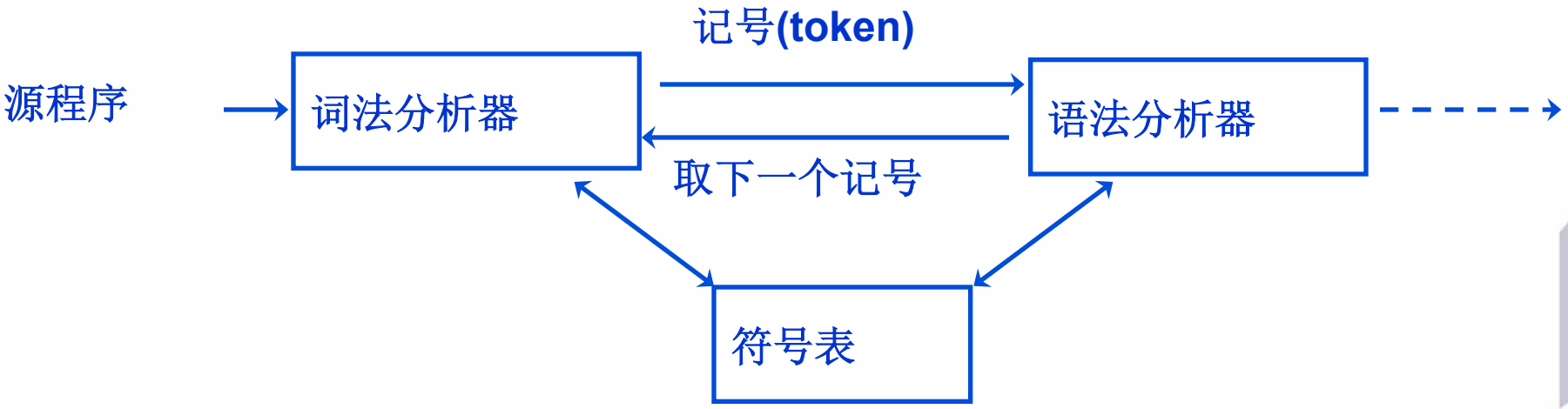
编译原理

第三章 词法分析

计算机科学与技术学院

第三章 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
- 介绍正则式、状态转换图和有限自动机概念

3.1 词法记号及属性

3.1.1 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”之间任意不含引号本身的字符串

3.1 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

    DO 8 I = 3. 75　　等同于　　**DO8I = 3. 75**

    DO 8 I = 3, 75
  - 关键字不保留

    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

3.1.2 词法记号的属性

position = initial + 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⟩

3.1.3 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“数字串.数字串”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

3.2 词法记号的描述与识别

3.2.1 串和语言

- 字母表：符号的有限集合， 例：$\sum = \{0, 1\}$
- 串：符号的有穷序列，例：0110， ε
- 语言：字母表上的一个串集

  $\{\varepsilon, 0, 00, 000, \cdots\}$， $\{\varepsilon\}$， $\varnothing$
- 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$s^{i-1}s$（$i > 0$）

3.2 词法记号的描述与识别

- 语言的运算
  - 并：　$L \cup M = \{s \mid s \in L$ 或 $s \in M\}$
  - 连接：　$LM = \{st \mid s \in L$ 且 $t \in M\}$
  - 幂：　L^0是$\{\varepsilon\}$，L^i是$L^{i-1}L$
  - 闭包：　$L^* = L^0 \cup L^1 \cup L^2 \cup \ldots$
  - 正闭包：　$L^+ = L^1 \cup L^2 \cup \ldots$
- 例

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

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

3.2.2 正则式

正则式用来表示简单的语言，叫做正则集

正则式	定义的语言	备注
ε	$\{\varepsilon\}$	
a	$\{a\}$	$a \in \sum$
$(r) \mid (s)$	$L(r) \cup L(s)$	r和s是正则式
$(r)(s)$	$L(r)L(s)$	r和s是正则式
$(r)^*$	$(L(r))^*$	r是正则式
(r)	$L(r)$	r是正则式

$((a)\ (b)^*) \mid (c)$可以写成$ab^* \mid c$

3.2 词法记号的描述与识别

- 正则式的例子　$\sum$ = {*a*, *b*}
  - ***a*** | ***b***　　{*a*, *b*}
  - (***a*** | ***b***) (***a*** | ***b***)　　{*aa*, *ab*, *ba*, *bb*}
  - ***aa*** | ***ab*** | ***ba*** | ***bb***　　{*aa*, *ab*, *ba*, *bb*}
  - a^*　　由字母*a*构成的所有串集
  - $(a \mid b)^*$　　由*a*和*b*构成的所有串集
- 复杂的例子

(00 | 11 | ((01 | 10) (00 | 11) * (01 | 10))) *

句子：01001101000010000010111001

3.2 词法记号的描述与识别

- 正则式等价：表示同样语言的正则式

公理	描述
$r \mid s = s \mid r$	|是可交换的
$r \mid (s \mid t) = (r \mid s) \mid t$	|是可结合的
$(rs)\ t = r\ (st)$	连接是可结合的
$r(s \mid t) = rs \mid rt$ $(s \mid t)\ r = sr \mid tr$	连接对|是可分配的
$\varepsilon r = r$ $r\varepsilon = r$	ε是连接的恒等元素
$r^* = (r \mid \varepsilon)^*$	$*$和 ε之间的关系
$r^{**} = r^*$	$*$是幂等的

3.2.3 正则定义

- 对正则式命名，使表示简洁

$$d_1 \rightarrow r_1$$
$$d_2 \rightarrow r_2$$
$$\ldots$$
$$d_n \rightarrow r_n$$

- 各个d_i的名字都不同
- 每个r_i都是$\sum \cup \{d_1, d_2, \ldots, d_{i-1}\}$上的正则式

3.2 词法记号的描述与识别

- 正则定义的例子
  - C语言的标识符是字母、数字和下划线组成的串

letter_ → *A* | *B* | … | *Z* | *a* | *b* | … | *z* | _

digit → 0 | 1 | … | 9

id → letter_(letter_ | digit)*

3.2 词法记号的描述与识别

- 正则定义的例子
  - 无符号数集合，例1946, 11.28, 63**E8**, 1.99**E-6**

digit → 0 | 1 | … | 9

digits → digit digit*

optional_fraction → . digits | ε

optional_exponent → (E (+ | - | ε) digits) | ε

number→digits optional_fraction optional_exponent

  - 简化表示

number → digit+ (.digit+)? (E(+|-)? digit+)?

3.2 词法记号的描述与识别

- 正则定义的例子（进行下一步讨论的例子）

while → while

do → do

relop → **< | < = | = | < > | > | > =**

letter → ***A* | *B* | … | *Z* | *a* | *b* | … | *z***

id → **letter (letter | digit)***

number → **digit$^+$ (.digit$^+$)? (E (+ | -)? digit$^+$)?**

delim → **blank | tab | newline**

ws → **delim$^+$**

3.2 词法记号的描述与识别

- 词法单元的识别

stmt → **if** *expr* **then** *stmt*
| **if** *expr* **then** *stmt* **else** *stmt*
| ε
expr → *term* **relop** *term*
| *term*
term → **id**
| **number**

3.2 词法记号的描述与识别

- 词法单元的识别

stmt → **if** *expr* **then** *stmt*
| **if** *expr* **then** *stmt* **else** *stmt*
| ε
expr → *term* **relop** *term*
| *term*
term → **id**
| **number**

digit → [0-9]
digits → *digit*+
number → *digits*(.*digits*)?(E[+-]?*digits*)?
letter → [A-Za-z]
id → *letter* (*letter* | *digit*)*
if → if
then → then
else → else
relop → < | > | <= | >= | = | <>

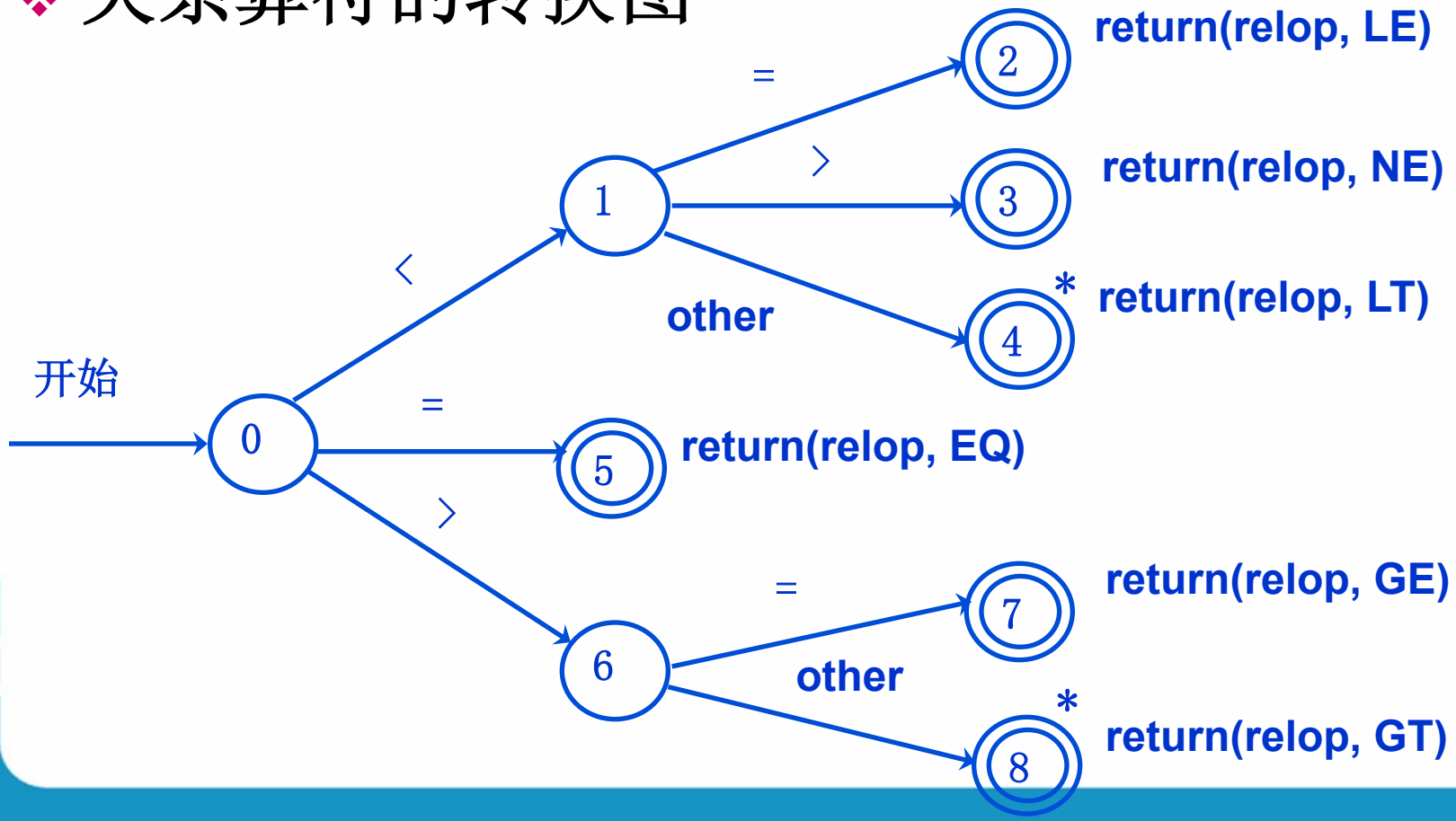
3.2 词法记号的描述与识别

- 词法单元的识别

词素	词法单元名字	属性值
Any *ws*	-	-
if	**if**	-
Then	**then**	-
else	**else**	-
Any *id*	**id**	指向符号表条目的指针
Any *number*	**number**	指向符号表条目的指针
<	**relop**	LT
<=	**relop**	LE
=	**relop**	EQ
<>	**relop**	NE
>	**relop**	GT
>=	**relop**	GE

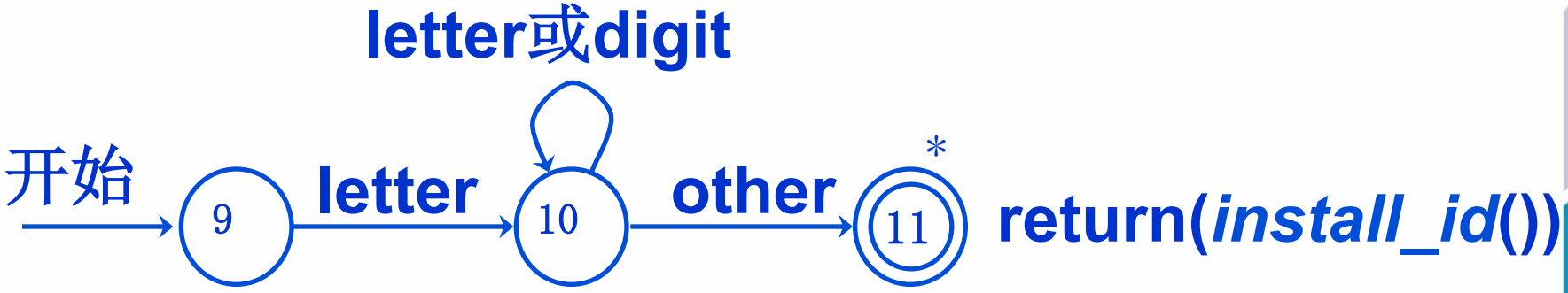
3.2.4 转换图

- 关系算符的转换图



3.2 词法记号的描述与识别

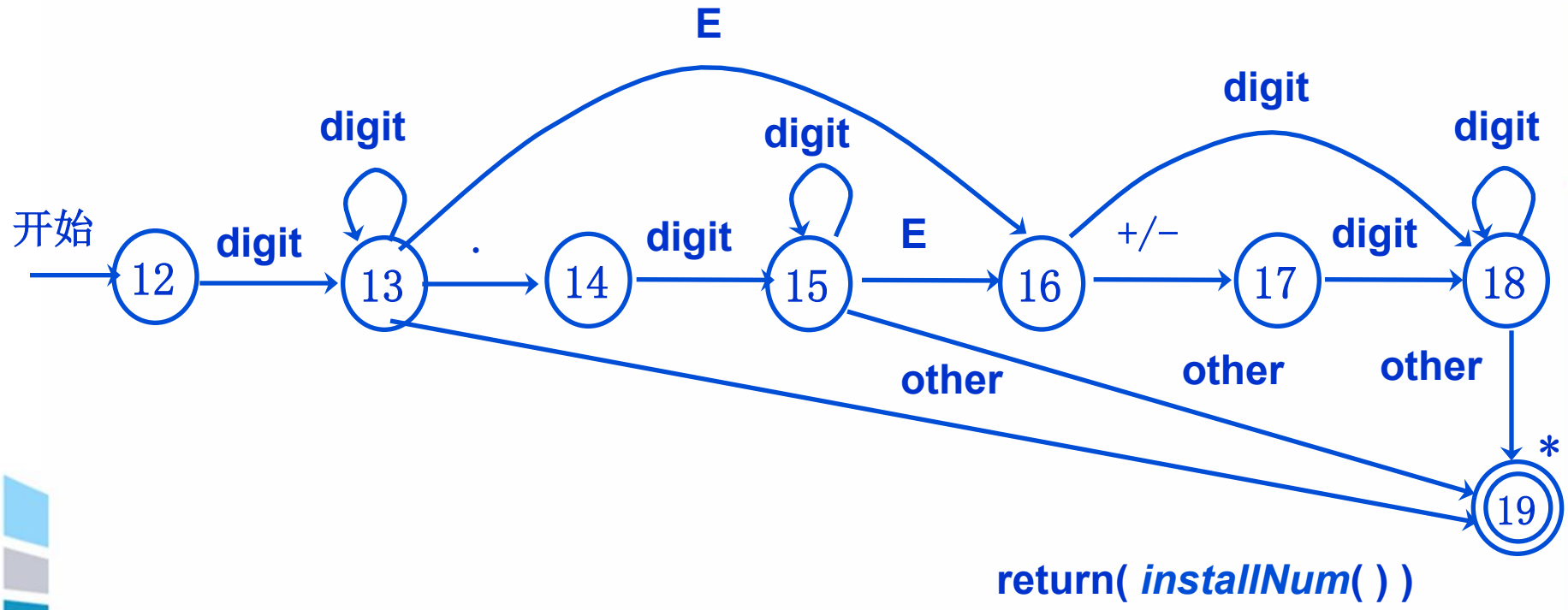
- 标识符和关键字的转换图



3.2 词法记号的描述与识别

- 无符号数的转换图

number → digit+ (.digit+)? (E (+ | -)? digit+)?

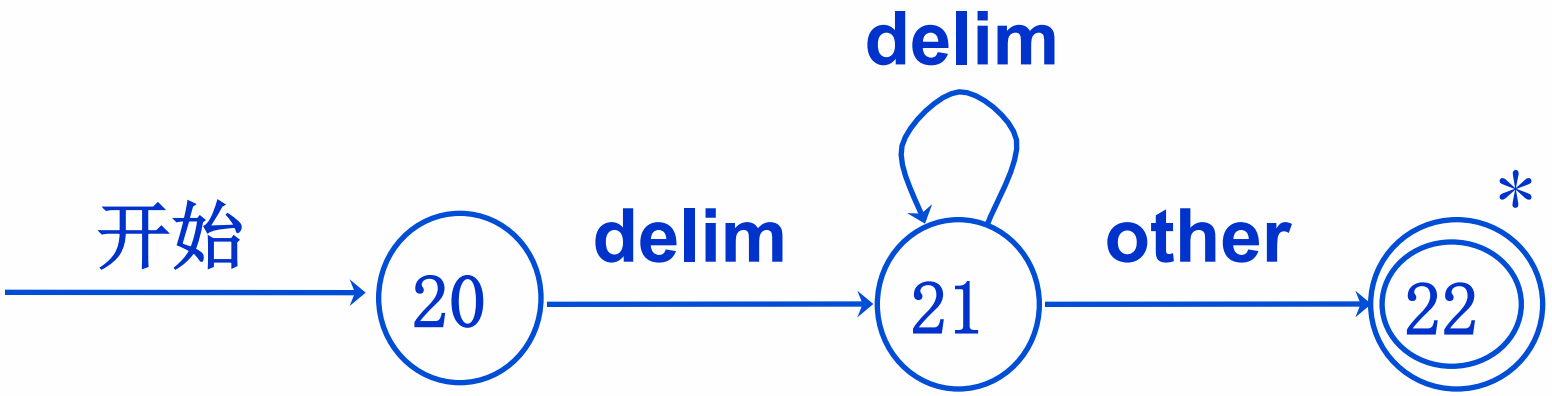


3.2 词法记号的描述与识别

- 空白的转换图

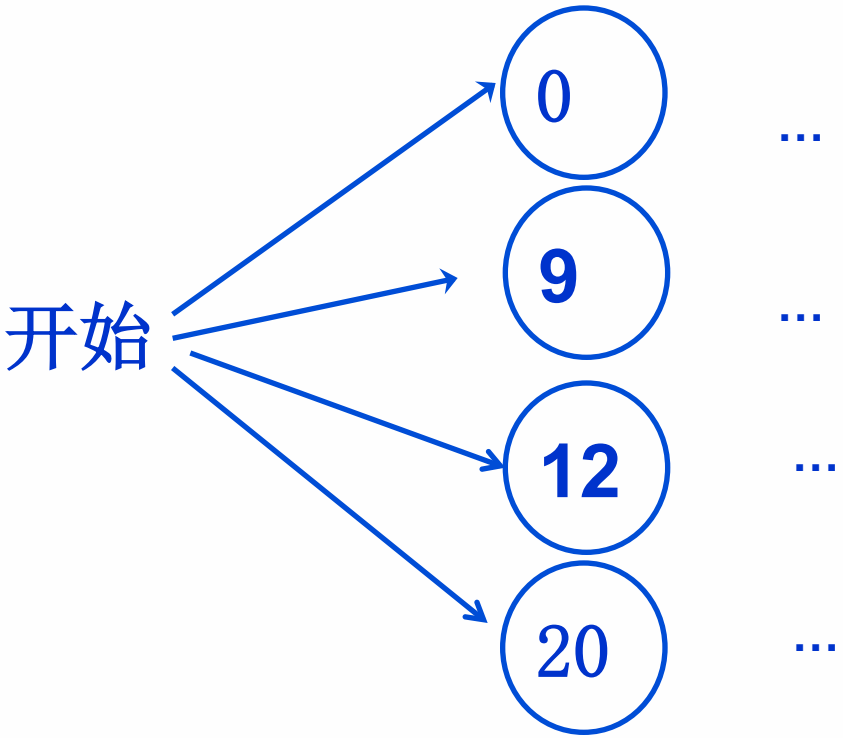
  delim → blank | tab | newline

  ws → delim+



3.2 词法记号的描述与识别

- 合成整体转换图

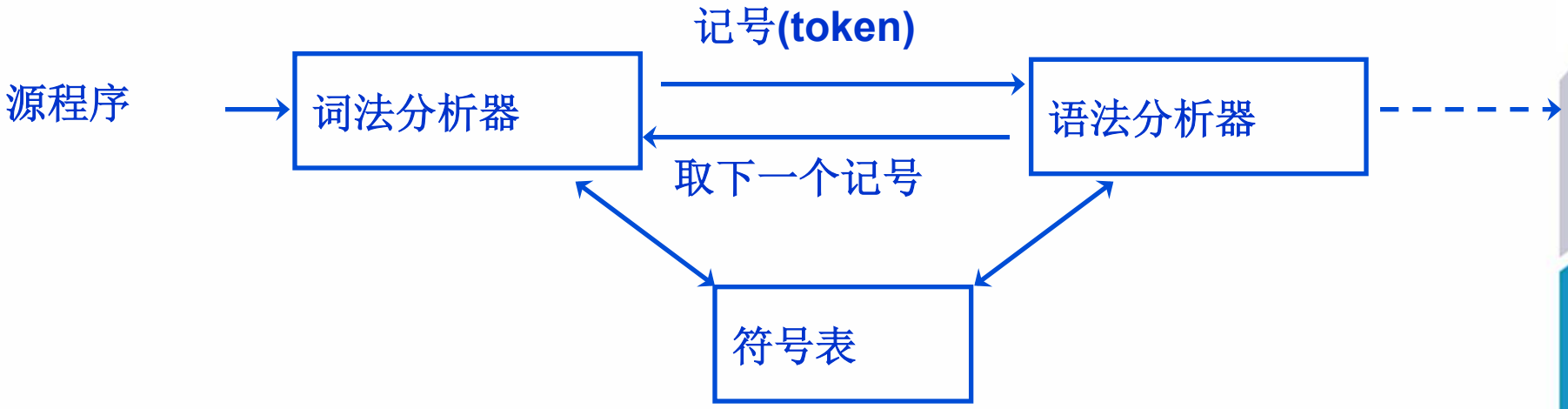


编译原理

第三章 词法分析 上次课回顾

计算机科学与技术学院

第三章 词法分析 上次课回顾



第三章 词法分析 上次课回顾

- 词法记号的属性

position = initial + 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⟩

第三章 词法分析 上次课回顾

- 正则式

  正则式用来表示简单的语言，叫做正则集

正则式	定义的语言	备注
ε	{ε}	
a	{***a***}	$a \in \sum$
(***r***) | (***s***)	$L(r) \cup L(s)$	***r***和***s***是正则式
(***r***)(***s***)	$L(r)L(s)$	***r***和***s***是正则式
$(r)^*$	$(L(r))^*$	***r***是正则式
(***r***)	$L(r)$	***r***是正则式

((***a***) $(b)^*$)| (***c***)可以写成ab^*| ***c***

第三章 词法分析 上次课回顾

- 正则定义
  - 对正则式命名，使表示简洁

$$d_1 \rightarrow r_1$$
$$d_2 \rightarrow r_2$$
$$\ldots$$
$$d_n \rightarrow r_n$$

  - 各个d_i的名字都不同
  - 每个r_i都是$\sum \cup \{d_1, d_2, \ldots, d_{i-1}\}$上的正则式

第三章 词法分析 上次课回顾

- 正则定义的例子

while → while

do → do

relop → **< | < = | = | < > | > | > =**

letter → ***A*** **|** ***B*** **| … |** ***Z*** **|** ***a*** **|** ***b*** **| … |** ***z***

id → **letter (letter | digit)***

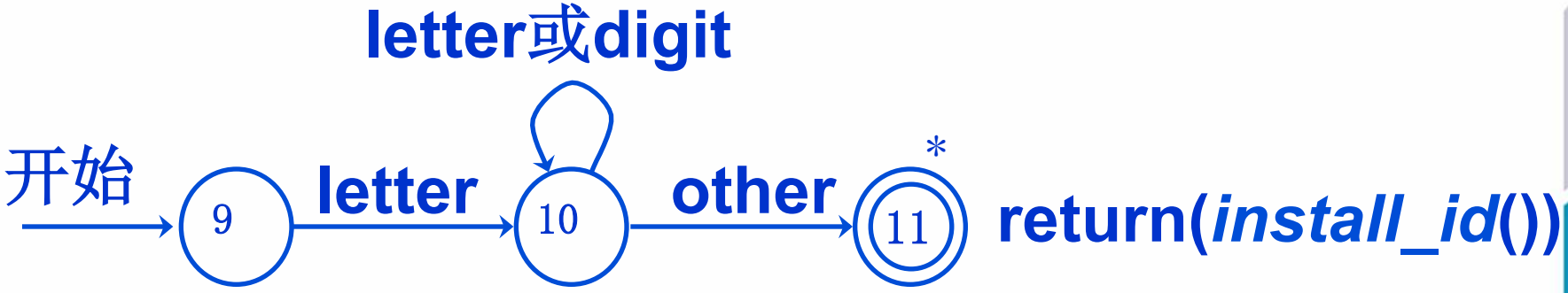
number → **digit**$^+$ **(.digit**$^+$**)? (E (+ | -)? digit**$^+$**)?**

delim → **blank | tab | newline**

ws → **delim**$^+$

第三章 词法分析 上次课回顾

- 标识符和关键字的转换图



3.3 有限自动机

3.3.1 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

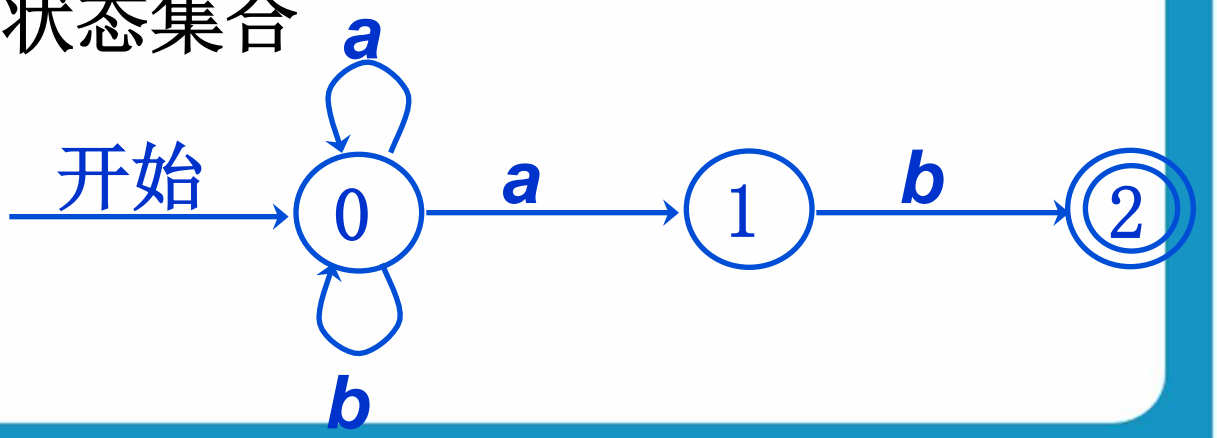
2、输入符号集合$\sum$

3、转换函数$move: S \times (\sum \cup \{\varepsilon\}) \rightarrow P(S)$

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

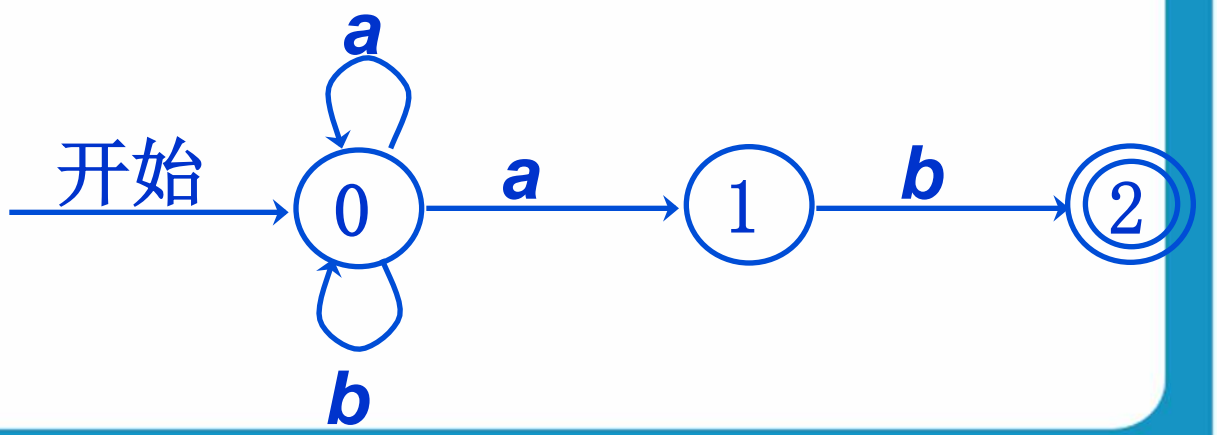


3.3 有限自动机

- NFA的转换表

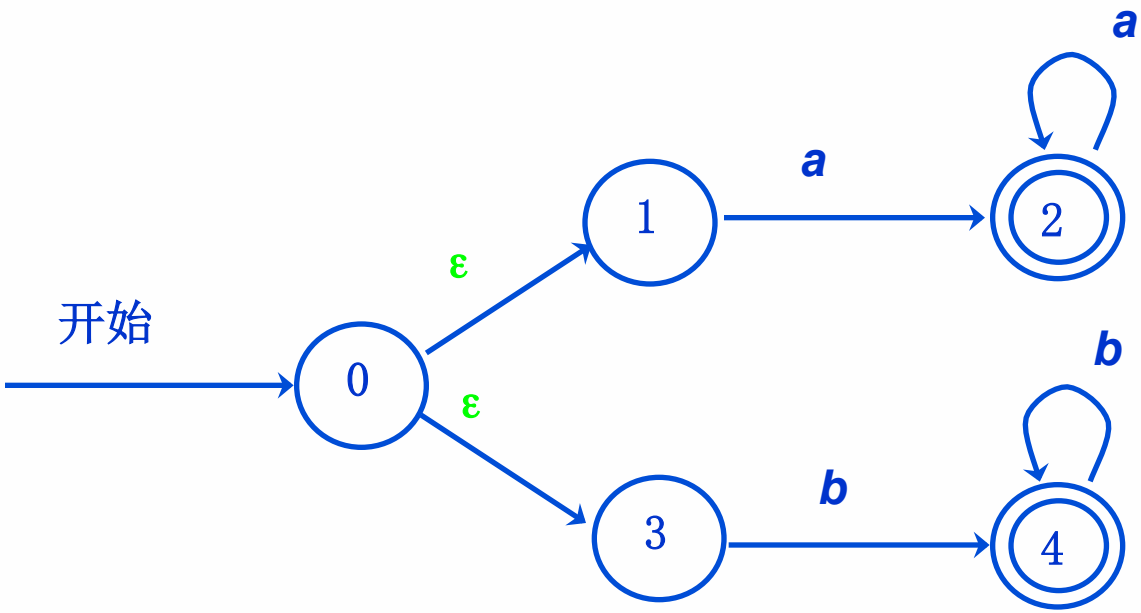
状　态	输　入　符　号	
	a	*b*
0	{0, 1}	{0}
1	∅	{2}
2	∅	∅

