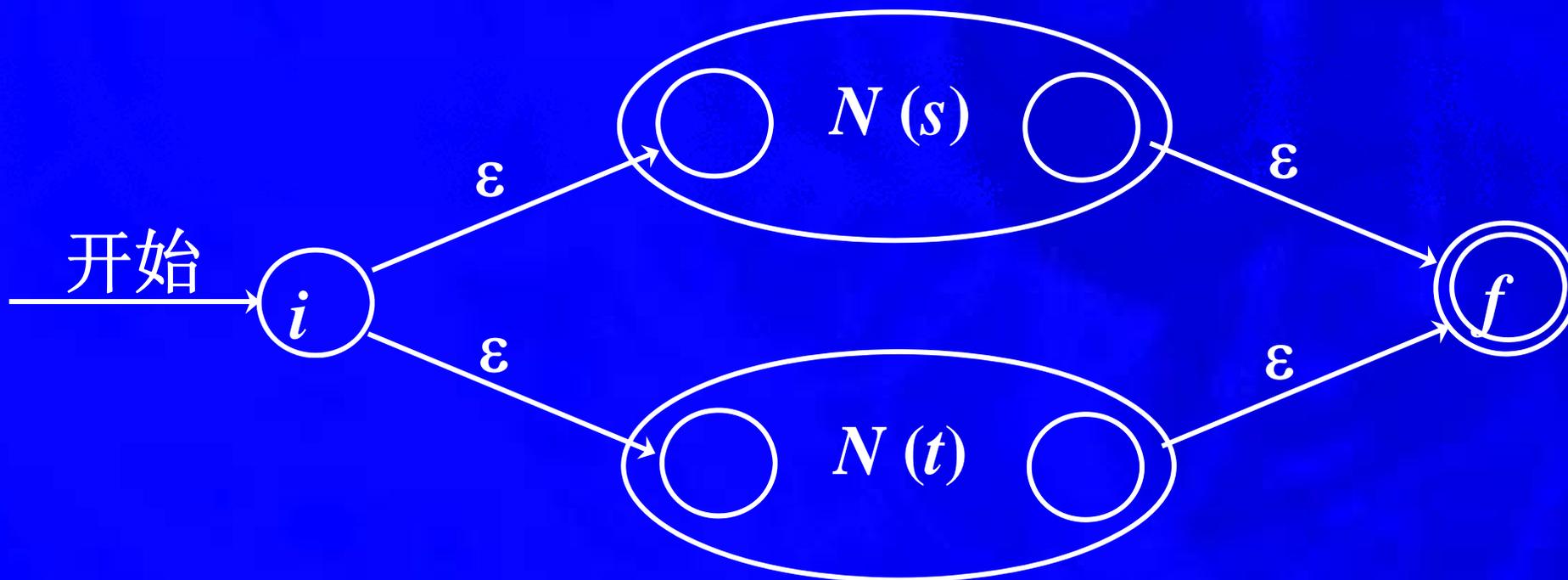
识别正规式 ε 的NFA



识别正规式 a 的NFA

2.4 从正规式到有限自动机

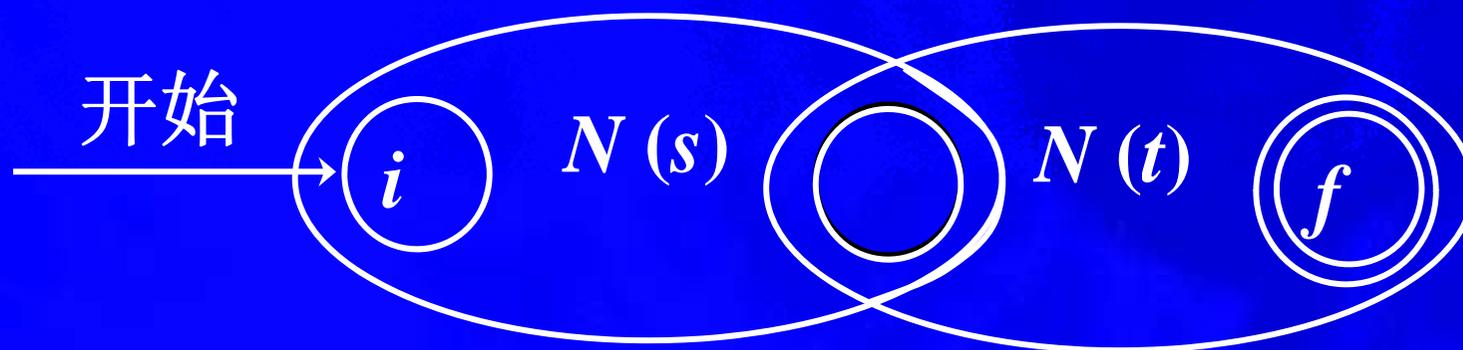
- 构造识别主算符为选择的正规式的NFA
重要特点：仅一个接受状态，它没有向外的转换



识别正规式 $s | t$ 的NFA

2.4 从正规式到有限自动机

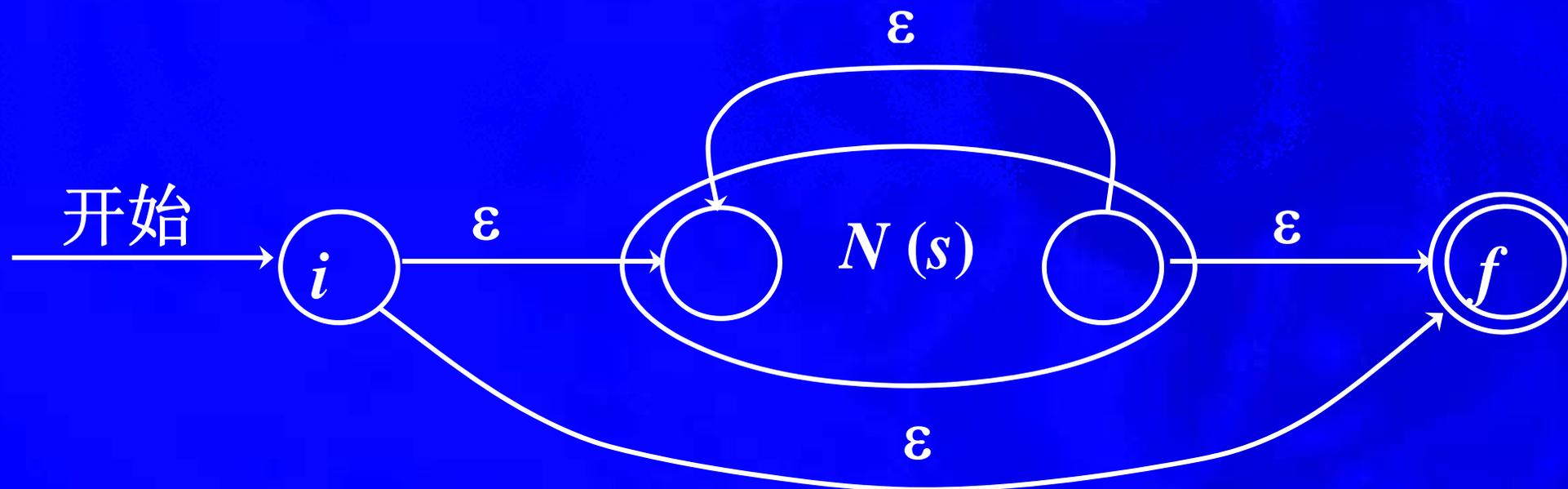
- 构造识别主算符为连接的正规式的NFA
重要特点：仅一个接受状态，它没有向外的转换



