



中国科学技术大学  
University of Science and Technology of China

# 词法分析

《编译原理和技术》

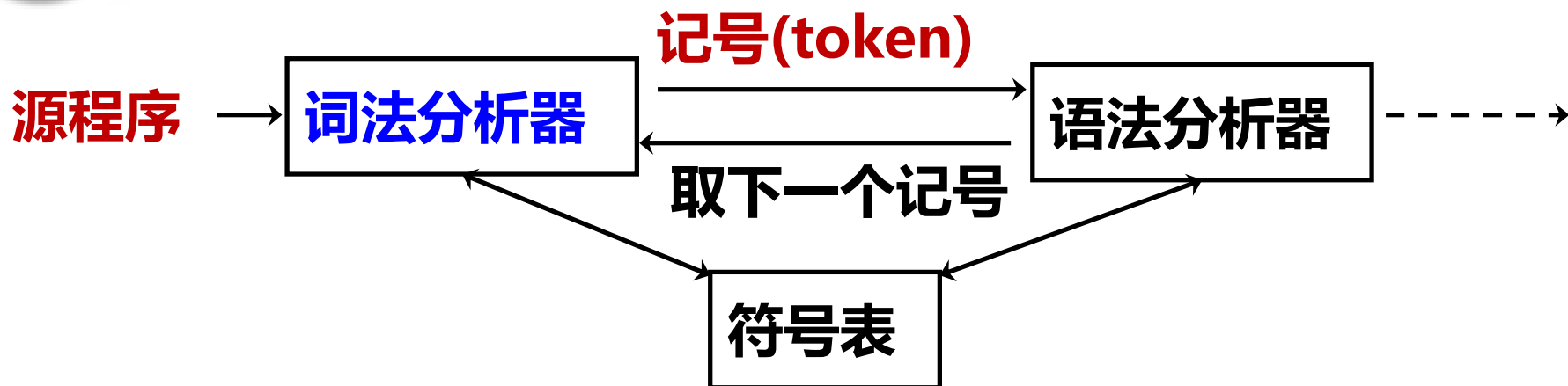
张昱

0551-63603804, [yuzhang@ustc.edu.cn](mailto:yuzhang@ustc.edu.cn)

中国科学技术大学  
计算机科学与技术学院



# 本章内容



## □ 词法分析及要解决的问题

- 向前看(Lookahead)、歧义(Ambiguities)

## □ 词法分析器的自动生成

- 词法记号的描述：正规式；词法记号的识别：转换图
- 有限自动机：NFA、DFA



## 2.1 词法记号及属性

- 词法单元(lexeme, 词素)
- 记号(token)
- 模式(pattern)



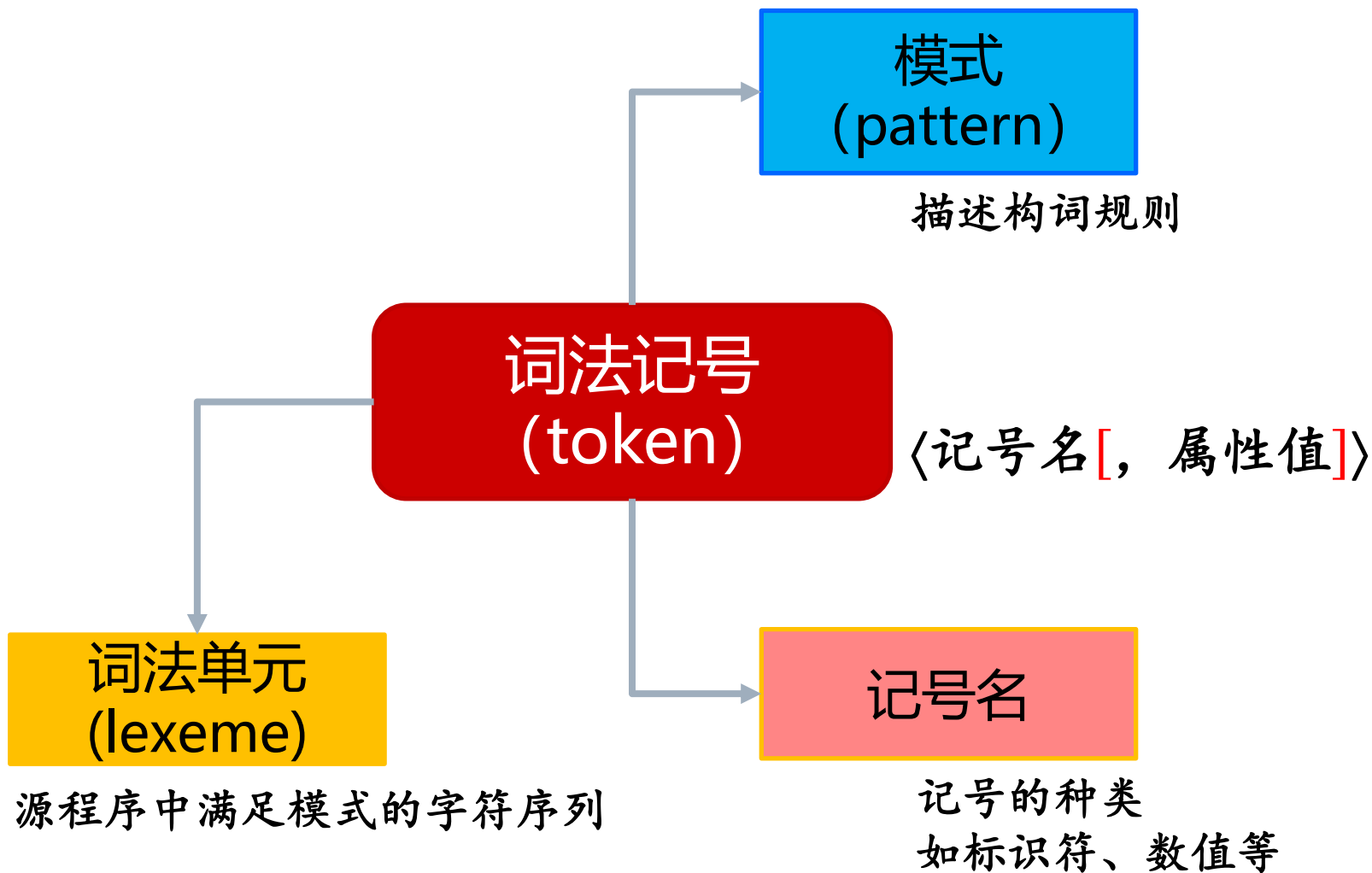
# 词法记号、词法单元、模式

记号名	词法单元实例	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
whitespace	换行符	换行符\n

空白字：如空格、\t、换行  
无意义，被丢弃，不提供给语法分析器



# 词法记号、词法单元、模式





# 词法记号的属性

**position = initial + rate \* 60** 的记号

⟨记号名[, 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⟩

符 号 表

1	position	...
2	initial	...
3	rate	...

lexeme  
(词素)



# 词法定义中的问题

## □ 关键字 ≠ 保留字

- **关键字(keyword)**: 有专门的意义和用途, 如if、else
- **保留字**: 有专门的意义, 不能当作一般的标识符使用  
例如, C语言中的关键字是保留字

## □ 历史上词法定义中的一些问题

- 忽略空格带来的困难, 例如 Fortran

DO 8 I = 3. 75    等同于    DO8I = 3. 75

DO 8 I = 3, 75

空格不是  
分隔符

- 关键字不保留

IF THEN THEN THEN=ELSE; ELSE ...



中国科学技术大学  
University of Science and Technology of China

## 2.2 词法记号的描述与识别

- 描述：正规式
- 识别：转换图





# 串和语言

## □ 术语

■ **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 $\Sigma$ 上的一个串集

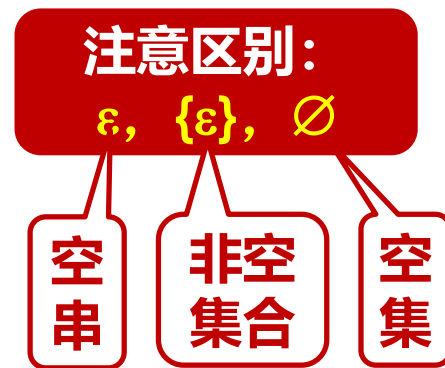
$\{\varepsilon, 0, 00, 000, \dots\}$ ,  $\{\varepsilon\}$ ,  $\emptyset$

■ **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 $\varepsilon$ ,  $s^i$ 为 $s^{i-1}s$  ( $i > 0$ )





# 串和语言

## □ 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  $\{\epsilon\}$ ,  $L^i$  是  $L^{i-1}L$

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

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

优先级:  
幂 > 连接 > 并

## □ 例

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

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



# 正规式(regular expression)

**正规式(正则表达式)用来表示简单的语言, 叫做正规集**

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

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

**优先级:**  
闭包 $*$  > 连接 > 选择  $\mid$



# 正规式举例

□  $\Sigma = \{a, b\}$

■  $a \mid b$   $\{a, b\}$

■  $(a \mid b)(a \mid b)$   $\{aa, ab, ba, bb\}$

■  $aa \mid ab \mid ba \mid bb$   $\{aa, ab, ba, bb\}$

■  $a^*$  由字母 $a$ 构成的所有串的集合

■  $(a \mid b)^*$  由 $a$ 和 $b$ 构成的所有串的集合

□ 复杂的例子

$(00 \mid 11 \mid ((01 \mid 10)(00 \mid 11)^*(01 \mid 10)))^*$

句子: **01001101000010000010111001**



# 正规定义(regular definition)

- 对正规式命名，使正规式表示简洁

$$d_1 \rightarrow r_1$$

$$d_2 \rightarrow r_2$$

...

$$d_n \rightarrow r_n$$

自底向上定义  
Bottom-up

- 各个 $d_i$  的名字都不同，是新符号，不在字母表  $\Sigma$  中
- 每个 $r_i$  都是  $\Sigma \cup \{d_1, d_2, \dots, d_{i-1}\}$  上的正规式
- C语言的标识符是字母、数字和下划线组成的串

$$\text{letter\_} \rightarrow A | B | \dots | Z | a | b / \dots | z / \_$$

$$\text{digit} \rightarrow 0 | 1 | \dots | 9$$

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



# 正规定义举例

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

**number  $\rightarrow \text{digits optional\_fraction optional\_exponent}$**



# 正规定义举例

□ 无符号数集合，例1946, 11.28, 63E8, 1.99E-6

$\text{digit} \rightarrow 0 \mid 1 \mid \dots \mid 9$

简记为[0-9] --- 字符组

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

$\text{optional\_fraction} \rightarrow \text{.digits} \mid \varepsilon$

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

$\text{number} \rightarrow \text{digits optional\_fraction optional\_exponent}$

□ 简化的表示

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

注意区分? 和 \*

? 表示0个或1个, \*表示0个或多个, +表示1个或多个



中国科学技术大学  
University of Science and Technology of China

## 2.2 词法记号的描述与识别

- 描述：正规式
- 识别：转换图





# 词法分析

## □ 词法分析

- 从左到右读取输入串，每次识别出一个token实例
- 可能需要“lookahead”来判断当前是否是token的结尾或下一个token的开始（尤其是在Fortran语言中）

D	O		8		I		=		3	,	2	5
---	---	--	---	--	---	--	---	--	---	---	---	---

↑  
lexeme的开始

↑  
lookahead

DO 8 I = 3

无法判断，需  
lookahead

DO 8 I = 3.25

点→DO8I为id

DO 8 I = 3, 25

逗号→DO为  
关键字



# 词法分析

## □ 词法分析

- 从左到右读取输入串，每次识别出一个token实例
- 可能需要“lookahead”来判断当前是否是一个token的结尾、下一个token的开始（尤其是在Fortran语言中）
- 可能需要结合上下文来识别是否是关键字（当关键字不是保留字时）

```
if (then .gt. else) then  
    then = else  
else  
    else = then  
endif
```

需要结合上下文  
识别是否是关键字



# 词法分析器：实现

## 一个词法分析器的实现必须做两件事

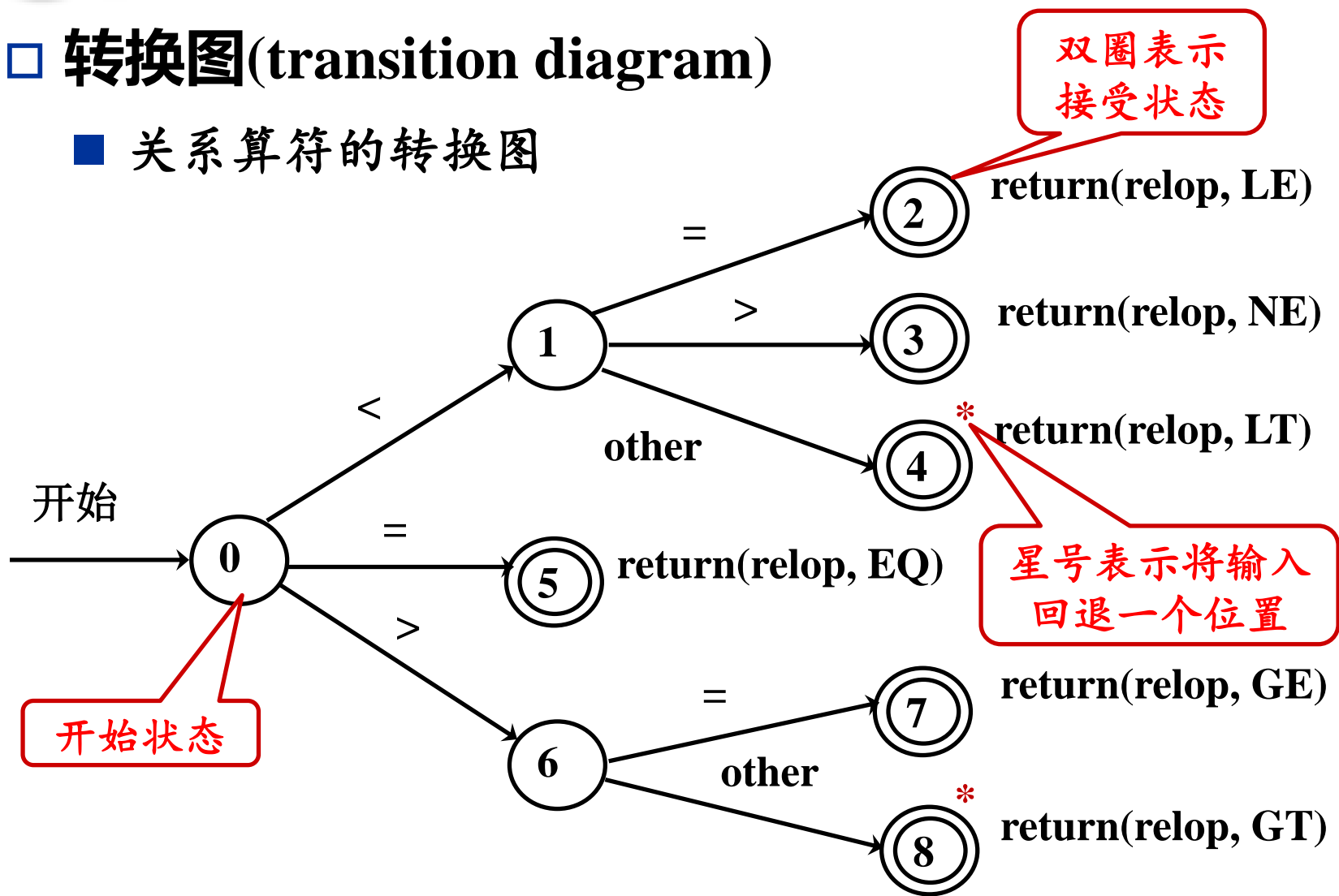
1. 识别子串并对应到 tokens
2. 返回token的值或词法单元(lexeme, 词素)
  - 词法单元是子串 (token的实例)