识别语言 $(a|b)^*ab$ 的NFA



3.3 有限自动机

- 例 识别*aa**|*bb**的NFA



3.3.2 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

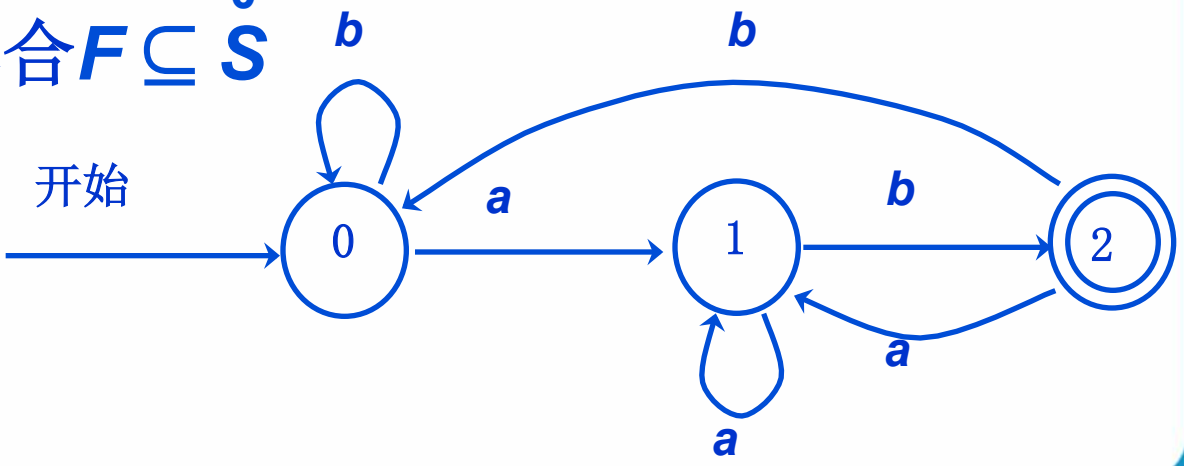
2、输入字母集合Σ

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



3.3 有限自动机

- 例 模拟DFA
  - 输入：输入串*x*，以文件结束符eof结尾。一个DFA *D*，其开始状态S_0, 结束状态集合是F。
  - 输出：如果*D*接受*x*，则回答*yes*，否则回答*no*

3.3 有限自动机

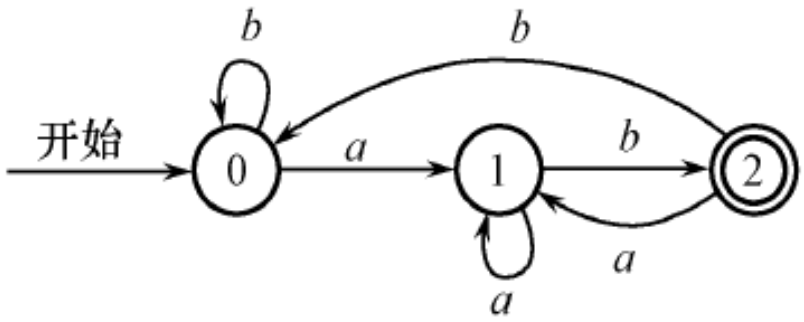
例 模拟DFA

- 输入：输入串*x*，以文件结束符eof结尾。一个DFA *D*，其开始状态S_0, 结束状态集合是F。
- 输出：如果*D*接受*x*，则回答*yes*，否则回答*no*
- 方法：
  - *move*(*s*,*c*) 状态转移
  - *nextchar*()返回输入串*x*的下一个字符

```
s := s0;
c := nextchar();
while c≠ eof do
    s := move (s, c);
    c := nextchar();
end;
if s属于 F then
    return yes
else return no ;
```

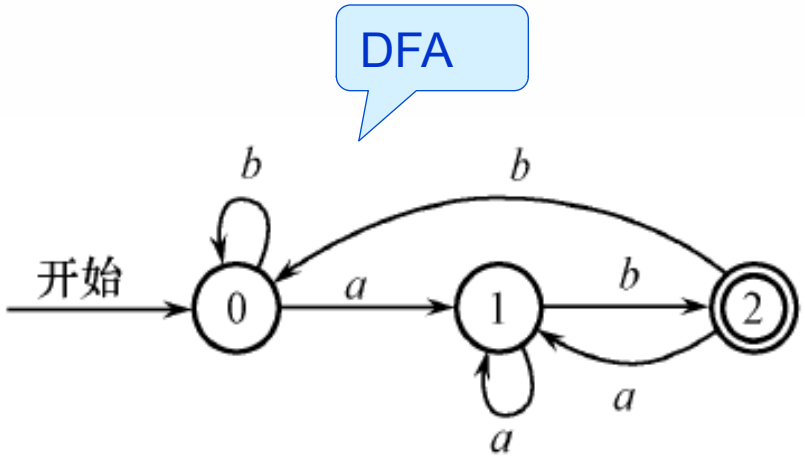
3.3 有限自动机

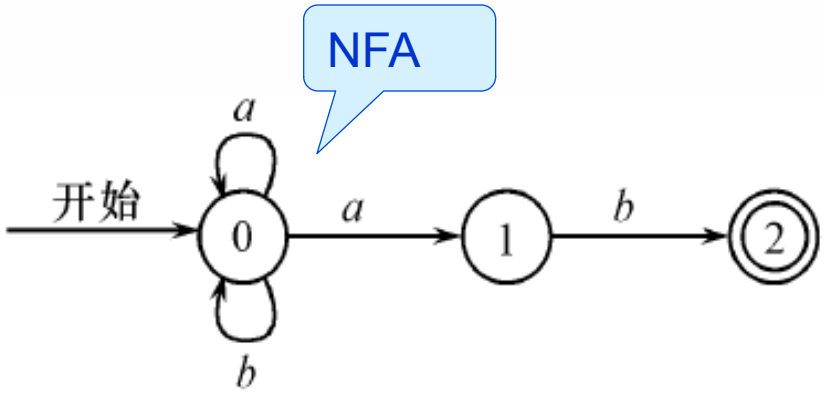
- 例 识别 $(a \mid b)^* a b$ 的DFA



3.3 有限自动机

- 例 识别 $(a \mid b)^* a b$ 的DFA





3.3 有限自动机

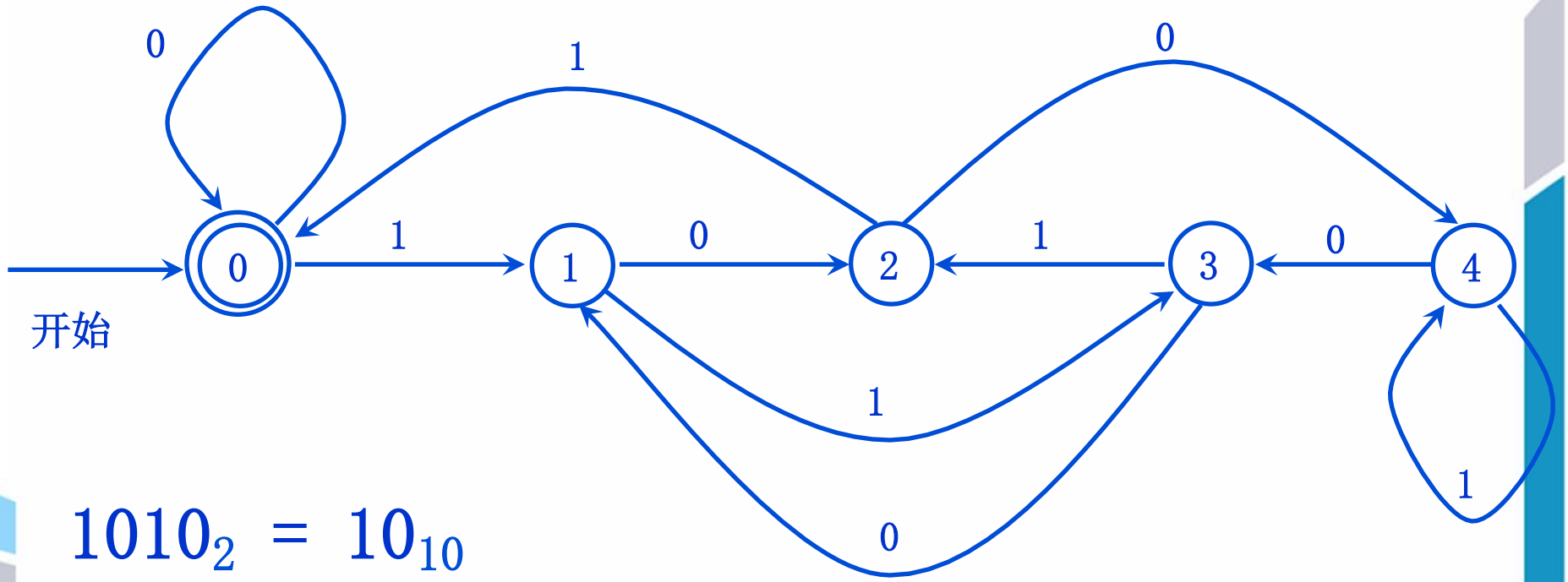
- 例 DFA，识别{0, 1}上能被5整除的二进制数

	已读过	尚未读	已读部分的值
某时刻	101	0111000	5
读进0	1010	111000	5 × 2 = 10
读进1	10101	11000	10 × 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

3.3 有限自动机

- 例 DFA，识别{0, 1}上能被5整除的二进制数



$1010_2 = 10_{10}$

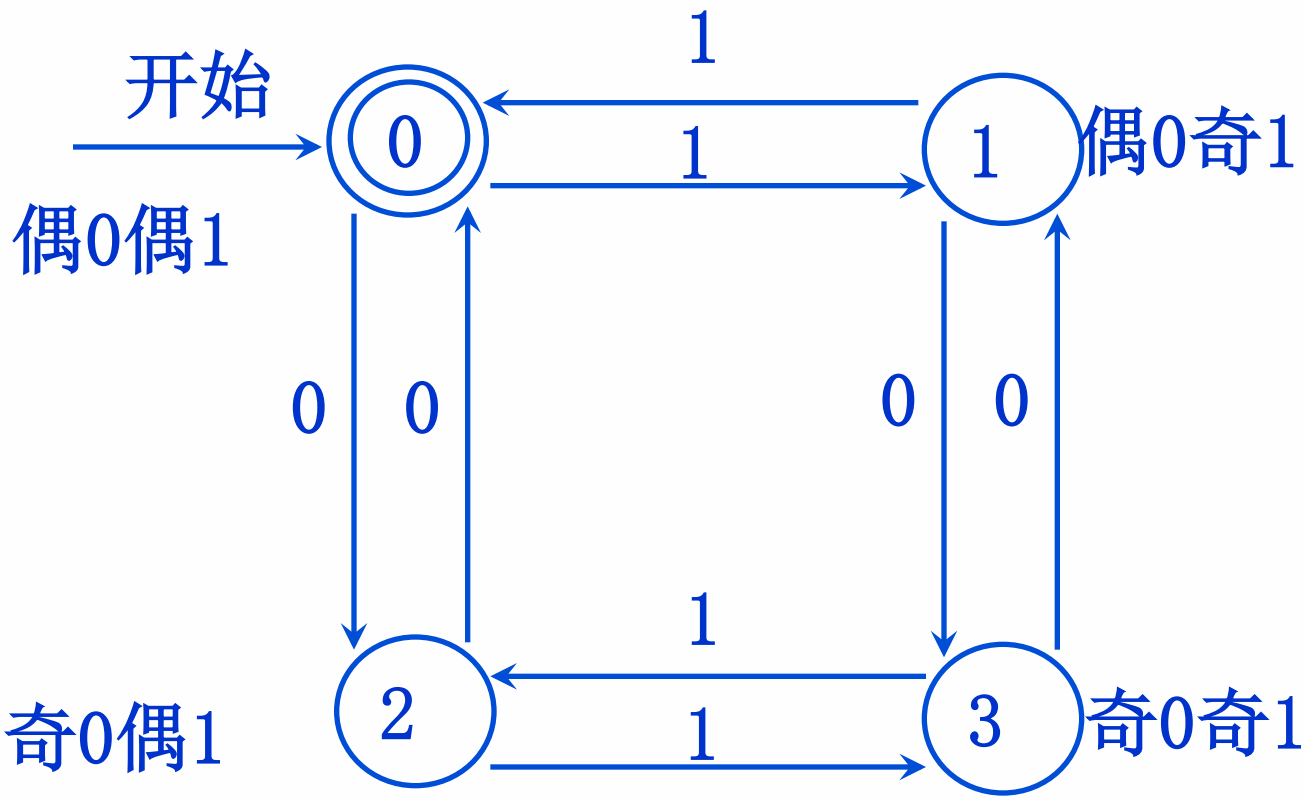
$111_2 = 7_{10}$

3.3 有限自动机

- 例 DFA, 接受 0和1的个数都是偶数的字符串

3.3 有限自动机

- 例 DFA, 接受 0和1的个数都是偶数的字符串



3.3.3 NFA到DFA的变换

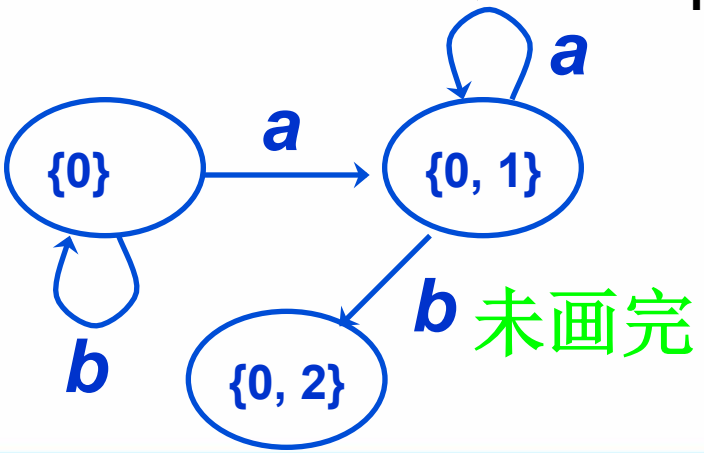
子集构造法

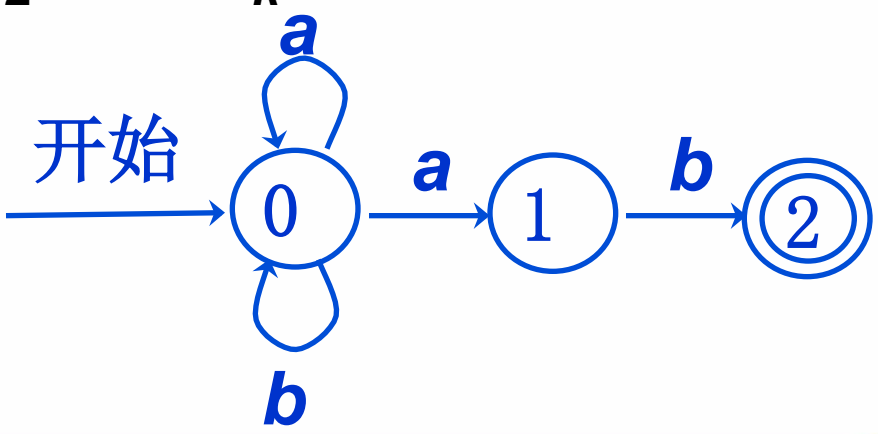
1、DFA的一个状态是NFA的一个状态集合

2、读了输入$a_1\ a_2\ \ldots\ a_n$后，

NFA能到达的所有状态：$s_1, s_2, \ldots, s_k$，则

DFA到达状态$\{s_1, s_2, \ldots, s_k\}$





3.3.3 NFA到DFA的变换

子集构造法

运算	描述
$\varepsilon-closure(s)$	从NFA的状态 s 出发，只用 ε 转换能到达的NFA状态集合
$\varepsilon-closure(T)$	NFA的状态集合 $\{s \mid s \in \varepsilon-closure(t)\ \&t \in T\}$
$move(T, a)$	NFA的状态集合 $\{s \mid s = move(t, a)\ \&t \in T\}$

3.3.3 NFA到DFA的变换

子集构造法： *ε-closure(T)*的计算

```
把 T 的所有状态压入栈；
ε— closure (T) 的初值置为 T;
while 栈非空 do begin
    把栈顶元素 t 弹出栈;
    for 每个状态 u (条件是从 t 到 u 的边上的标记为 ε) do
        if u 不在 ε— closure (T) 中 do begin
            把 u 加入 ε — closure (T) ;
            把 u 压入栈
        end
end
```

3.3.3 NFA到DFA的变换

子集构造法：状态转换表的构造

初始，ε－*closure*(*s*)是 Dstates仅有的状态，并且尚未标记；

while *Dstates*有尚未标记的状态 *T* do begin

　　标记 *T*;

　　for 每个输入符号 *a* do begin

　　　　U := ε－*closure*(*move*(*T*, *a*));

　　　　if *U*不在 *Dstates*中 then

　　　　　　把 U作为尚未标记的状态加入 Dstates;