识别正规式 st 的NFA

2.4 从正规式到有限自动机

- 构造识别主算符为闭包的正规式的NFA
重要特点：仅一个接受状态，它没有向外的转换



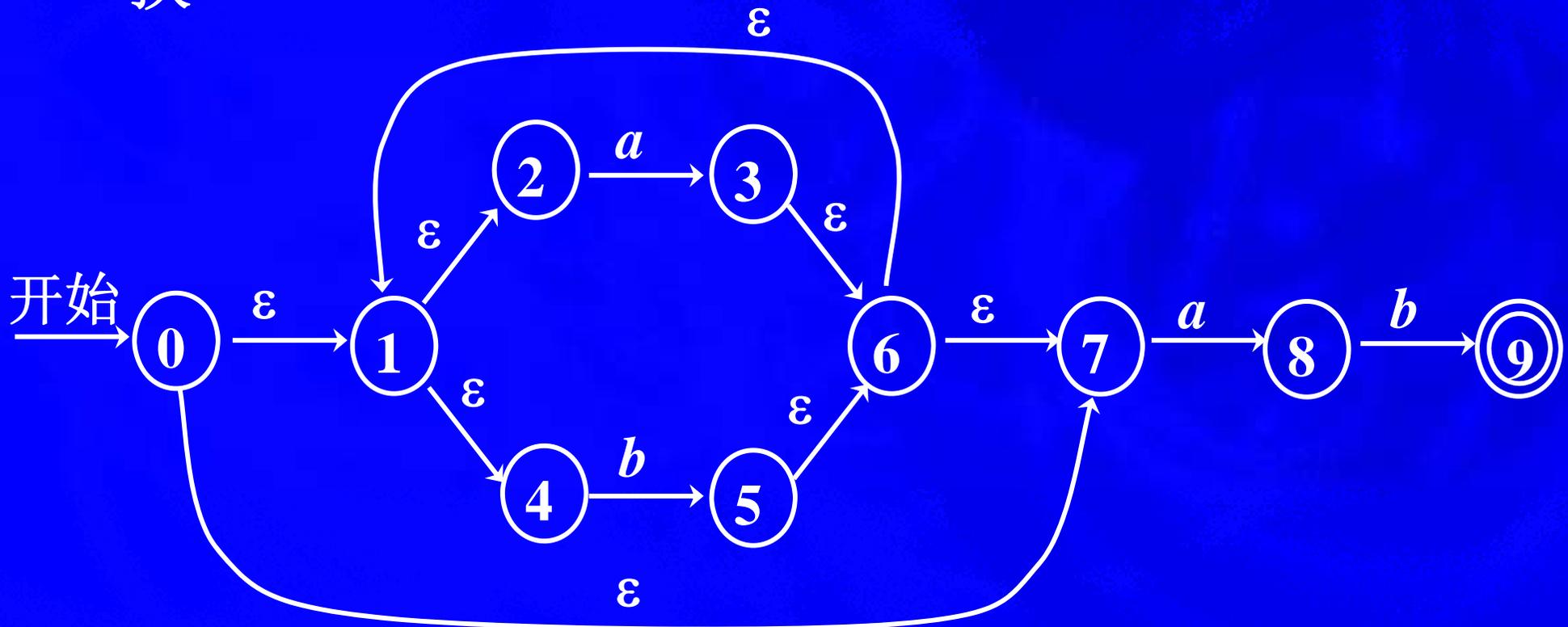
识别正规式 s^* 的NFA

2.4 从正规式到有限自动机

- 对于加括号的正规式(s), 使用 $N(s)$ 本身作为它的**NFA**

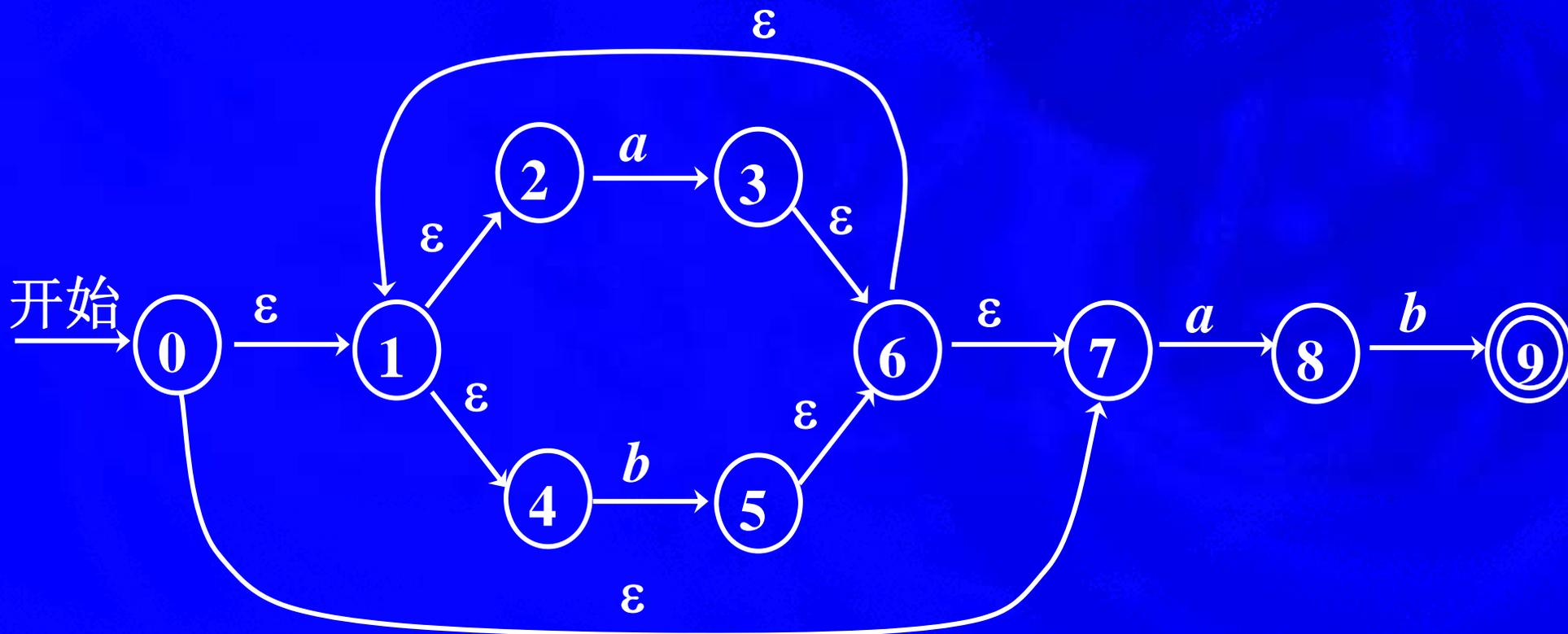
2.4 从正规式到有限自动机

- 本方法产生的NFA有下列性质
 - $N(r)$ 的状态数最多是 r 中符号和算符总数的两倍
 - $N(r)$ 只有一个接受状态，接受状态没有向外的转换



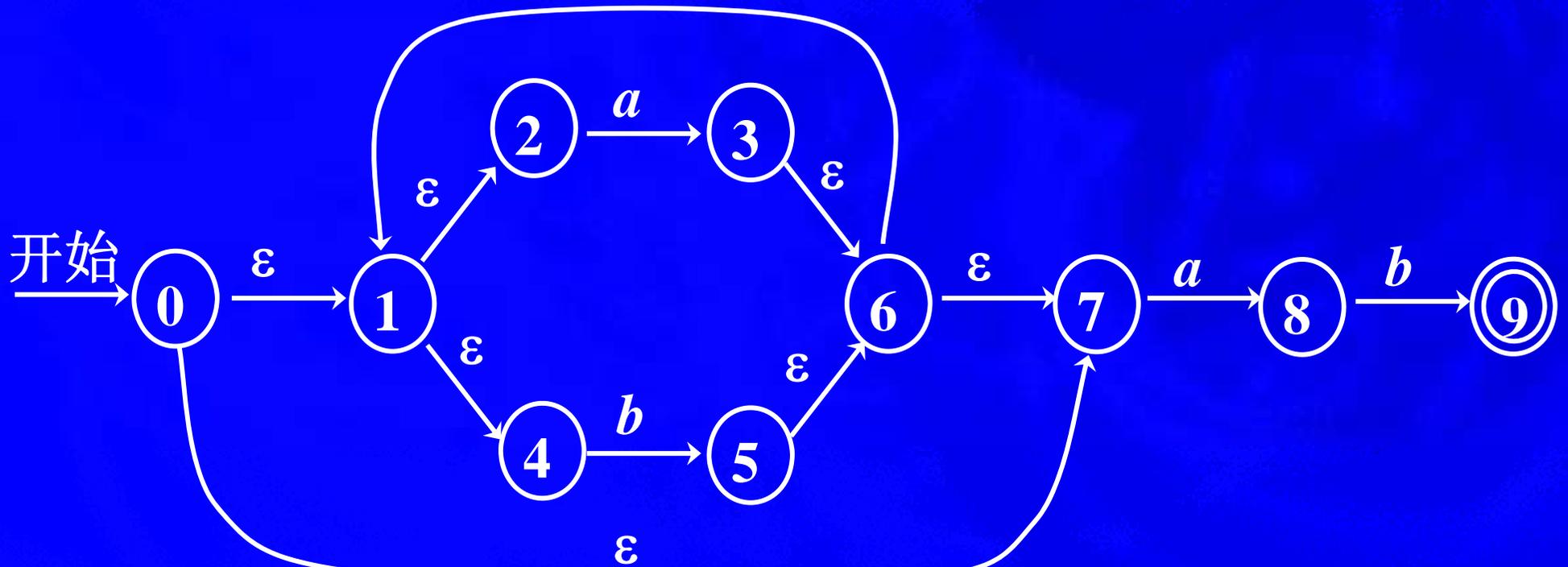
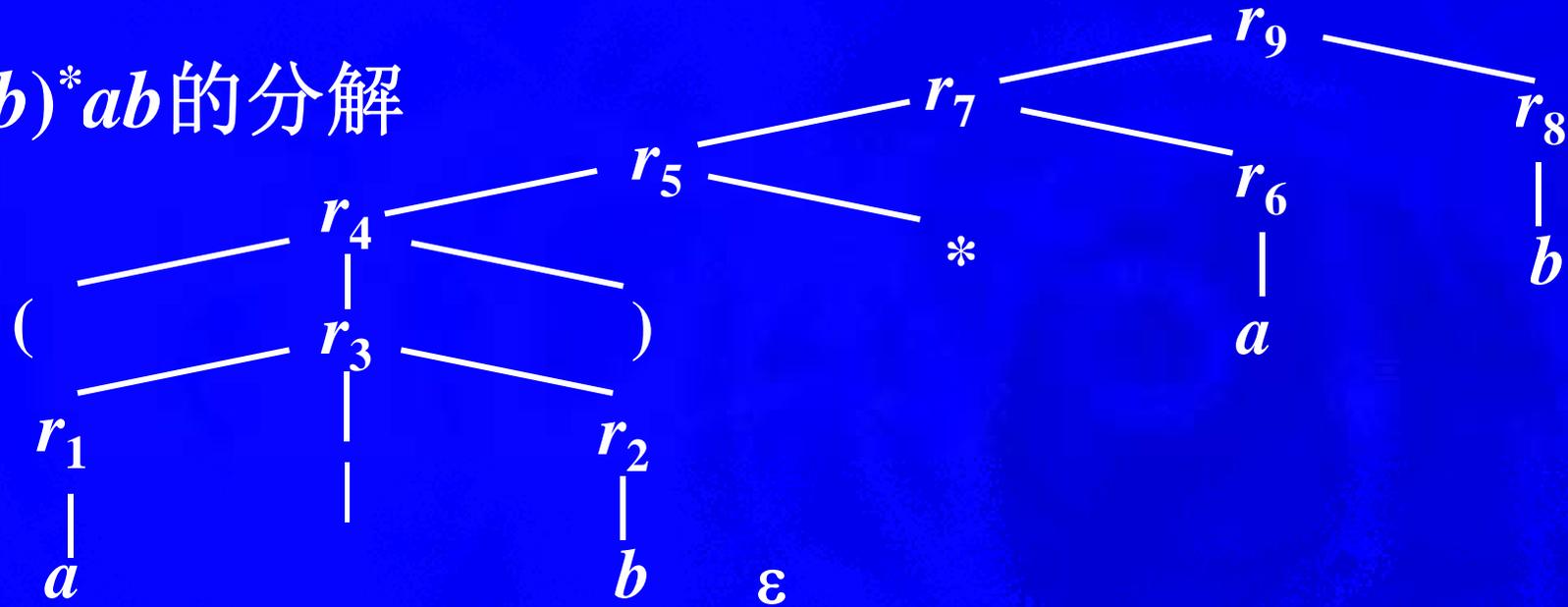
2.4 从正规式到有限自动机