# 词法记号的识别：状态转换图

## □ 转换图(transition diagram)

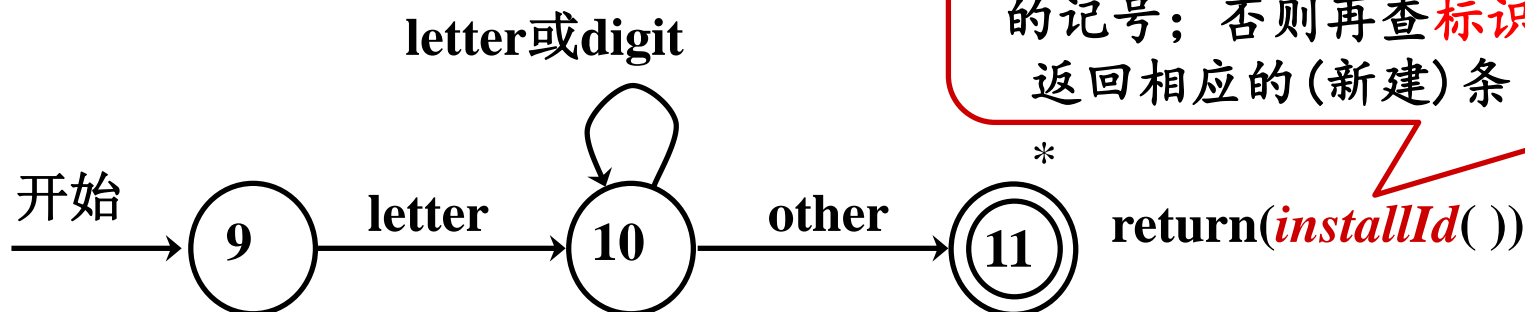
### ■ 关系算符的转换图





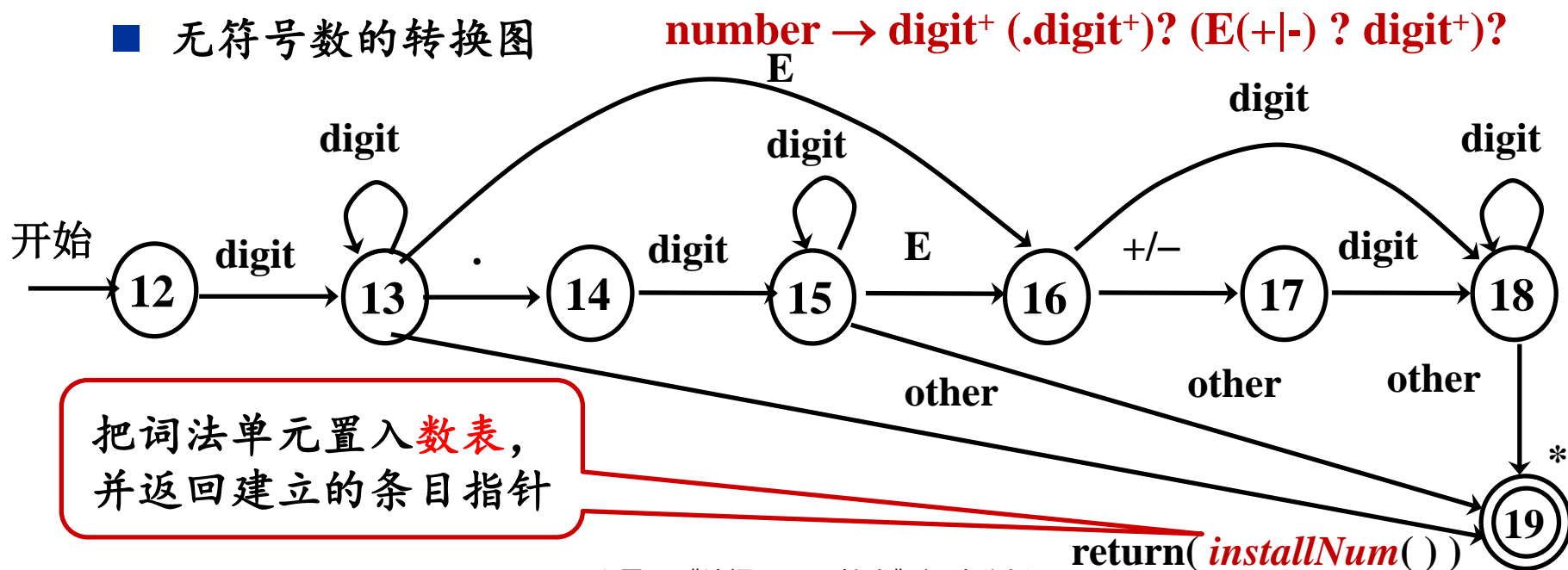
# 转换图

## 标识符和关键字的转换图



先查看**关键字表**，若当前词法单元构成关键字，则返回相应的记号；否则再查**标识符表**，返回相应的(新建)条目指针

## 无符号数的转换图



把词法单元置入**数表**，并返回建立的条目指针

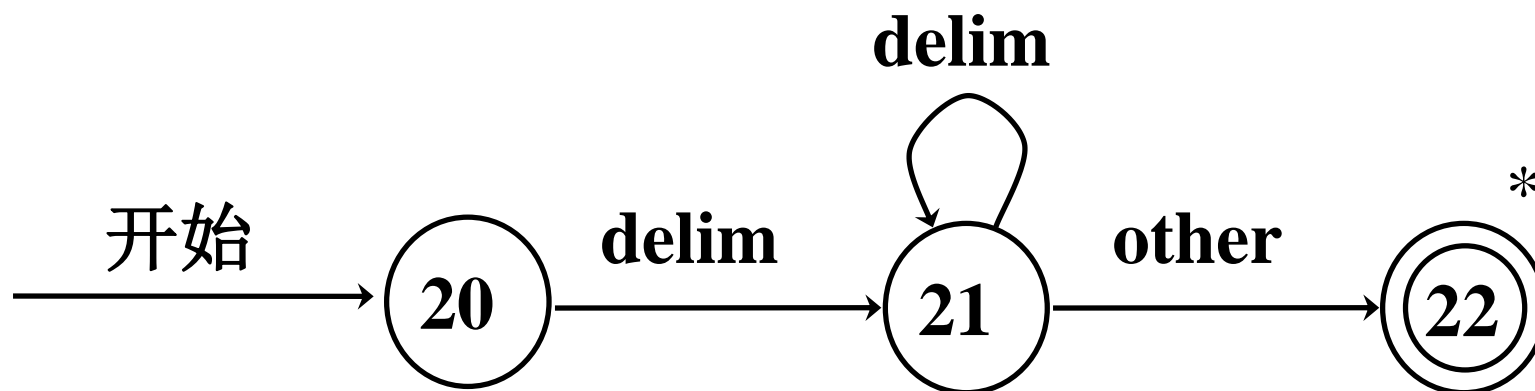


# 转换图

## □ 空白的转换图

**delim** → blank | tab | newline

**ws** → **delim**+

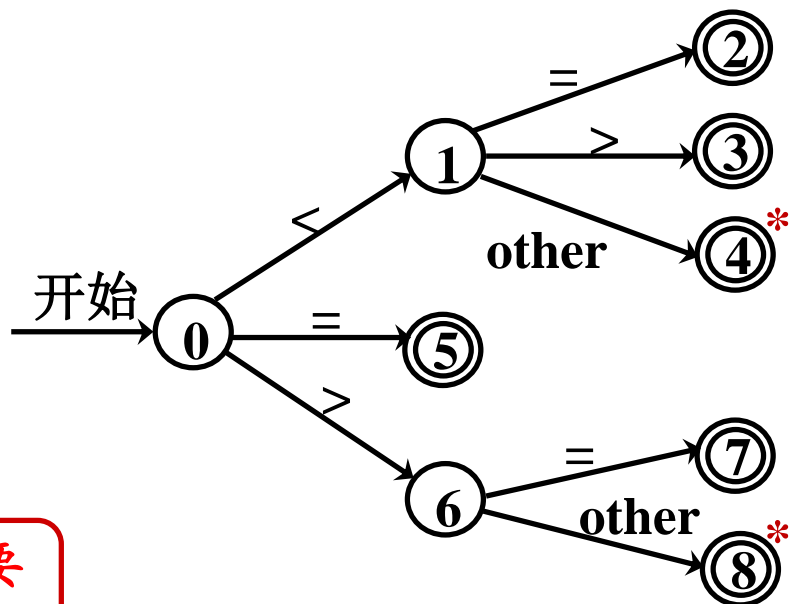




# 基于转换图的词法分析

## ■ 例：relop的转换图的概要实现

```
TOKEN getRelop() {  
    TOKEN retToken = new(RELOP);  
    while (1) {  
        switch (state) {  
            case 0: c = nextChar();  
                if (c == '<') state = 1;  
                else if (c == '=') state = 5;  
                else if (c == '>') state = 6;  
                else fail();  
                break;  
            case 1: ...  
            ...  
            case 8: retract();  
                retToken.attribute = GT;  
                return(retToken);  
        }  
    }  
}
```



出错处理，要  
能从错误恢复

回退



# 词法分析中的冲突及解决

**R = Whitespace | Integer | Identifier | ‘+’**

分析 “foo+3”

→ “f” 匹配 **R**, 更精确地说是 **Identifier**

→ 但是 “fo” 也匹配 **R**, “foo” 也匹配, 但 “foo+” 不匹配

如何处理输入? 如果

$x_1 \dots x_i \in L(R)$  并且  $x_1 \dots x_k \in L(R)$ , 假设  $i < k$

则可按“**Maximal munch**”规则来选择:

→ 选择匹配 **R** 的最长前缀, 即选择  $x_1 \dots x_k$

最长匹配规则在实现时: lookahead, 不符合则回退





# 词法分析：分类的不确定性

**$R = \text{Whitespace} \mid \text{'new'} \mid \text{Integer} \mid \text{Identifier}$**

分析 “**new** foo”

- “**new**” 匹配  **$R$** , 更精确地说是 **‘new’**
- 但是也匹配 **Identifier**, 此时该选哪个?

一般地, 如果  **$x_1 \dots x_i \in L(R_j)$**  和  **$x_1 \dots x_i \in L(R_k)$**

**规则: 选择先列出的模式 ( $R_j$  如果  $j < k$ )**

- 必须将 **‘new’** 列在 **Identifier** 的前面



# 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`

注：数字串长度不小于1

- 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



# 例题 1

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

解答：

$answer \rightarrow (0 \mid no\_0 \ 0) (no\_0 \ 0)^* (no\_0 \mid \varepsilon) \mid no\_0$

$no\_0 \rightarrow (1 \mid no\_0-1 \ 1) (no\_0-1 \ 1)^* (no\_0-1 \mid \varepsilon) \mid no\_0-1$

...

$no\_0-8 \rightarrow 9$

将这些正规定义逆序排列就是答案



## 例题2

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      此处遗漏了分号
    else
        /* else part */
        return gcd(q, p%q);
}
```



## 例题2

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);
}
```



中国科学技术大学  
University of Science and Technology of China

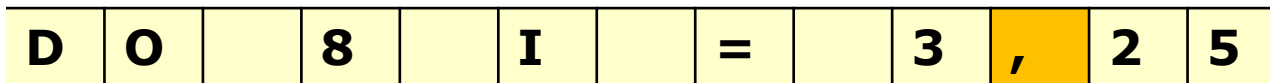
## 2.3 词法分析器的生成器

□ Lex: flex、jflex、antlr



# 词法分析器的生成器

Input buffer



lexeme的开始

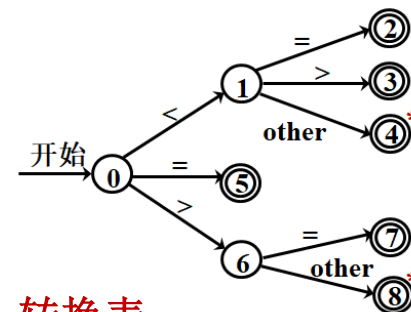
lookahead

Automation simulator

Lex program

Lex compiler

Translation Table  
转换表



转换表

状态	<	=	>	Other
0	1	5	6	Error
1	4	2	3	4
2				



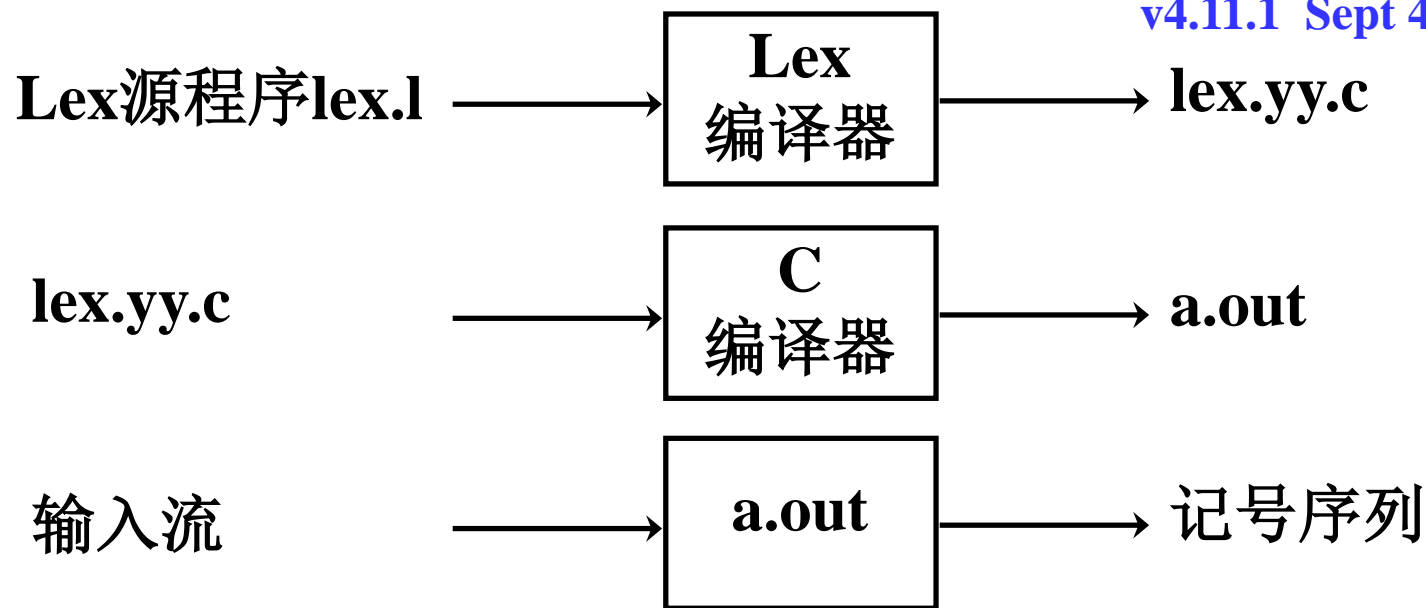
# 用Lex建立词法分析器

词法分析器 — Lexical analyzer, scanner

生成器 — generator

**Flex** <https://github.com/westes/flex> ; **JFlex** <http://jflex.de/>

**ANTLR** <https://www.antlr.org/> v4.8 Jan 16, 2020 v4.9.2 Mar 11, 2021  
v4.11.1 Sept 4, 2022







# Lex程序

## □ 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 $n$ }



# Lex文件举例—声明部分

%{

/\* 常量LT, LE, EQ, NE, GT, GE,  
WHILE, DO, ID, NUMBER, RELOP的定义\*/

%{和%}包含的代码  
将直接复制到生成的  
分析器源程序文件中

%}

/\* 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}+)?