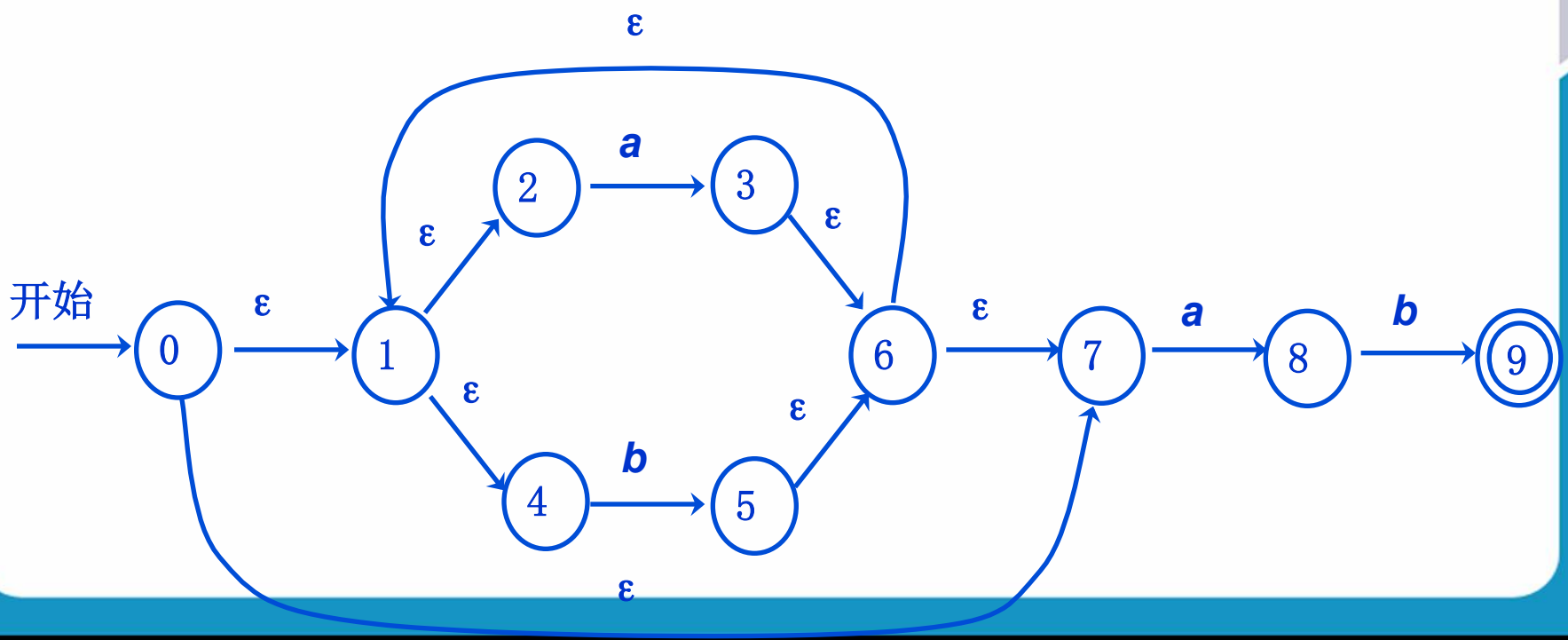
　　　　Dtran[*T*, *a*] := *U*

　　end

end

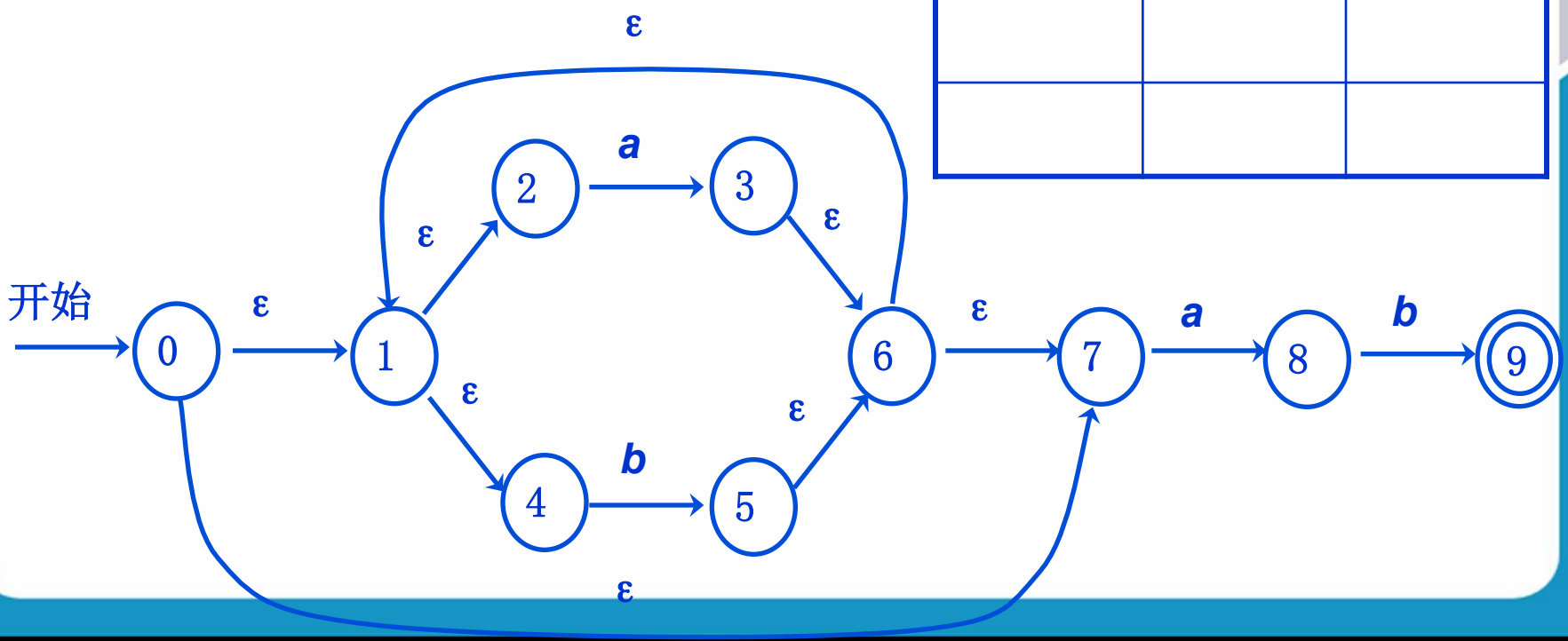
3.3 有限自动机

- 例　　(*a*|*b*)**ab*，NFA如下，把它变换为DFA



3.3 有限自动机

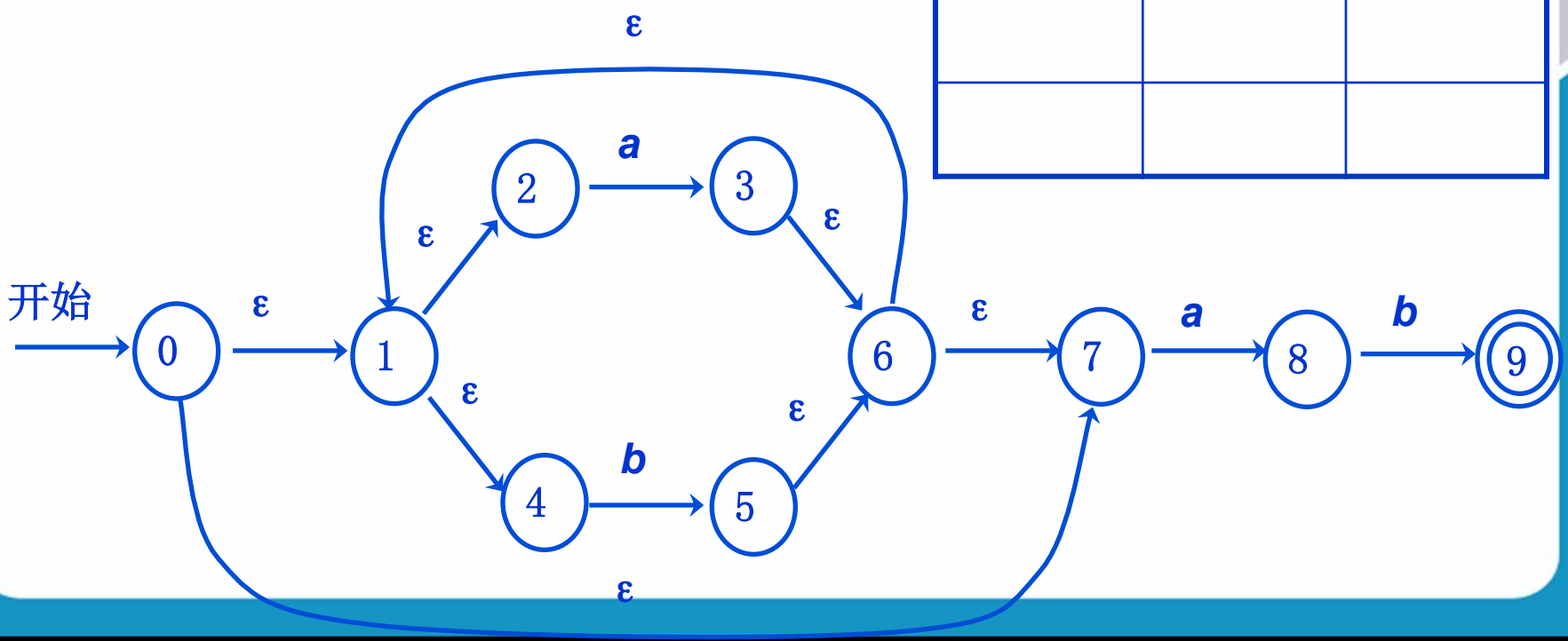
状态	输入符号	
	a	*b*



3.3 有限自动机

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

状态	输入符号	
	a	*b*
A		

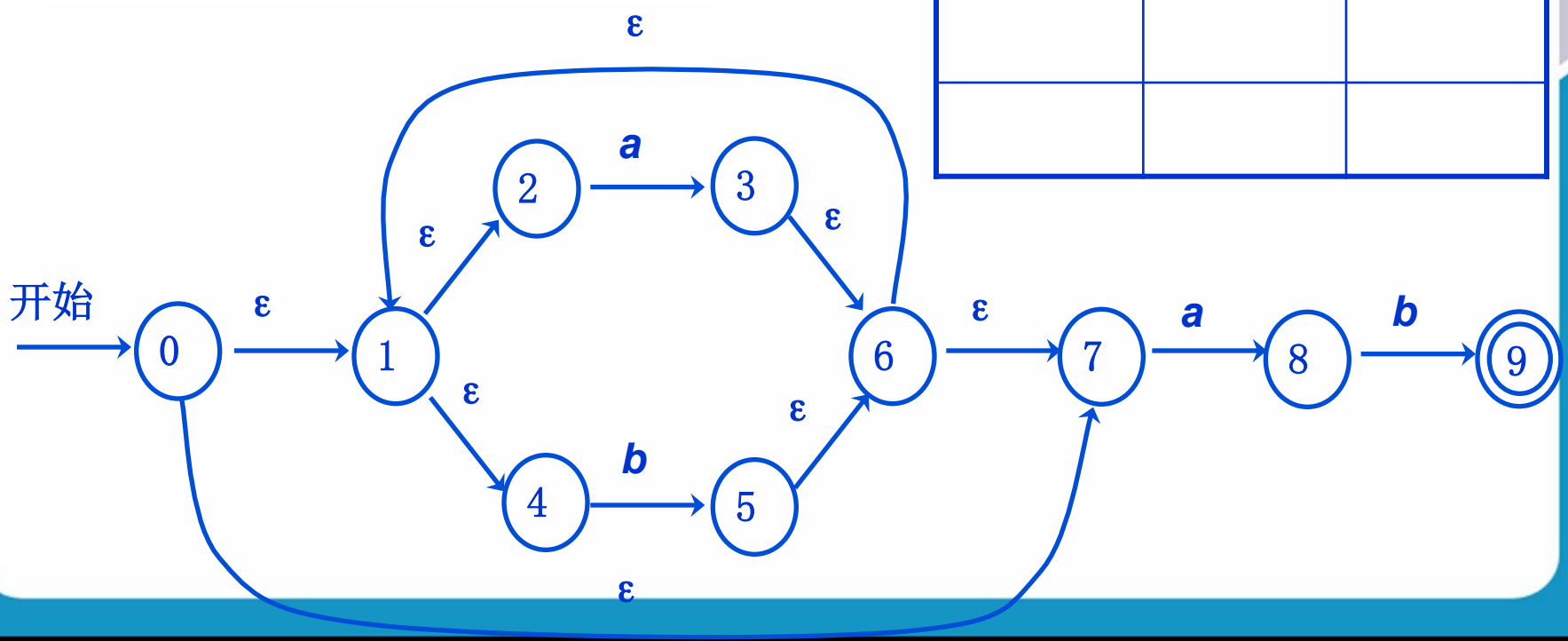


3.3 有限自动机

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

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

状态	输入符号	
	a	*b*
A	*B*	

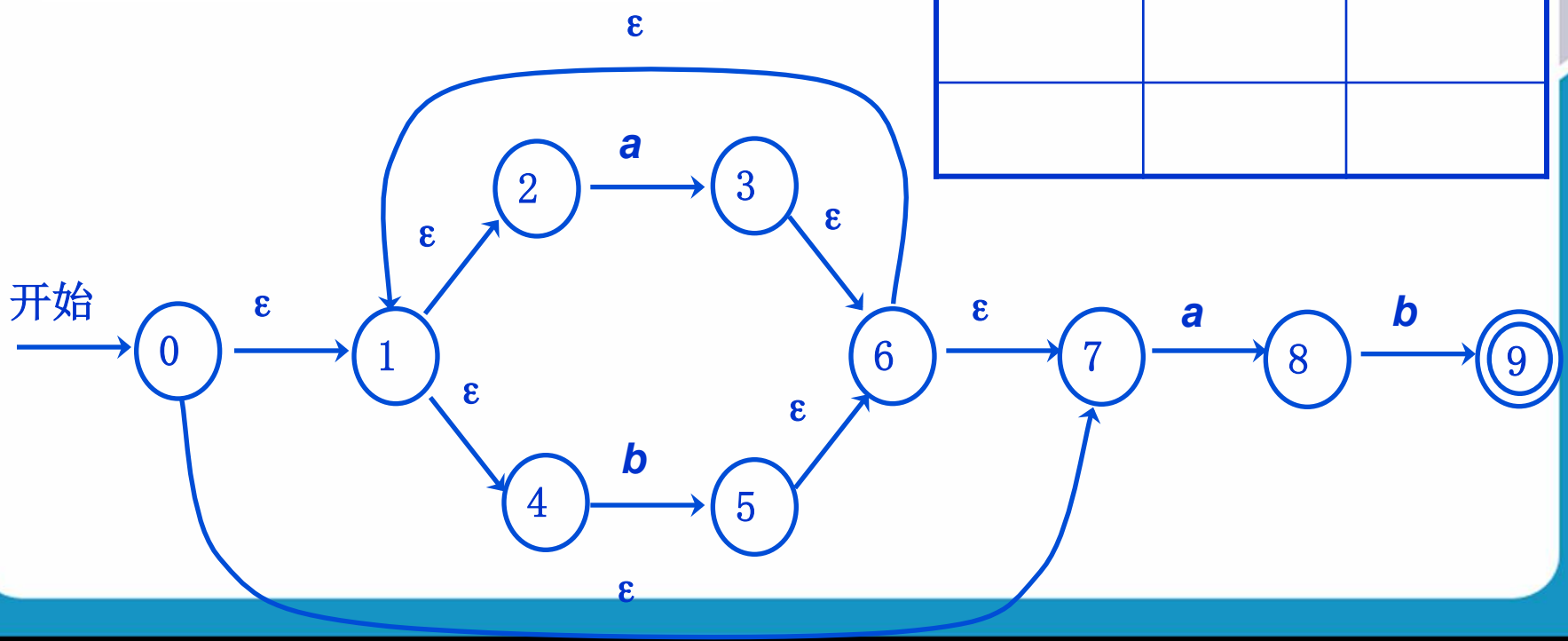


3.3 有 限 自 动 机

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

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

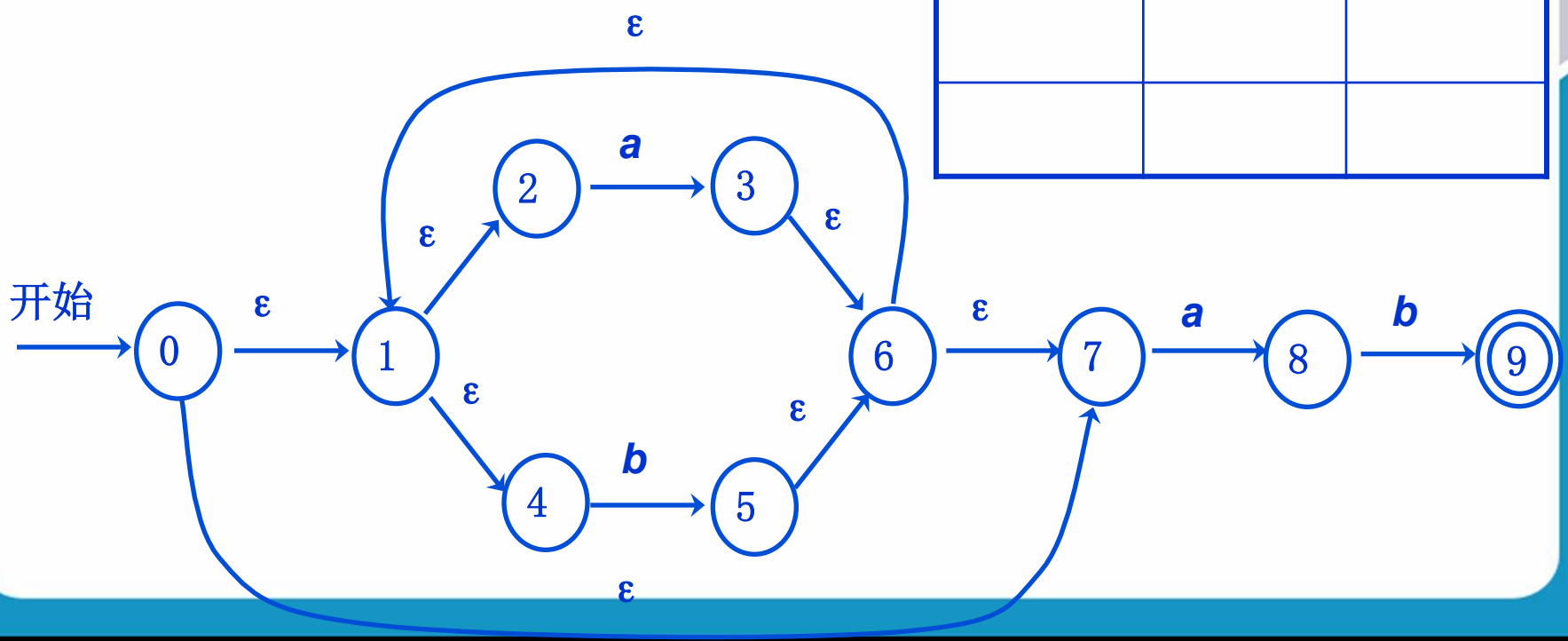
状态	输入符号	
	a	*b*
A	*B*	
B		



3.3 有 限 自 动 机

$A = \{0, 1, 2, 4, 7\}$
$B = \{1, 2, 3, 4, 6, 7, 8\}$
$C = \{1, 2, 4, 5, 6, 7\}$

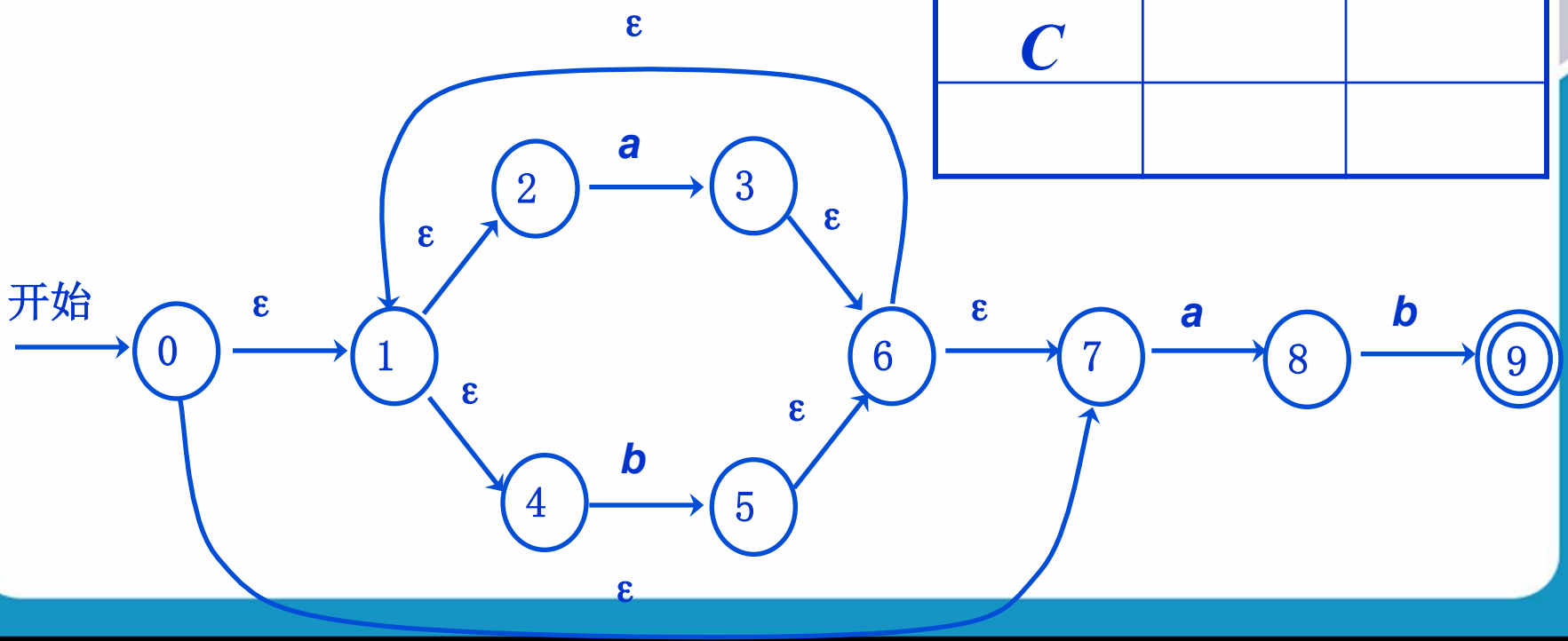
状态	输入符号	
	a	*b*
A	*B*	*C*
B		



3.3 有 限 自 动 机

$A = \{0, 1, 2, 4, 7\}$
$B = \{1, 2, 3, 4, 6, 7, 8\}$
$C = \{1, 2, 4, 5, 6, 7\}$

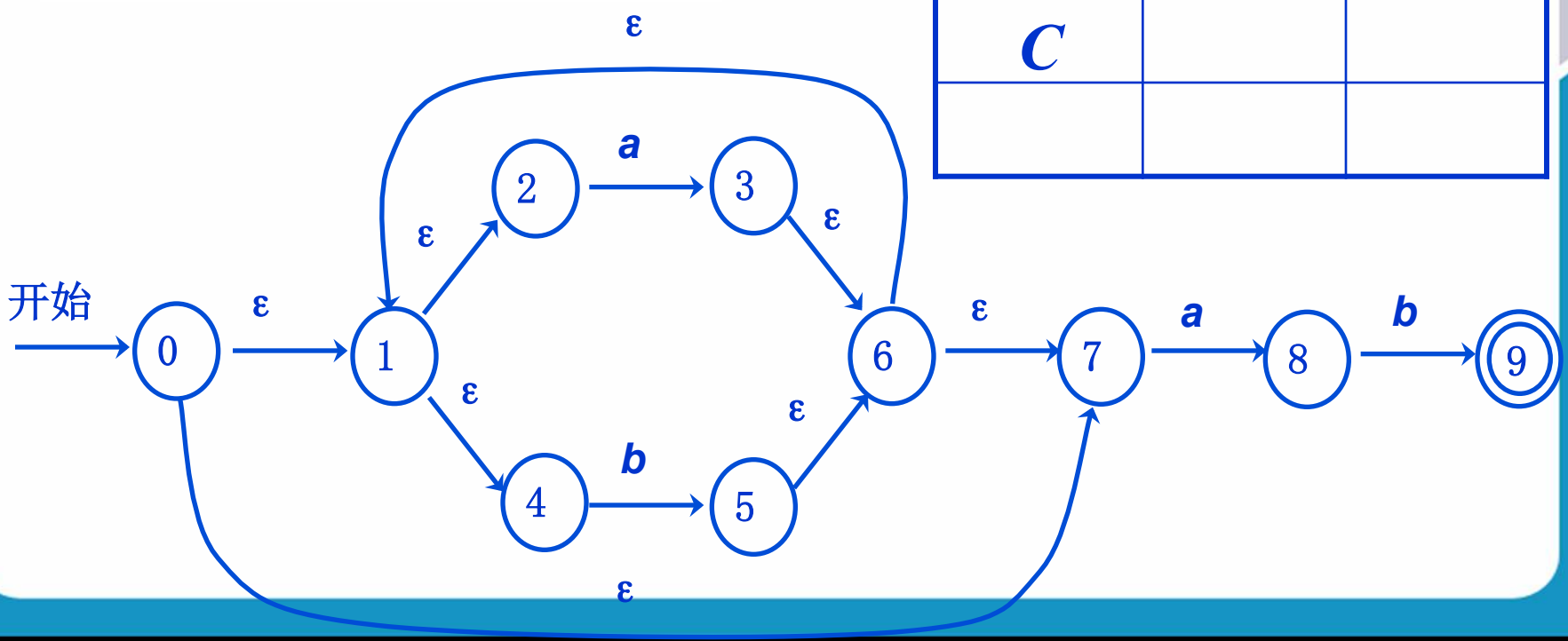
状态	输入符号	
	a	*b*
A	*B*	*C*
B		
C		



3.3 有限自动机

$A = \{0, 1, 2, 4, 7\}$
$B = \{1, 2, 3, 4, 6, 7, 8\}$
$C = \{1, 2, 4, 5, 6, 7\}$

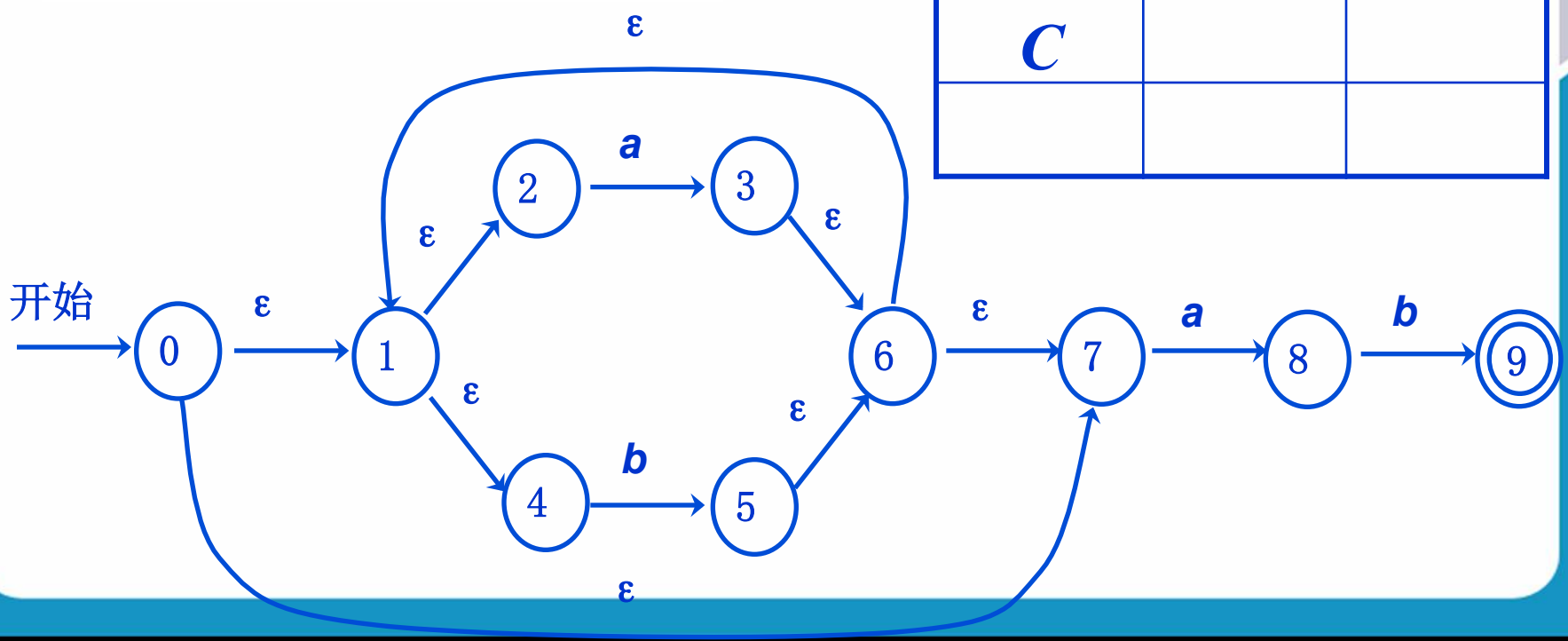
状态	输入符号	
	a	*b*
A	*B*	*C*
B	*B*	
C		



3.3 有限自动机

$A = \{0, 1, 2, 4, 7\}$
$B = \{1, 2, 3, 4, 6, 7, 8\}$
$C = \{1, 2, 4, 5, 6, 7\}$
$D = \{1, 2, 4, 5, 6, 7, 9\}$

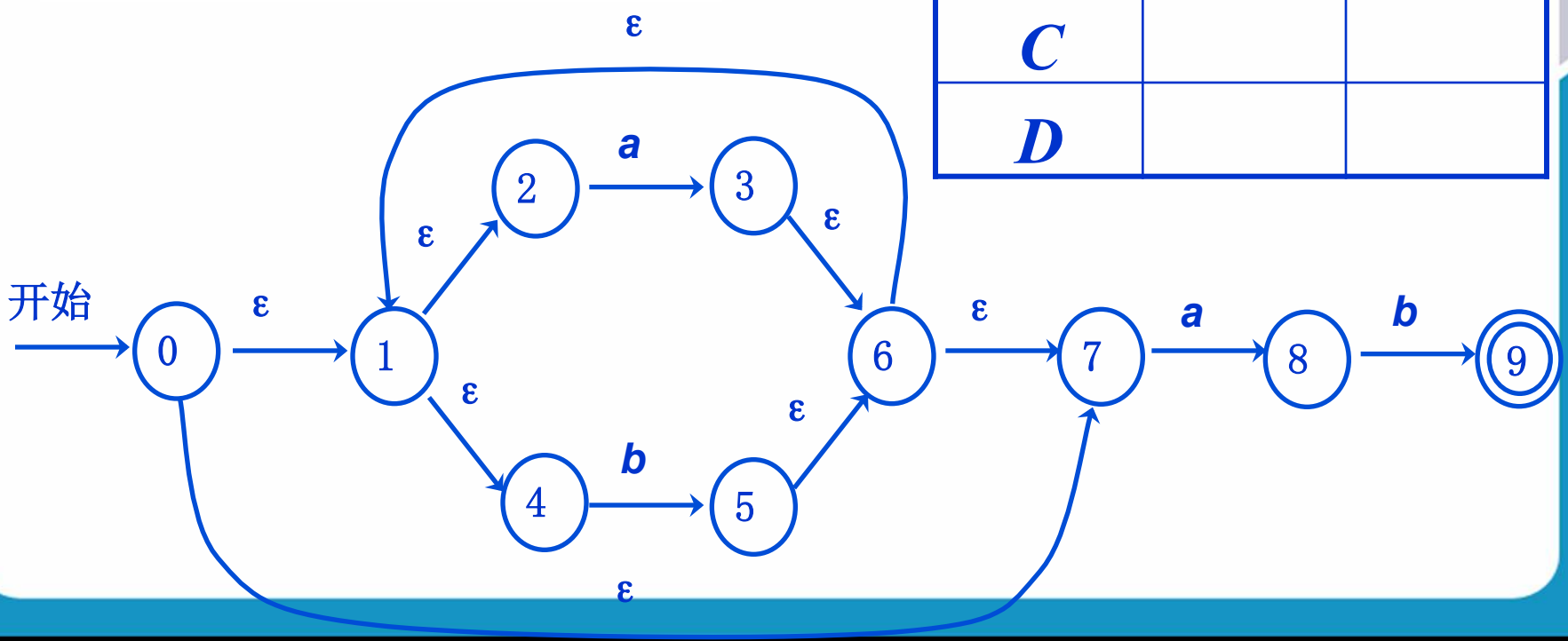
状态	输入符号	
	a	*b*
A	*B*	*C*
B	*B*	*D*
C		



3.3 有限自动机

$A = \{0, 1, 2, 4, 7\}$
$B = \{1, 2, 3, 4, 6, 7, 8\}$
$C = \{1, 2, 4, 5, 6, 7\}$
$D = \{1, 2, 4, 5, 6, 7, 9\}$

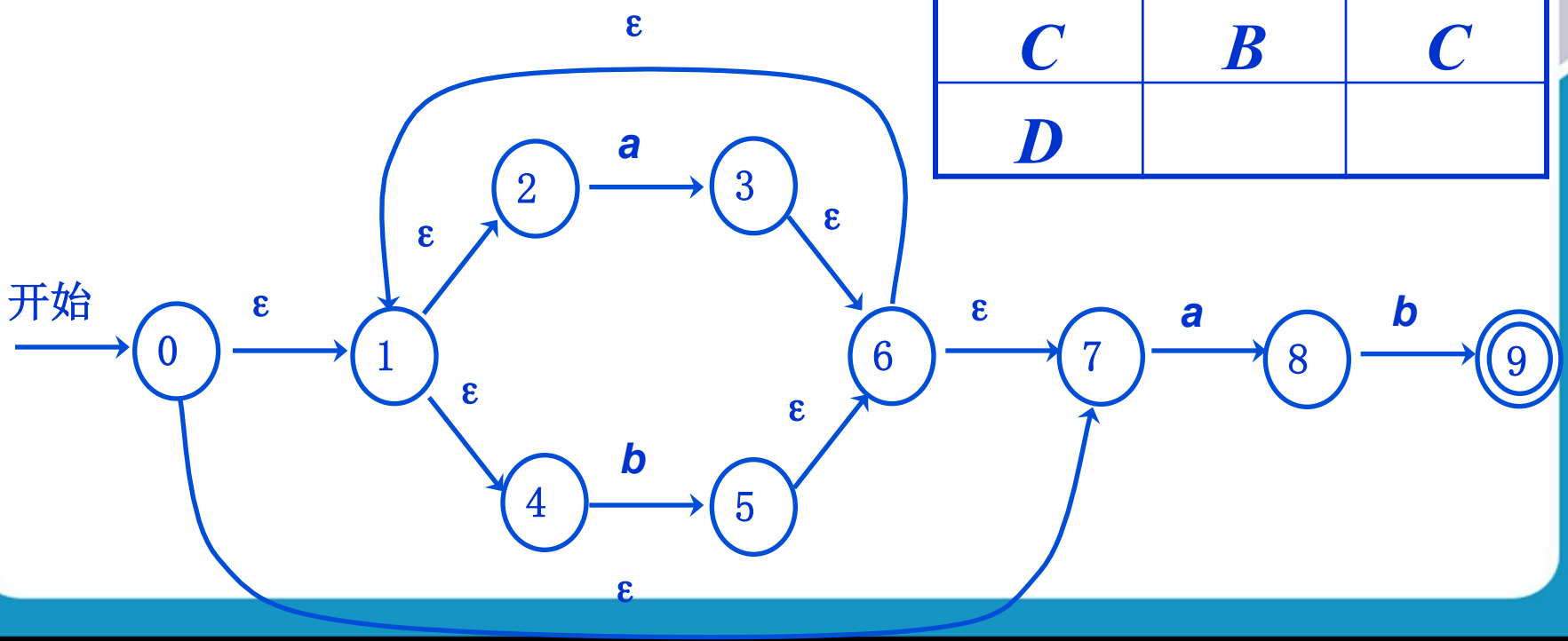
状态	输入符号	
	a	*b*
A	*B*	*C*
B	*B*	*D*
C		
D		



3.3 有 限 自 动 机

$A = \{0, 1, 2, 4, 7\}$
$B = \{1, 2, 3, 4, 6, 7, 8\}$
$C = \{1, 2, 4, 5, 6, 7\}$
$D = \{1, 2, 4, 5, 6, 7, 9\}$

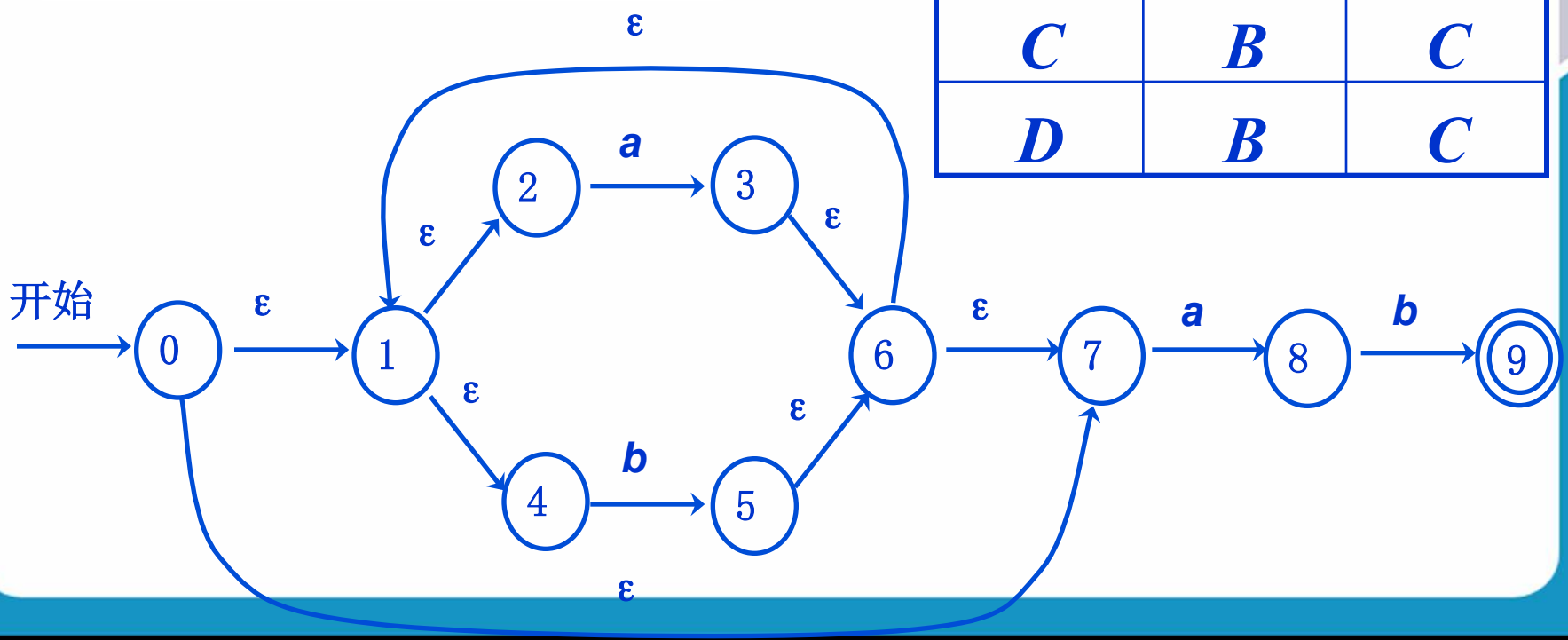
状态	输入符号	
	a	*b*
A	*B*	*C*
B	*B*	*D*
C	*B*	*C*
D		