- 本方法产生的NFA有下列性质
 - $N(r)$ 的每个状态有一个用 Σ 的符号标记的指向其他状态的转换，或者最多两个指向其他状态的 ϵ 转换



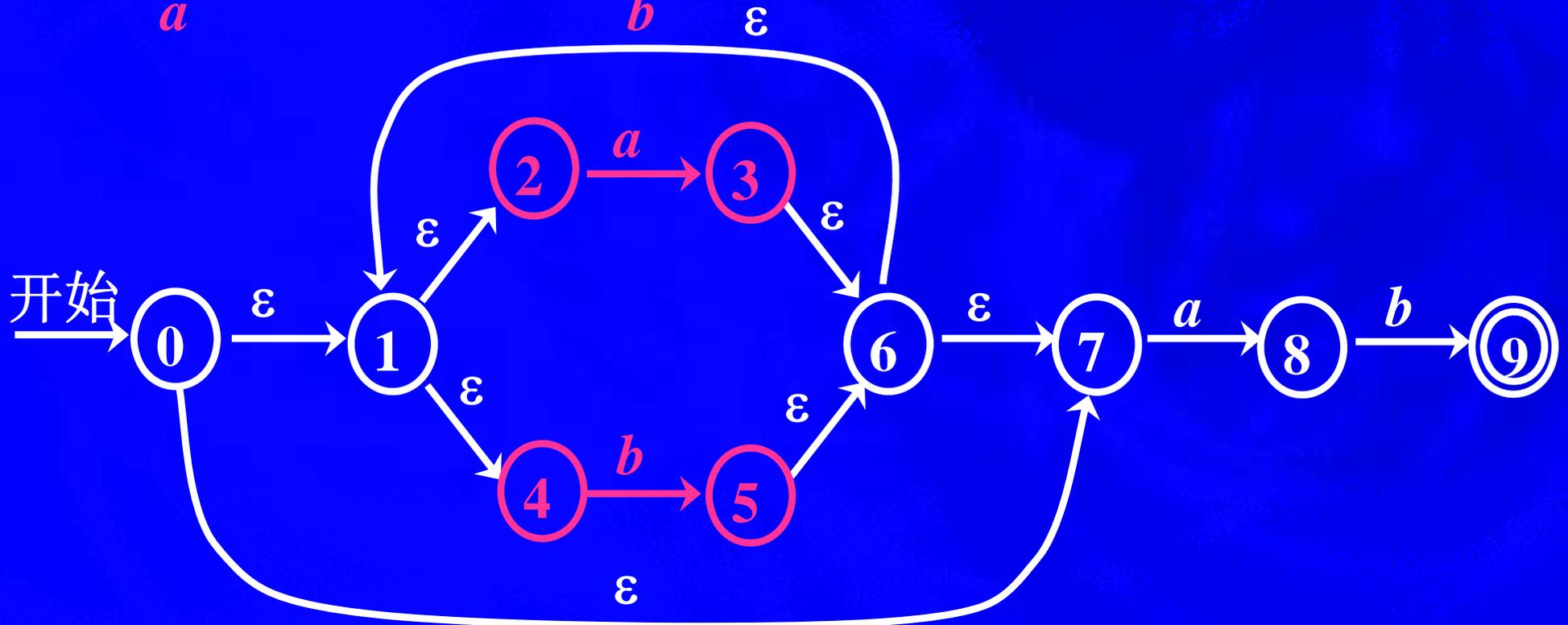
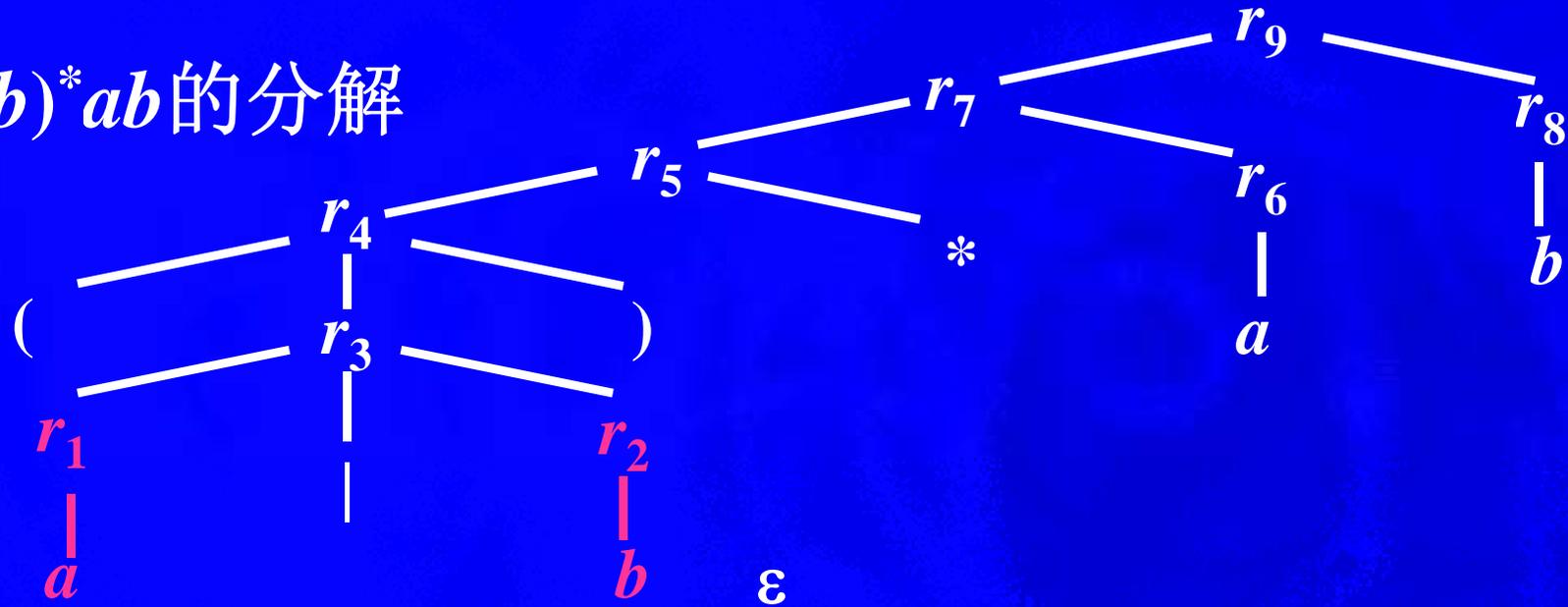
2.4 从正规式到有限自动机