# Lex文件举例—翻译规则部分

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



# Lex文件举例—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 */
```

```
}
```



# ANTLR 的语法文件 .g4

\$ antlr4 myg.g4

使用说明: [lexer-rules](#), [tool-options](#)

## □ 格式

```
grammar MyG;  
options { ... }  
import ... ;  
tokens { ... }  
@actionName { ... }  
ruleName : <stuff> ;  
.....
```

正规定义, DIGIT不是记号

模式定义, 词法状态

点通配任一字符

## □ ruleName

■ 词法: 大写 字母开头

■ 语法: 小写 字母开头

## □ 纯词法分析器

```
lexer grammar MyG;
```

## □ 词法规则

```
INT : DIGIT+ ;
```

```
fragment DIGIT : [0-9] ;
```

```
LQUOTE : '"' -> more, mode(STR) ;
```

```
mode STR;
```

```
STRING : '"' -> mode(DEFAULT_MODE);
```

```
TEXT : . -> more
```

取另一个记号但  
不抛出当前文本

模式调用



# ANTLR: Lexer规则中的命令

## □ 命令格式

TokenName : 选项1|...|选项N [-> 命令名 [(参数)]];

## □ 命令

- **skip**: 不返回记号给parser, 如识别出空白符或注释
- **more**: 取另一个记号但不抛出当前的文本
- **type(T)**: 设置当前记号的类型
- **channel(C)**: 设置当前记号的通道, 缺省为  
Token.DEFAULT\_CHANNEL(值为0);  
Token.HIDDEN\_CHANNEL(值为1)
- **mode(M)**: 匹配当前记号后, 切换到模式M
- **pushMode(M)**: 与mode(M)类似, 但将当前模式入栈
- **popMode**: 从模式栈弹出模式, 使之成为当前模式



## 2.4 有限自动机

- 描述分析器：NFA、DFA  