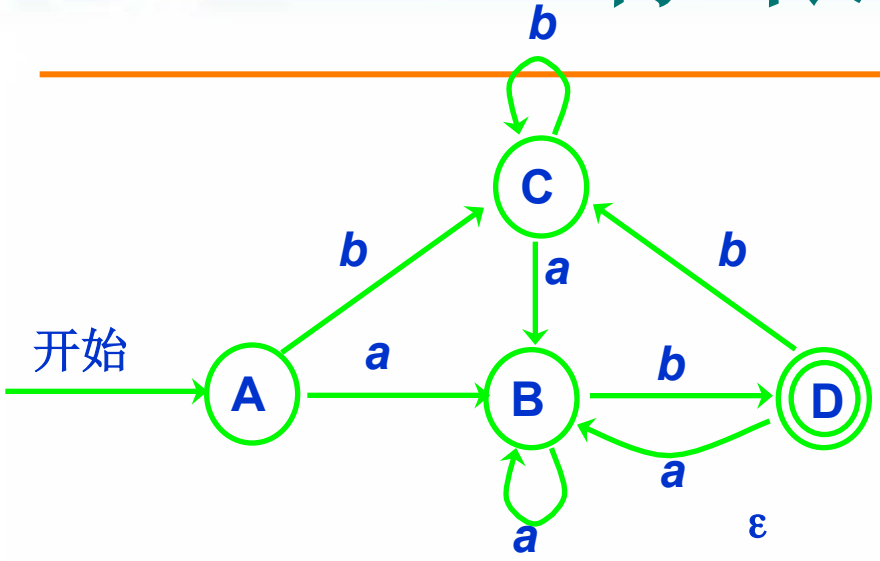
3.3 有限自动机

$A = \{0, 1, 2, 4, 7\}$
$B = \{1, 2, 3, 4, 6, 7, 8\}$
$C = \{1, 2, 4, 5, 6, 7\}$
$D = \{1, 2, 4, 5, 6, 7, 9\}$

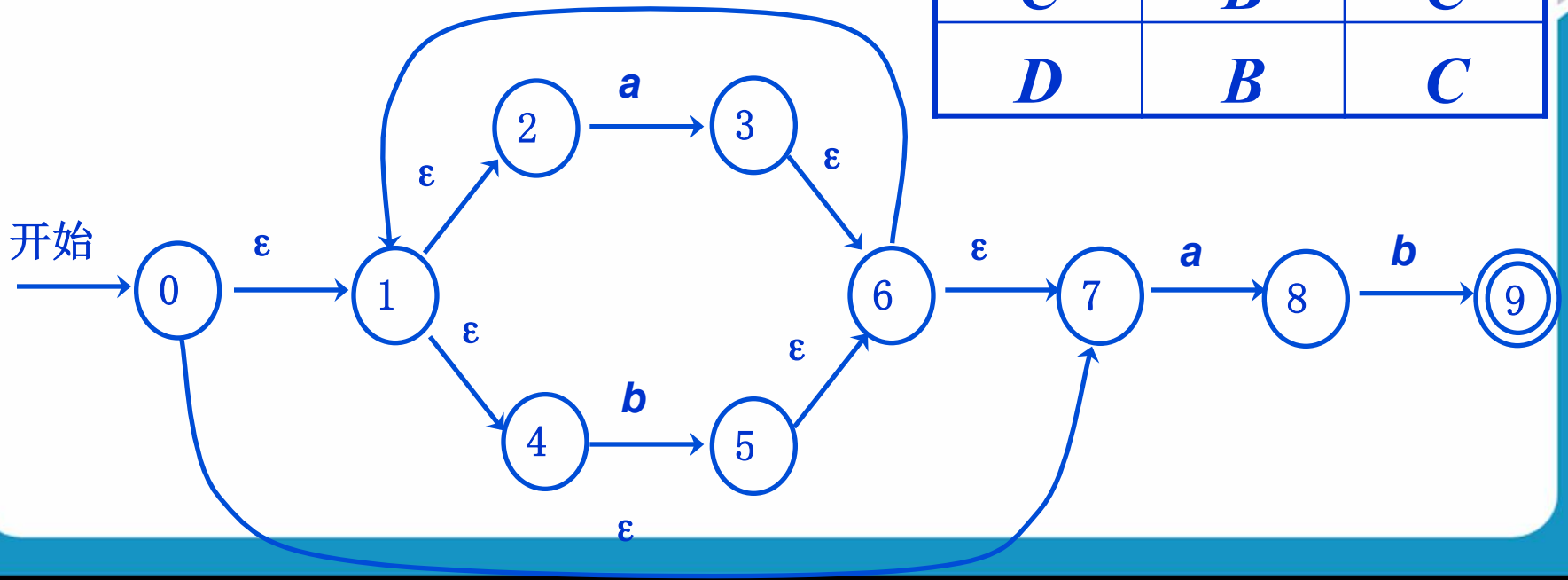
状态	输入符号	
	a	*b*
A	*B*	*C*
B	*B*	*D*
C	*B*	*C*
D	*B*	*C*



3.3 有限自动机

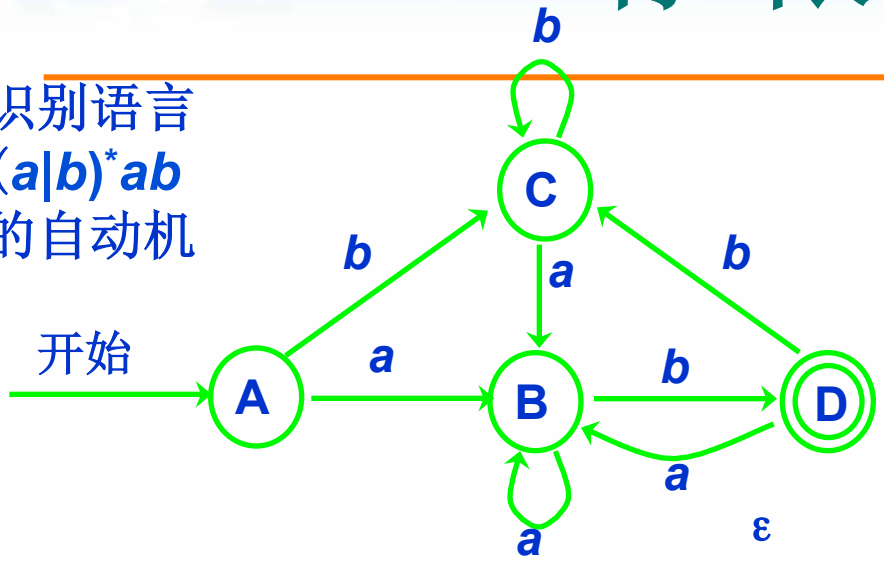


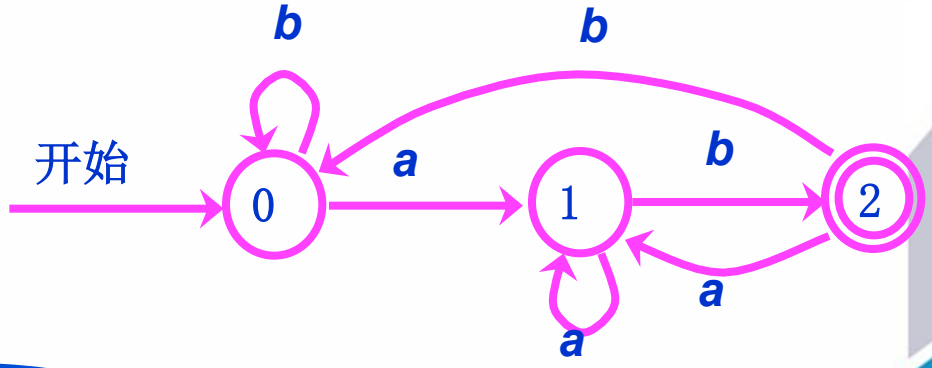
状态	输入符号	
	a	*b*
A	*B*	*C*
B	*B*	*D*
C	*B*	*C*
D	*B*	*C*

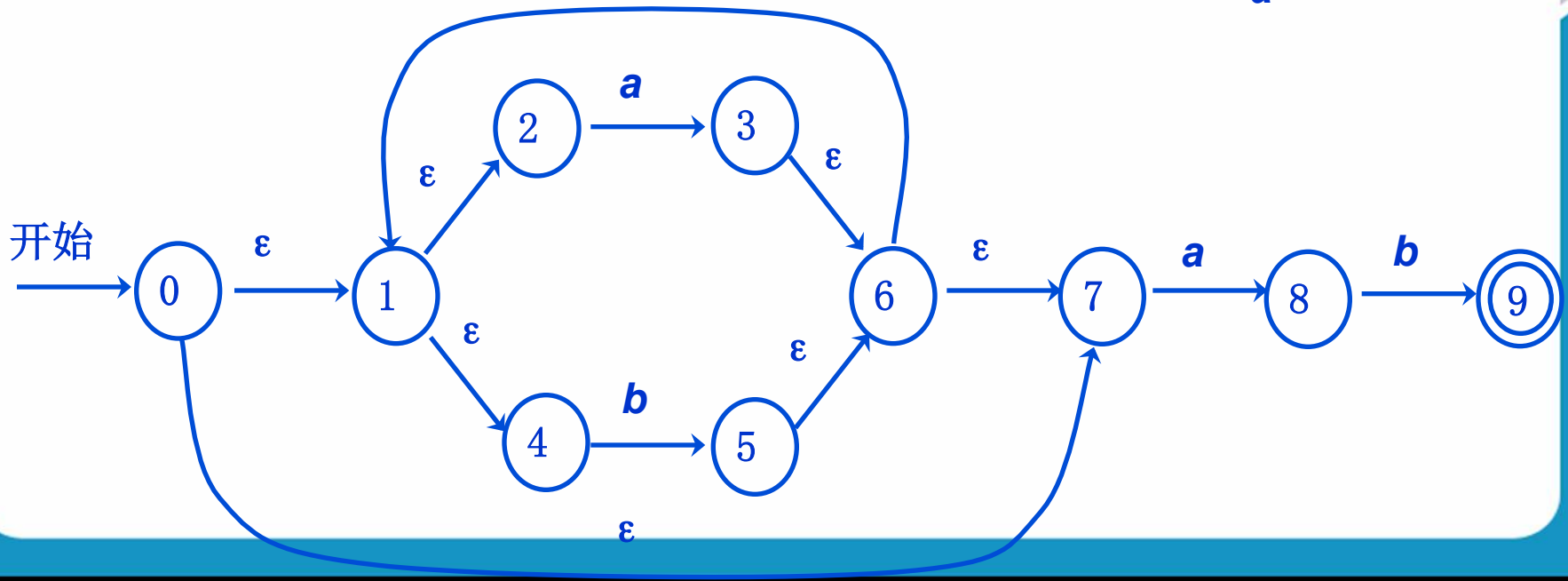


3.3 有限自动机

识别语言
$(a|b)^*ab$
的自动机

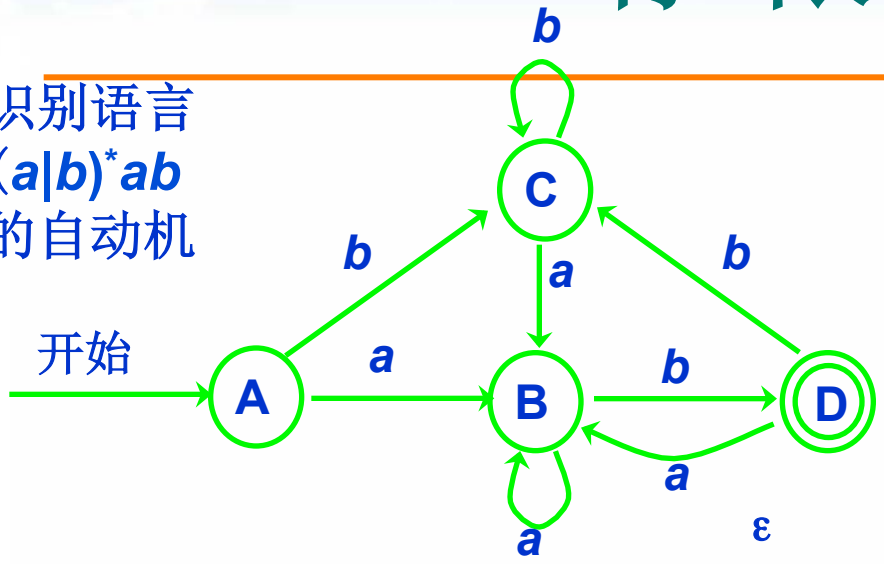




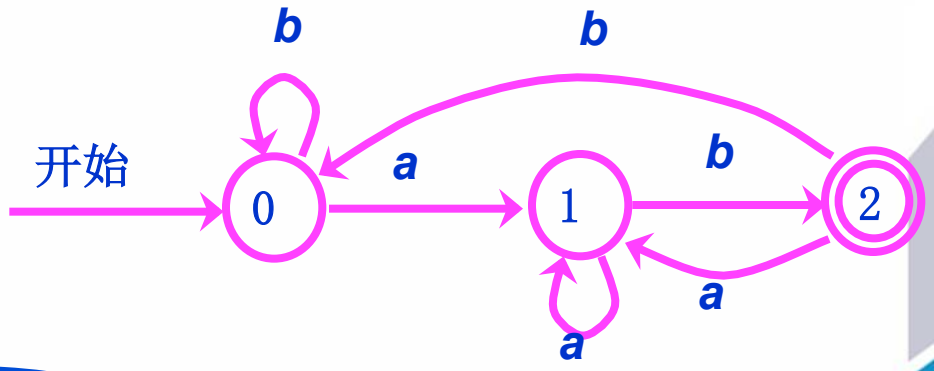


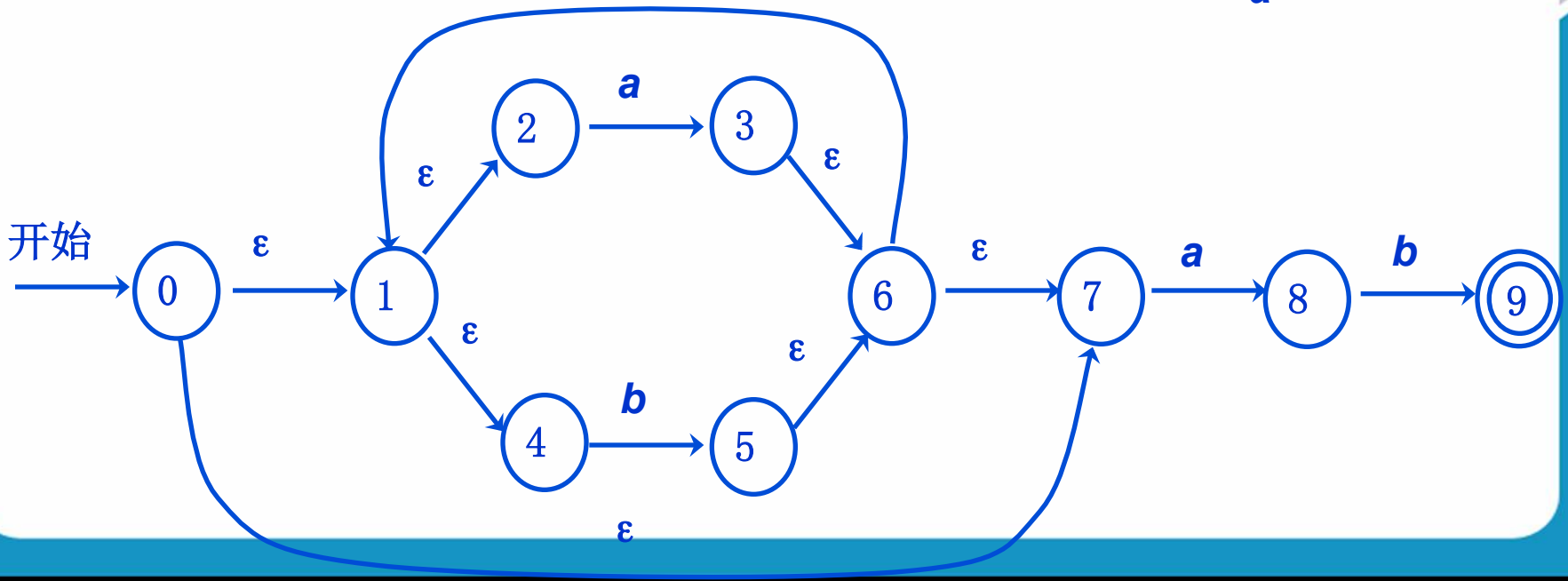
3.3 有 限 自 动 机

识别语言
$(a|b)^*ab$
的自动机



子集构造法不一定得到最简DFA





3.3 有限自动机

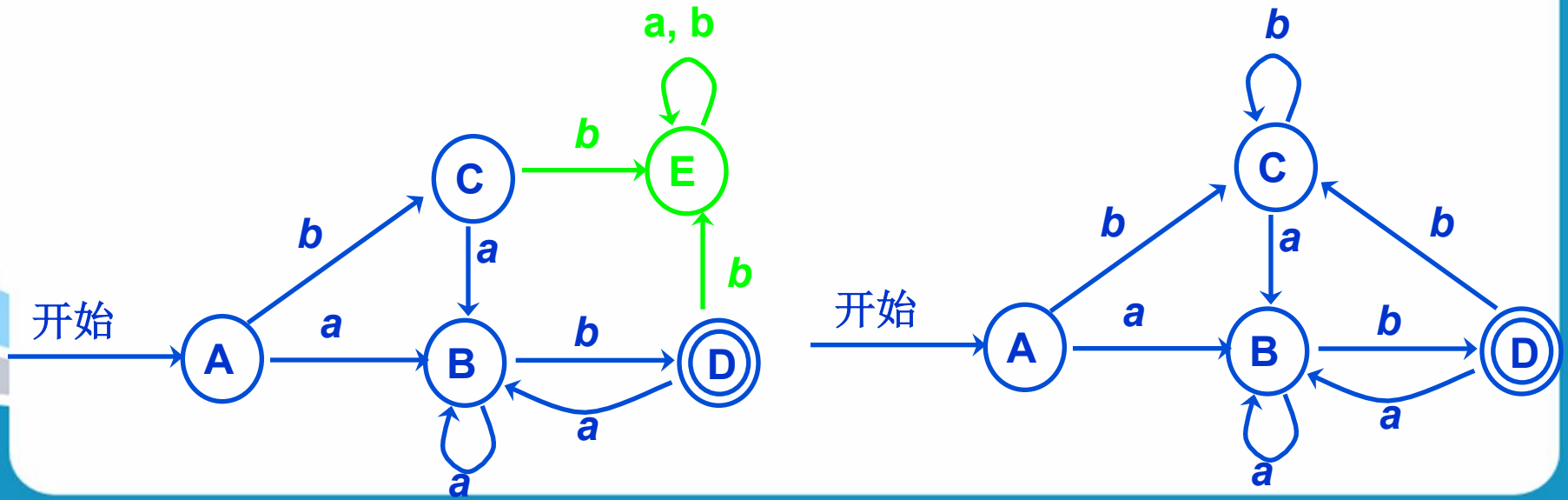
3.3.4 DFA的化简

- 每一个正则式可以由一个状态数最少的DFA识别，且这个DFA唯一
- 死状态
  - 在转换函数由部分函数改成全函数表示时引入
  - 死状态对所有输入符号都转换到本身

3.3 有 限 自 动 机

3.3.4 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
  - 死状态对所有输入符号都转换到本身
  - 左图需要引入死状态E ； 右图无须引入死状态

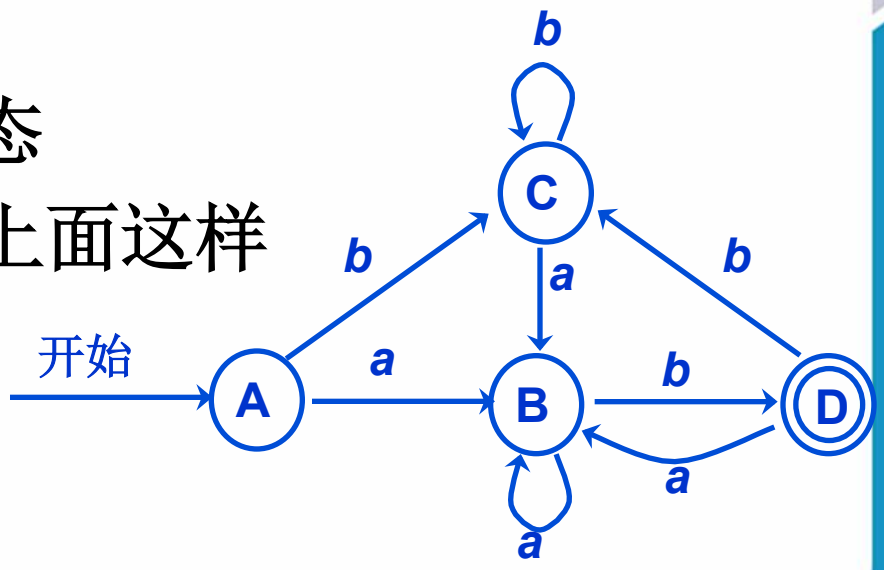


3.3 有 限 自 动 机

- 可区别的状态
  - A和B是可区别的状态

    从A出发，读过单字符*b*构成的串，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



3.3 有 限 自 动 机

- 方法

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

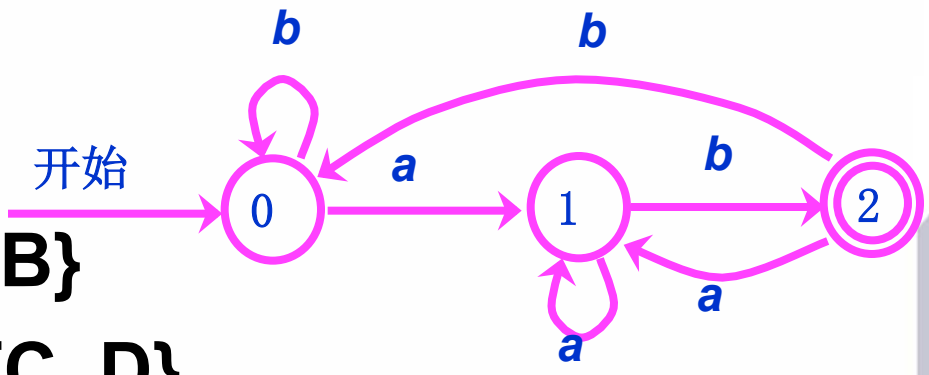
move({A, B, C}, *a*) = {B}

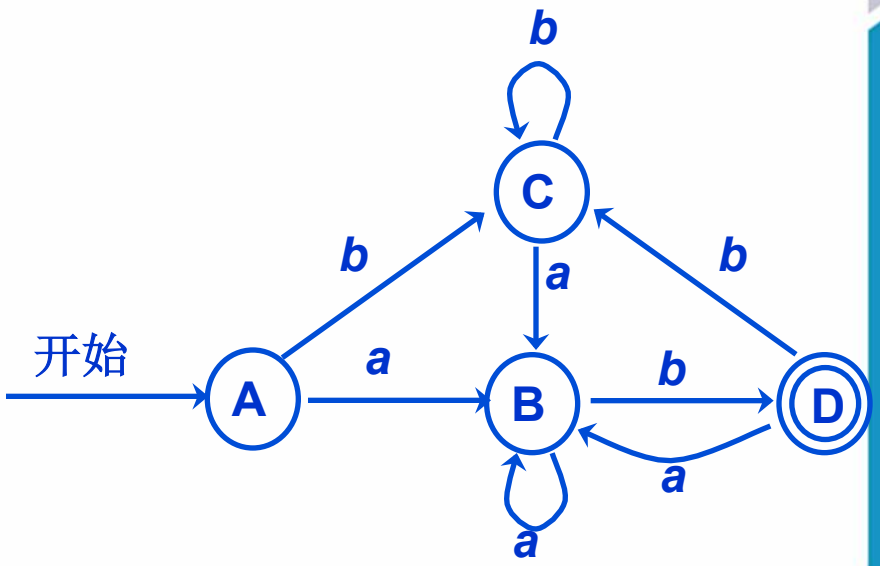
move({A, B, C}, *b*) = {C, D}

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

move({A, C}, *a*) = {B}

move({A, C}, *b*) = {C}





3.3 有 限 自 动 机

3.3.4 DFA的化简

- 构造最简DFA：
  - 构造状态集合的初始划分π ： 两个子集——接受状态子集F和非接受状态子集S – F

3.3 有限自动机

3.3.4 DFA的化简

- 构造最简DFA：
  - 构造状态集合的初始划分π：两个子集——接受状态子集F和非接受状态子集S – F
  - 应用下面的过程构造π_{new}
    - For π 中的每个子集G，do begin
      - 把G划分为若干子集，G的两个状态 s 和 t 在同一子集中，当且仅当对任意输入符号 a，s 和 t 的 a 转换都到 π 的同一子集中
      - 在π_{new} 中，用G的划分代替G
    - End

3.3 有限自动机

3.3.4 DFA的化简

- 构造最简DFA：
  - 构造状态集合的初始划分π：两个子集——接受状态子集F和非接受状态子集S – F
  - 应用下面的过程构造π_{new}
    - For π 中的每个子集G，do begin
      - 把G划分为若干子集，G的两个状态 s 和 t 在同一子集中，当且仅当对任意输入符号 a，s 和 t 的 a 转换都到 π 的同一子集中
      - 在π_{new} 中，用G的划分代替G
    - End
  - 如果$\pi_{new} = \pi$，则$\pi_{final} = \pi$；否则令$\pi = \pi_{new}$，转上步

3.3 有限自动机

3.3.4 DFA的化简

- 构造最简DFA：
  - 构造状态集合的初始划分π：两个子集——接受状态子集F和非接受状态子集S – F
  - 应用下面的过程构造π_{new}
    - For π 中的每个子集G，do begin
      - 把G划分为若干子集，G的两个状态 s 和 t 在同一子集中，当且仅当对任意输入符号 a，s 和 t 的 a 转换都到 π 的同一子集中
      - 在π_{new} 中，用G的划分代替G
    - End
  - 如果$\pi_{new} = \pi$，则$\pi_{final} = \pi$；否则令$\pi = \pi_{new}$，转上步
  - 在π_{final}的每个状态子集中选一个状态代表它，即为最简DFA的状态

编译原理

第三章 词法分析 上次课回顾

计算机科学与技术学院

上次课回顾

- 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

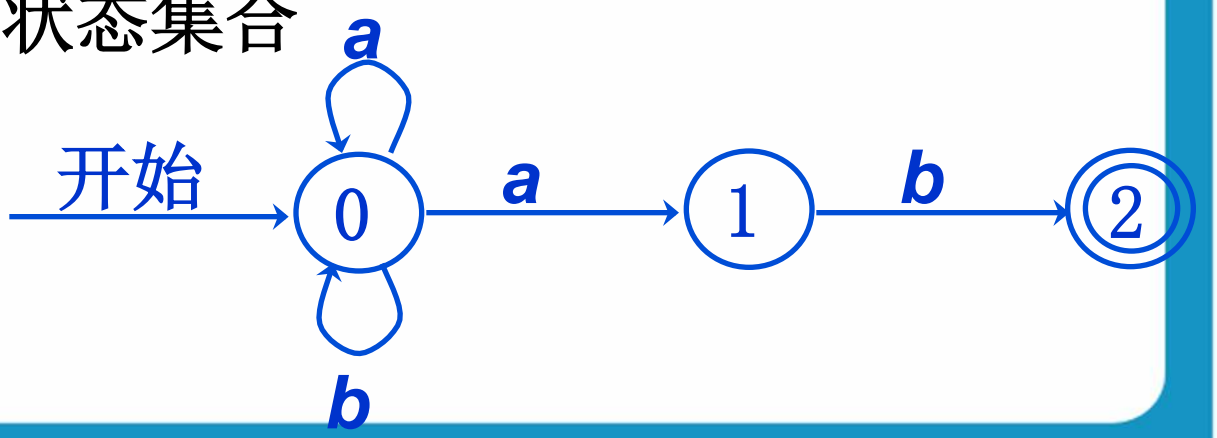
  2、输入符号集合$\sum$

  3、转换函数$move: S \times (\sum \cup \{\varepsilon\}) \rightarrow P(S)$

  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



上次课回顾

- 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

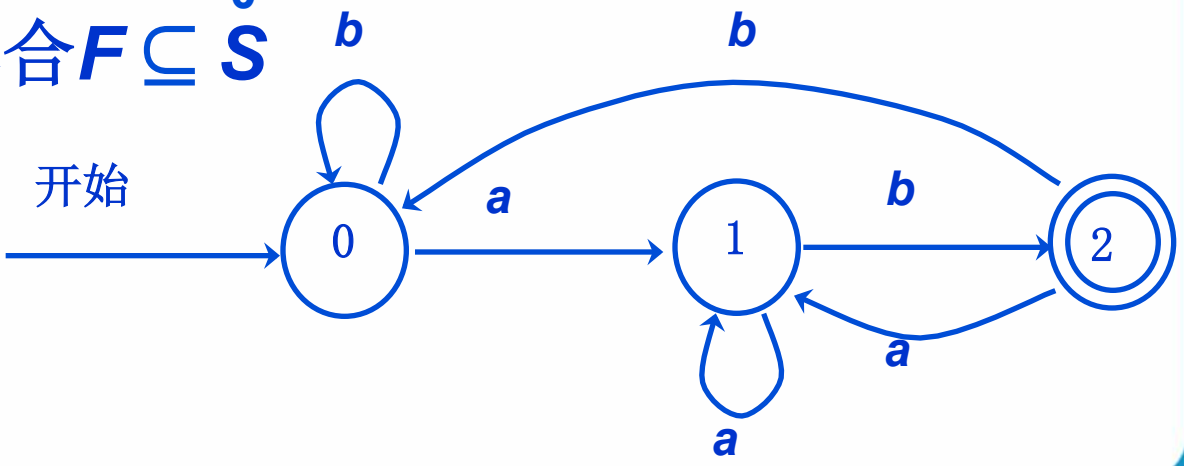
  2、输入字母集合Σ

  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



上次课回顾

- **NFA到DFA的变换**

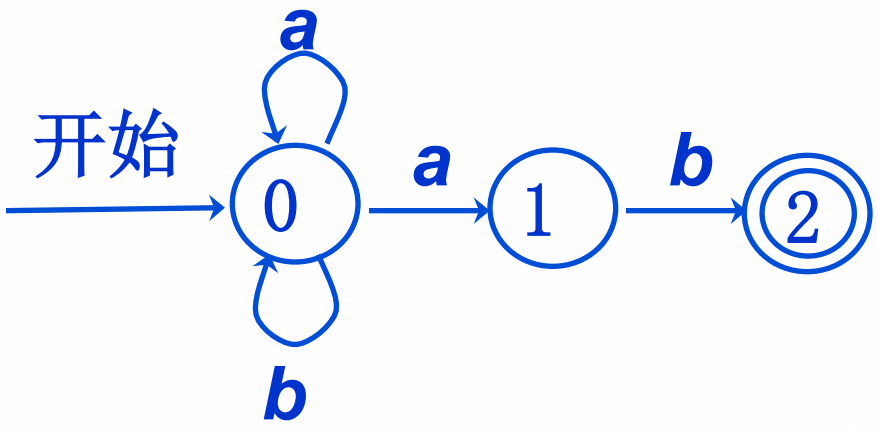
  子集构造法

  1、DFA的一个状态是NFA的一个状态集合

  2、读了输入$a_1\ a_2\ \ldots\ a_n$后，

  NFA能到达的所有状态：$s_1, s_2, \ldots, s_k$，则

  DFA到达状态$\{s_1, s_2, \ldots, s_k\}$



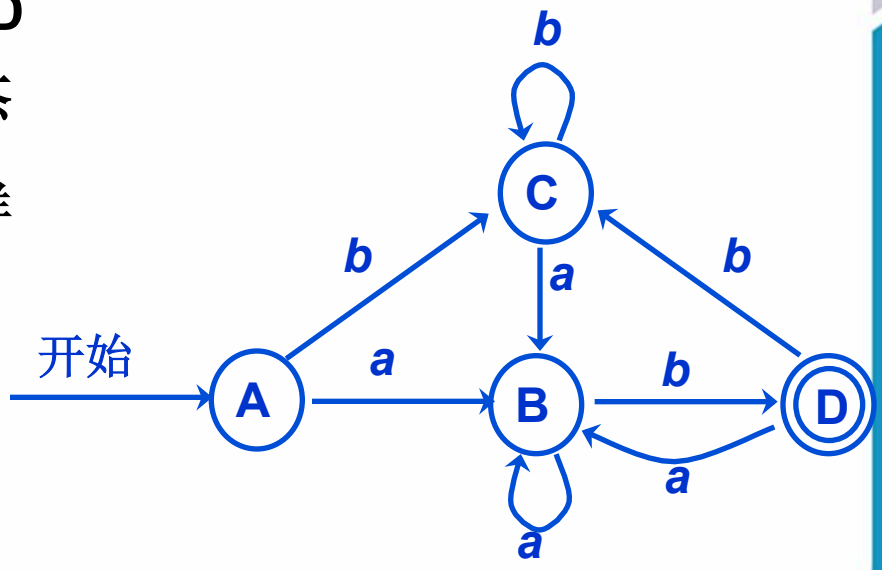
上次课回顾

- **DFA的化简**
  - 可区别的状态
    - **A和B是可区别的状态**

从**A**出发，读过单字符***b***构成的串，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



3.4 从正则式到有限自动机

- 从正则式建立识别器的步骤

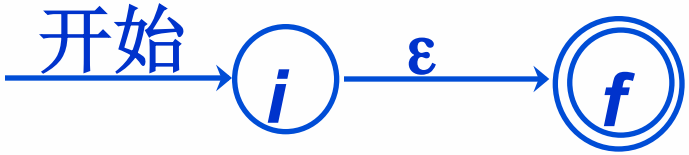
  - 从正则式构造**NFA**（本节介绍）

    用语法制导的算法，它用正则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 （合并不可区别状态，也已介绍）