$(a|b)^*ab$ 的分解



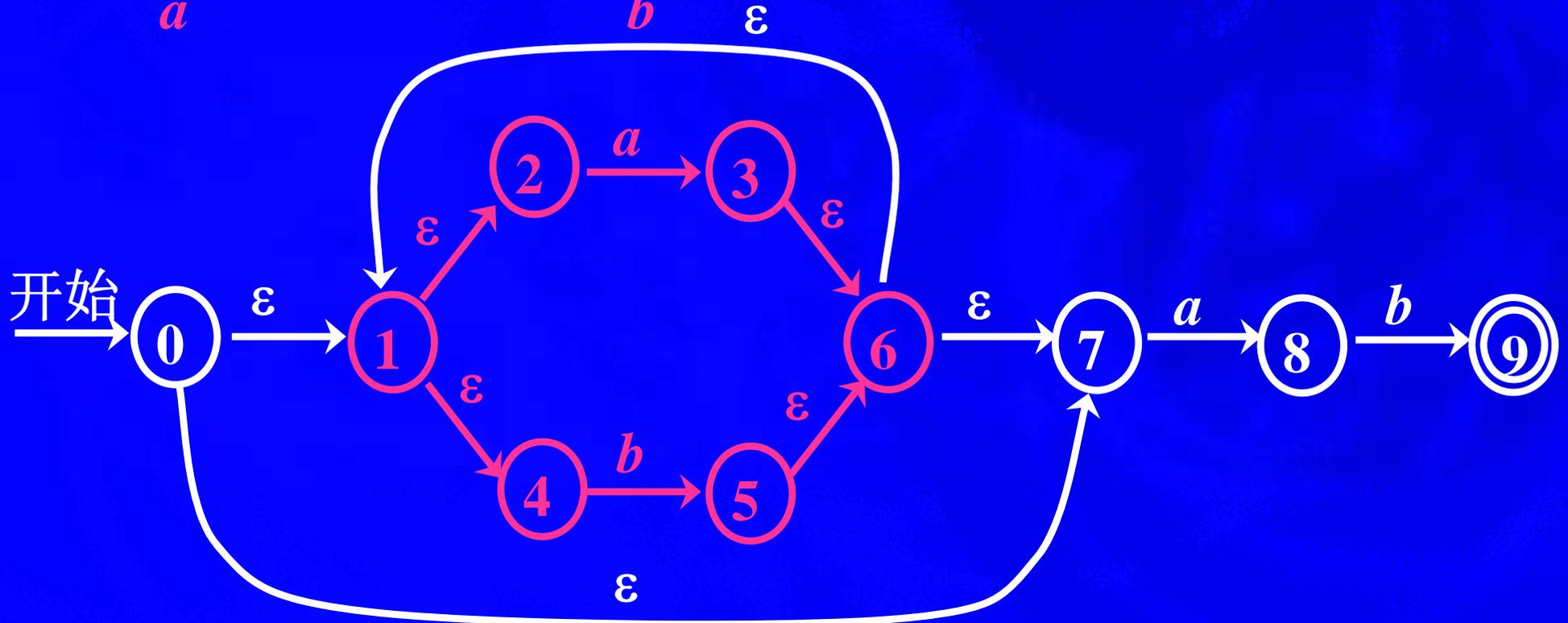
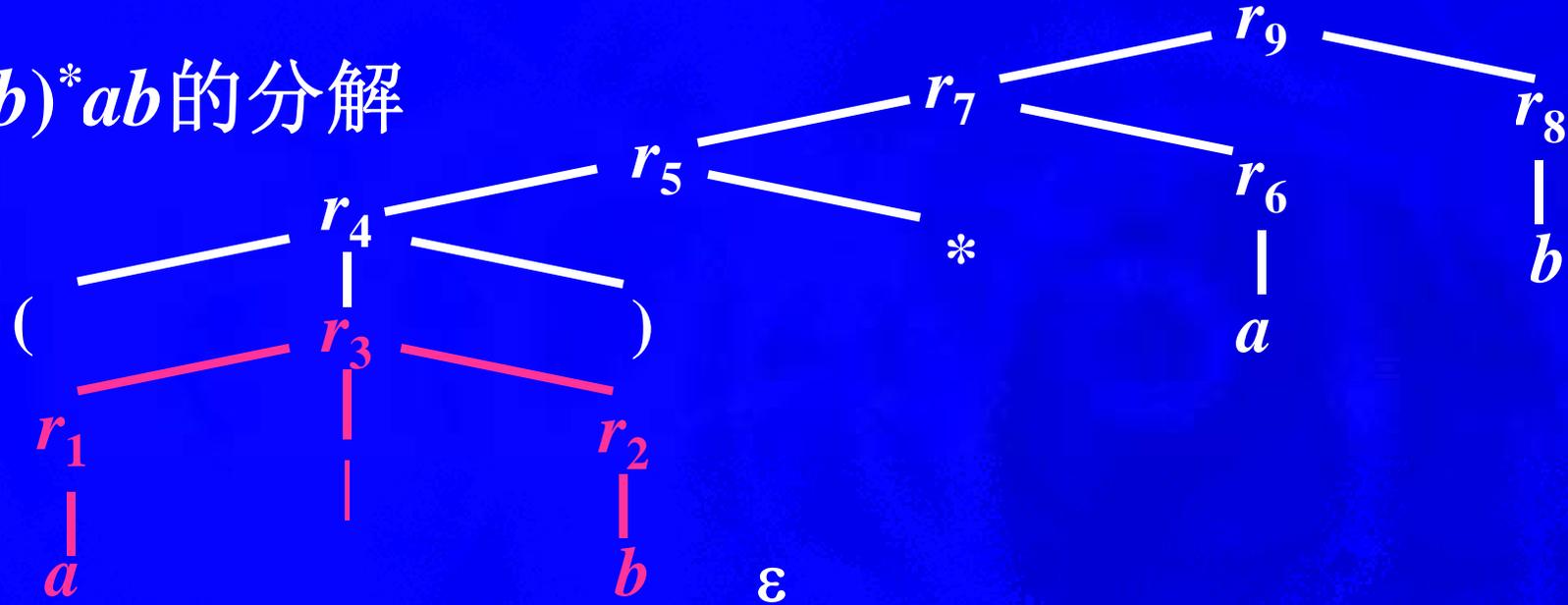
2.4 从正规式到有限自动机

$(a|b)^*ab$ 的分解



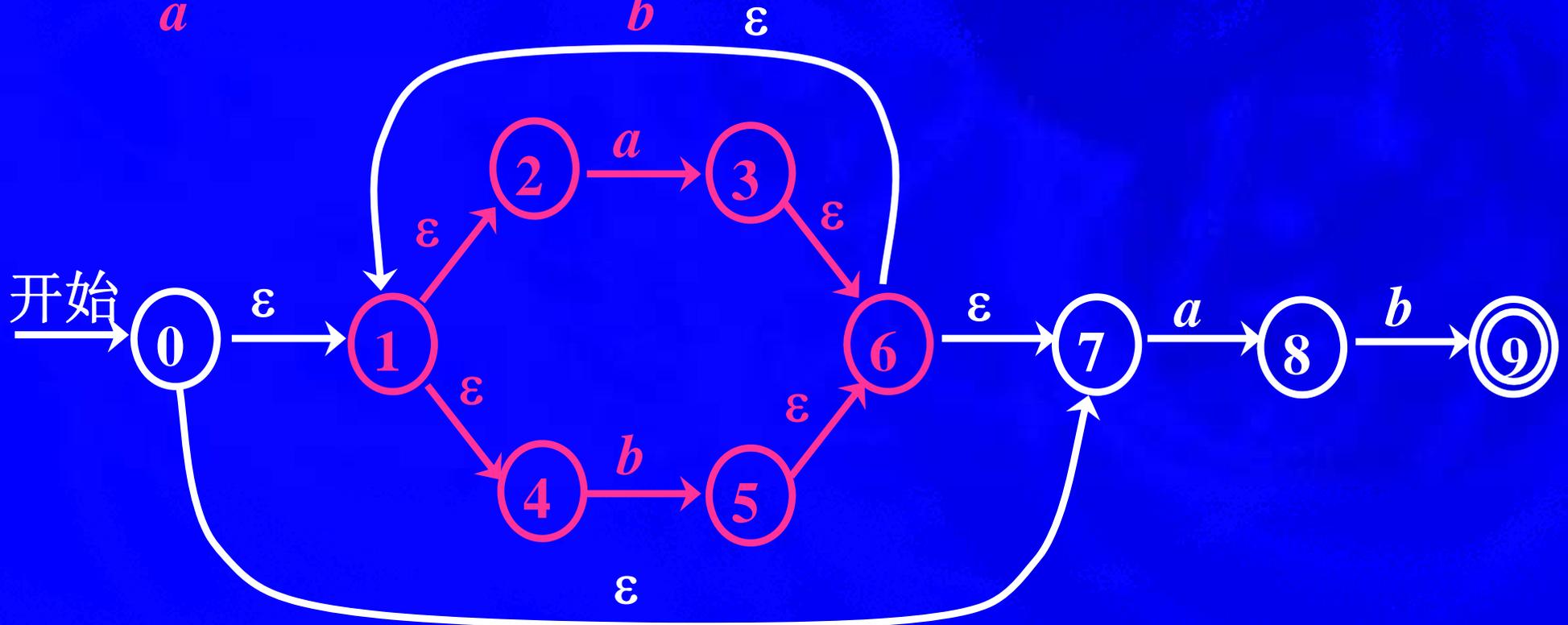
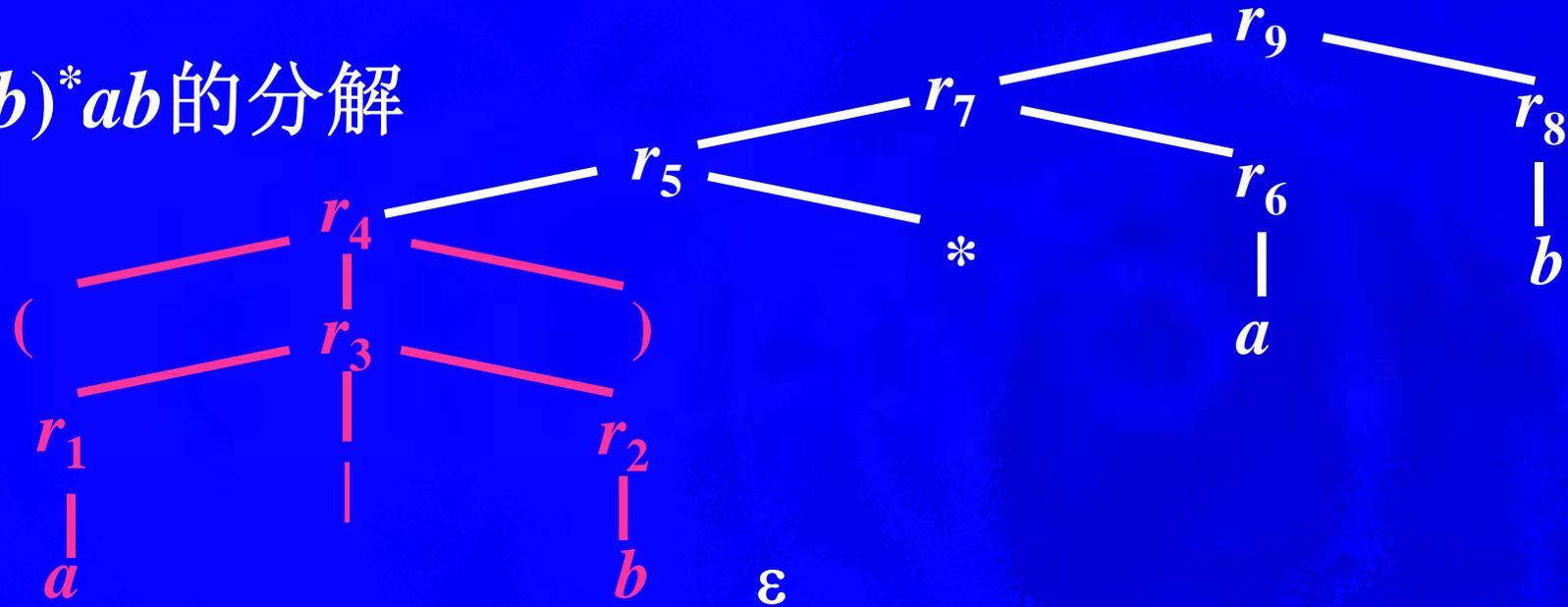
2.4 从正规式到有限自动机