(对转换图的形式定义和分类)



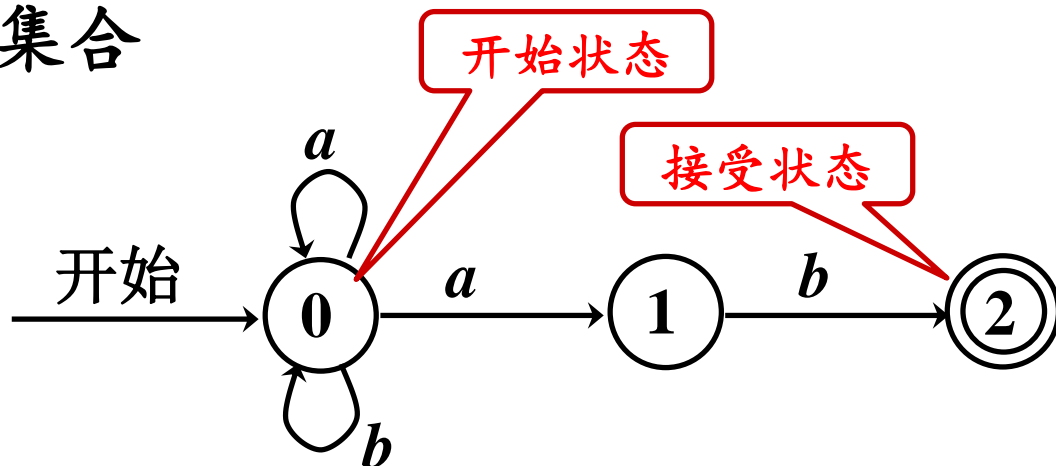
# 有限自动机

## □ 不确定的有限自动机 (NFA)

一个数学模型，它包括： (nondeterministic finite automaton)

- 1、有限的状态集合  $S$
- 2、输入符号集合  $\Sigma$
- 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





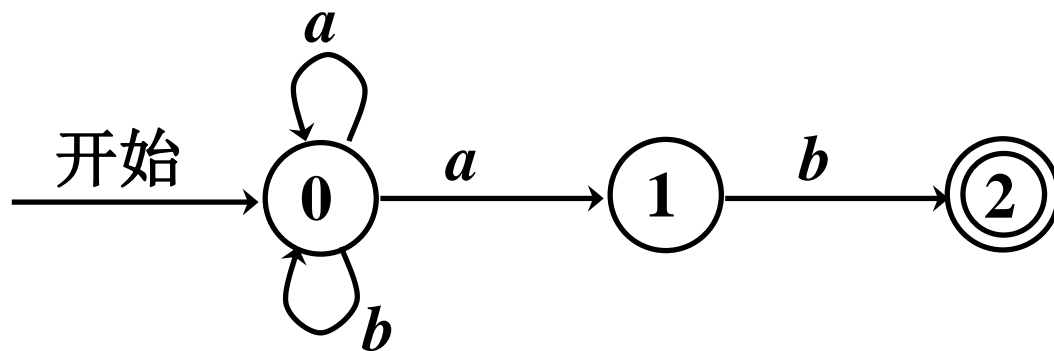


# 转换表

## □ NFA的转换表

	输入符号	
	$a$	$b$
0	$\{0, 1\}$	$\{0\}$
1	$\emptyset$	$\{2\}$
2	$\emptyset$	$\emptyset$

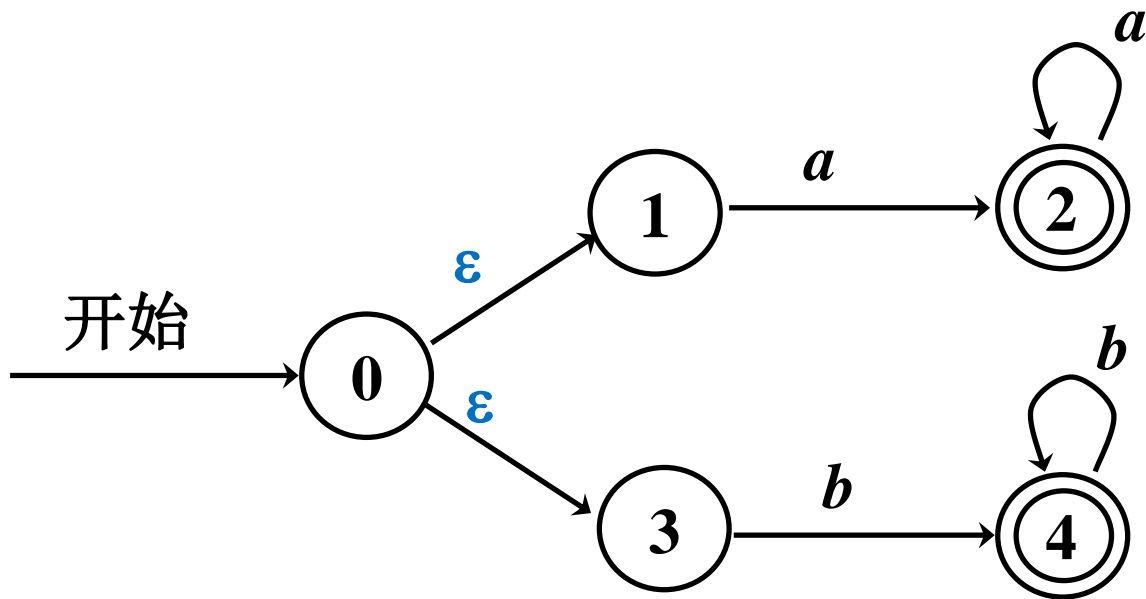
识别语言  
 $(a|b)^*ab$   
的NFA





# 例题3

□ 识别 $aa^* | bb^*$ 的NFA





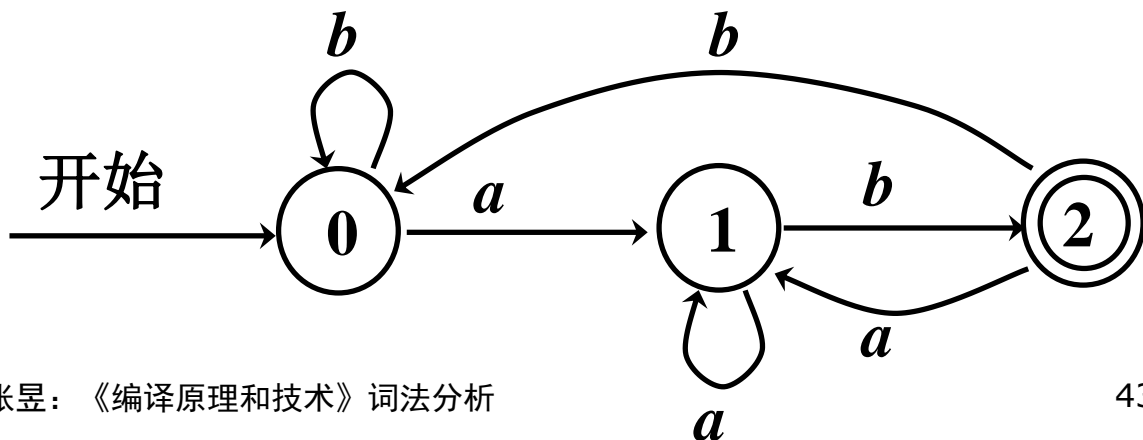
# 确定的有限自动机

## □ 确定的有限自动机 (DFA)

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 $\Sigma$
- 3、转换函数 $move : S \times \Sigma \rightarrow S$ ，且可以是部分函数
- 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DFA





# NFA vs. DFA

## □ 主要差异在转换函数

■ NFA:  $move : S \times (\Sigma \cup \{\epsilon\}) \rightarrow P(S)$

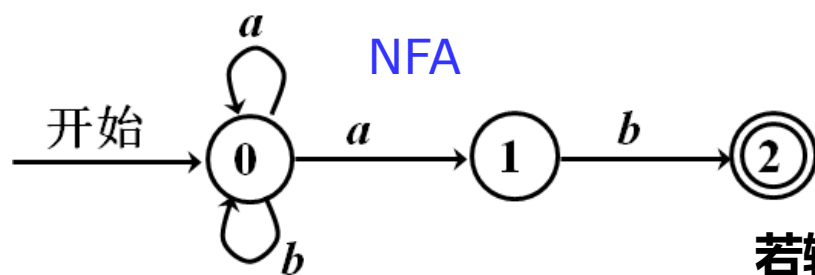
效率低

□ 要识别一个token, 需要对多种可能的路径试探+失败回退

■ DFA:  $move : S \times \Sigma \rightarrow S$ , 可以是部分函数

快速高效

□ 对于面临的 $\Sigma$ 中的符号, 状态转换是明确的



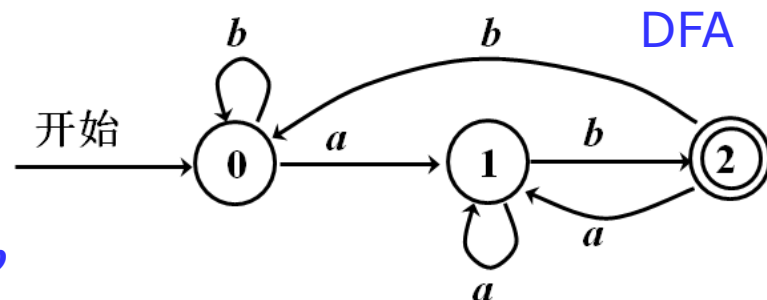
识别语言  
 $(a|b)^*ab$

若输入串是  $ab$

0-(a)→0-(b)→0-结束符  
0←回退b