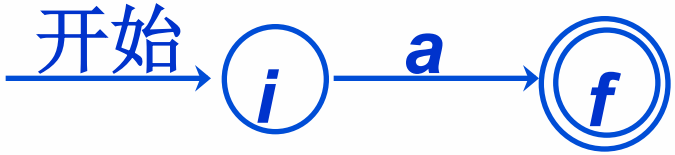
3.4 从正则式到有限自动机

- 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



识别正则式ε的NFA

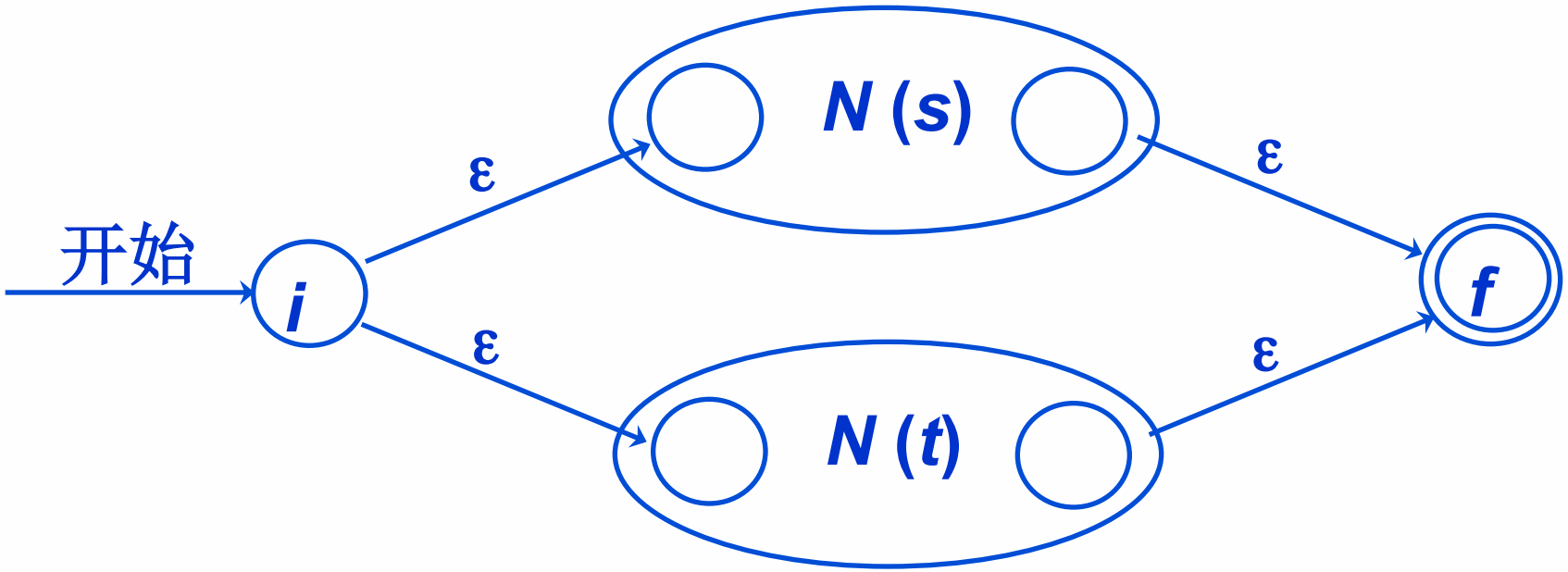


识别正则式*a*的NFA

3.4 从正则式到有限自动机

- 构造识别主算符为选择的正则式的NFA

  重要特点：仅一个接受状态，它没有向外的转换

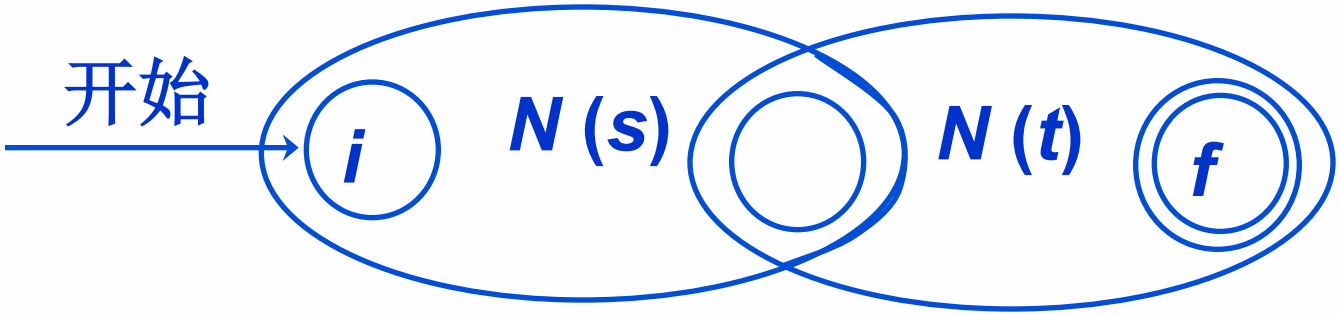


识别正则式*s* | *t* 的NFA

3.4 从正则式到有限自动机

- 构造识别主算符为连接的正则式的NFA

  重要特点：仅一个接受状态，它没有向外的转换

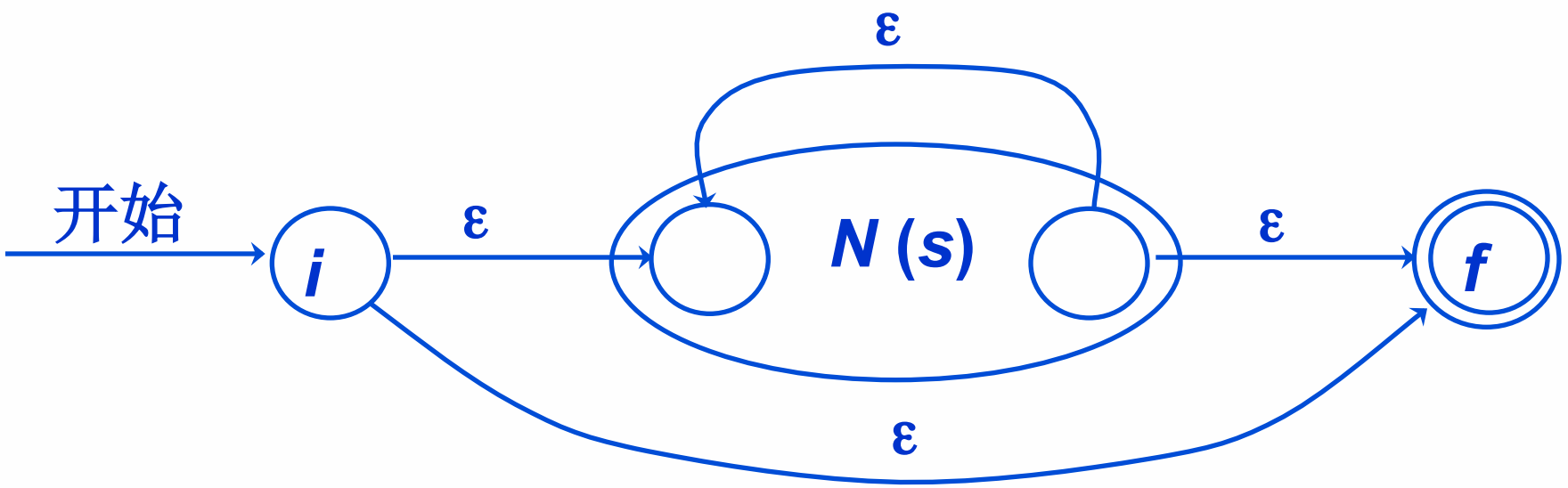


识别正则式 *st* 的NFA

3.4 从正则式到有限自动机

- 构造识别主算符为闭包的正则式的NFA

  重要特点：仅一个接受状态，它没有向外的转换



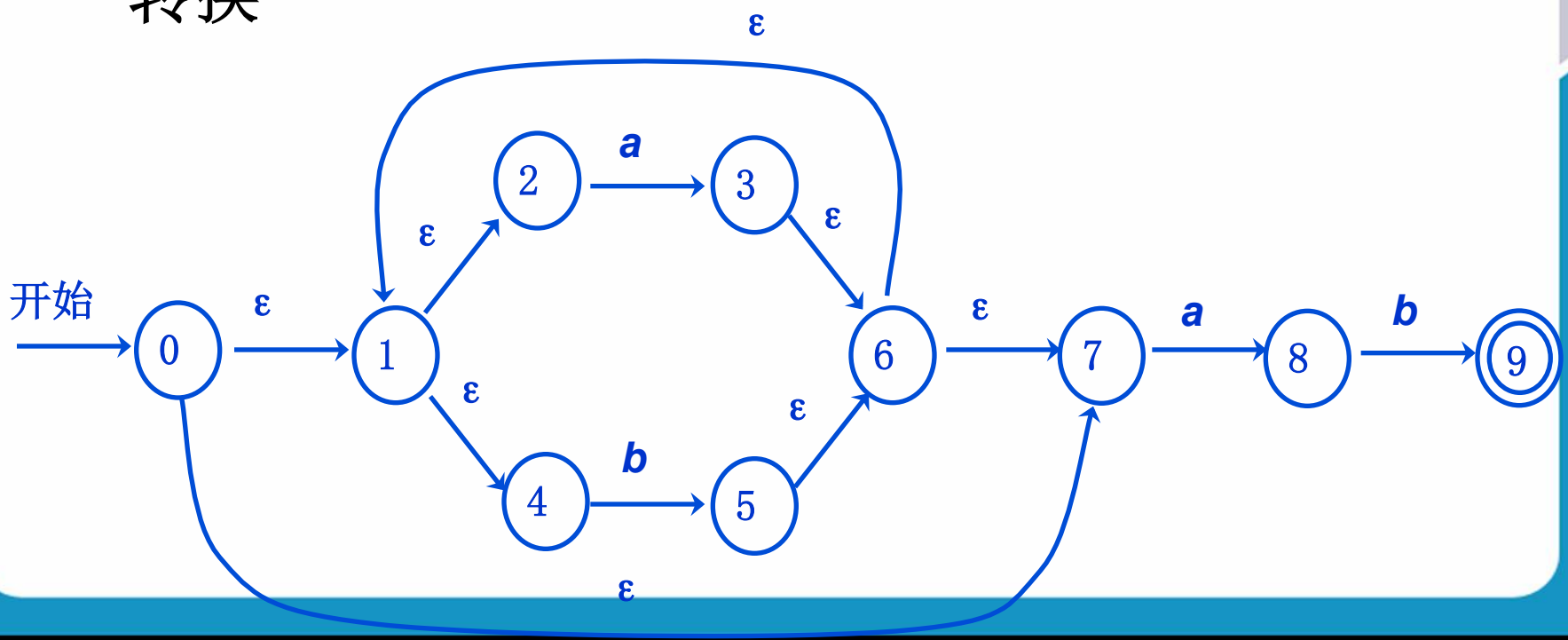
识别正则式 s^* 的NFA

3.4 从正则式到有限自动机

- 对于加括号的正则式(s)，使用*N*(s)本身作为它的NFA

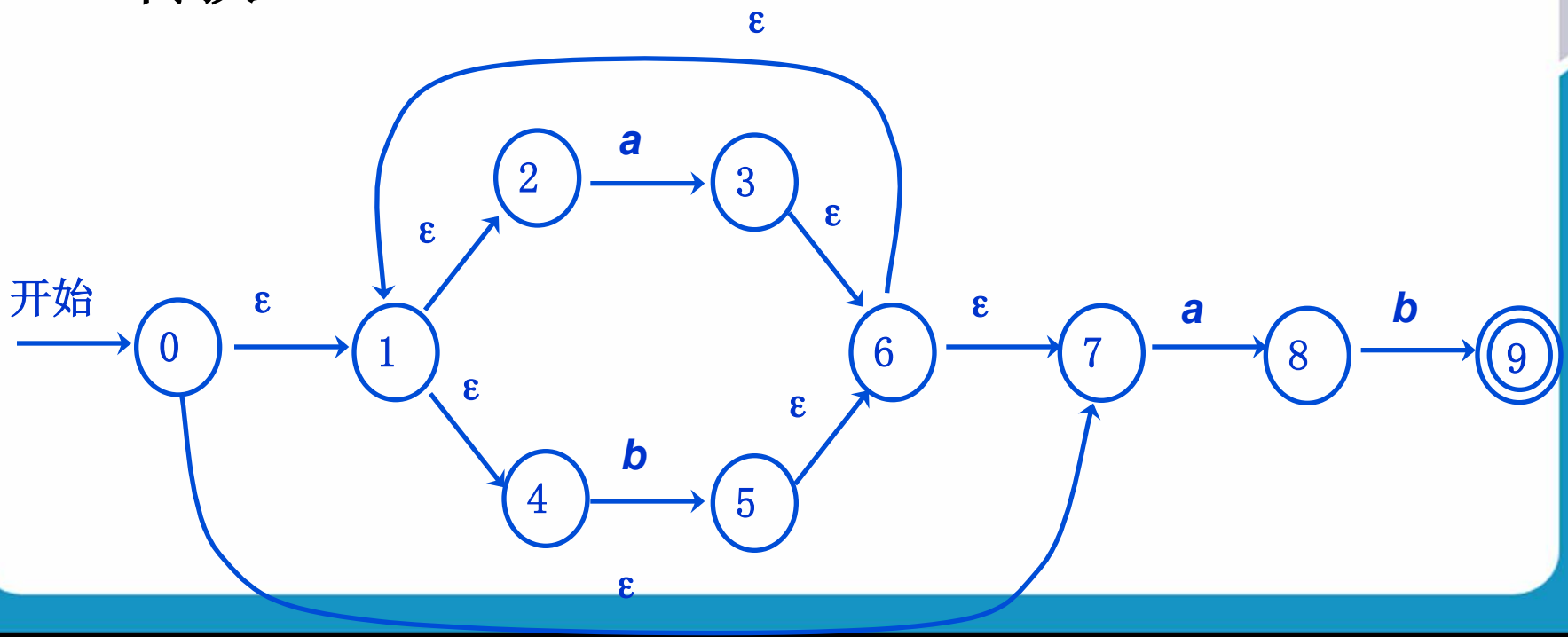
3.4 从正则式到有限自动机

- 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



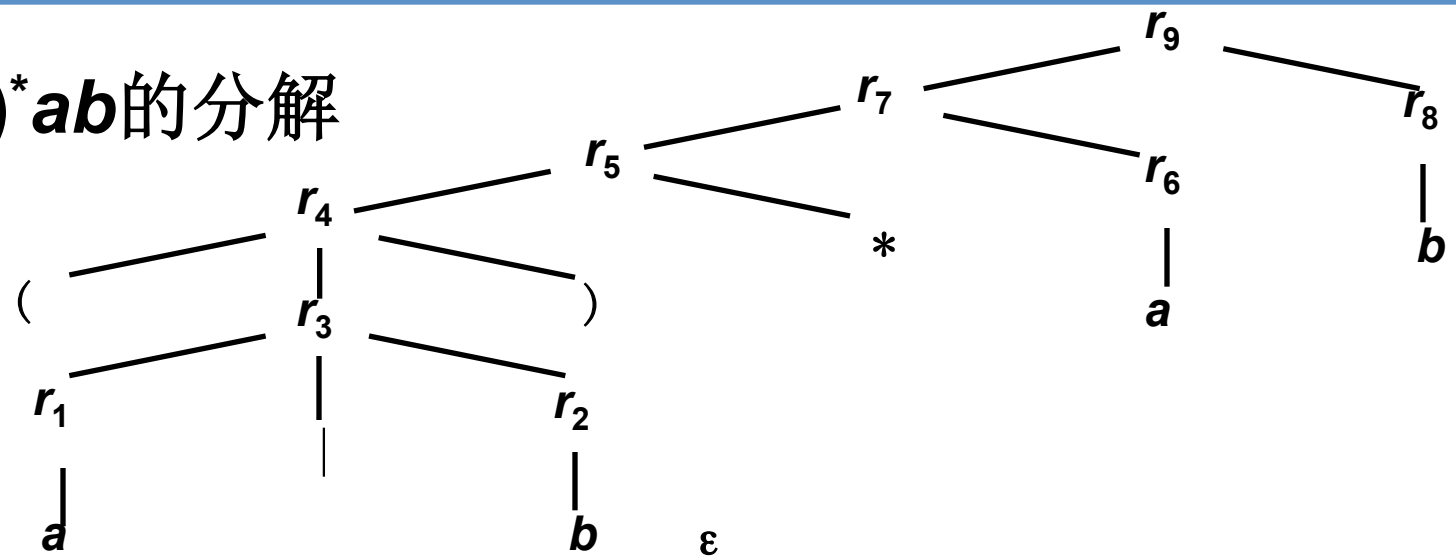
3.4 从正则式到有限自动机

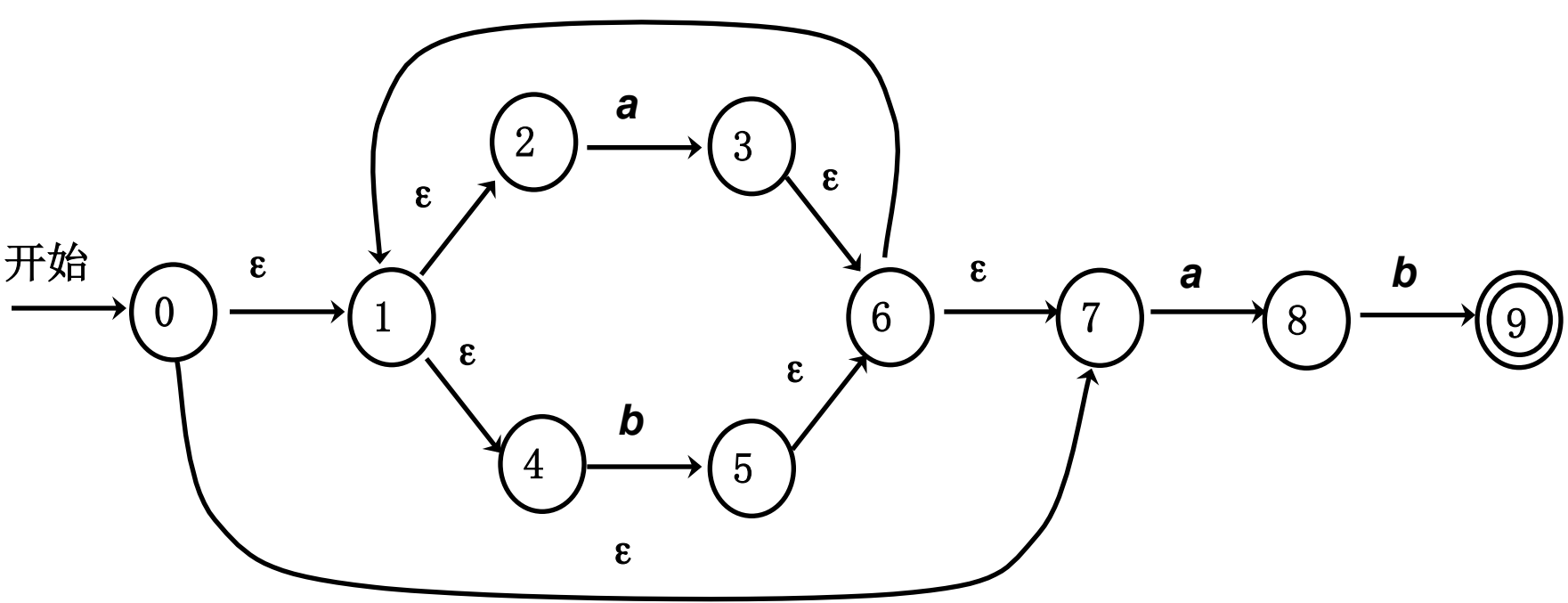
- 本方法产生的NFA有下列性质
  - *N*(*r*)的每个状态有一个用Σ的符号标记的指向其它结点的转换，或者最多两个指向其它结点的ε转换