$(a|b)^*ab$ 的分解



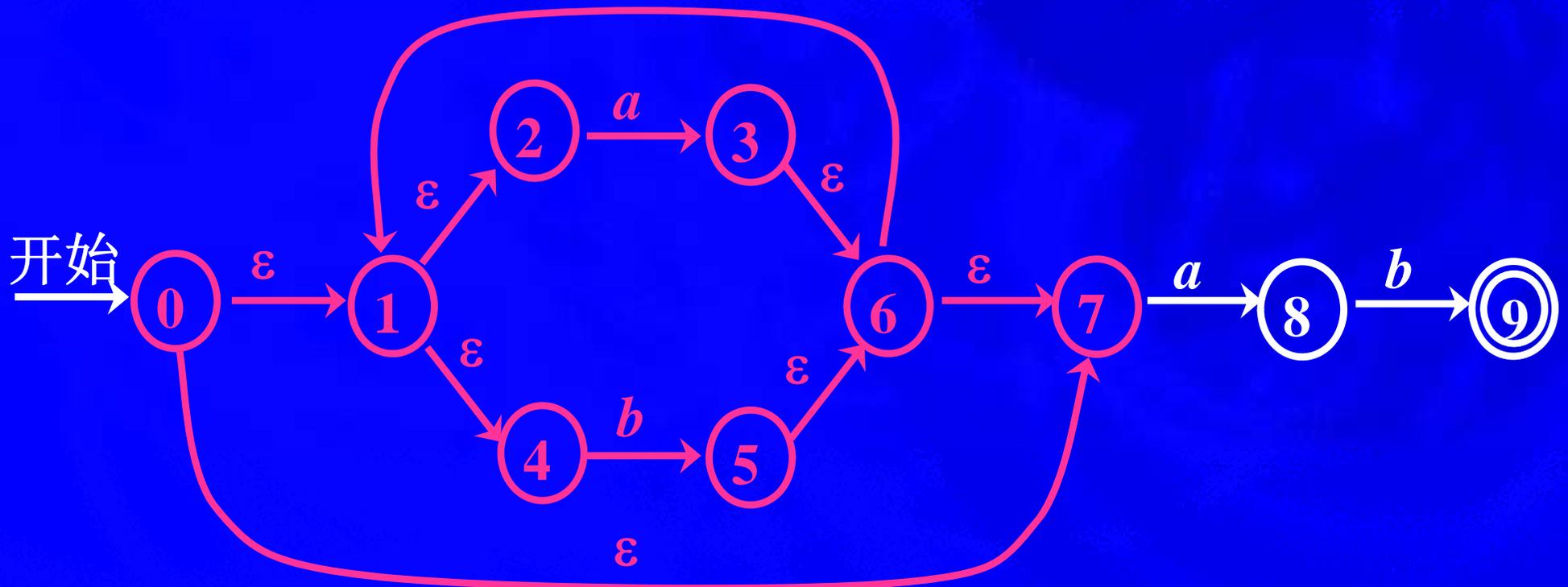
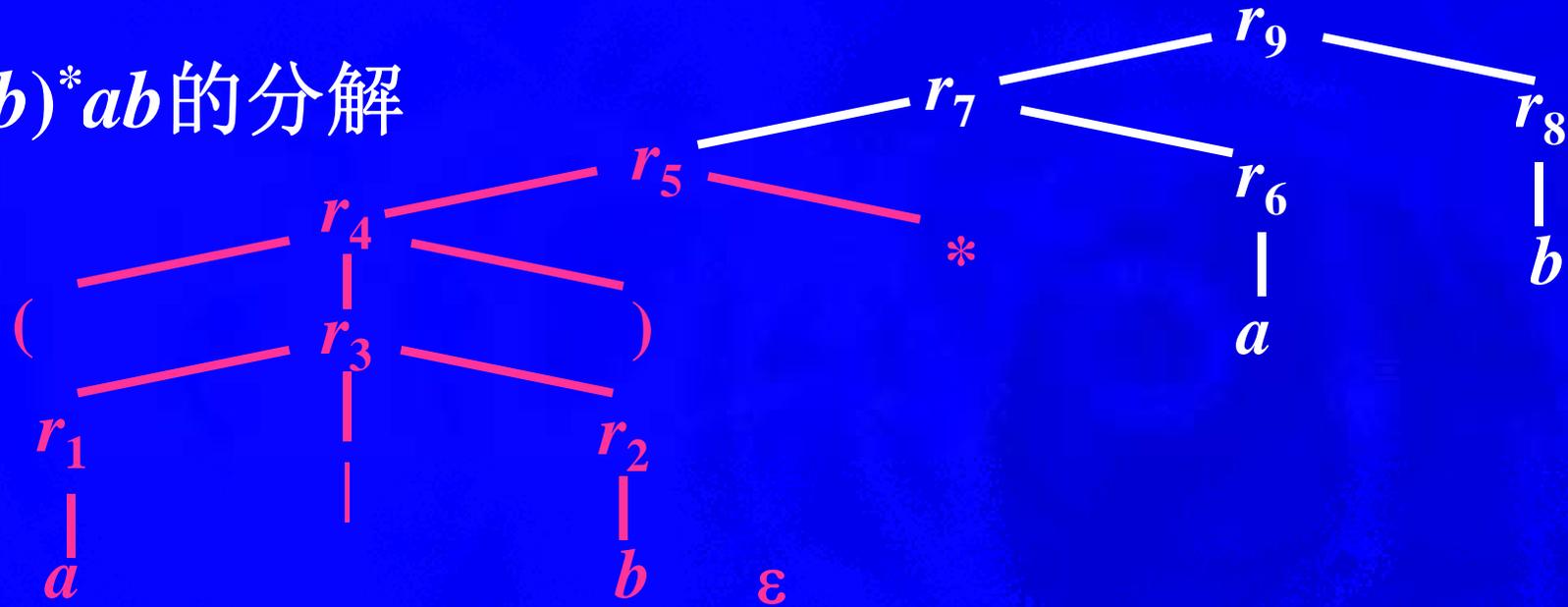
2.4 从正规式到有限自动机

$(a|b)^*ab$ 的分解



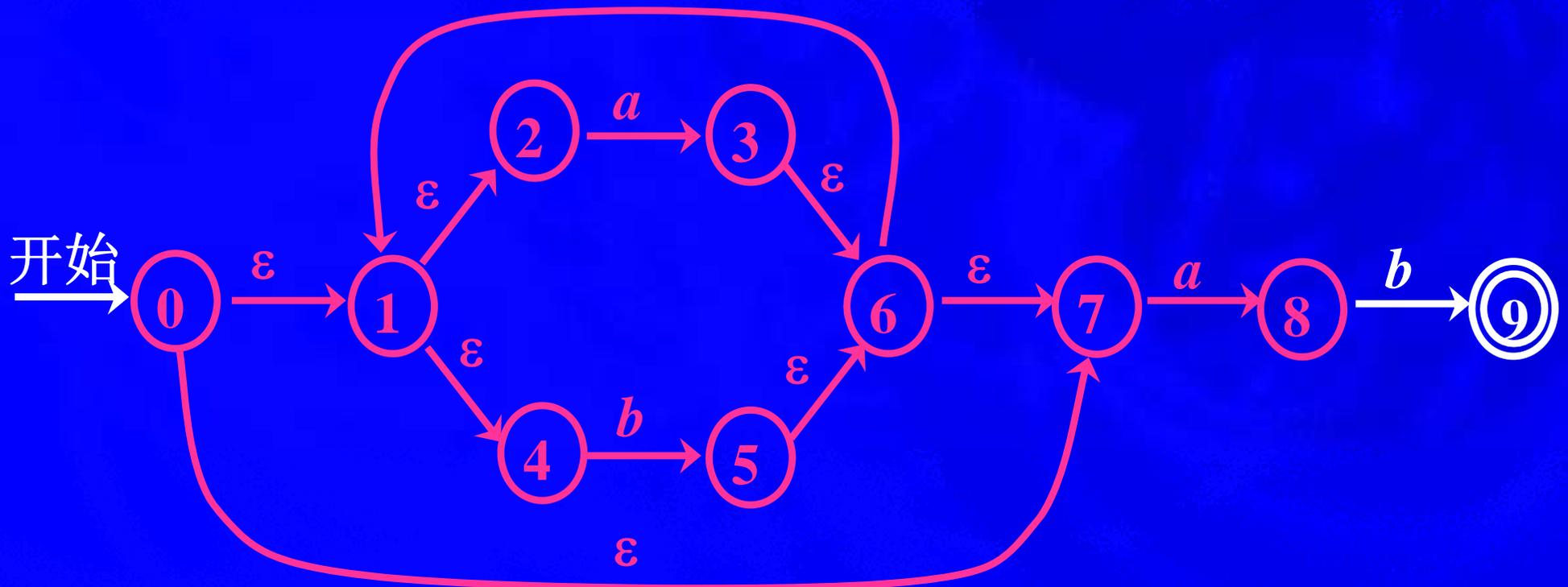
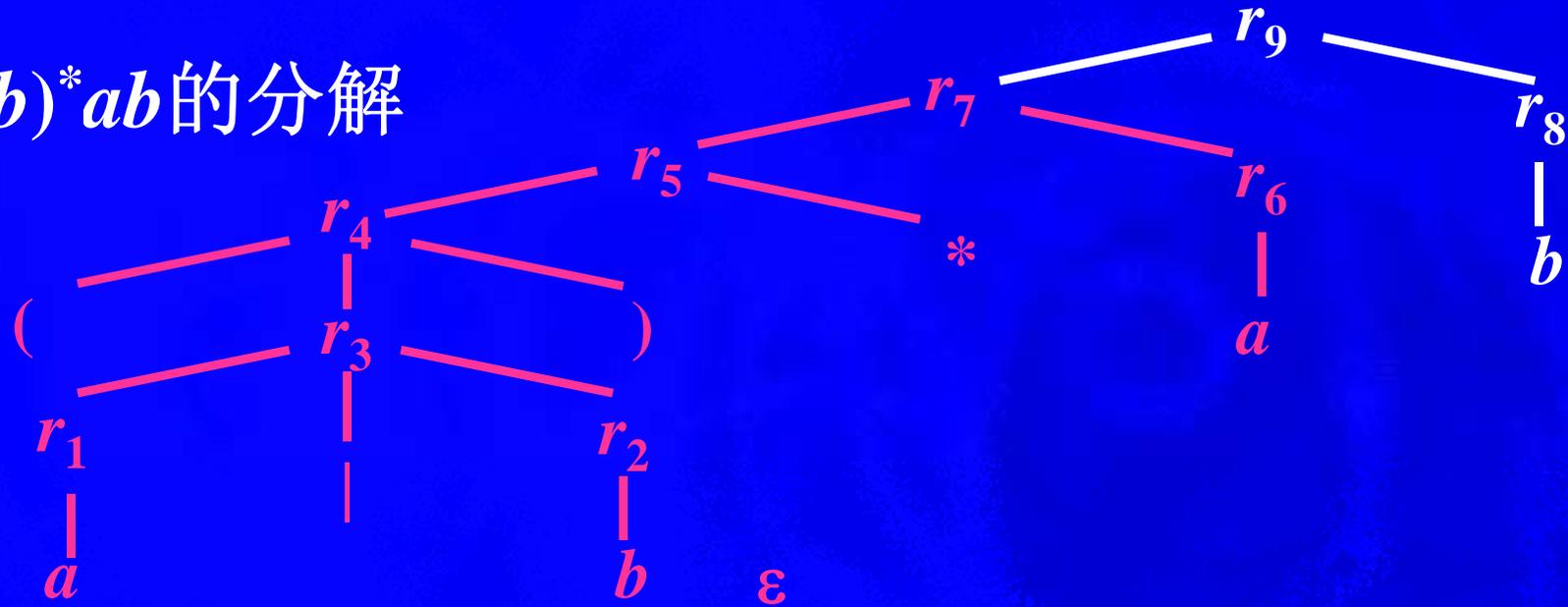
2.4 从正规式到有限自动机