0←回退a

0-(a)→1-(b)→2-结束(接受状态)



0-(a)→1-(b)→2-结束(接受状态)

但是, 由正规式不易构造DFA



## 例题4

构造一个DFA，它能识别 $\{0,1\}$ 上能被5整除的二进制数。

**解答**

	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	$10 \times 2 + 1 = 21$

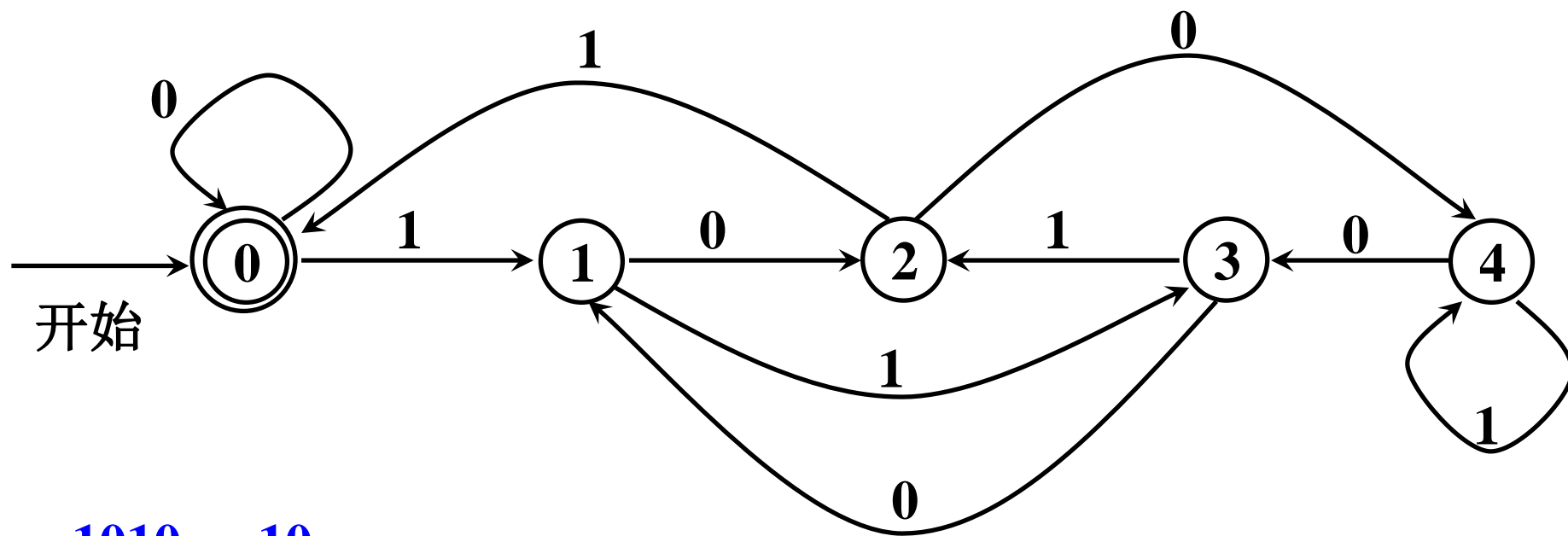
引入5个状态即可，分别代表已读部分的值除以5的余数



## 例题4

构造一个DFA，它能识别 $\{0,1\}$ 上能被5整除的二进制数。

解答



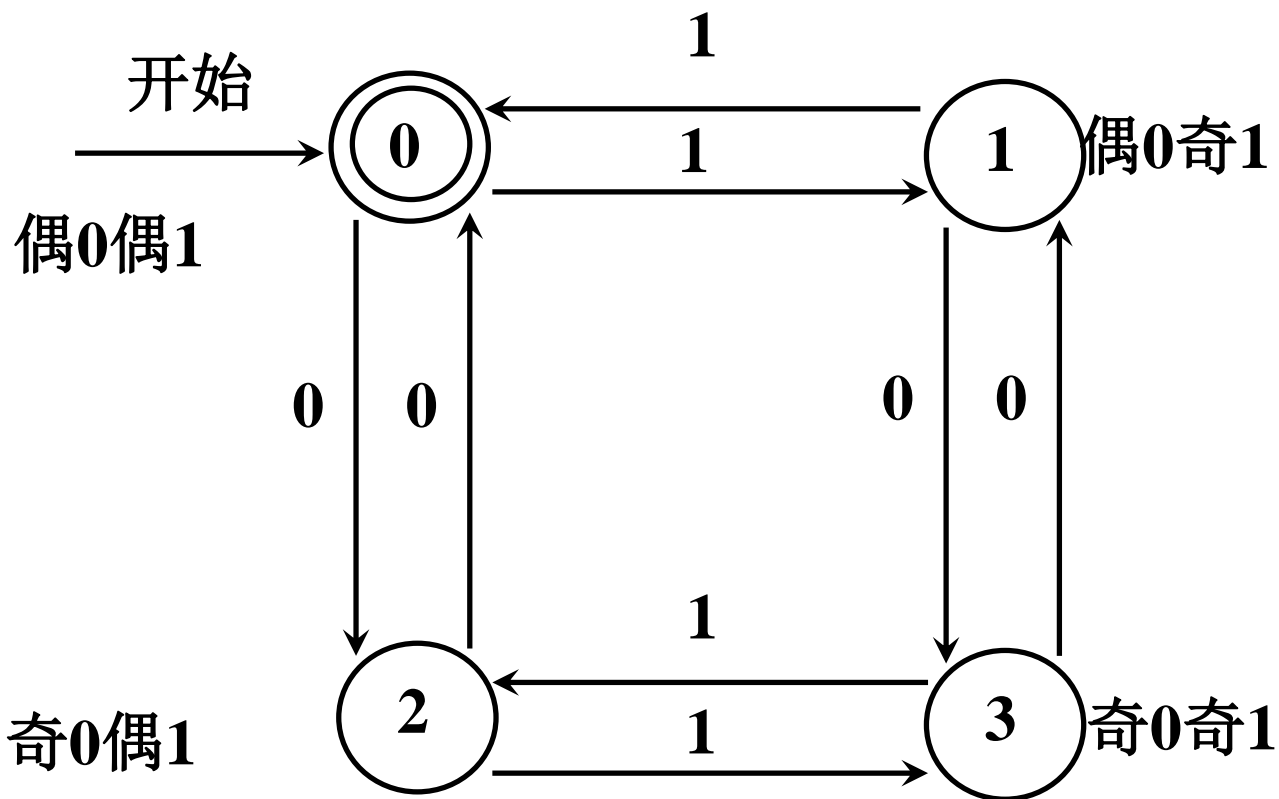
$$1010_2 = 10_{10}$$

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



## 例题5

构造一个DFA，它能接受0和1的个数都是偶数的字符串。





中国科学技术大学  
University of Science and Technology of China

## 2.5 从正规式到有限自动机

### □ 分析器的自动构造

■ 正规式  $\rightarrow$  NFA  $\rightarrow$  DFA  $\rightarrow$  化简的DFA  
采用语法制导的算法来构造NFA





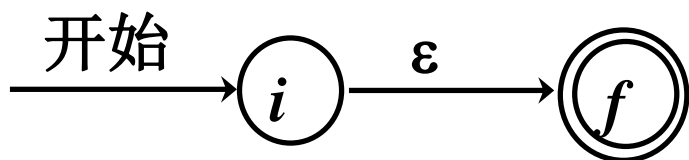
# 词法分析器的自动生成技术

- 正规式：描述语言的词法
- 有限自动机：刻画词法分析的实现
  
- 词法分析器自动生成的主要过程
  - 正规式→NFA（语法制导的构造算法）
  - NFA→DFA（子集构造法）
  - DFA化简
  - 根据DFA构造词法分析器源码

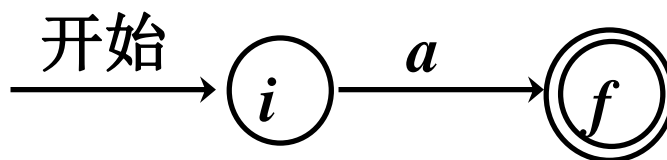


# 语法制导的NFA构造算法

- 语法制导(Syntax-directed): 按正规式的语法结构来指导构造
- 首先, 构造识别  $\varepsilon$  和字母表中一个符号的NFA
  - 重要特点: 仅有一个接受状态, 接受状态没有出边



识别正规式  $\varepsilon$  的NFA



识别正规式  $a$  的NFA

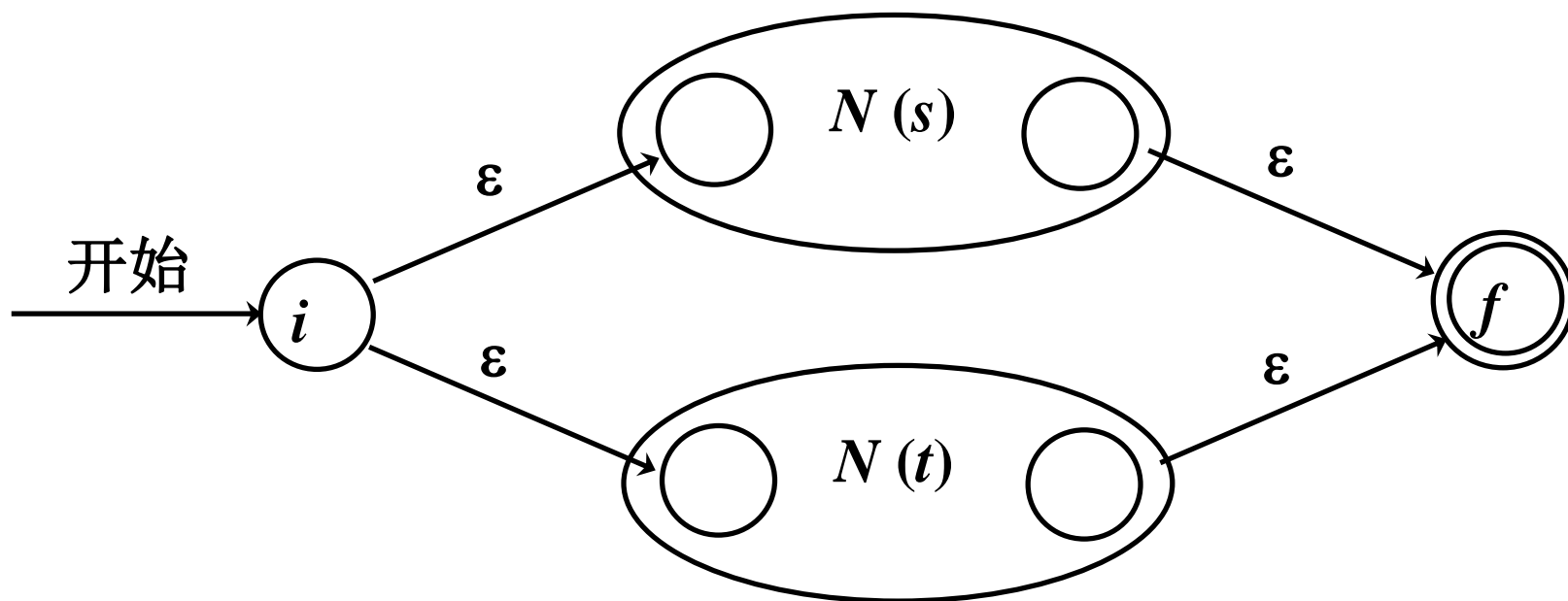