3.4 从正则式到有限自动机

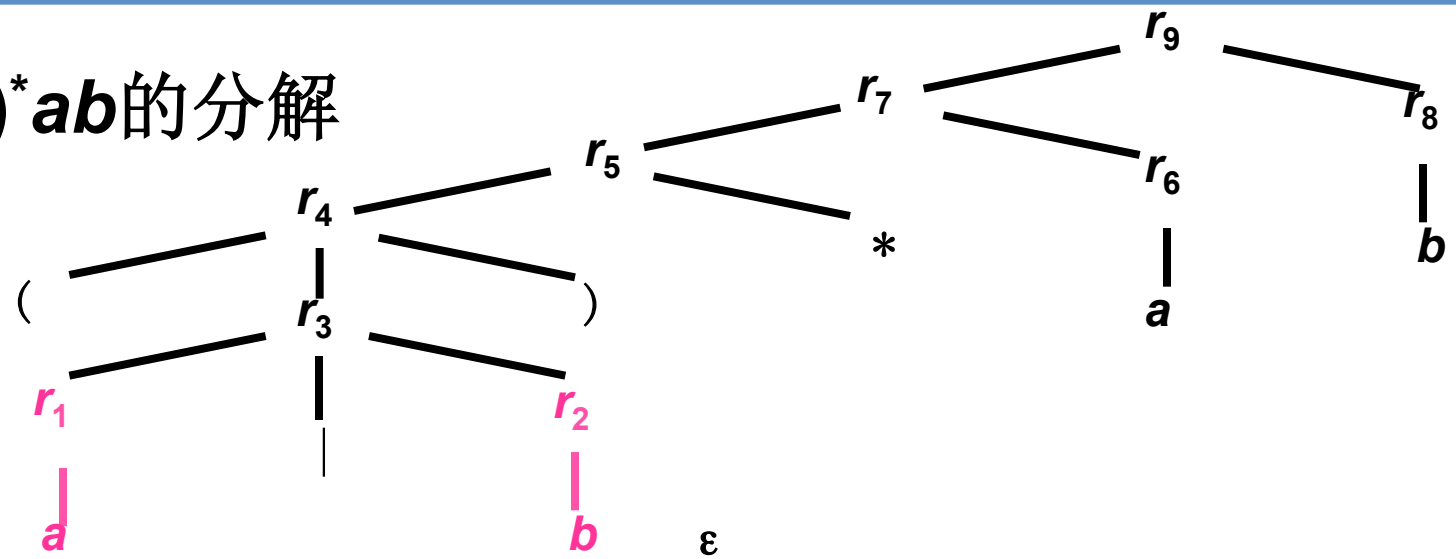
$(a|b)^*ab$的分解

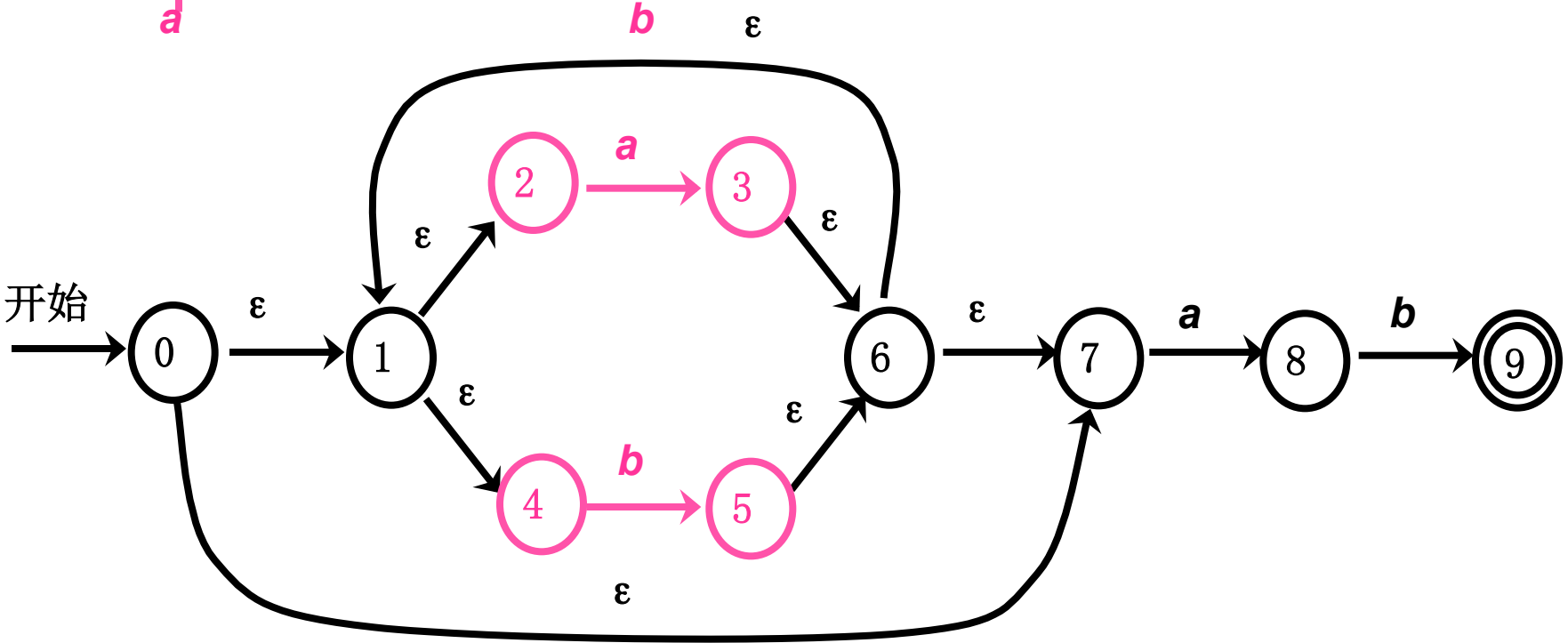




3.4 从正则式到有限自动机

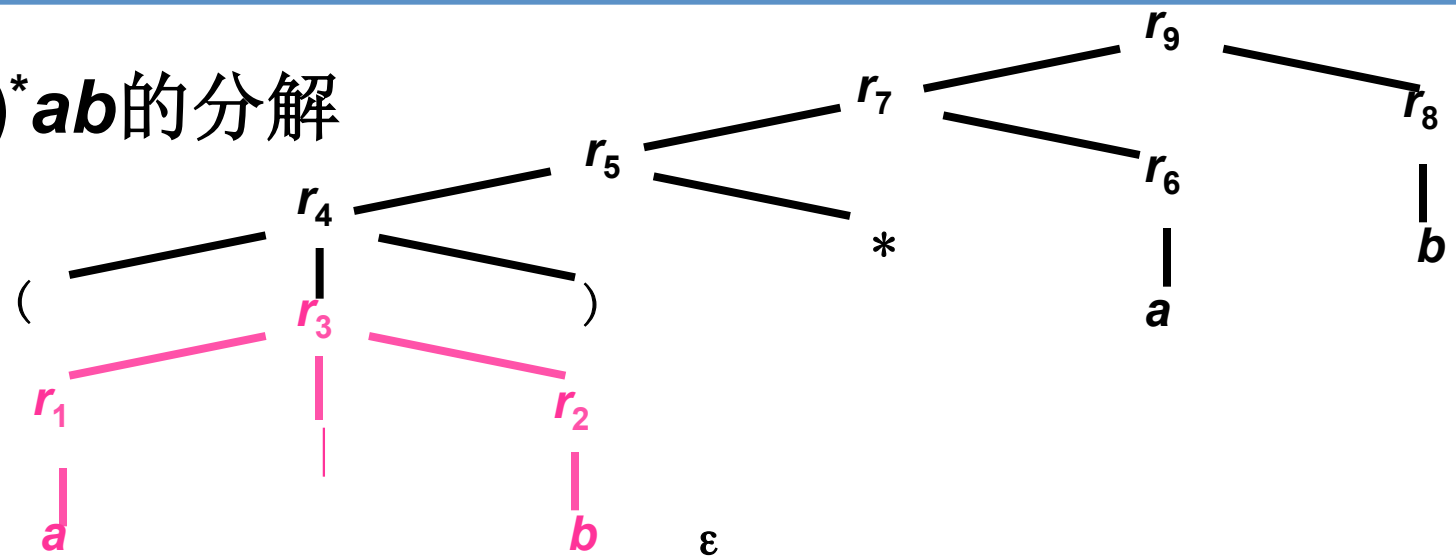
(*a*|*b*)***ab*的分解

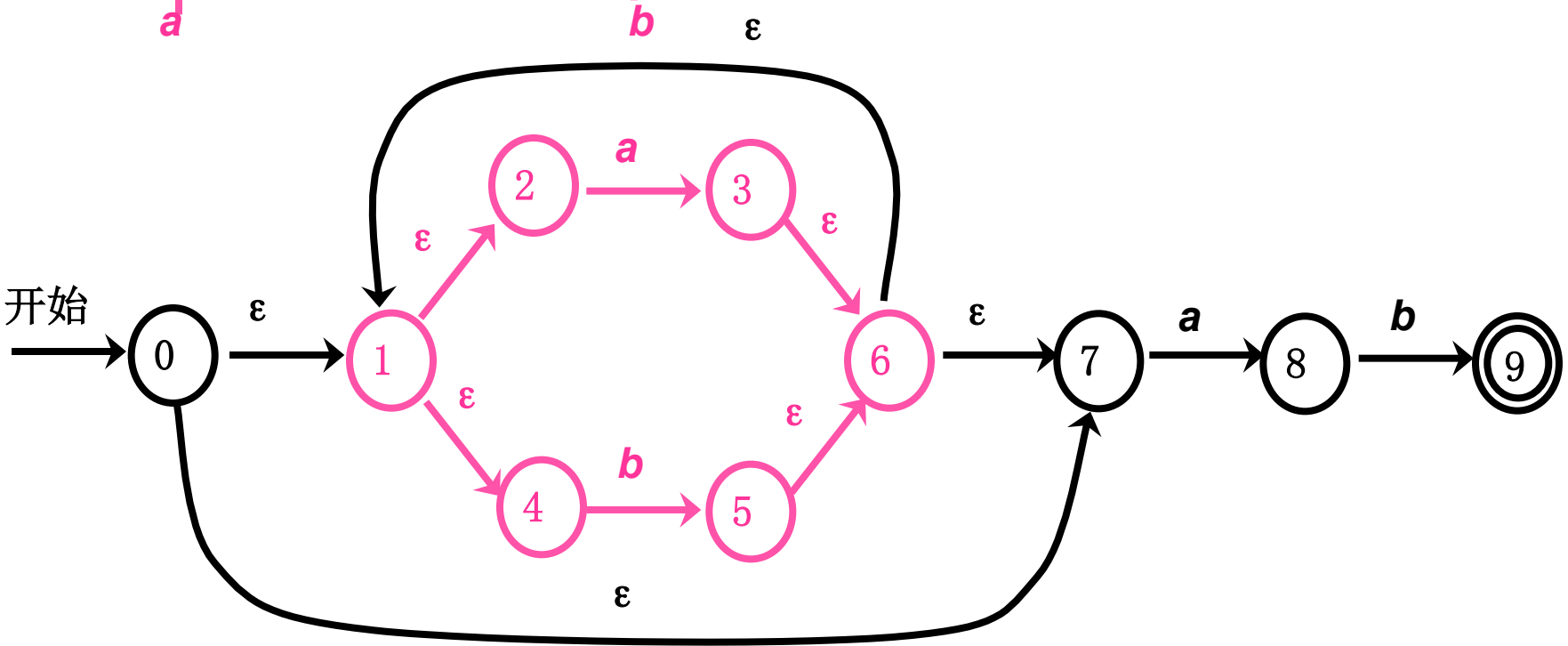




3.4 从正则式到有限自动机

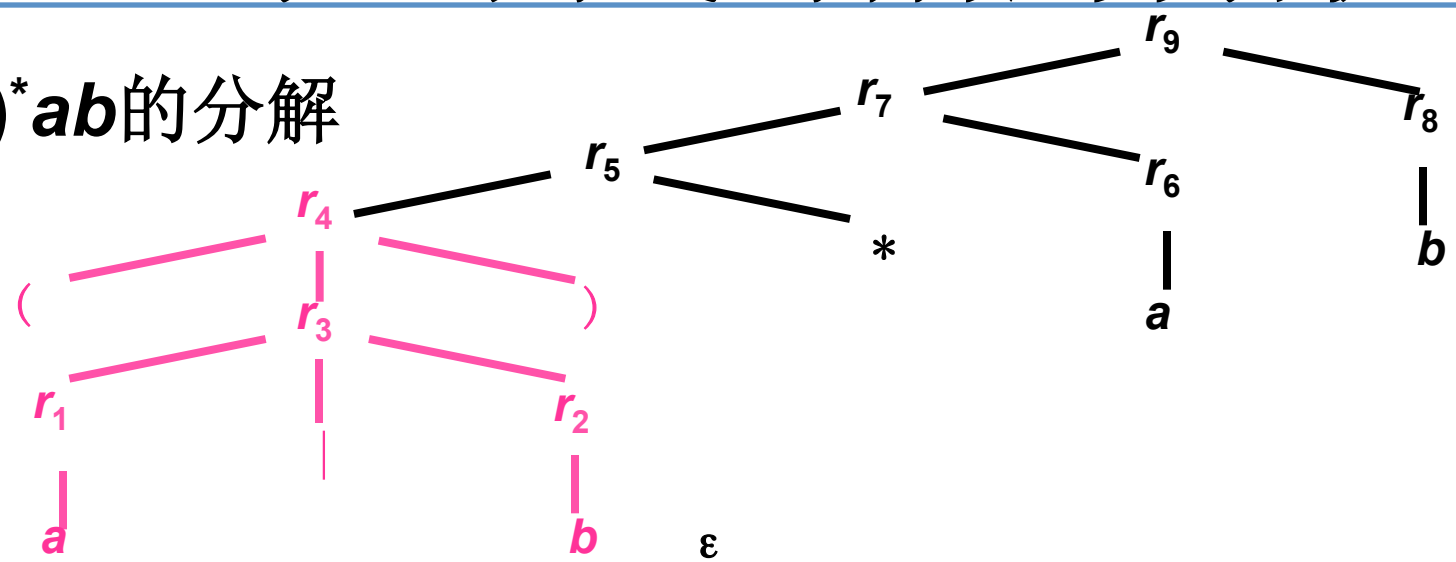
(*a*|*b*)***ab*的分解

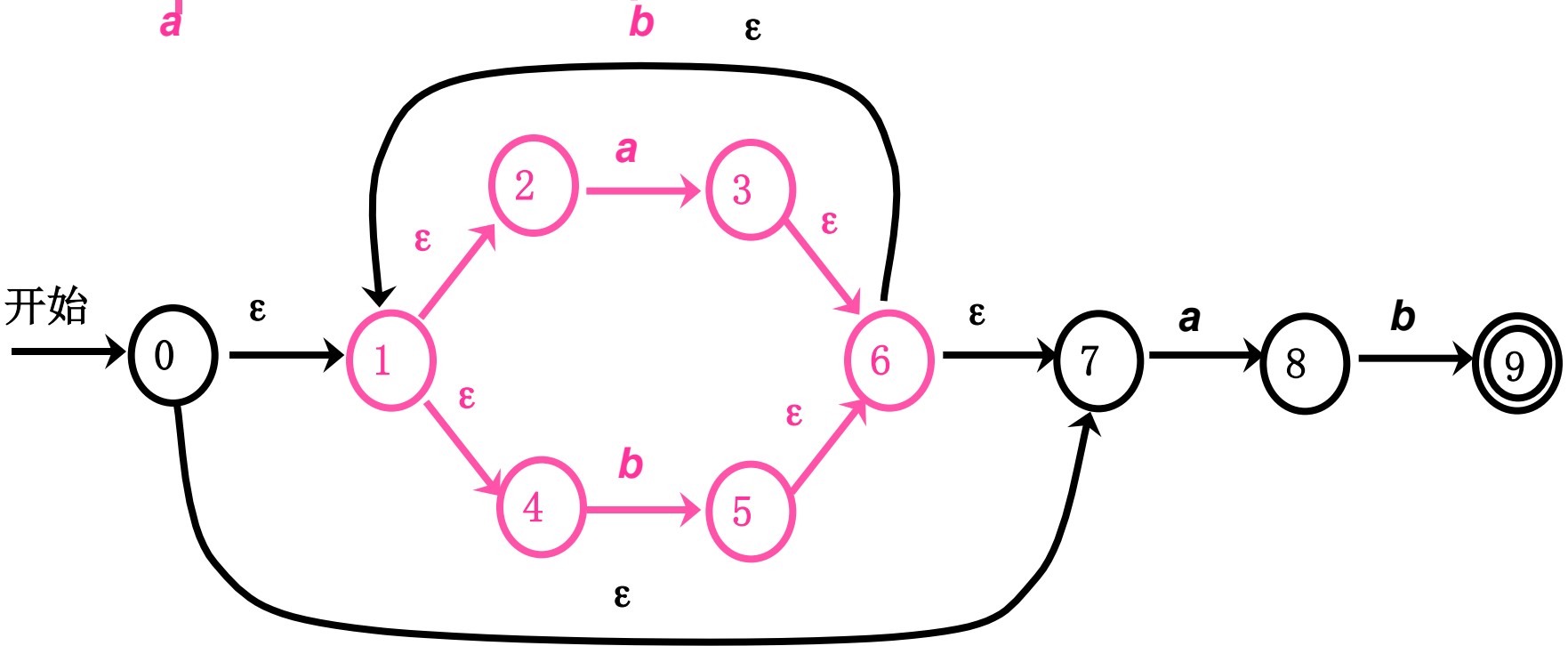




3.4 从正则式到有限自动机

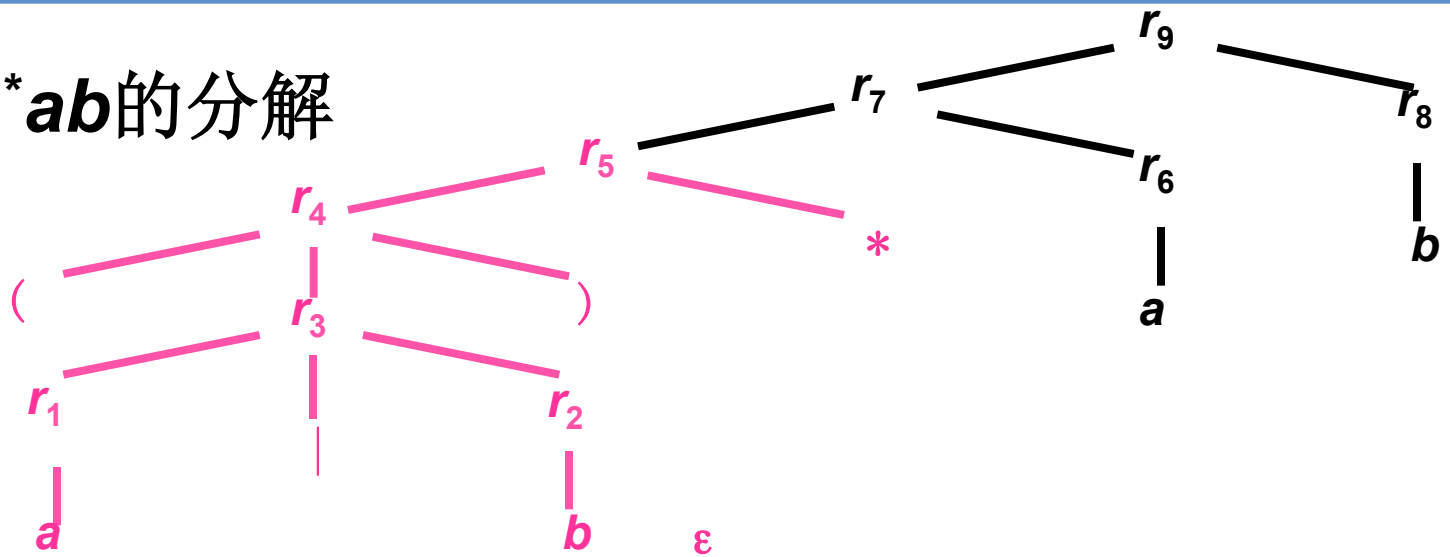
(*a*|*b*)***ab*的分解

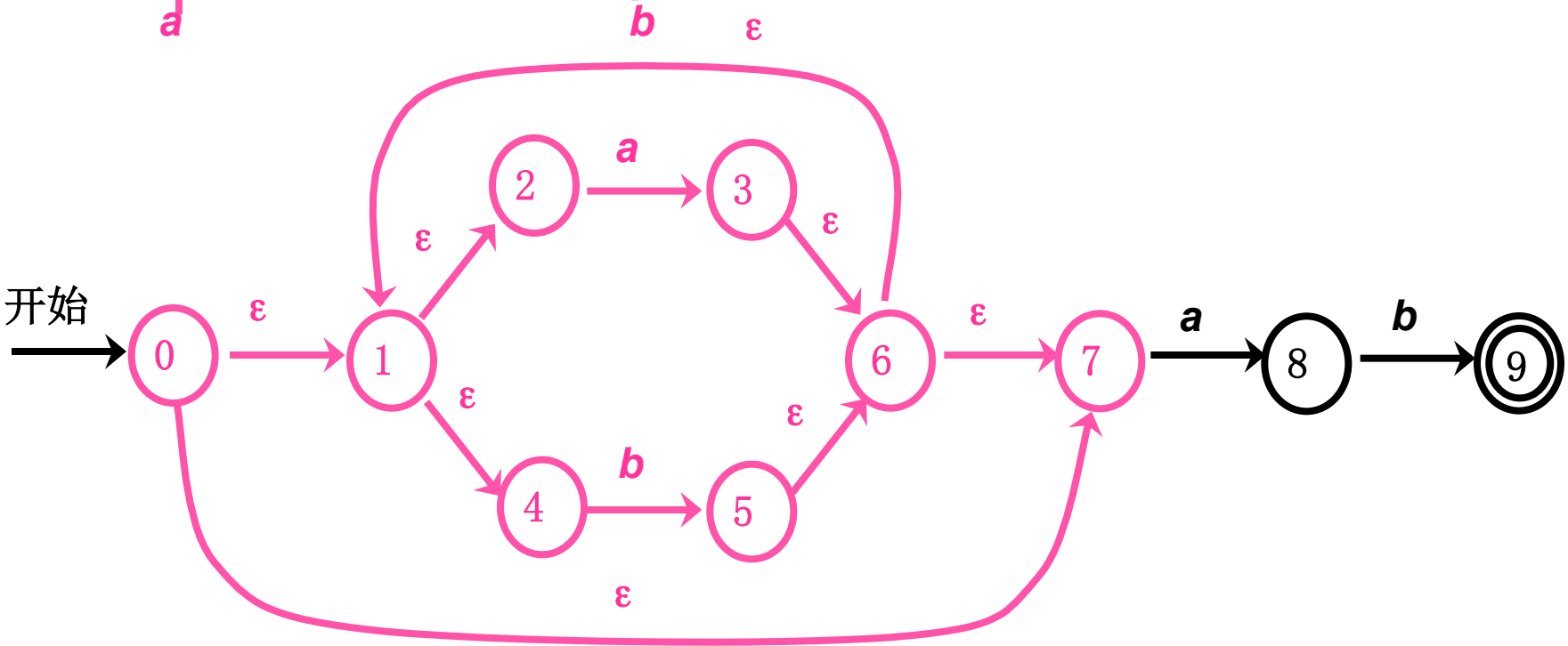




3.4 从正则式到有限自动机

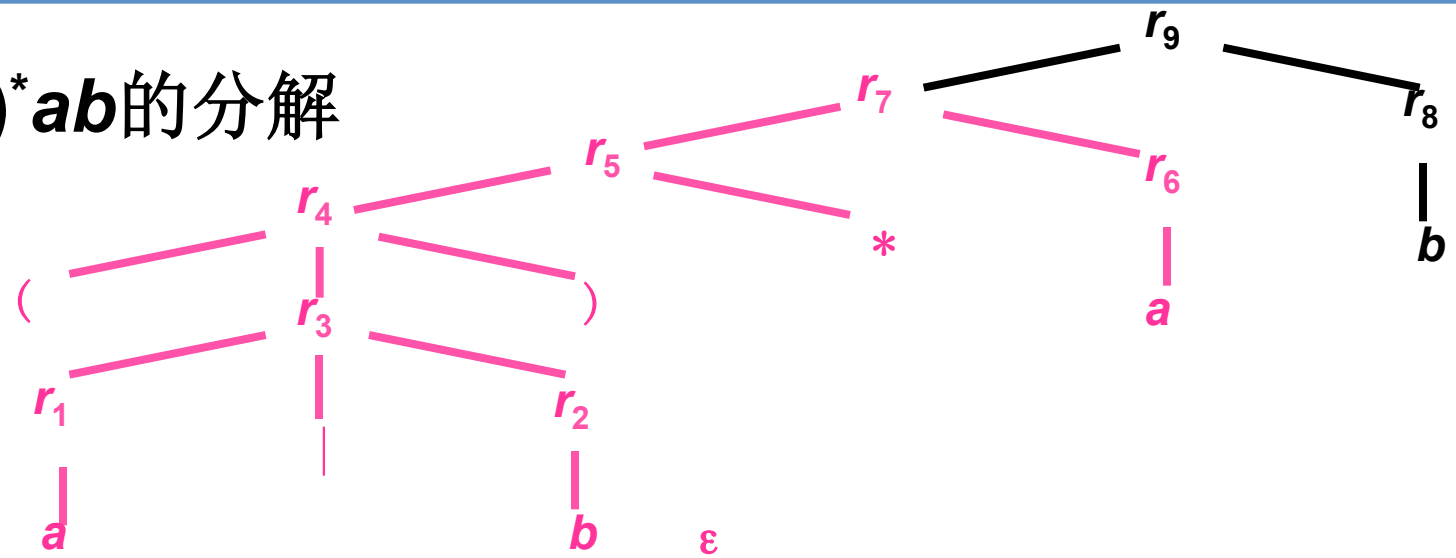
$(a|b)^*ab$的分解

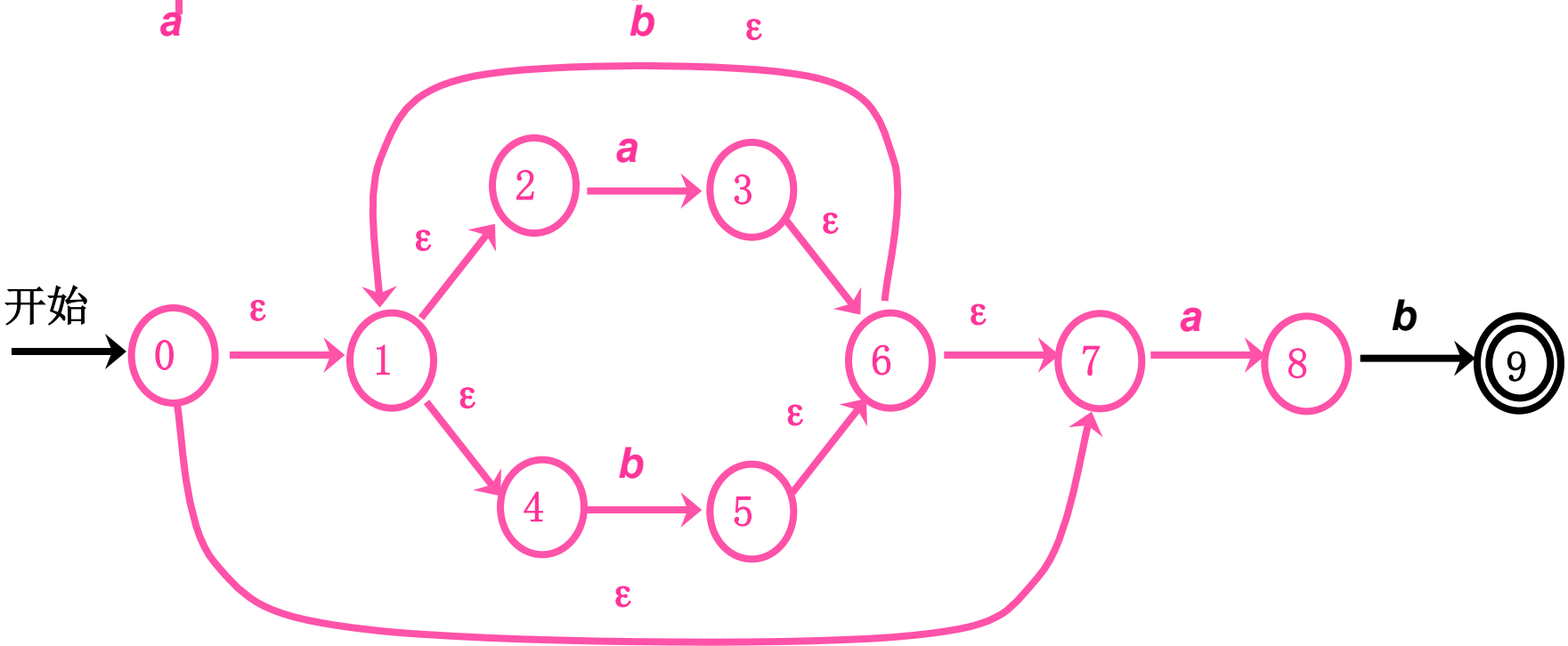




3.4 从正则式到有限自动机

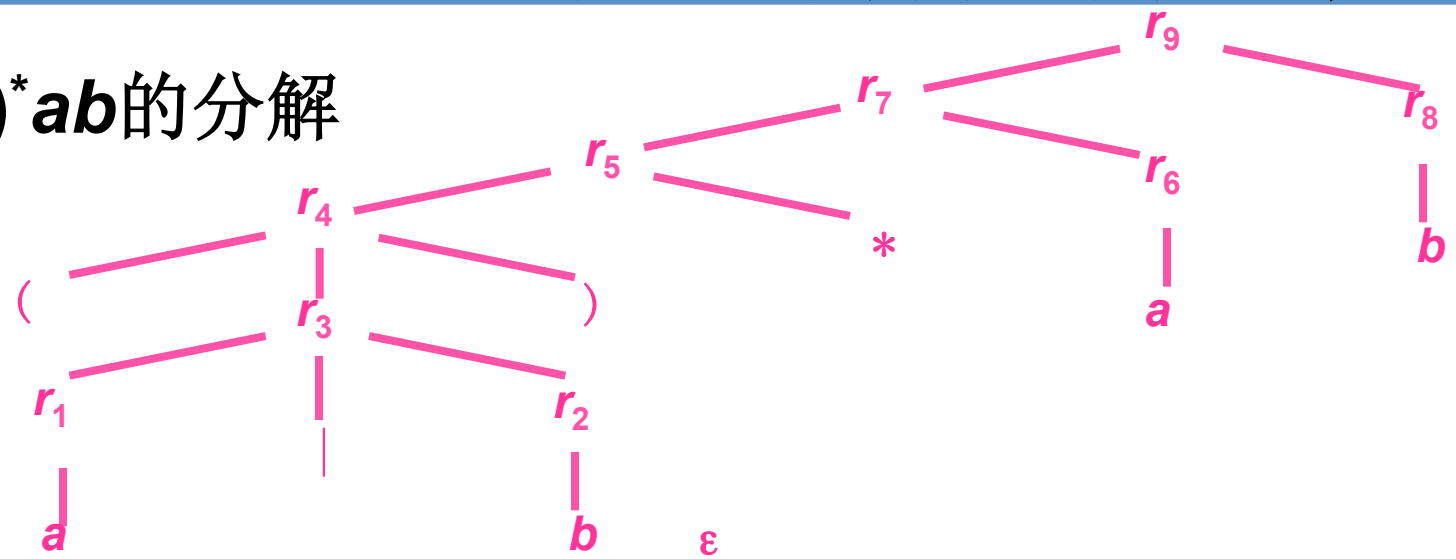
$(a|b)^*ab$的分解

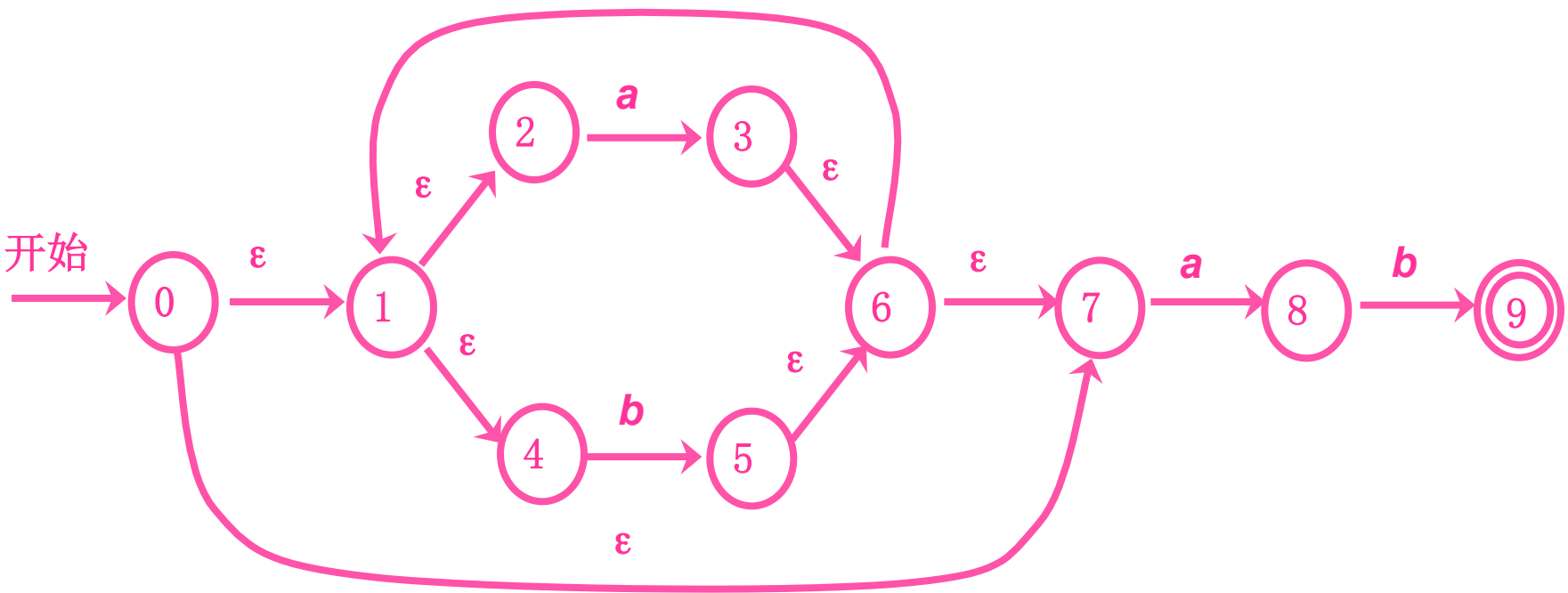




3.4 从正则式到有限自动机

(*a*|*b*)***ab*的分解

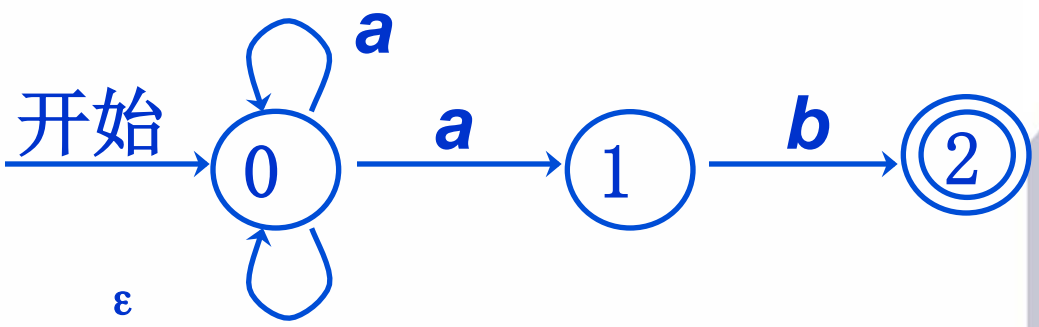




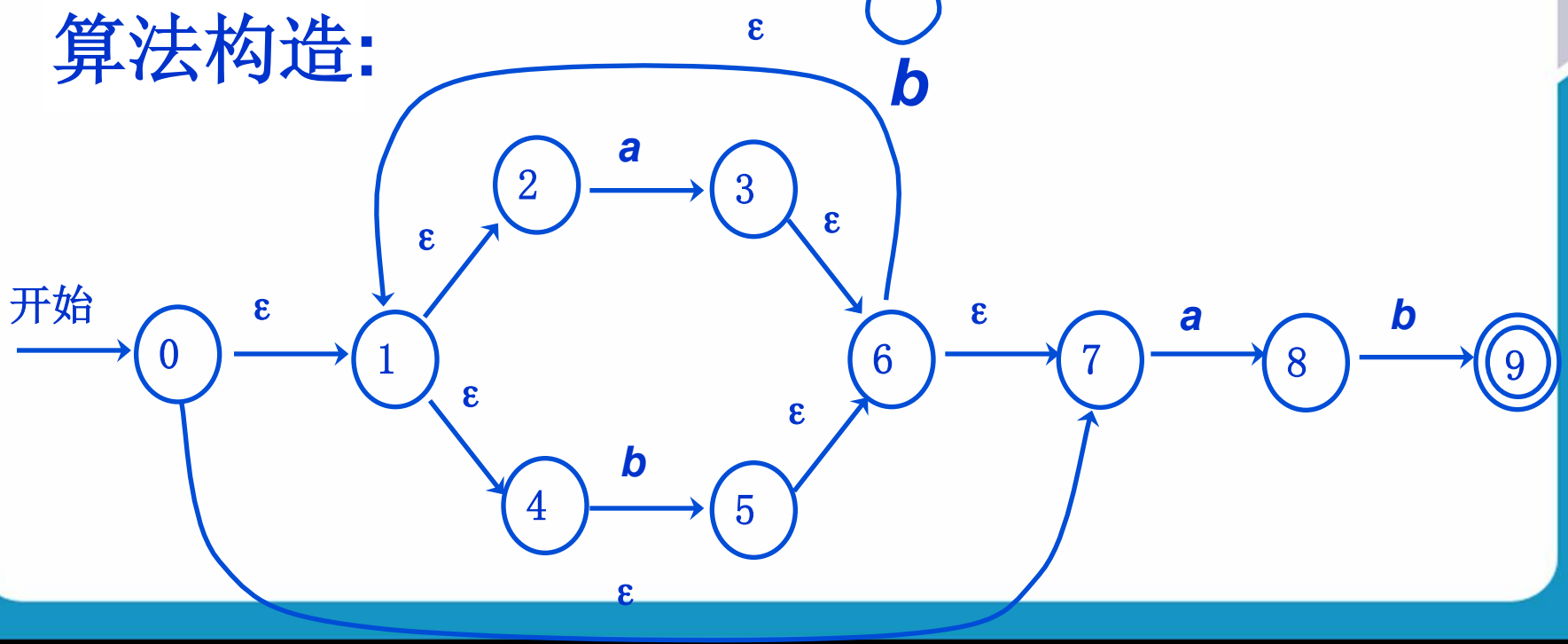
3.4 从正则式到有限自动机

- $(a|b)^*ab$的两个NFA的比较

手工构造:



算法构造:

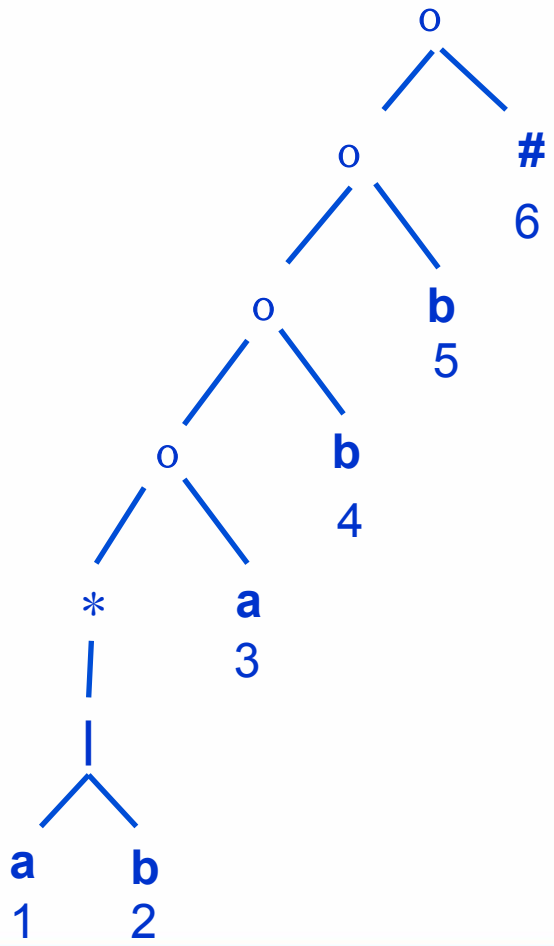


3.4 从正则式到有限自动机

- 小结：从正则式建立识别器的步骤
  - 从正则式构造NFA
  - 把NFA变成DFA
  - 将DFA化简

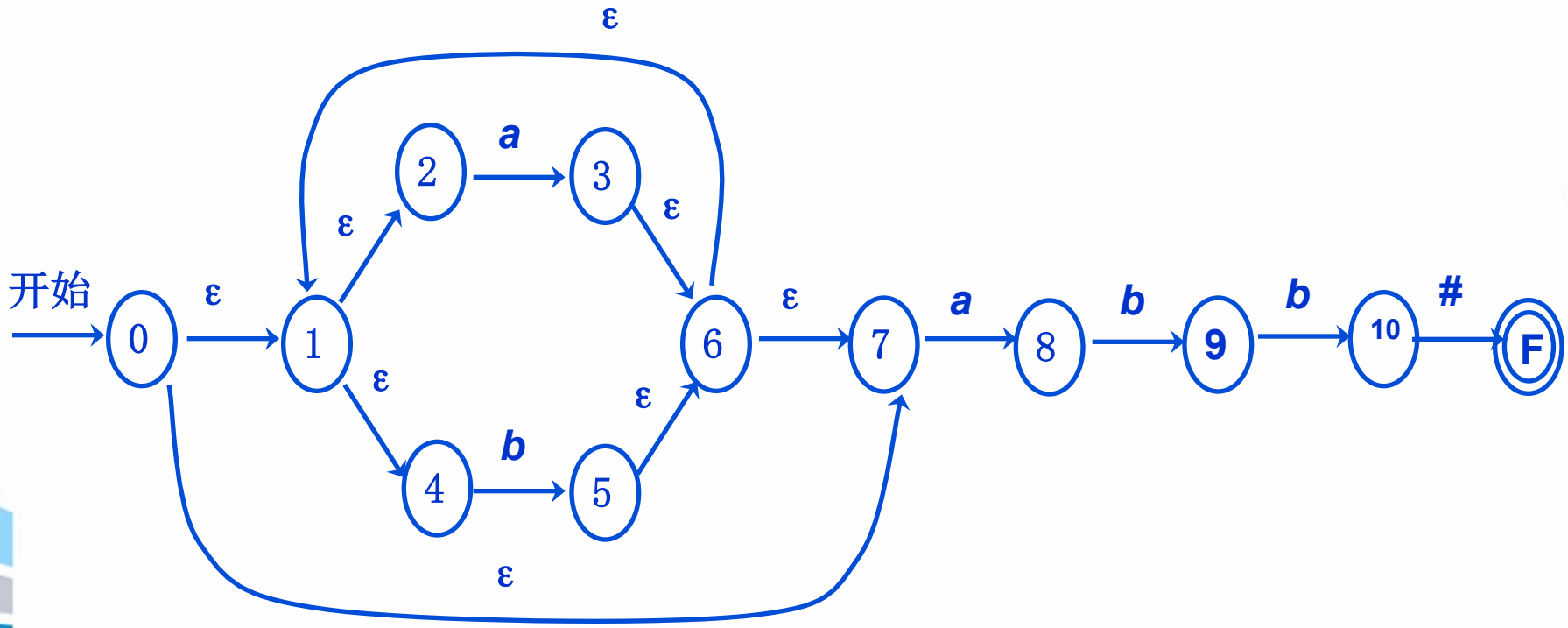
3.4 从正则式到有限自动机

- 直接从正则式到DFA，不需要构造中间的NFA
- 例：
  - (a|b)*abb#



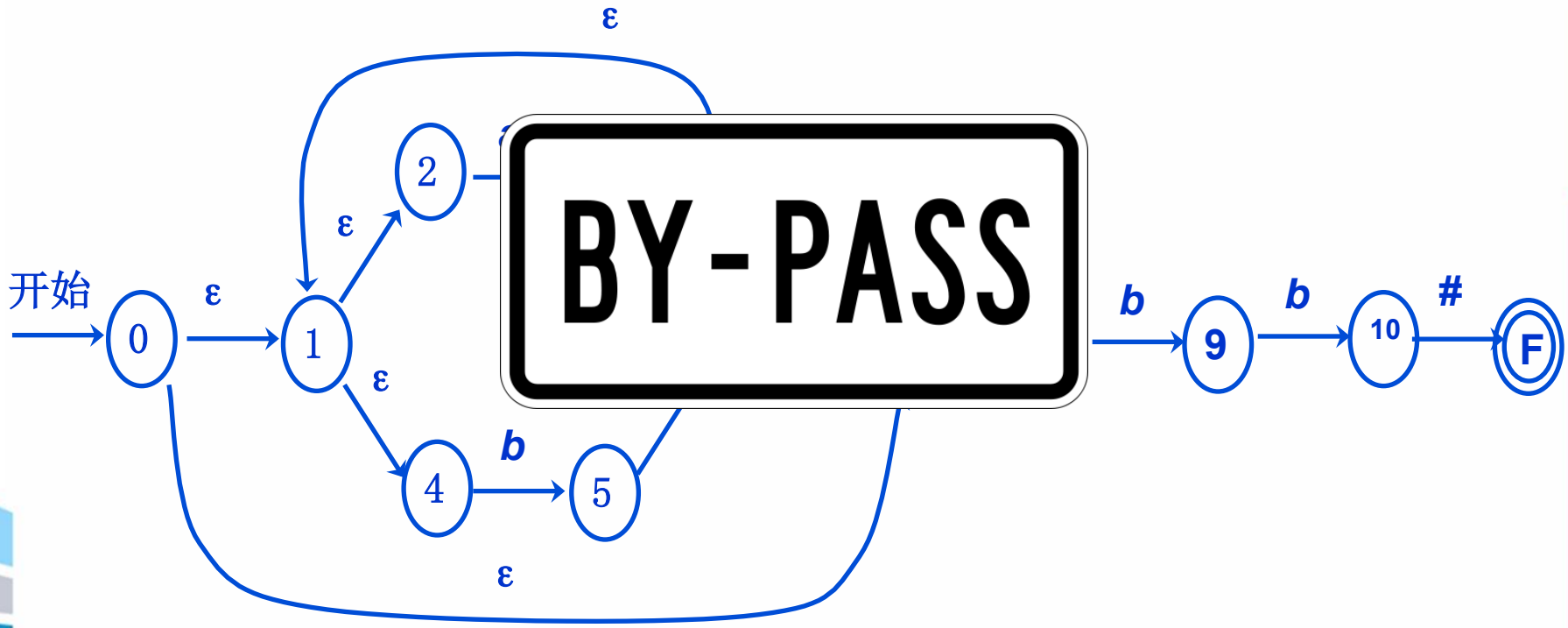
3.4 从正则式到有限自动机

- 直接从正则式到DFA，不需要构造中间的NFA
- 例：



3.4 从正则式到有限自动机

- 直接从正则式到DFA，不需要构造中间的NFA
- 例：



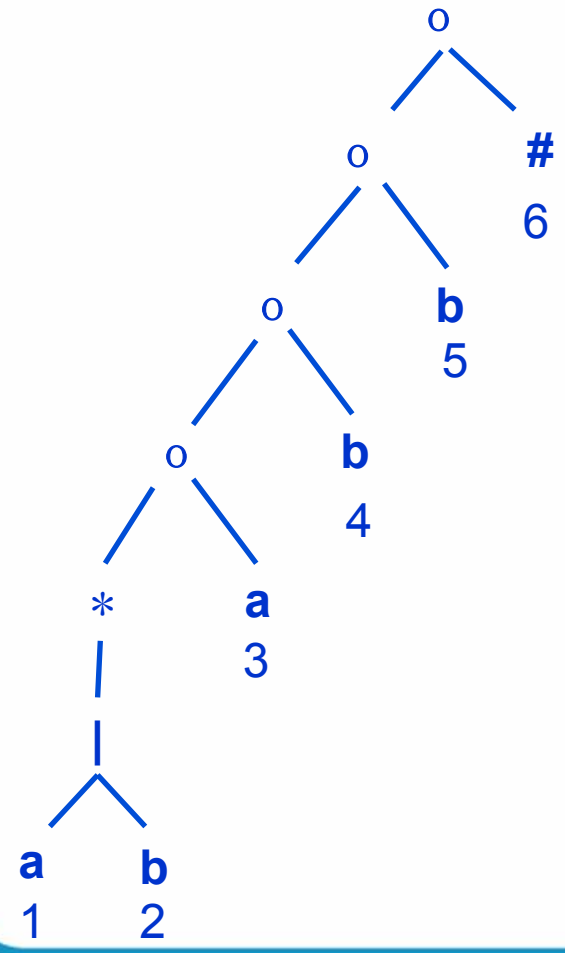
3.4 从正则式到有限自动机

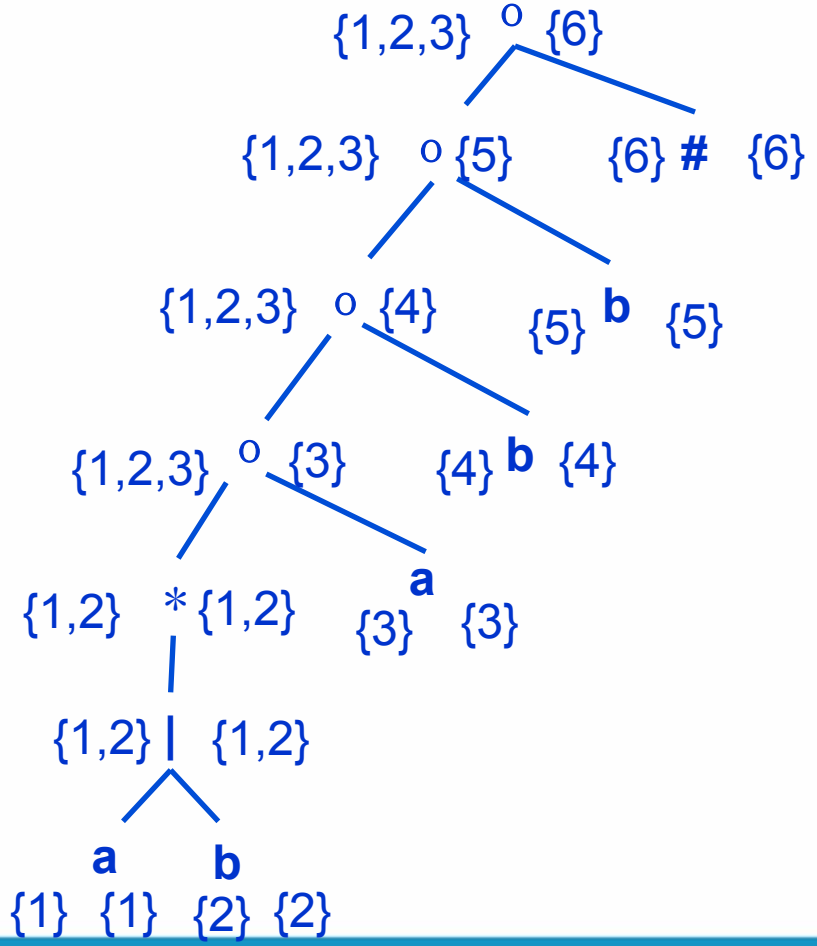
- 直接从正则式到DFA
  - nullable, firstpos, lastpos, followpos

结点*n*	***nullable*(*n*)**	***firstpos*(*n*)**
一个标号为ε的叶节点	true	Φ
一个位置为*i*的叶节点	false	{*i*}
一个**or-结点** $n = c_1 \mid c_2$	*nullable*(c_1) **or** *nullable*(c_2)	*firstpos*(c_1) ∪ *firstpos*(c_2)
一个**cat-结点** $n = c_1\ c_2$	*nullable*(c_1) **and** *nullable*(c_2)	**if**(*nullable*(c_1)) *firstpos*(c_1) ∪ *firstpos*(c_2) **else** *firstpos*(c_1)
一个**star-结点** $n = c^*$	true	*firstpos*(*c*)

3.4 从正则式到有限自动机

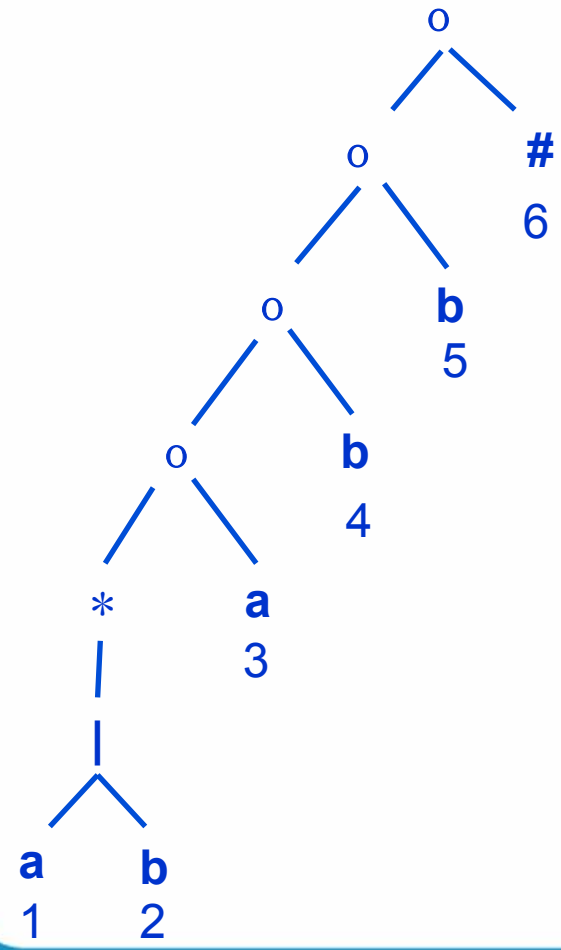
- 直接从正则式到DFA，不需要构造中间的NFA



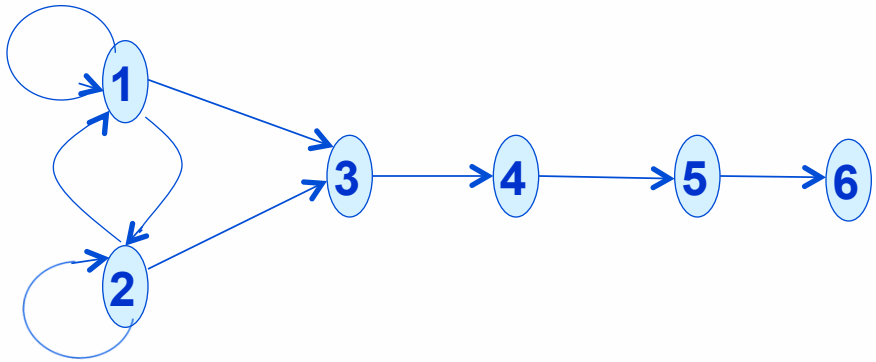


3.4 从正则式到有限自动机

- 直接从正则式到DFA，不需要构造中间的NFA



位置*n*	*followpos*(*n*)
1	{1,2,3}
2	{1,2,3}
3	{4}
4	{5}
5	{6}
6	Φ



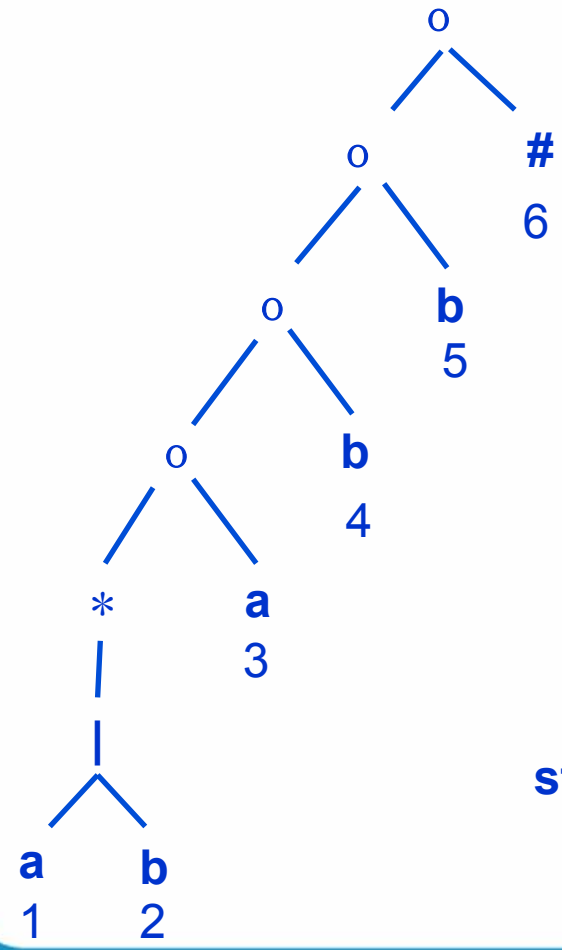
3.4 从正则式到有限自动机

- 直接从正则式到DFA，不需要构造中间的NFA

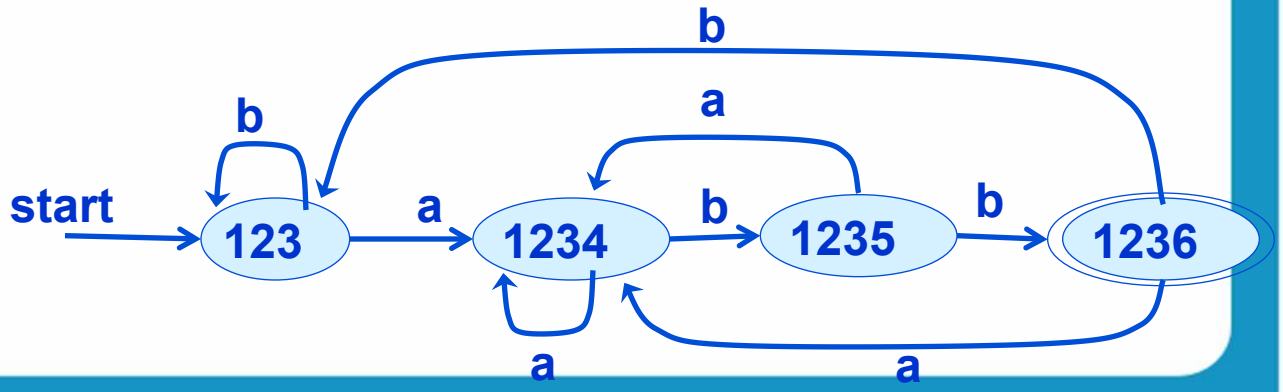
初始化*Dstates*，使之只包含未标记的状态 *firstpos*(n_0)，
　其中n_0 是(*r*)#的抽象语法树的根节点；
while(*Dstates*中存在未标记的状态*S*) **{**
　　标记*S*；
　　for(每个输入符号*a*) **{**
　　令*U*为*S*中和*a*对应的所有位置*p*的*followpos*(*p*)的并集；
　　if(*U*不在*Dstates*中)
　　将*U*作为未标记的状态加入到*Dstates*中；
　　Dtran[*S*,*a*] = *U*；
　　}
}

3.4 从正则式到有限自动机

- 直接从正则式到DFA，不需要构造中间的NFA

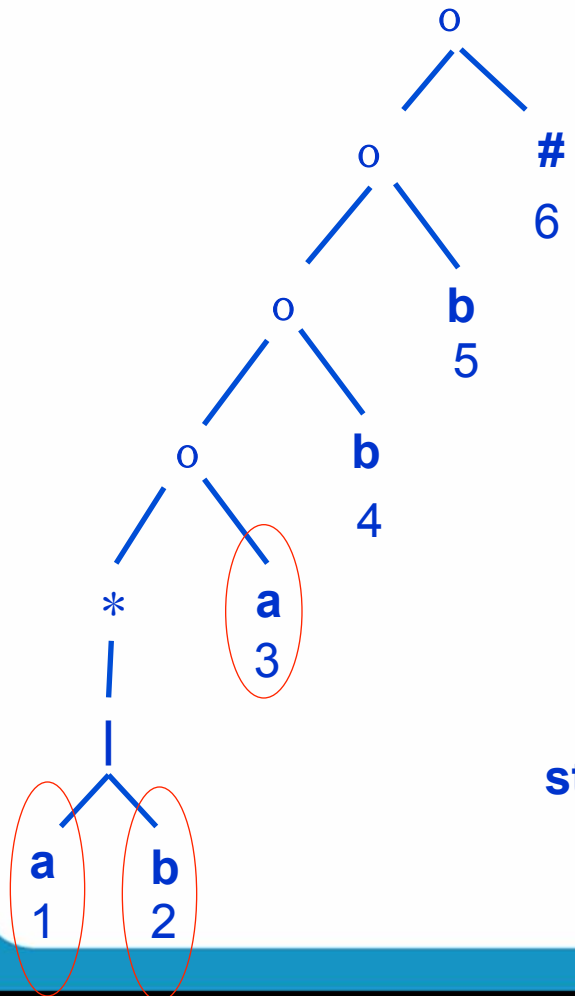


位置*n*	*followpos*(*n*)
1	{1,2,3}
2	{1,2,3}
3	{4}
4	{5}
5	{6}
6	Φ

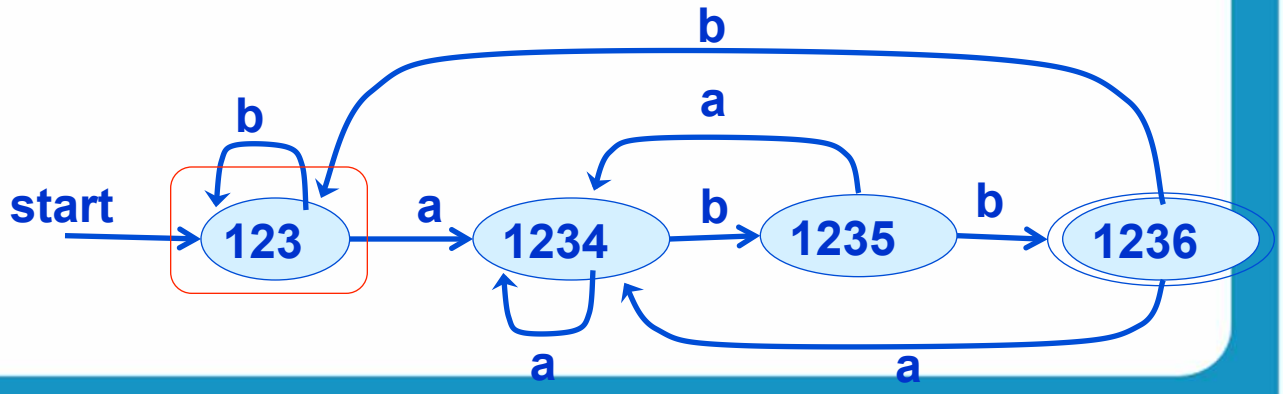


3.4 从正则式到有限自动机

- 直接从正则式到DFA，不需要构造中间的NFA

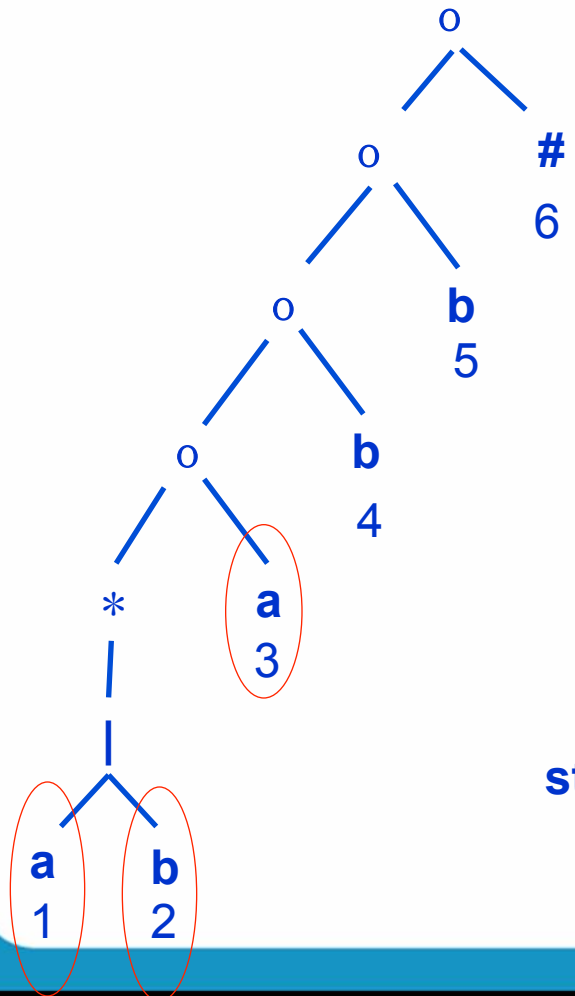


位置*n*	*followpos*(*n*)
1	{1,2,3}
2	{1,2,3}
3	{4}
4	{5}
5	{6}
6	Φ

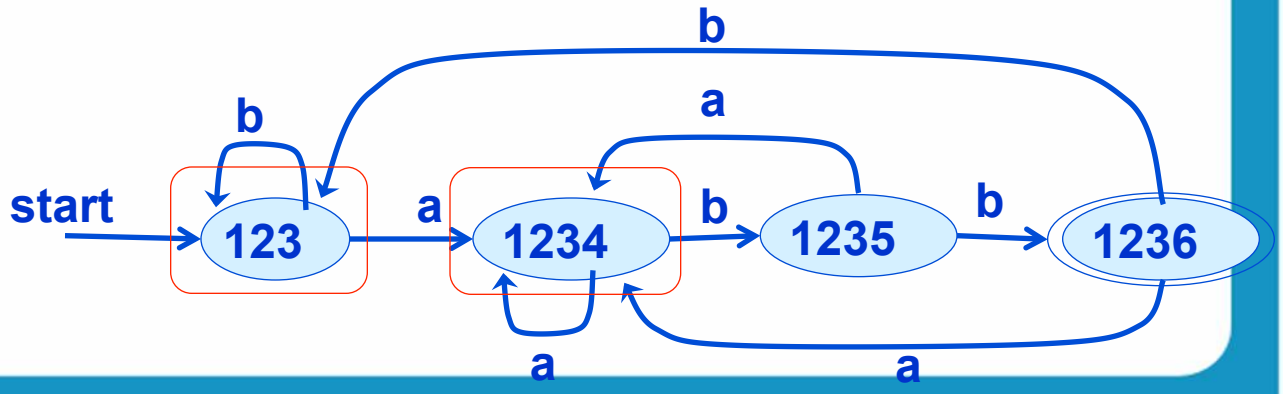


3.4 从正则式到有限自动机

- 直接从正则式到DFA，不需要构造中间的NFA

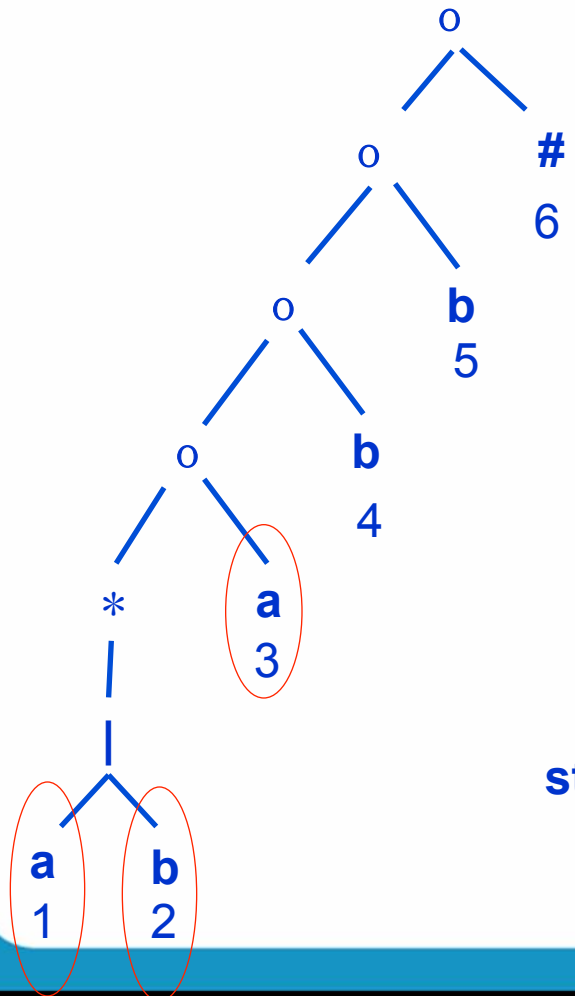


位置*n*	*followpos*(*n*)
1	{1,2,3}
2	{1,2,3}
3	{4}
4	{5}
5	{6}
6	Φ

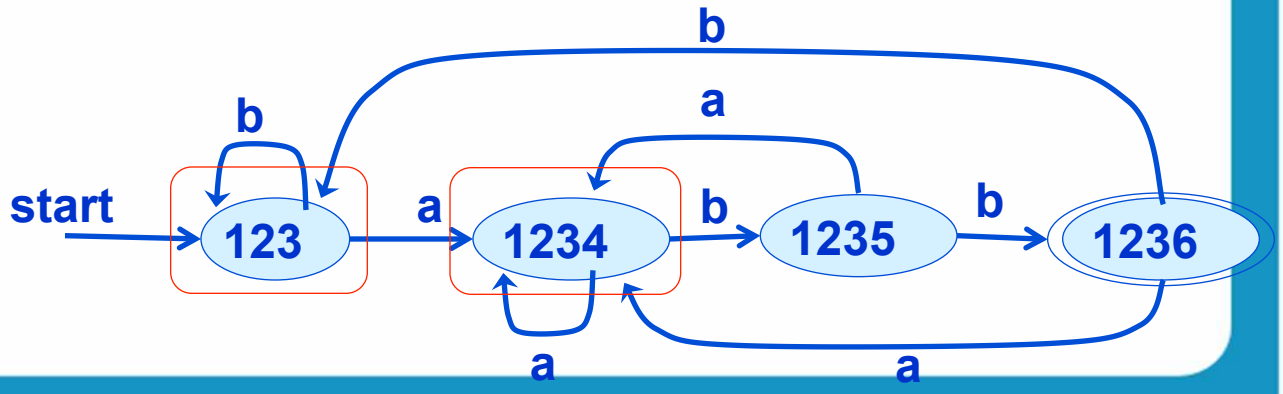


3.4 从正则式到有限自动机

- 直接从正则式到DFA，不需要构造中间的NFA

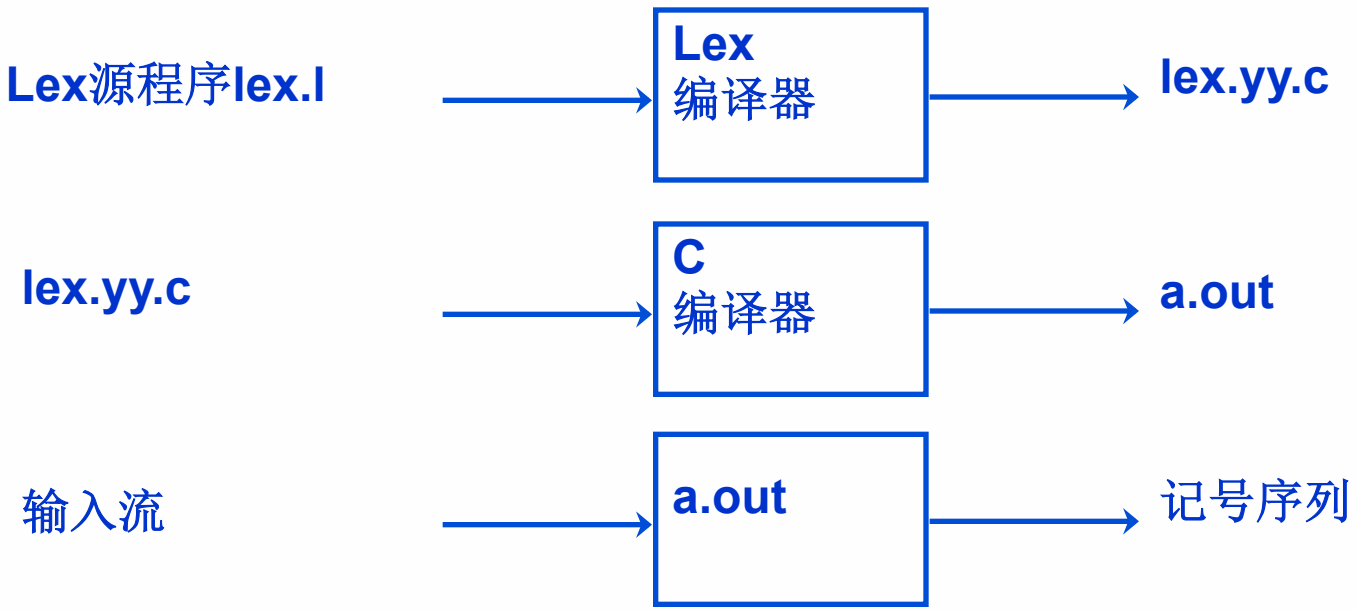


位置*n*	*followpos*(*n*)
1	{1,2,3}
2	{1,2,3}
3	{4}
4	{5}
5	{6}
6	Φ



3.5 词法分析器的生成器

- 用Lex建立词法分析器的步骤



3.5 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*}

3.5 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*/
%}
/* 正则定义 */
delim          [ \t \n ]
ws             {delim}+
letter         [A -Za - z]
digit          [0-9]
id             {letter}({letter}|{digit})*
number         {digit}+(\ .{digit}+)?(E[+\-]?{digit}+)?
```

3.5 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 = install_id ( ); return (ID);}
{number}    {yylval = install_num( );
                                  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);}
```