$(a|b)^*ab$ 的分解



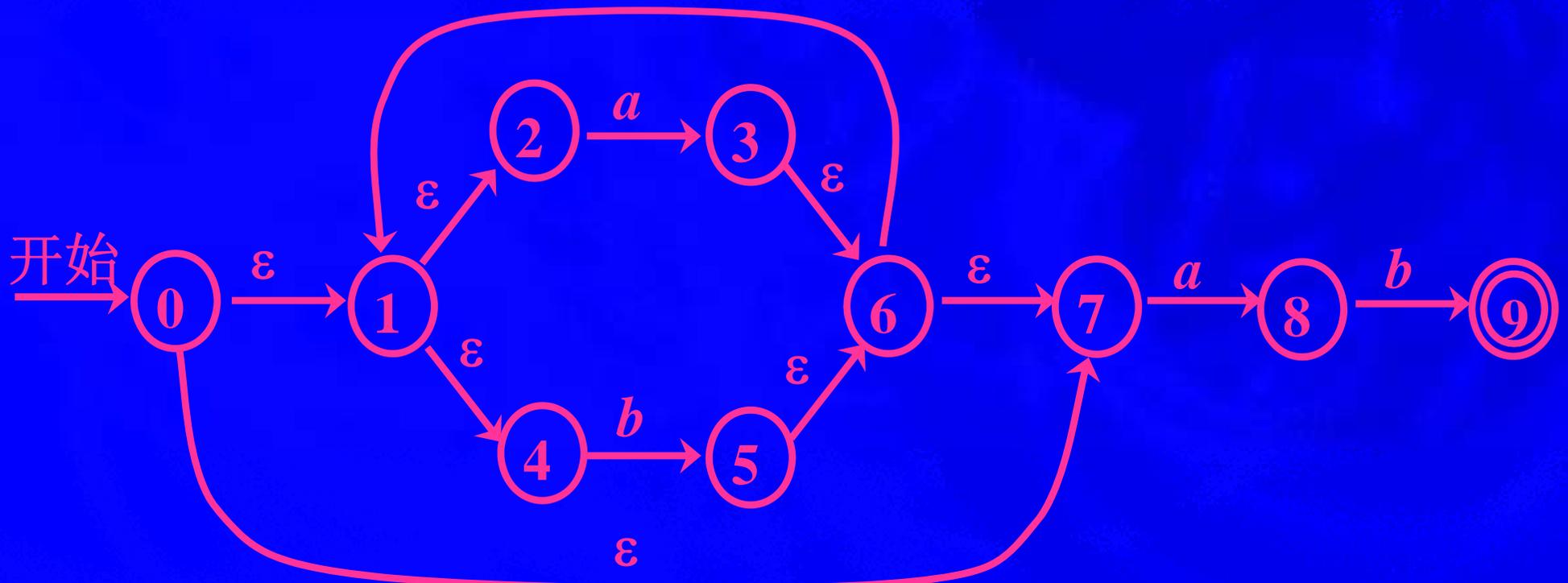
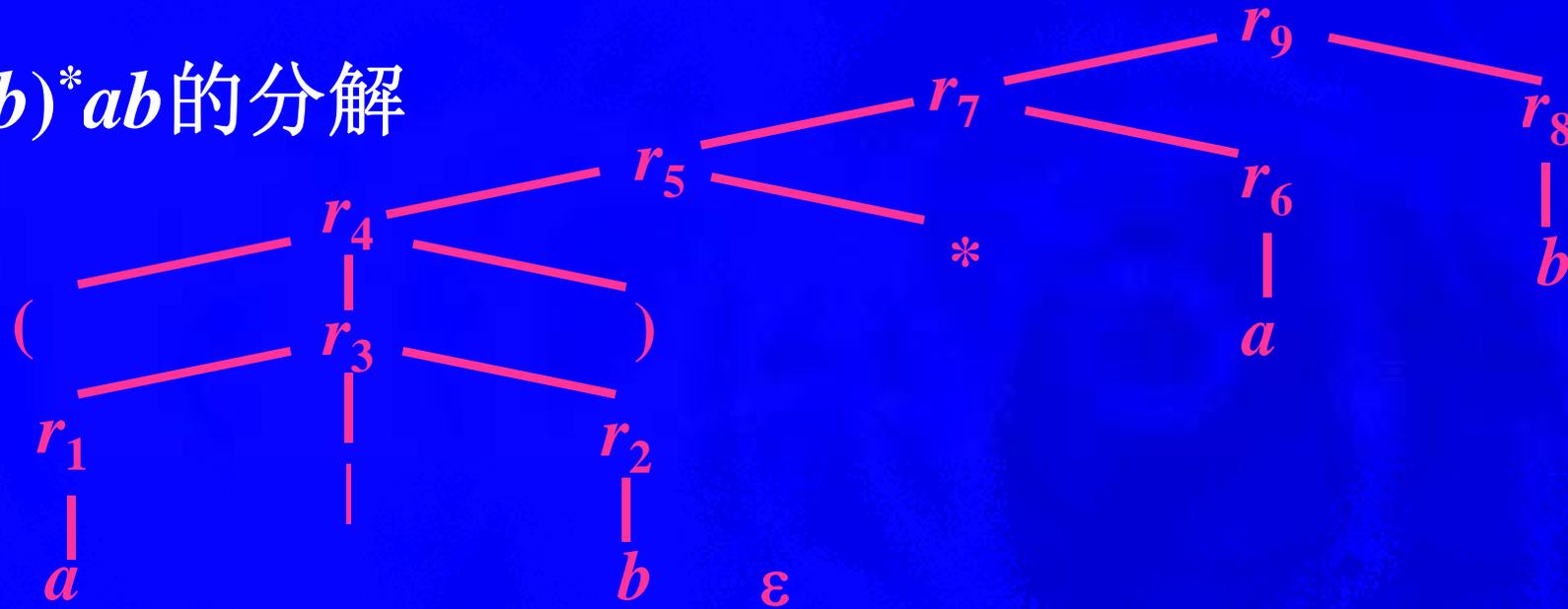
2.4 从正规式到有限自动机