- 对于带括号的正规式( $s$ ), 使用  $s$  对应的NFA  $N(s)$  本身作为( $s$ )的NFA



# 语法制导的NFA构造算法

## □ 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，接受状态没有出边



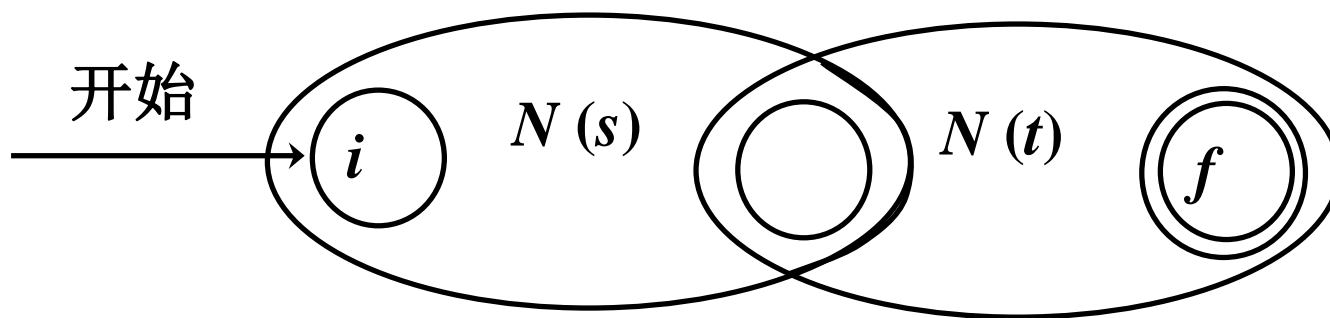
识别正规式 **(s) | (t)** 的NFA



# 语法制导的NFA构造算法

## □ 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，接受状态没有出边



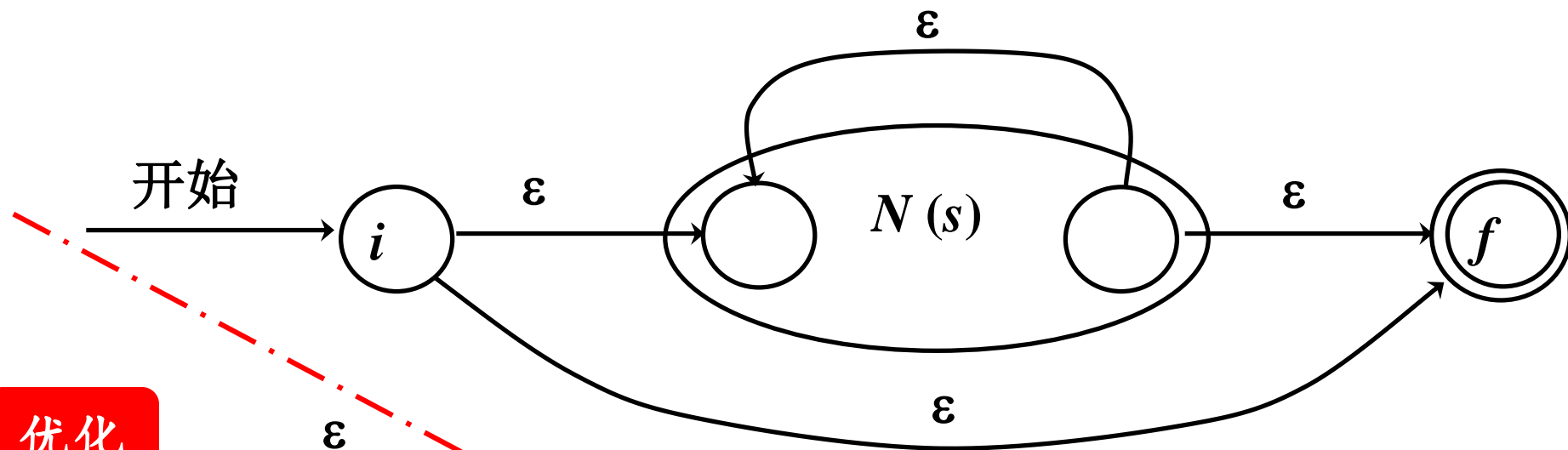
识别正规式  $(s)(t)$  的NFA



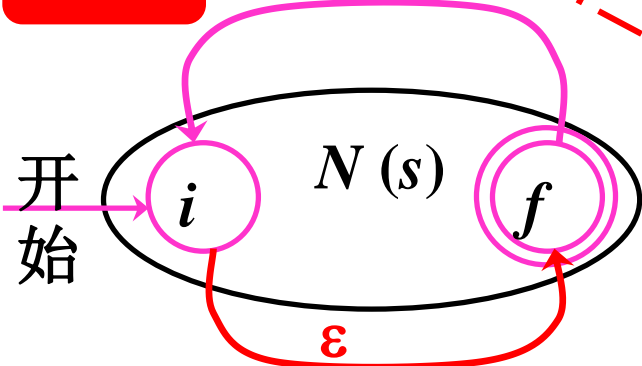
# 语法制导的NFA构造算法

## □ 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，接受状态没有出边



优化



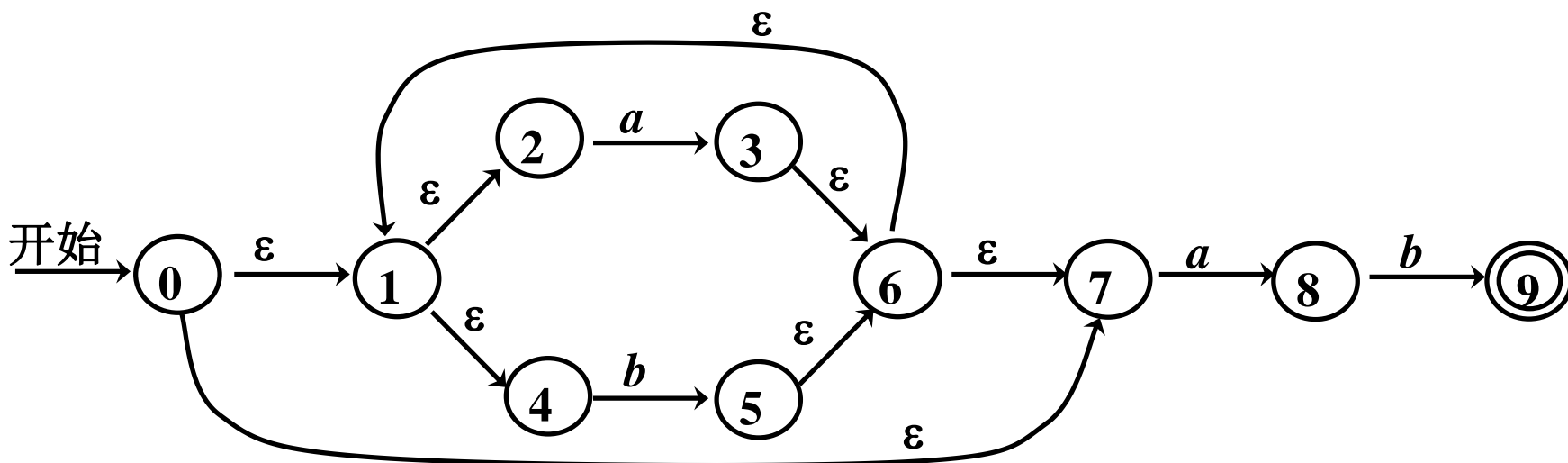
识别正规式  $(s)^*$  的NFA



# 语法制导的NFA构造算法

## □ 本方法产生的NFA有下列性质

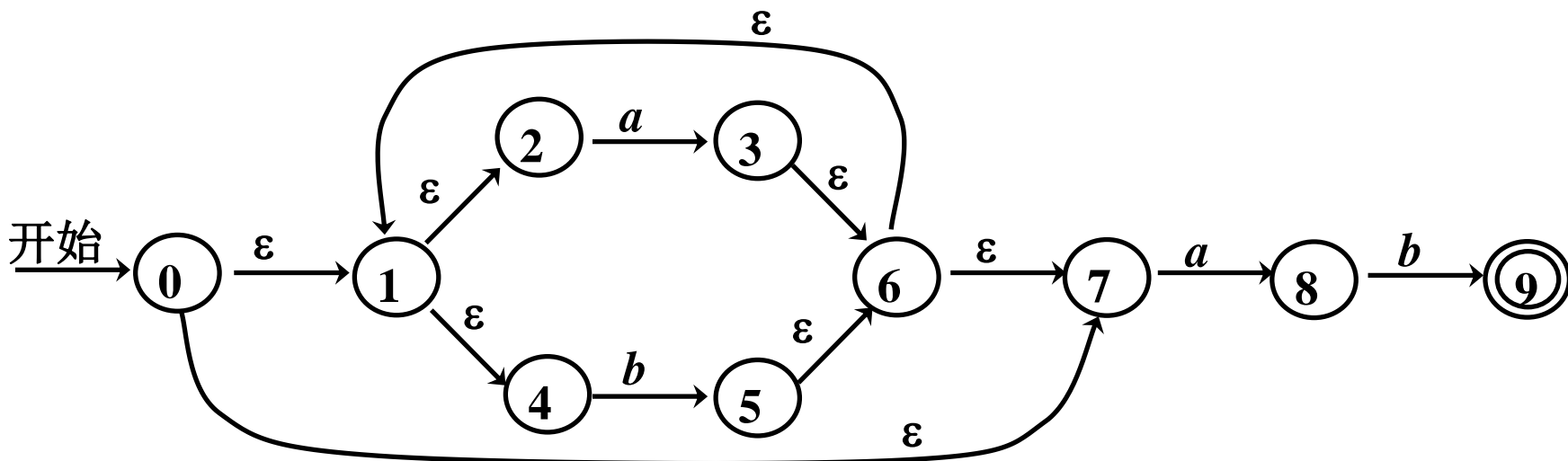
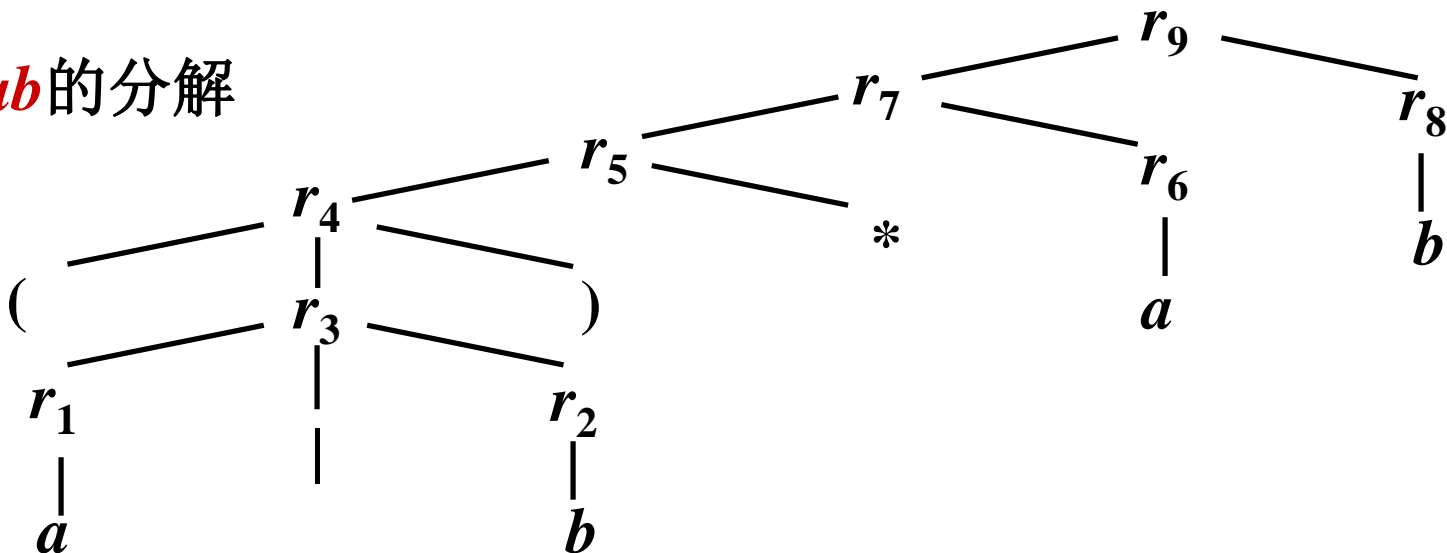
- $N(r)$ 的状态数最多是 $r$ 中符号和算符总数的两倍
- $N(r)$ 只有一个接受状态，接受状态没有向外的转换
- $N(r)$ 的每个非接受状态有
  - 一个用 $\Sigma$ 的符号标记指向其它结点的转换，或者
  - 最多两个指向其它结点的 $\epsilon$ 转换





# NFA构造过程举例

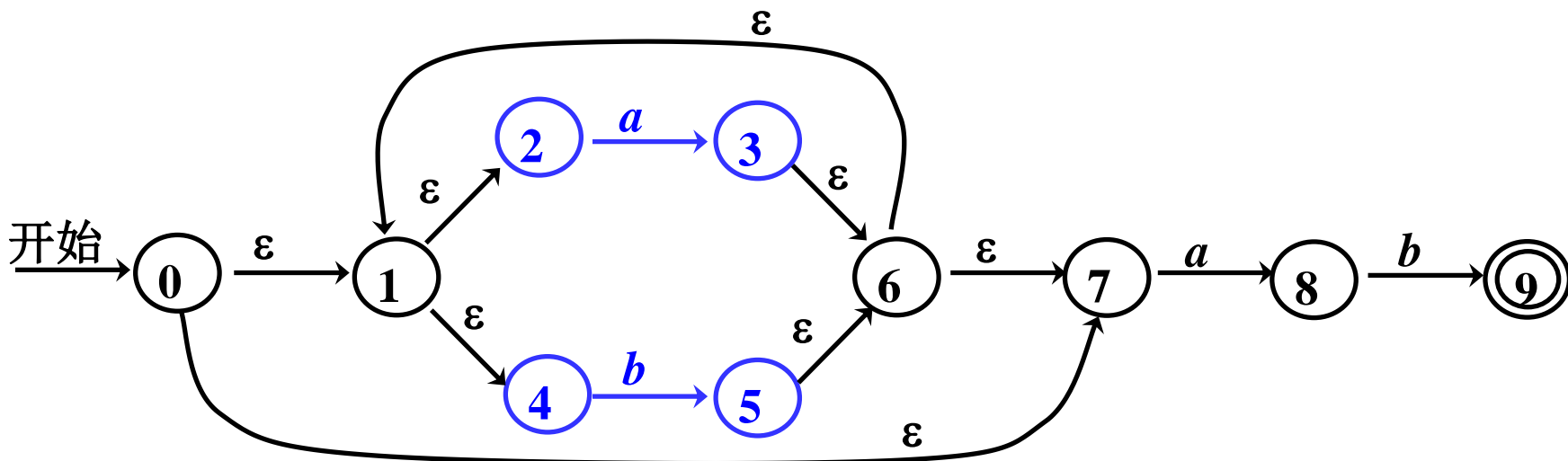
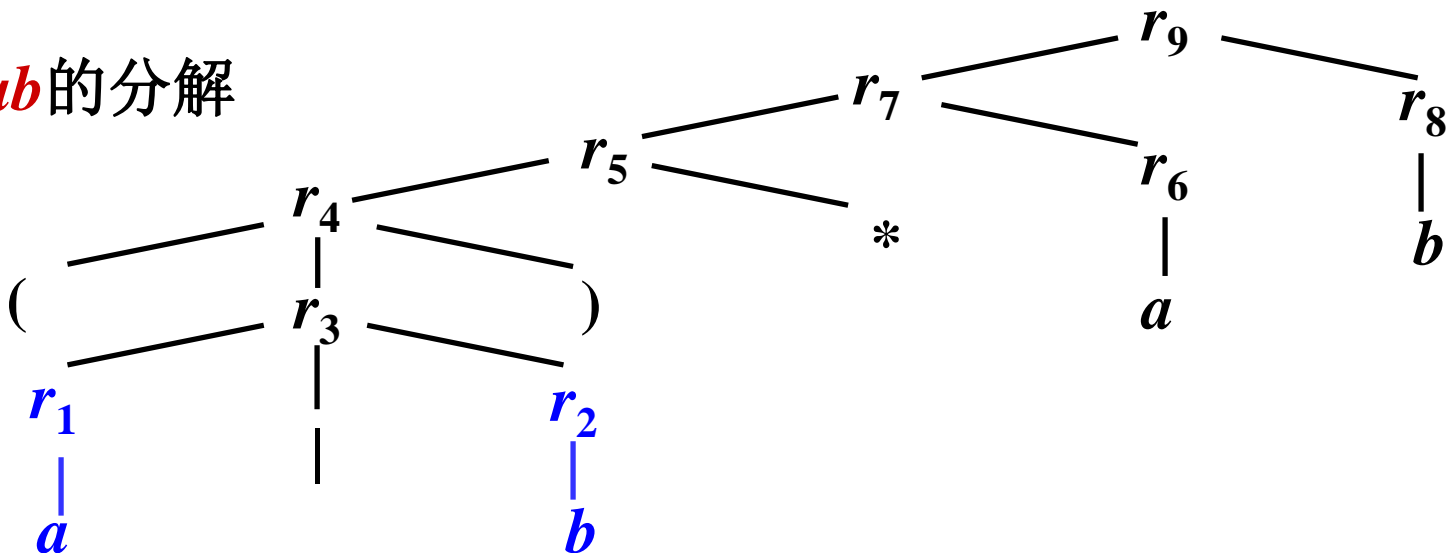
$(a|b)^*ab$  的分解





# NFA构造过程举例

$(a|b)^*ab$  的分解

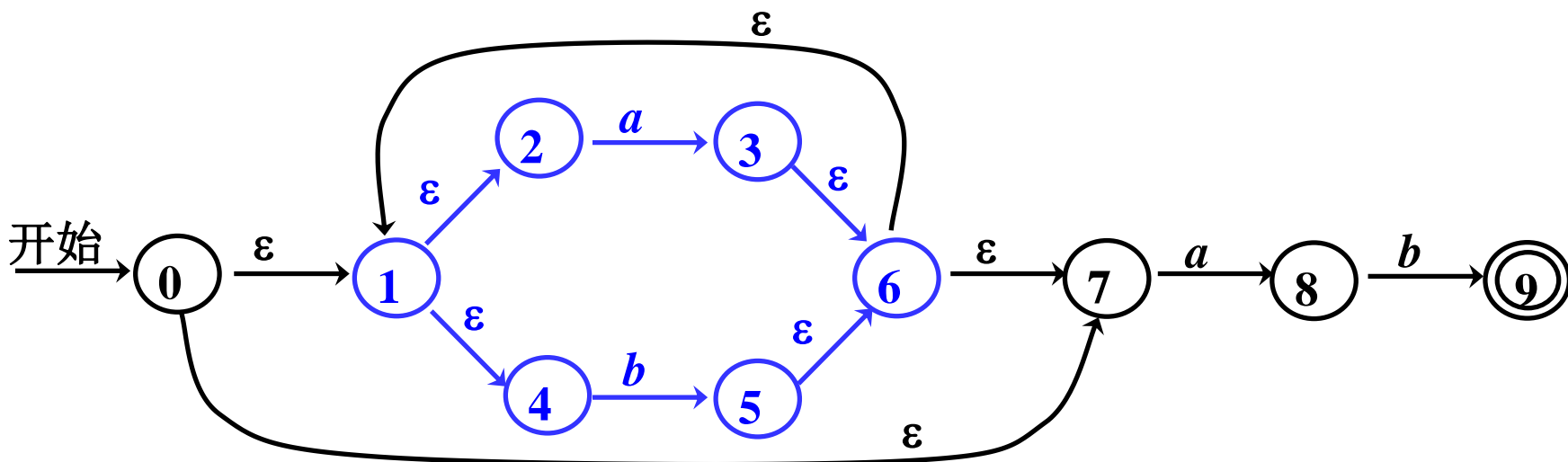
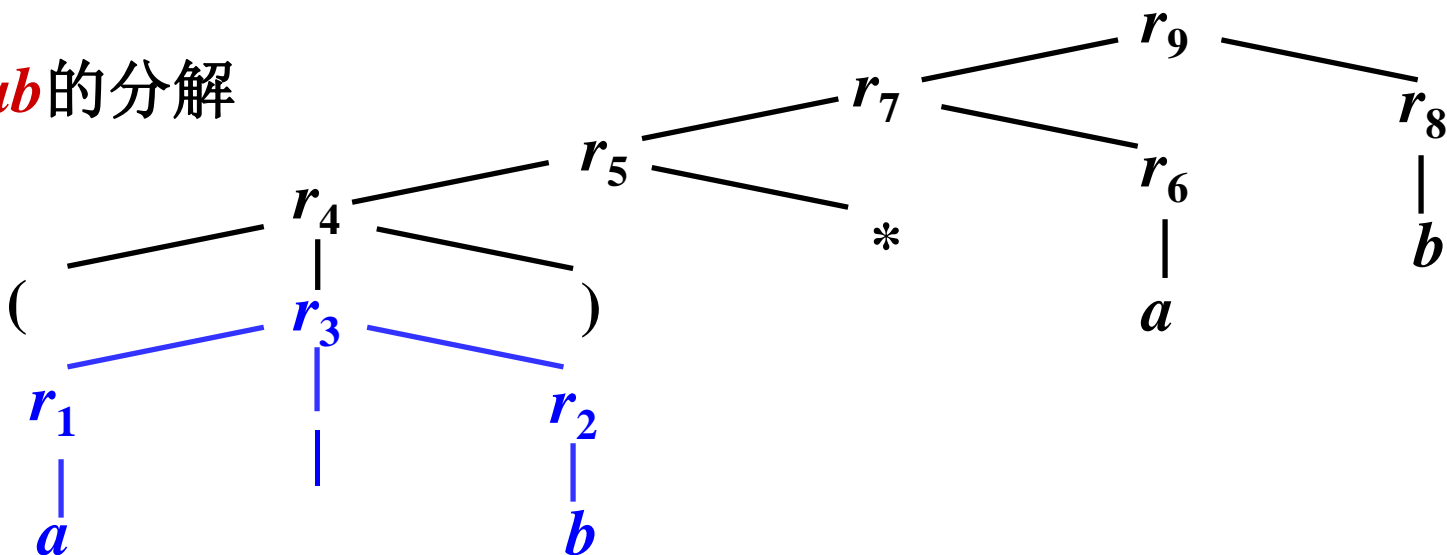






# NFA构造过程举例

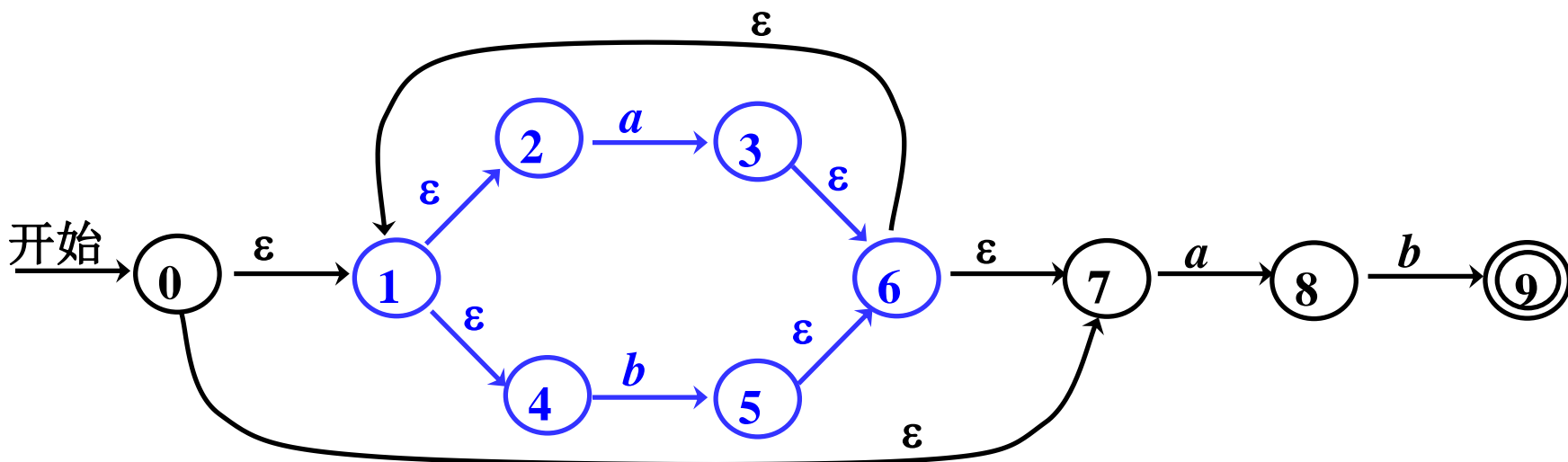
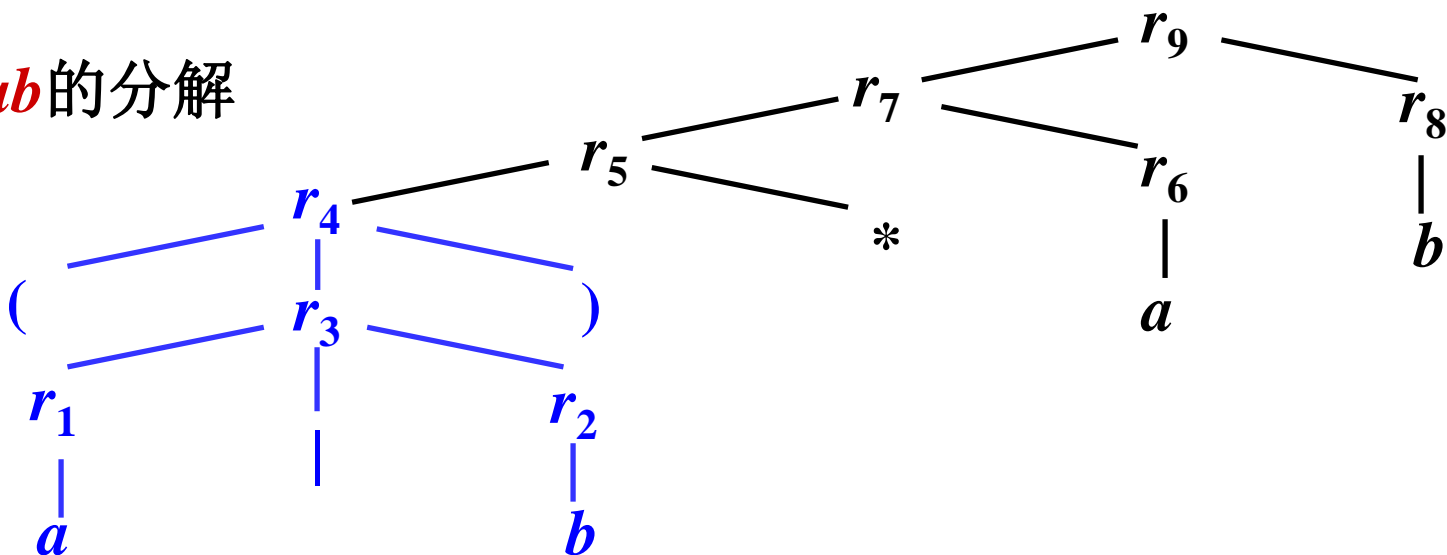