3.5 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针。
        yytext指向该词法单元的第一个字符，
        yyleng给出的它的长度          */
}

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

3.5 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 = install_id ( ); return (ID);}
{number}    {yylval = install_num( );
                                  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);}
```

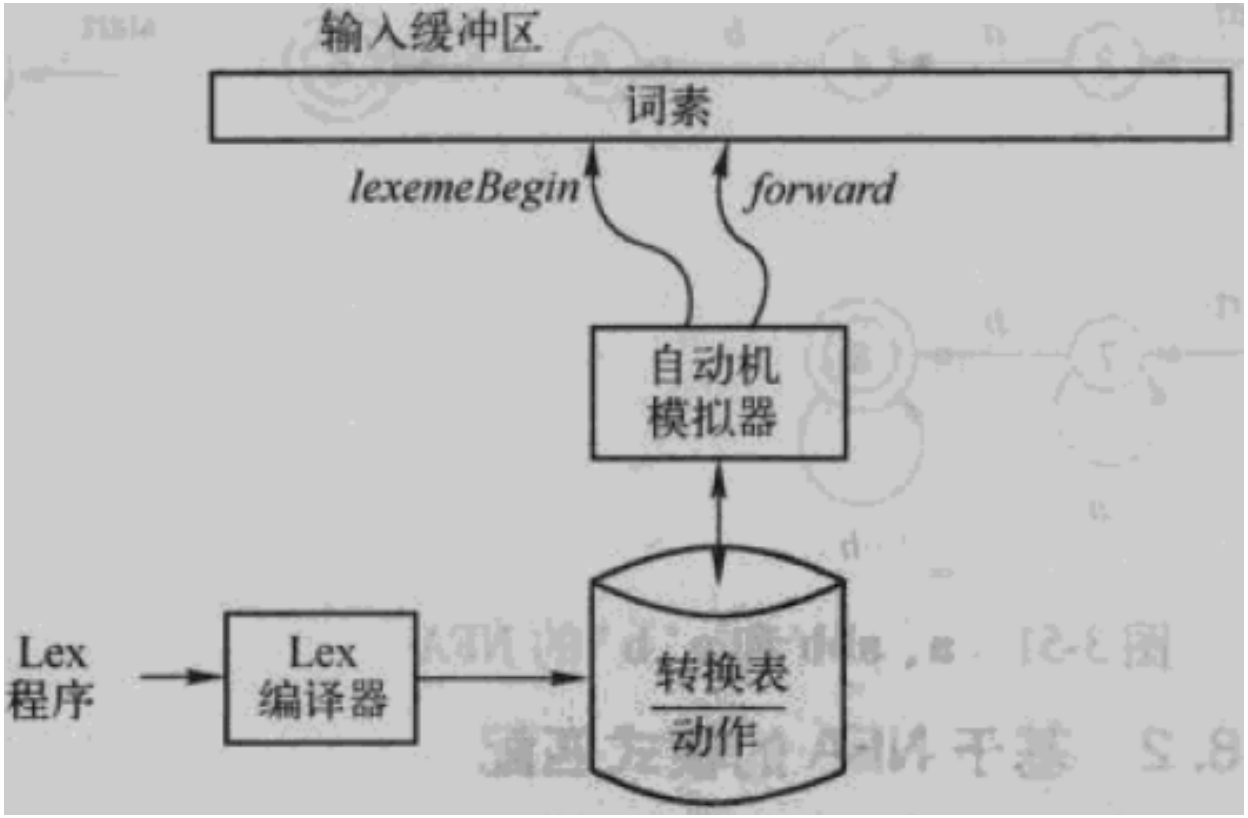
3.5 词法分析器的生成器

- 例——翻译规则部分

```
{ws}        {/* 没有动作，也不返回 */}
while       {return (WHILE);}
do          {return (DO);}
{id}        {yylval = install_id ( ); return (ID);}
{number}    {yylval = install_num( );
                                  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);}
```

3.5 词法分析器的生成器

- 词法分析器生成工具的设计

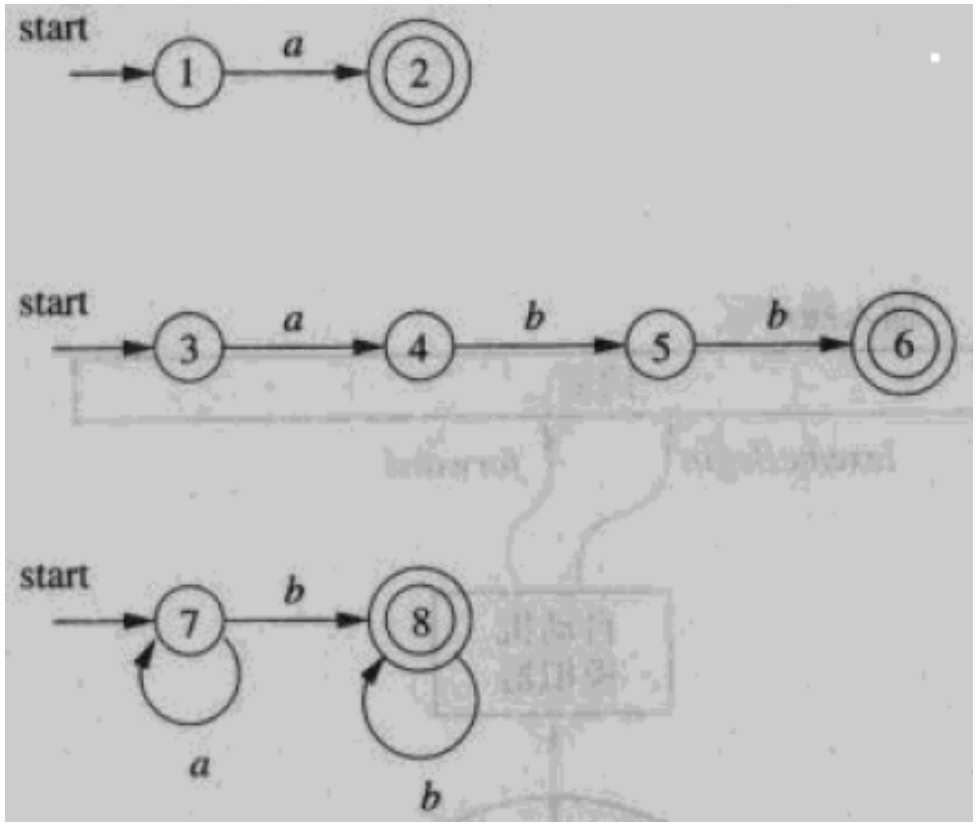


3.5 词法分析器的生成器

- 词法分析器生成工具的设计
- 例：
  - a {模式P1的动作A1}
  - abb {模式P2的动作A2}
  - $a*b^+$ {模式P3的动作A3}

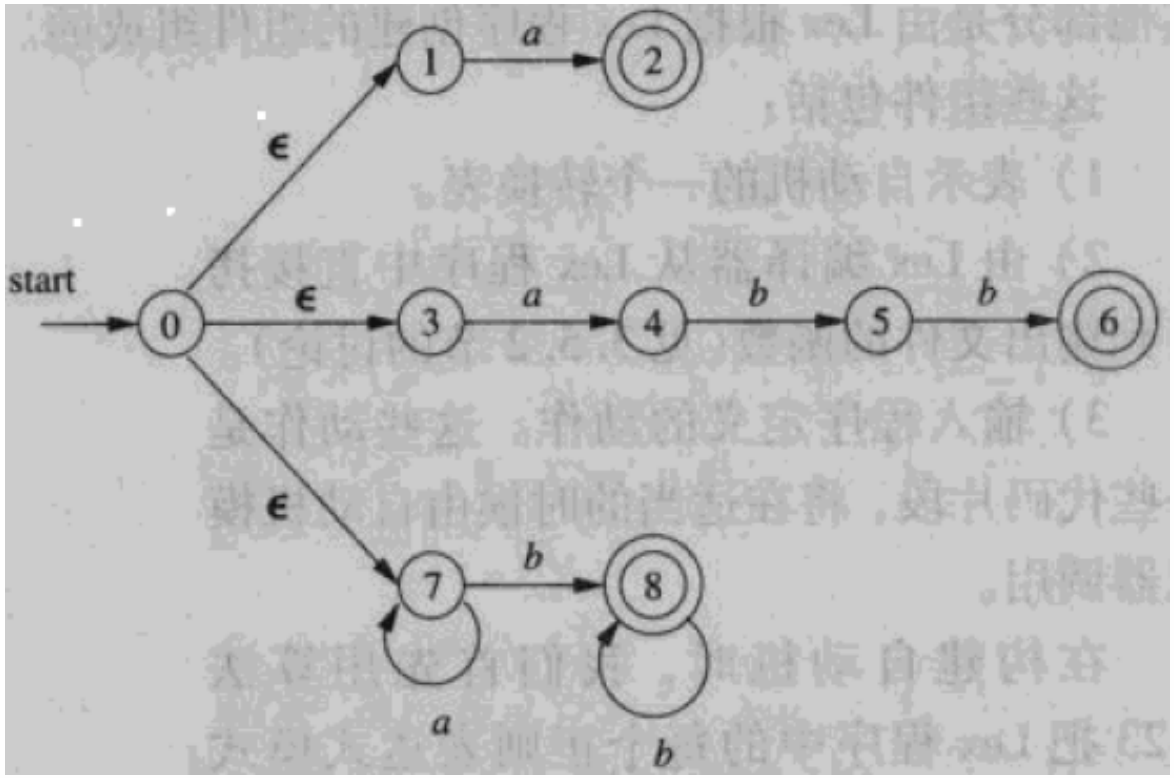
3.5 词法分析器的生成器

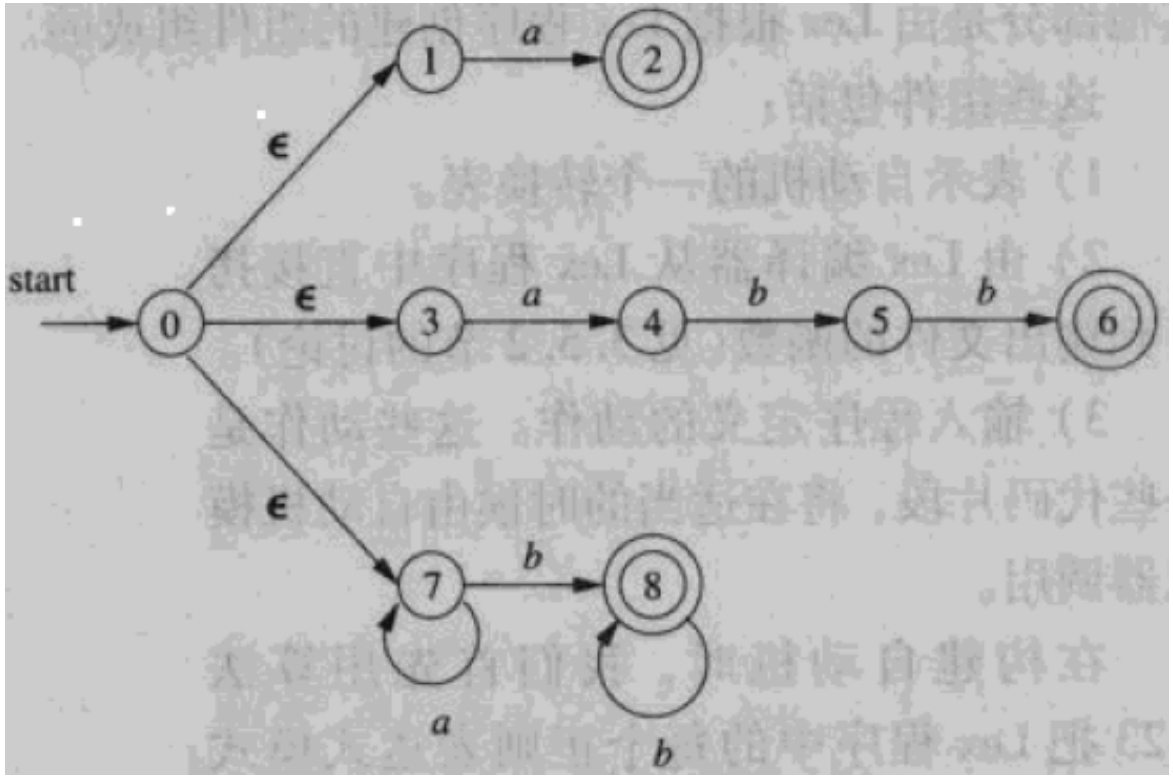
- 词法分析器生成工具的设计
- 例：
  - a
  - abb
  - a^*b^+

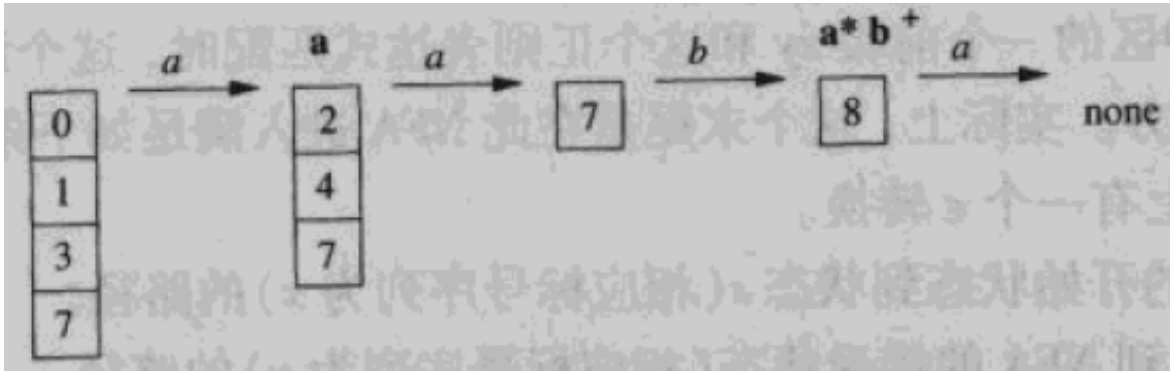


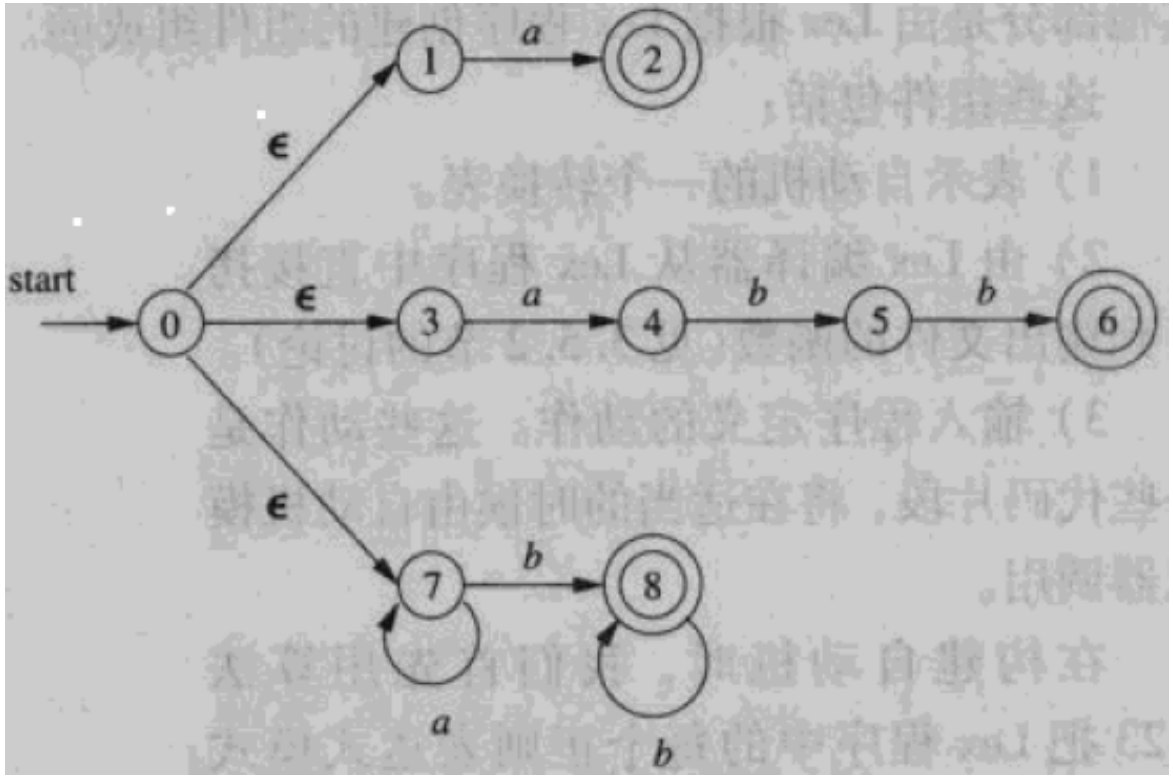
3.5 词法分析器的生成器

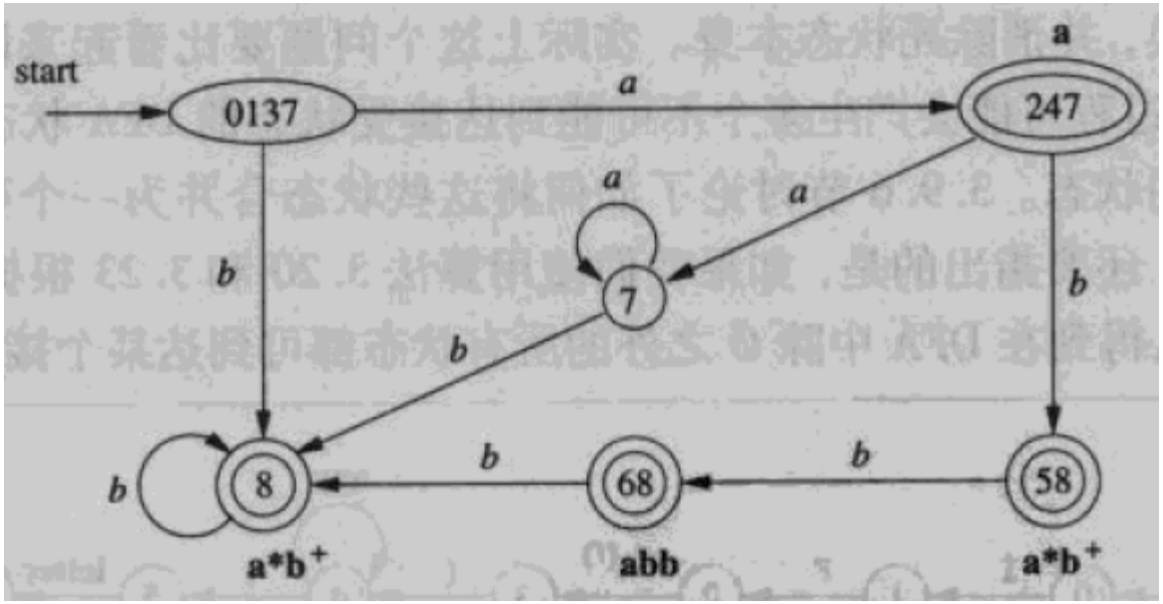
- 词法分析器生成工具的设计
- 例：
  - a
  - abb
  - a^*b^+




start
0
1
2
3
4
5
6
7
8
a
b
ϵ


0
1
3
7
a
a
2
4
7
a
7
b
a* b +
8
a
none


start
0
1
2
3
4
5
6
7
8
a
b
ϵ


start
0137
247
7
8
68
58
a
b
a*b+
abb
a*b+

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

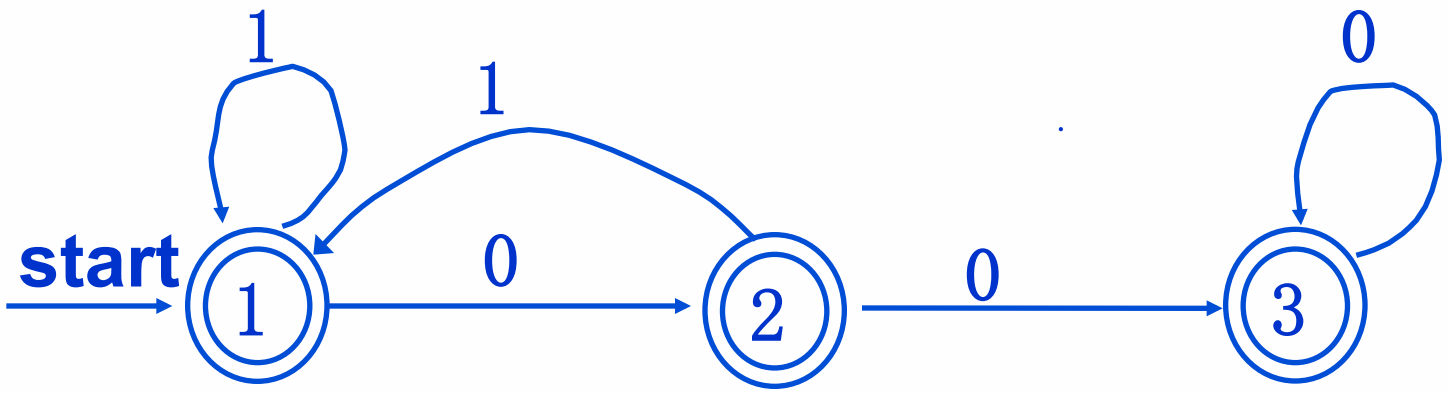
- 掌握下面涉及的一些概念，它们之间转换的技巧、方法或算法
  - 非形式描述的语言 ↔ 正则式
  - 正则式 → NFA
  - 非形式描述的语言 ↔ NFA
  - NFA → DFA
  - DFA → 最简DFA
  - 非形式描述的语言 ↔ DFA（或最简DFA）

例　题　1

- 叙述下面的正则式描述的语言，并画出接受该语言的最简**DFA**的状态转换图

  (1|01)* 0*

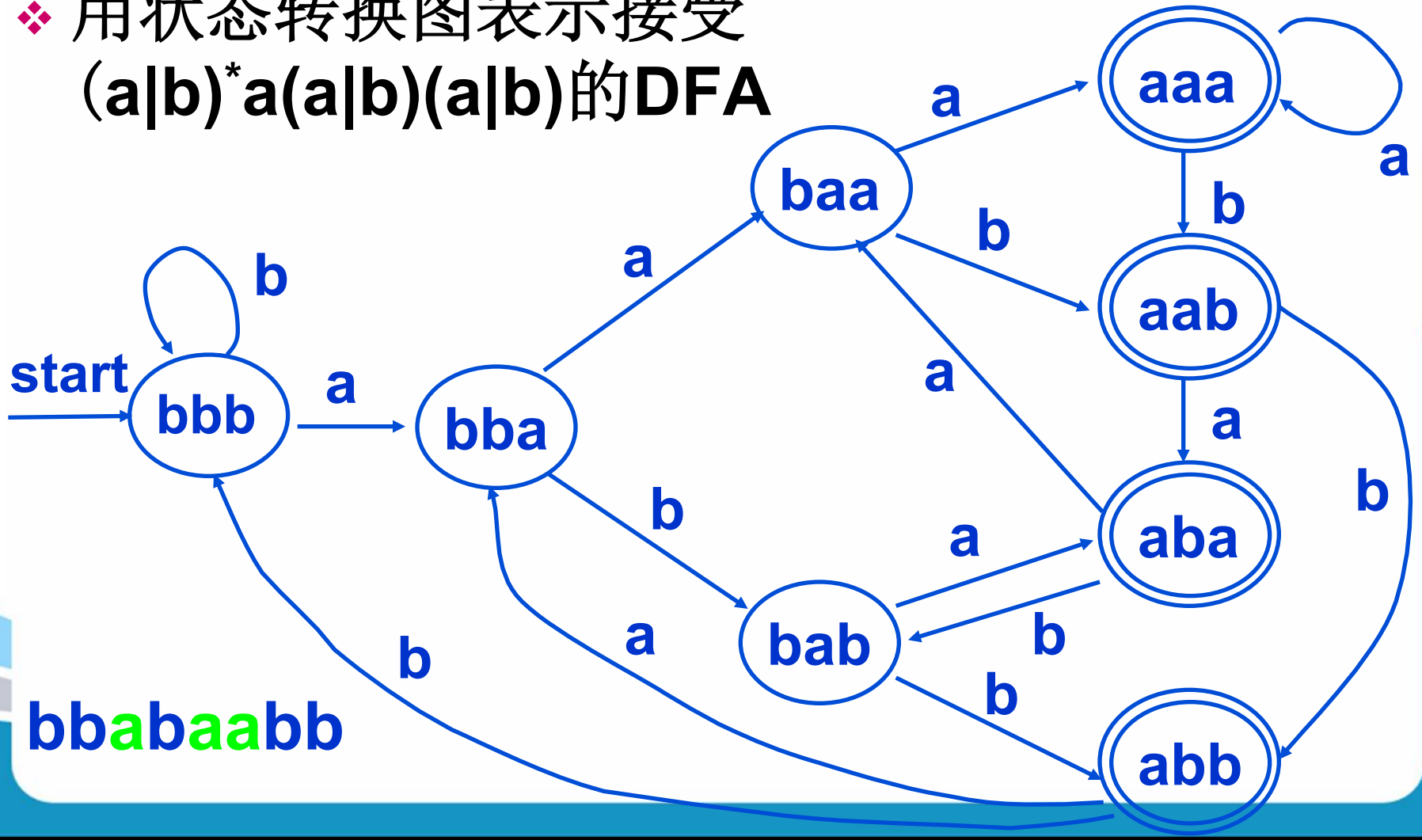
  - 描述的语言是，所有不含子串001的0和1的串



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”的正则定义

1230313571067980357901 23

answer → (0 | *no_0* 0) (*no_0* 0)* (*no_0* | ε) | *no_0*

no_0 → (1 | *no_0-1* 1) (*no_0-1* 1)* (*no_0-1* | ε) | *no_0-1*

. . .

no_0-8 → 9

将这些正则定义逆序排列就是答案