$(a|b)^*ab$ 的分解



2.4 从正规式到有限自动机

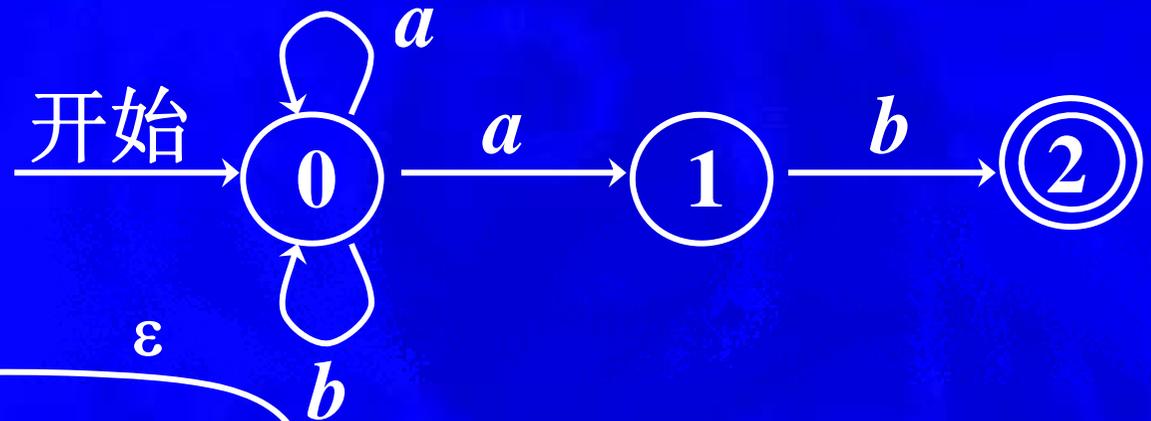
$(a|b)^*ab$ 的分解



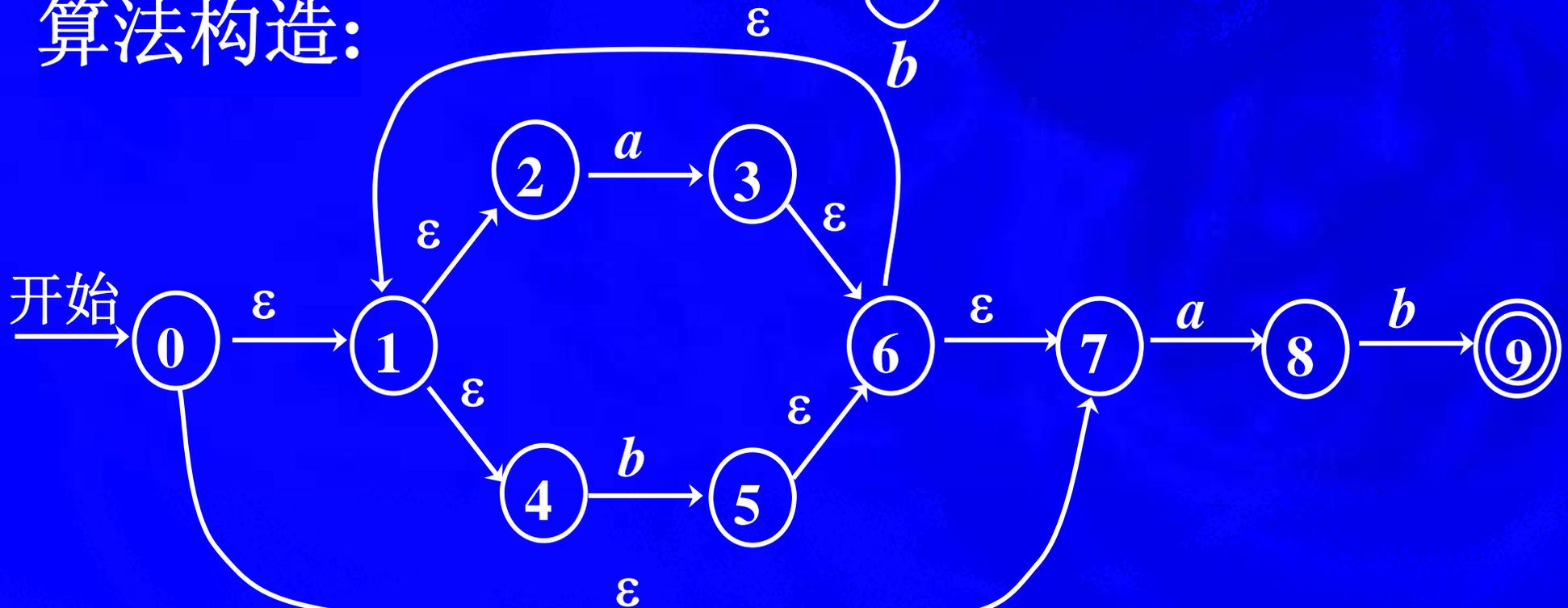
2.4 从正规式到有限自动机

- $(a|b)^*ab$ 的两个NFA的比较

手工构造:

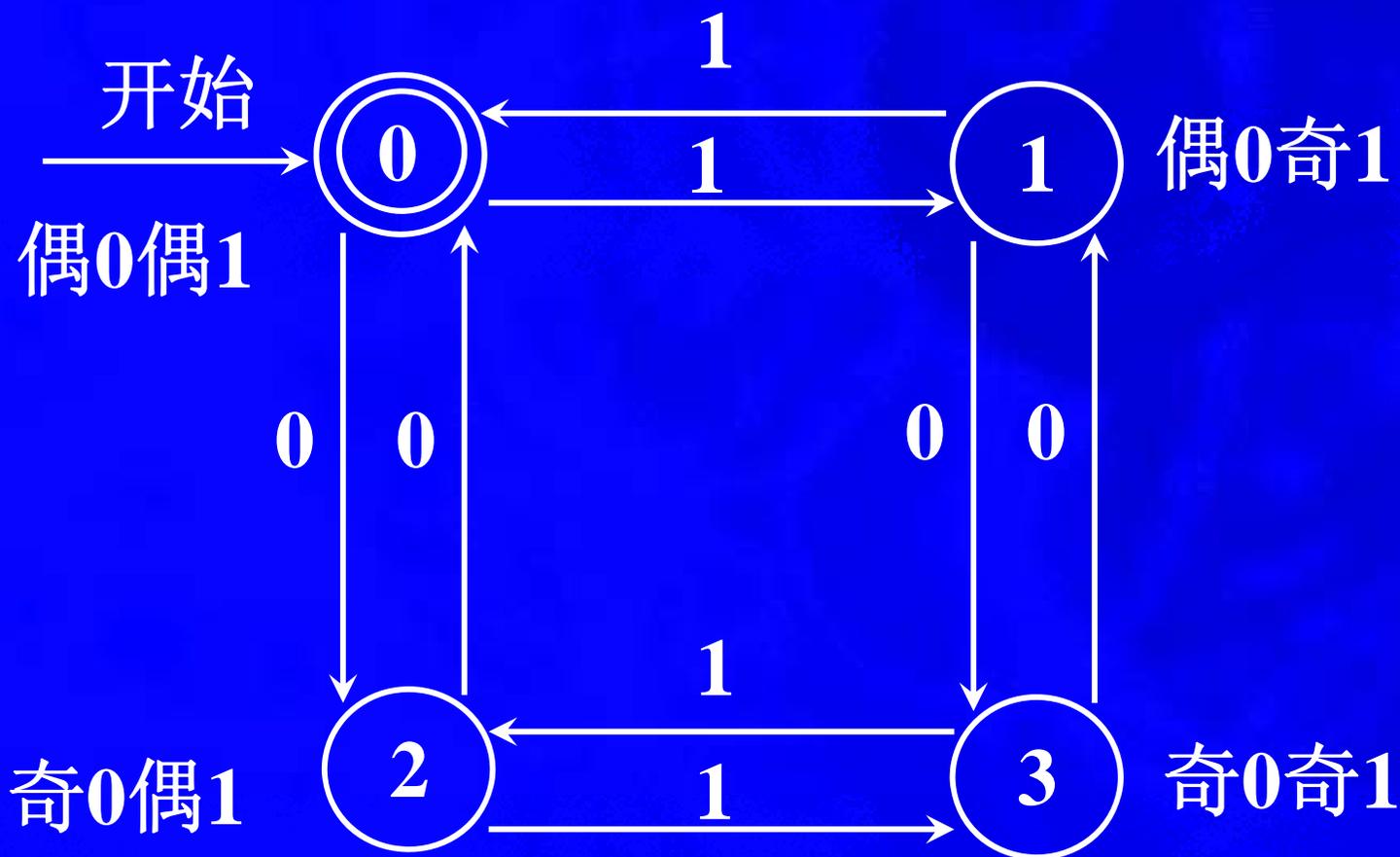


算法构造:



2.4 从正规式到有限自动机

- 例 DFA, 接受 0 和 1 的个数都是偶数的字符串

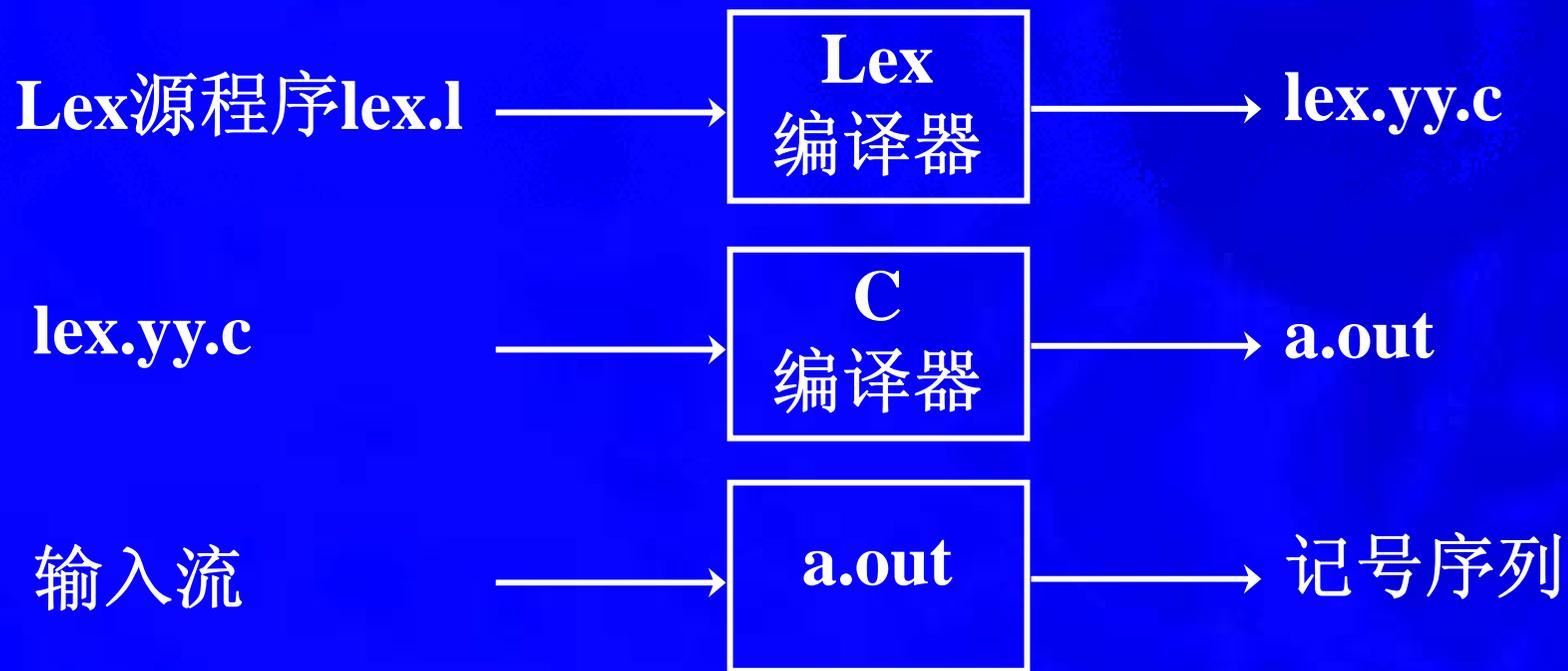


2.4 从正规式到有限自动机

- 小结：从正规式建立识别器的步骤
 - 从正规式构造NFA
 - 把NFA变成DFA
 - 将DFA化简
- 存在其他办法

2.5 词法分析器的生成器

- 用Lex建立词法分析器的步骤



2.5 词法分析器的生成器

- Lex程序包括三个部分

声明

%%

翻译规则

%%

辅助过程

- Lex程序的翻译规则

p_1 {动作1}

p_2 {动作2}

...