$(a|b)^*ab$  的分解





# NFA构造过程举例

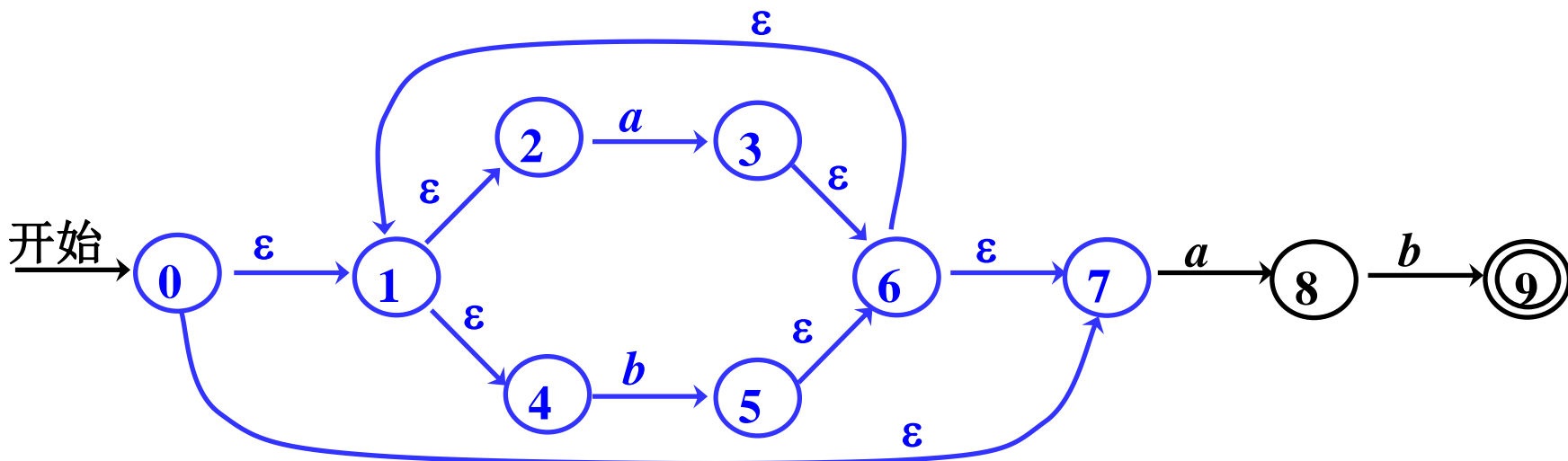
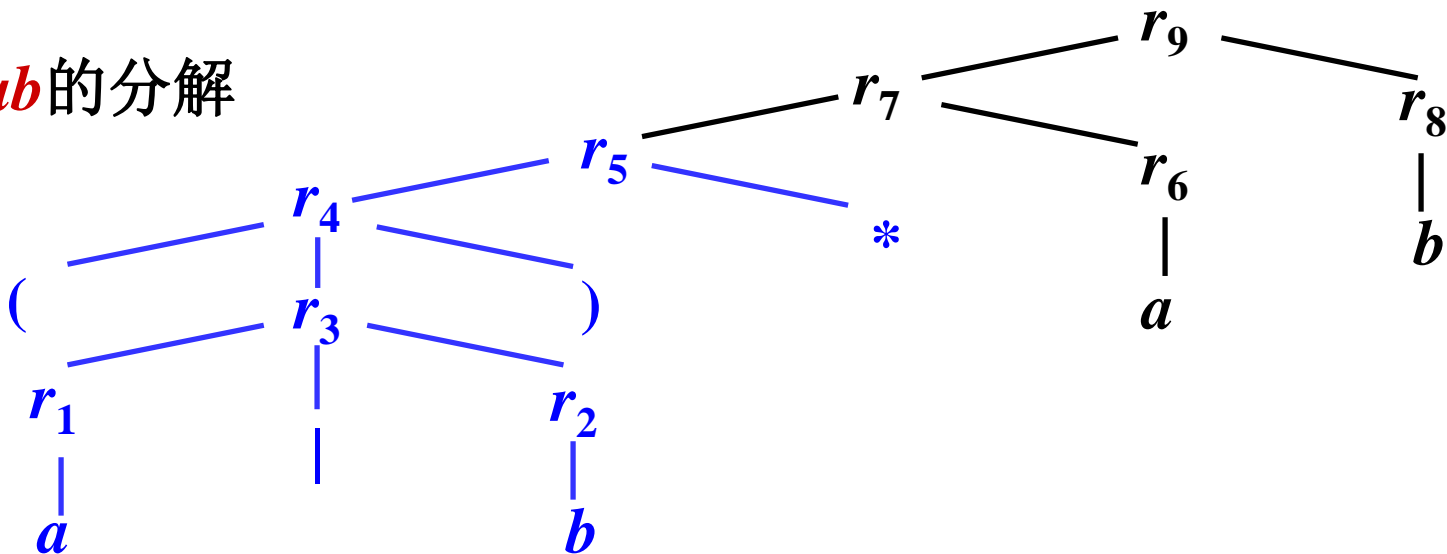
$(a|b)^*ab$  的分解





# NFA构造过程举例

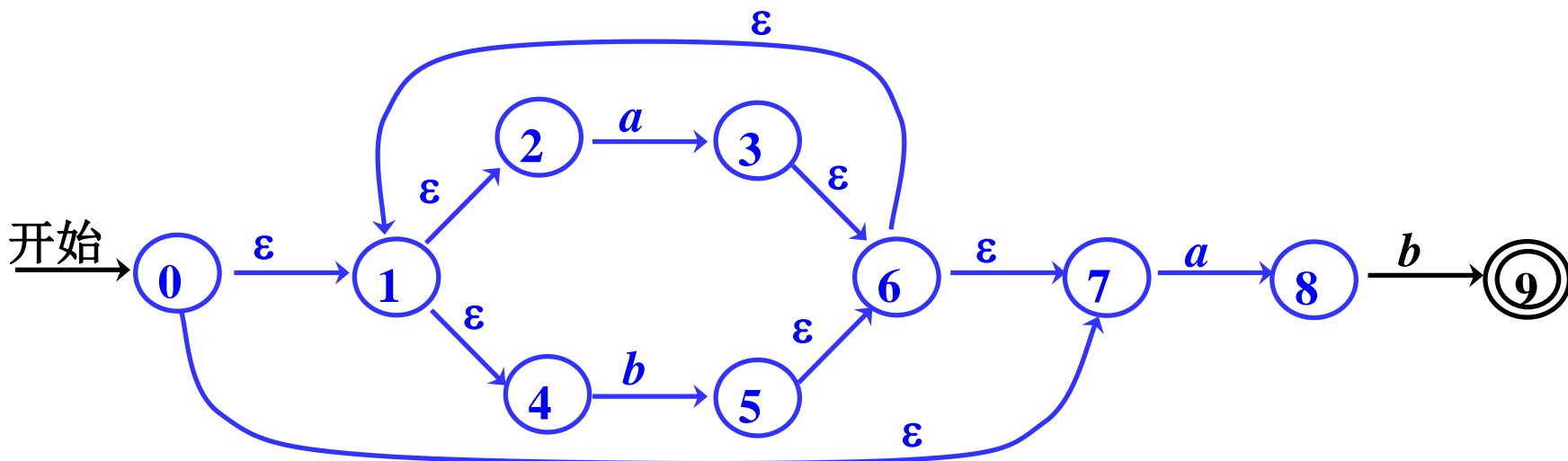
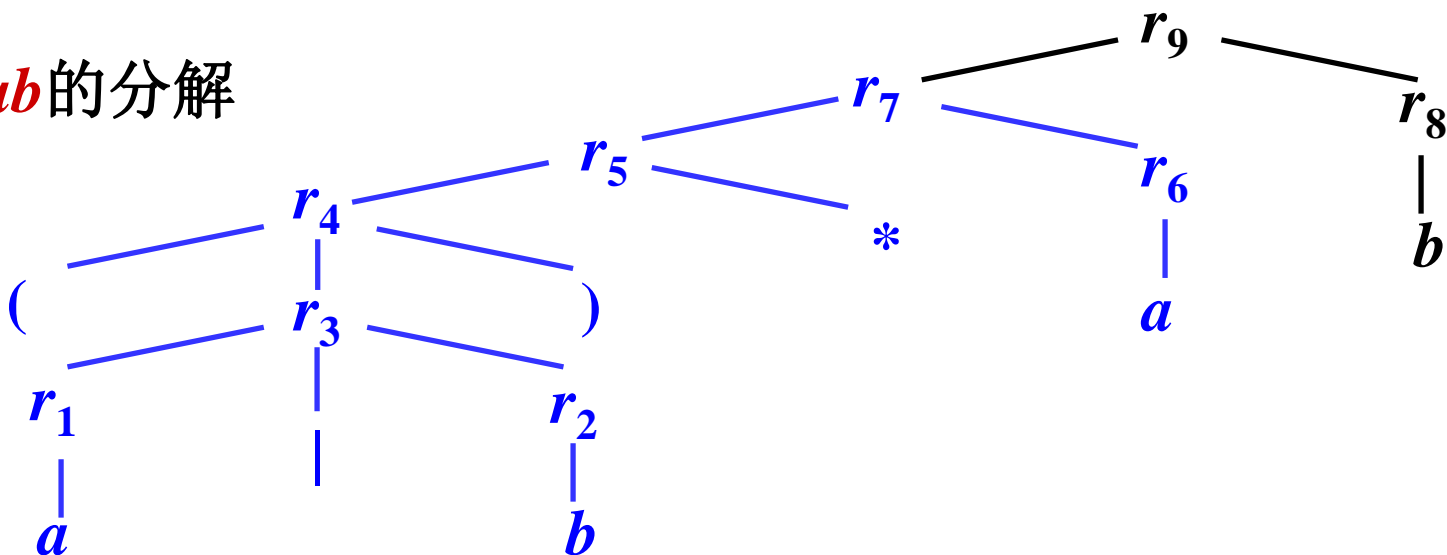
$(a|b)^*ab$  的分解





# NFA构造过程举例

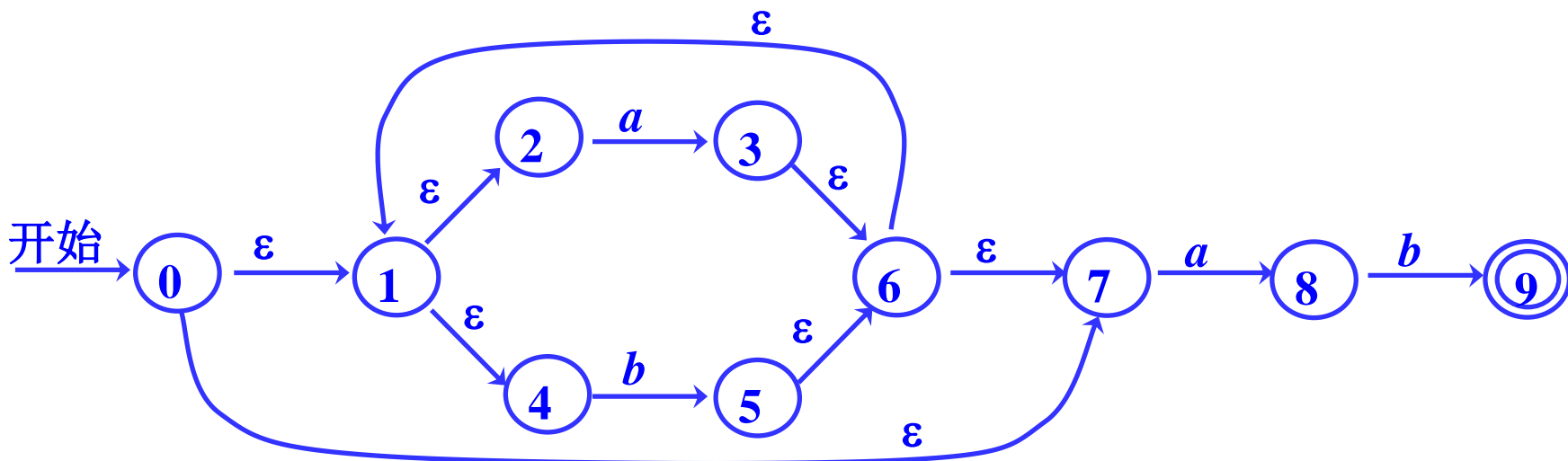
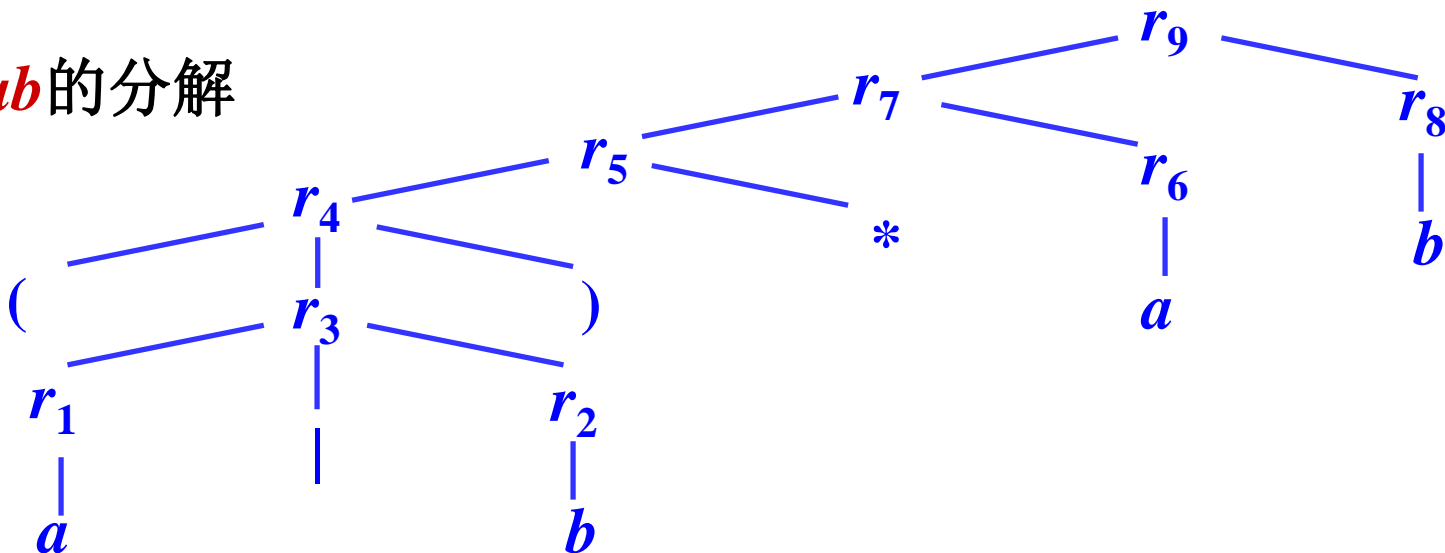
$(a|b)^*ab$  的分解





# NFA构造过程举例

$(a|b)^*ab$  的分解

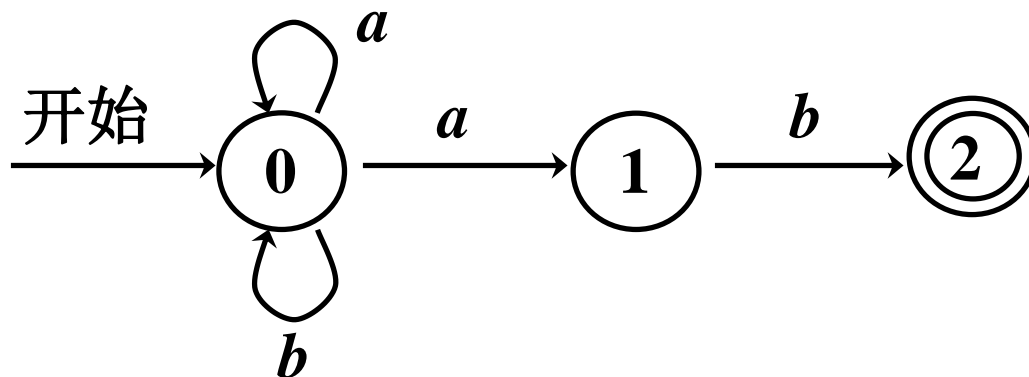




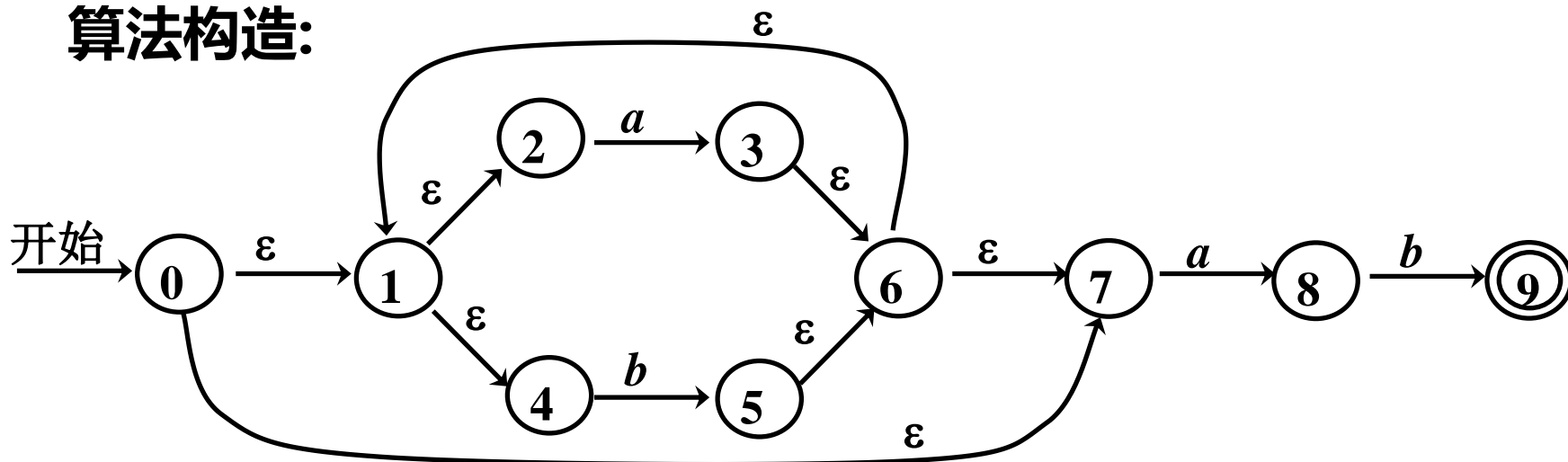
# NFA的手工构造和算法构造

## □ $(a|b)^*ab$ 的两个NFA的比较

手工构造:



算法构造:





# 词法分析器的自动生成技术

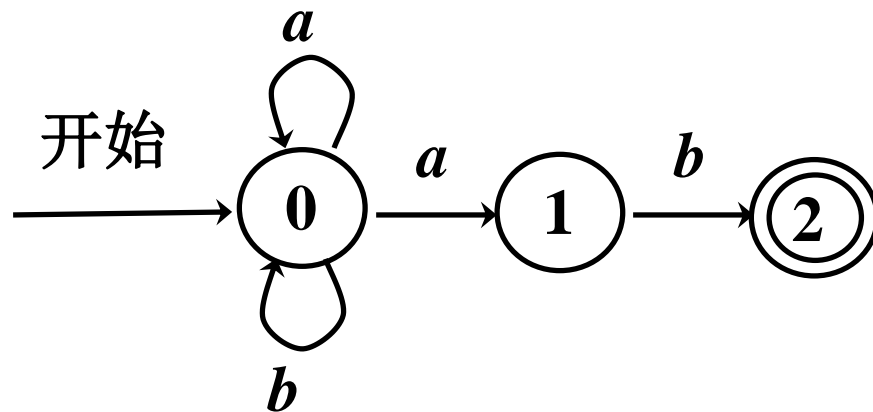
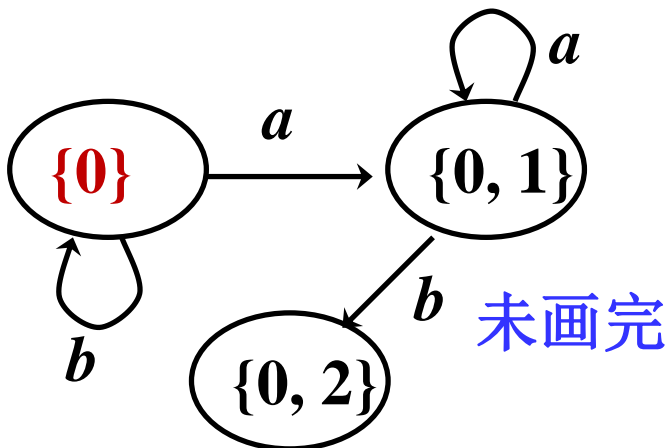
- 正规式：描述语言的词法
- 有限自动机：刻画词法分析的实现
  
- 词法分析器自动生成的主要过程
  - 正规式→NFA（语法制导的构造算法）
  - NFA→DFA (子集构造法)
  - DFA化简
  - 根据DFA构造词法分析器源码



# NFA到DFA的变换

## □ 子集构造法(subset construction)

1. **DFA的一个状态是NFA的一个状态集合**
2. **DFA的开始状态是包含NFA的开始状态的状态集合**
3. 读了输入  $a_i$  后,  
NFA能到达的所有状态:  $s_1, s_2, \dots, s_k$ , 则  
DFA到达一个状态, 对应于NFA的  $\{s_1, s_2, \dots, s_k\}$







# NFA到DFA的变换

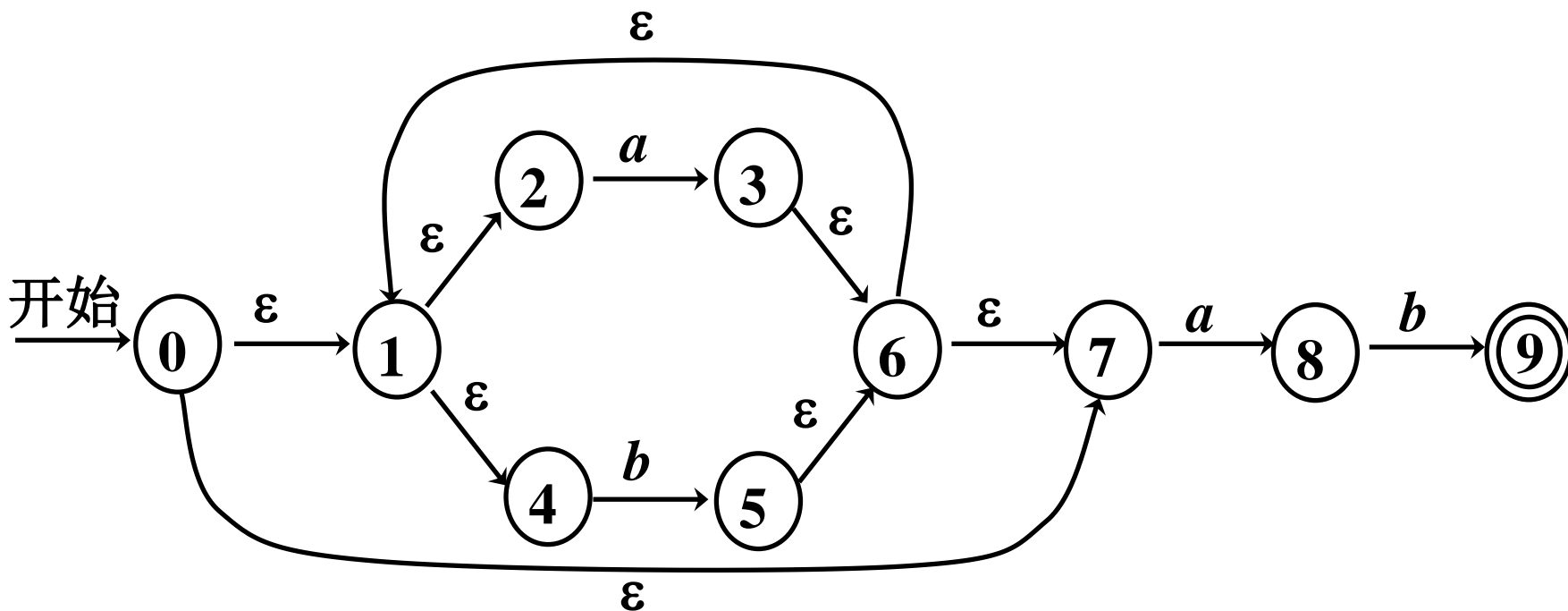