p_n {动作 n }

2.5 词法分析器的生成器

- 例——声明部分

```
%{  
/* 常量LT, LE, EQ, NE, GT, GE, WHILE, DO, ID,  
   NUMBER, RELOP的定义*/  
%}  
/* 正规定义 */  
delim      [ \t \n ]  
ws         {delim}+  
letter     [A -Za - z]  
digit      [0-9]  
id         {letter}({letter}|{digit})*  
number     {digit}+(\.{digit}+)?(E[+\-]?{digit}+)?
```

2.5 词法分析器的生成器

- 例——翻译规则部分

```
{ws}      { /* 没有动作, 也不返回 */ }
while     { return (WHILE); }
do        { return (DO); }
{id}      { yylval = installId ( ); return (ID); }
{number}  { yylval = installNum( ); return (NUMBER); }
“<”      { yylval = LT; return (RELOP); }
“<= ”    { yylval = LE; return (RELOP); }
“== ”    { yylval = EQ; return (RELOP); }
“<> ”    { yylval = NE; return (RELOP); }
“> ”    { yylval = GT; return (RELOP); }
“>= ”   { yylval = GE; return (RELOP); }
```

2.5 词法分析器的生成器

- 例——辅助过程部分

```
installId() {  
    /* 把词法单元装入符号表并返回指向它的指针。  
    yytext指向该词法单元的第一个字符，  
    yyleng给出它的长度          */  
}
```

```
installNum () {  
    /* 类似上面的过程，但词法单元不是标识符  
    而是数 */  
}
```

本章要点

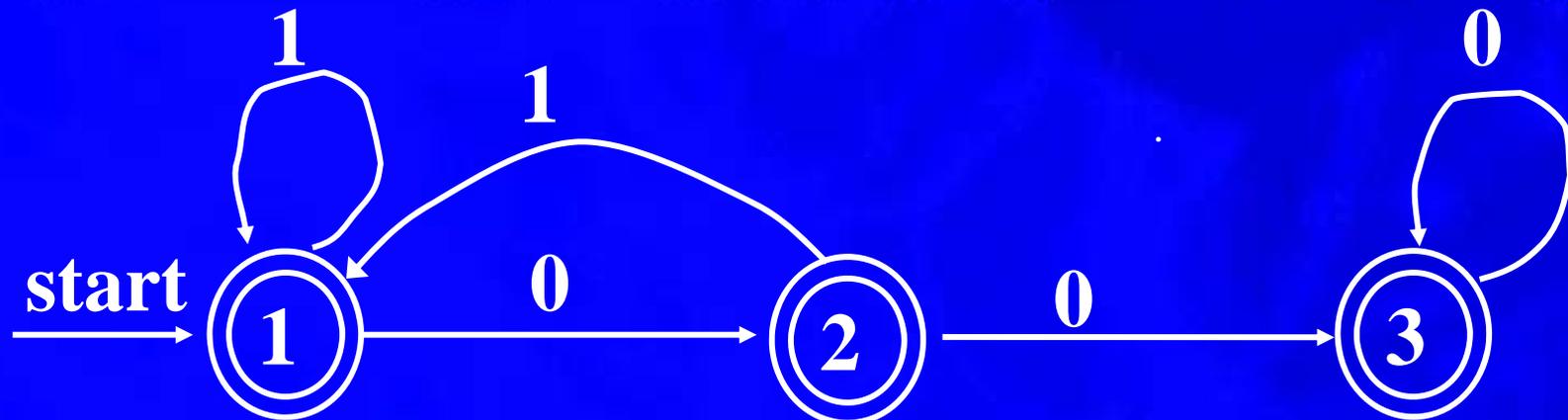
- 词法分析器的作用和接口，用高级语言编写词法分析器等内容
- 掌握下面涉及的一些概念，它们之间转换的技巧、方法或算法
 - 非形式描述的语言 \leftrightarrow 正规式
 - 正规式 \rightarrow NFA
 - 非形式描述的语言 \leftrightarrow NFA
 - NFA \rightarrow DFA
 - DFA \rightarrow 最简DFA
 - 非形式描述的语言 \leftrightarrow DFA（或最简DFA）

例题 1

- 叙述下面的正规式描述的语言，并画出接受该语言的最简DFA的状态转换图

$(1|01)^* 0^*$

- 描述的语言是：所有不含子串001的0和1的串



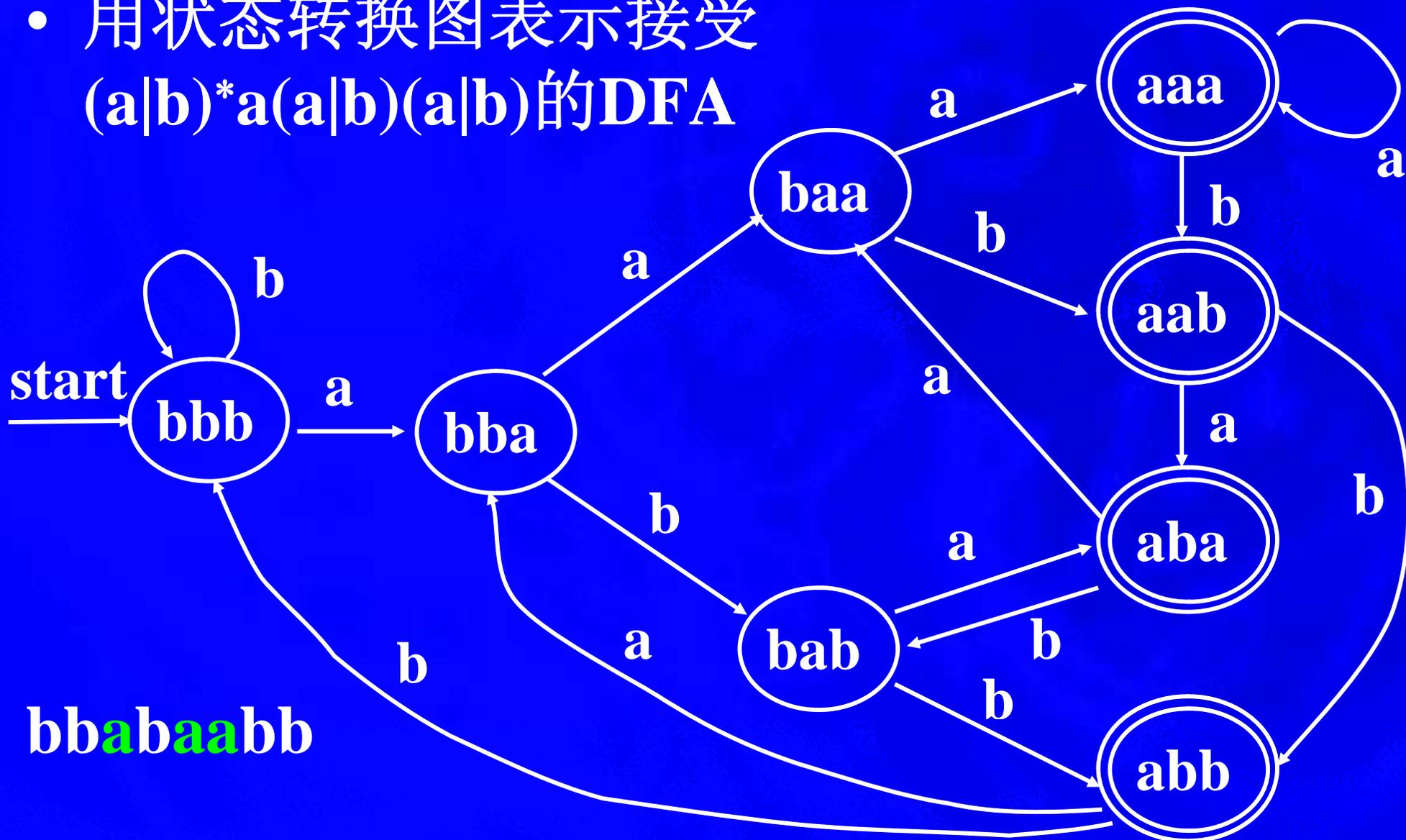
刚读过的不是0

连续读过一个0

连续读过
不少于两个0

例题 2

- 用状态转换图表示接受 $(a|b)^*a(a|b)(a|b)$ 的 DFA



例题 3

- 写出语言“所有相邻数字都不相同的非空数字串”的正规定义

123031357106798035790123

$answer \rightarrow (0 \mid no_0 \ 0) (no_0 \ 0)^* (no_0 \mid \varepsilon) \mid no_0$
 $no_0 \rightarrow (1 \mid no_0-1 \ 1) (no_0-1 \ 1)^* (no_0-1 \mid \varepsilon) \mid no_0-1$
 \dots
 $no_0-8 \rightarrow 9$

将这些正规定义逆序排列就是答案

例题 4

下面C语言编译器编译下面的函数时，报告

parse error before 'else'

```
long gcd(p,q)
```

```
long p,q;
```

```
{
```

```
    if (p%q == 0)
```

```
        /* then part */
```

```
        return q
```

此处遗漏分号

```
    else
```

```
        /* else part */
```

```
        return gcd(q, p%q);
```

```
}
```

例题 4

现在少了第一个注释的结束符号后，反而不报错了

```
long gcd(p,q)
long p,q;
{
    if (p%q == 0)
        /* then part
        return q
    else
        /* else part */
        return gcd(q, p%q);
}
```

习 题

- 第一次 2.3, 2.4 (d) (e)
- 第二次 2.7 (c) (d), 2.8 (仅为2.7 (c)),
2.9, 2.11(a)