## □ 子集构造法(subset construction)

1.  **$\epsilon$ -闭包 ( $\epsilon$ -closure)** 状态 $s$  的 $\epsilon$ -闭包是  $s$  经  **$\epsilon$ 转换**所能到达的状态集合
2. **NFA的初始状态的  $\epsilon$ -闭包**对应于DFA的初始状态
3. 针对每个DFA 状态 ——NFA状态子集 $A$ ，求输入每个 $a_i$ 后能到达的NFA状态的 $\epsilon$ -闭包并集，该集合对应于DFA中的一个已有状态，或者是一个要新加的DFA状态



## 例题6

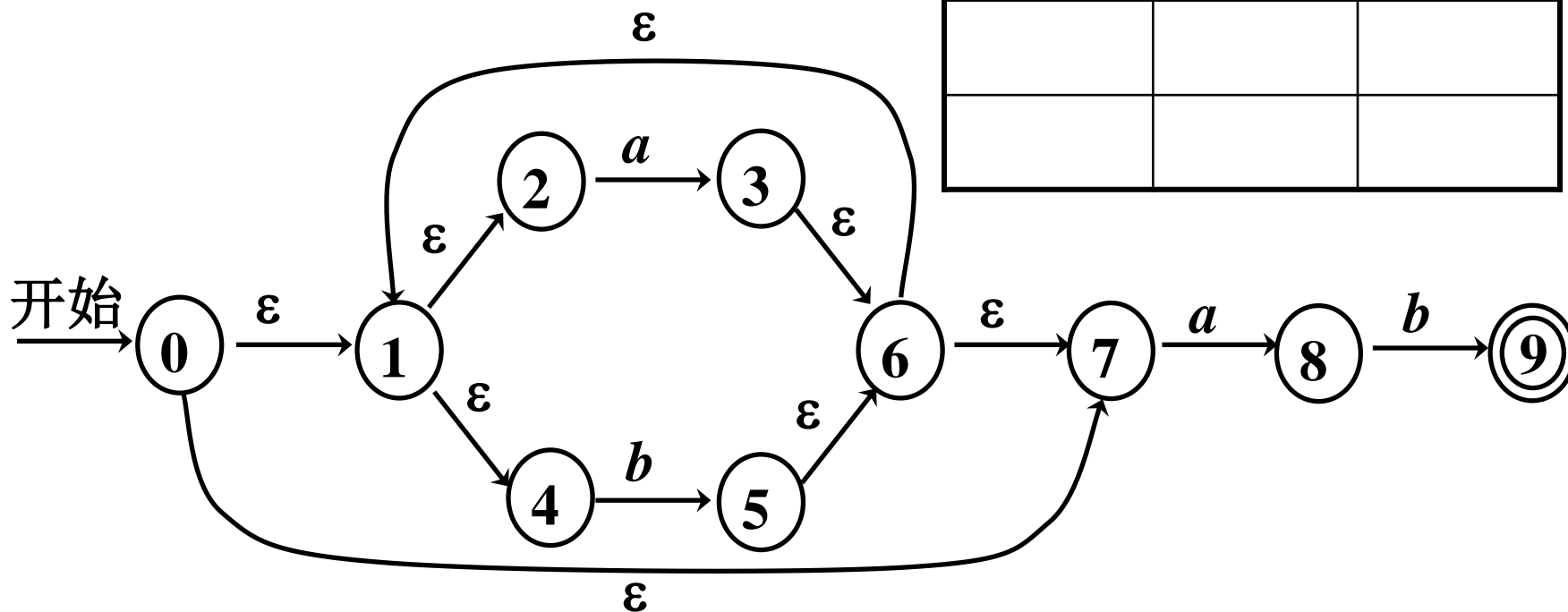
正规式  $(a|b)^*ab$  对应的NFA如下，把它变换为DFA





# 例题6

状态	输入符号	
	<i>a</i>	<i>b</i>

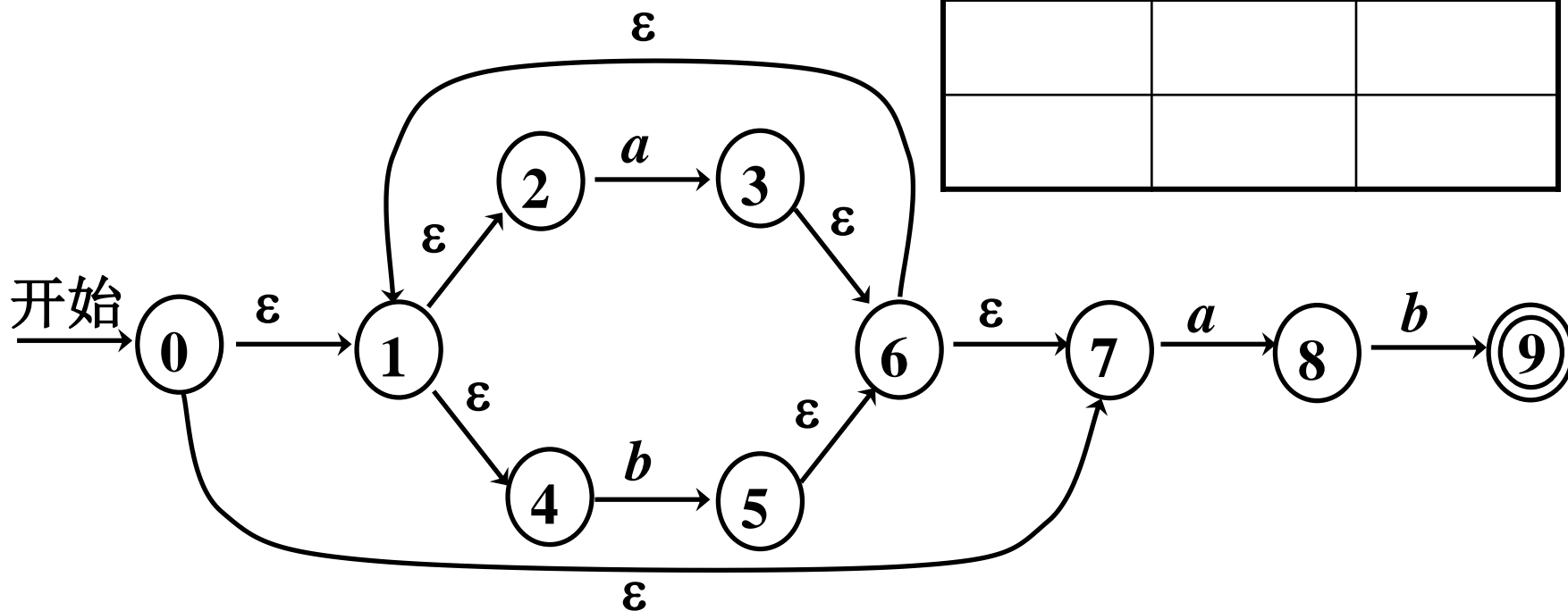




# 例题6

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

状态	输入符号	
	$a$	$b$
$A$		



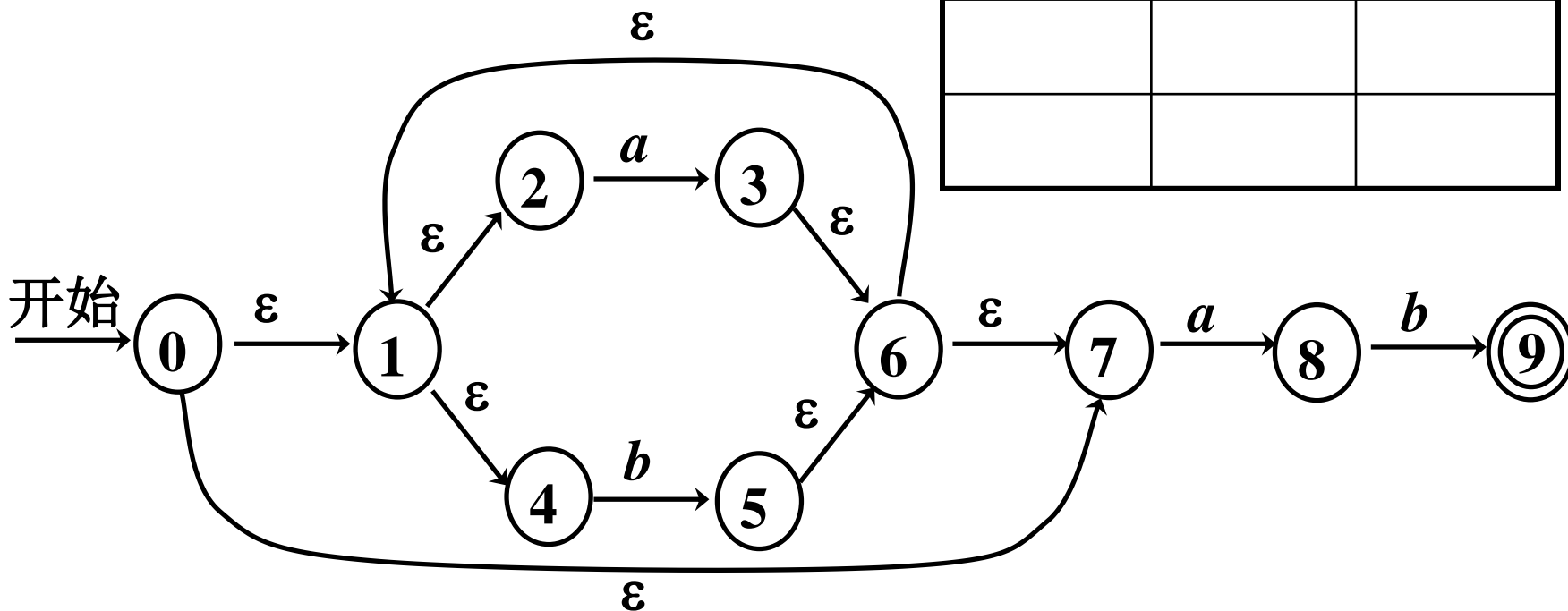


# 例题6

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

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

状态	输入符号	
	$a$	$b$
$A$	$B$	



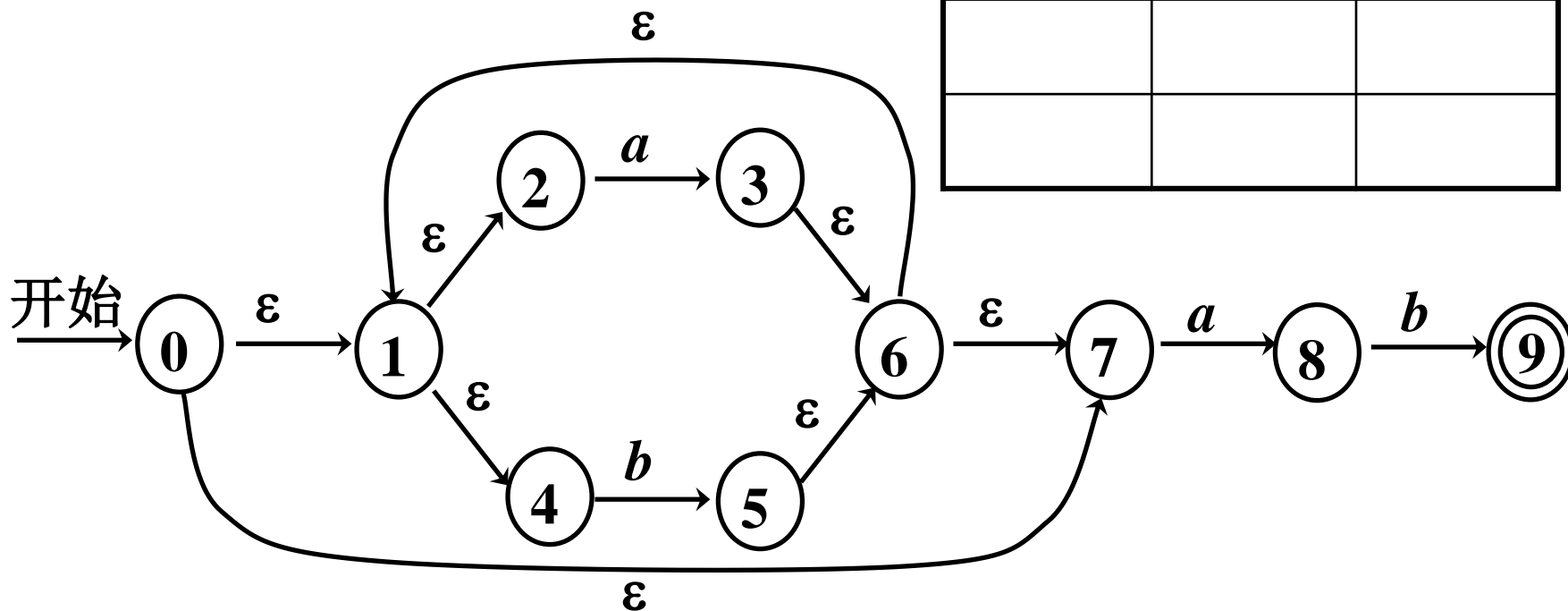


# 例题6

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

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

状态	输入符号	
	$a$	$b$
$A$	$B$	
$B$		





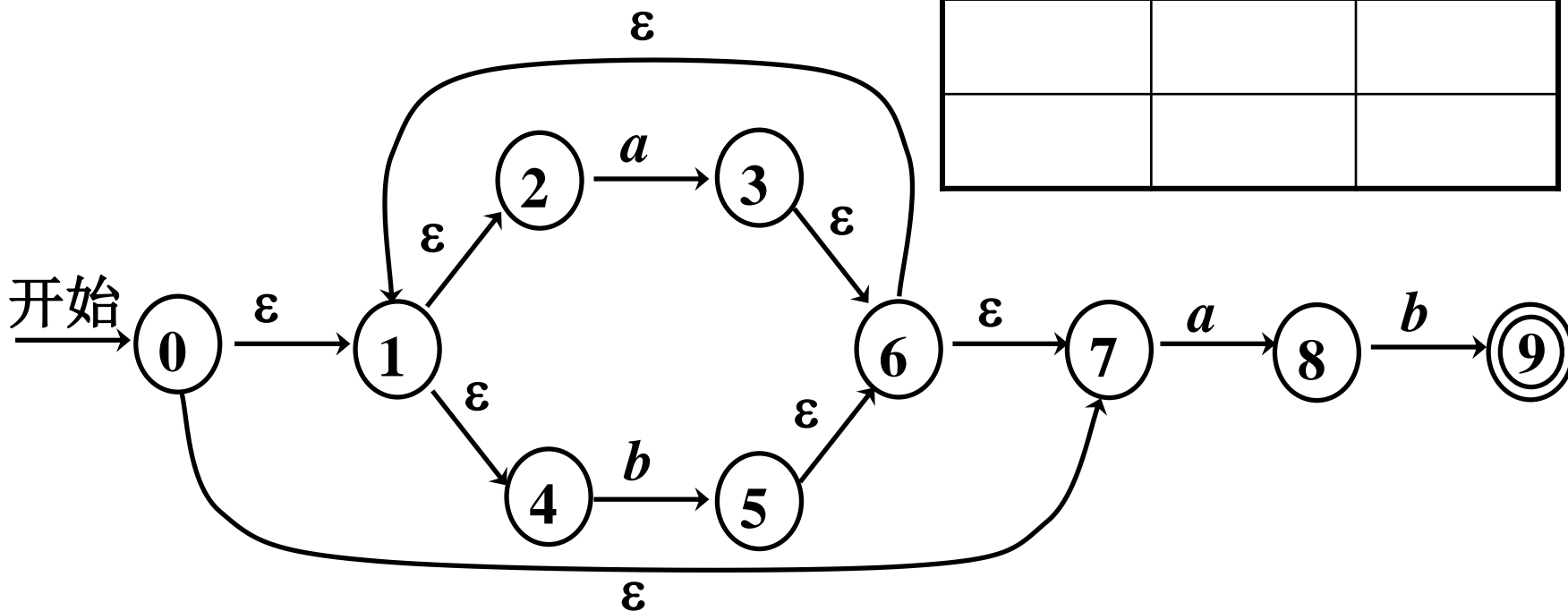
# 例题6

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

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

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

状态	输入符号	
	$a$	$b$
$A$	$B$	$C$
$B$		





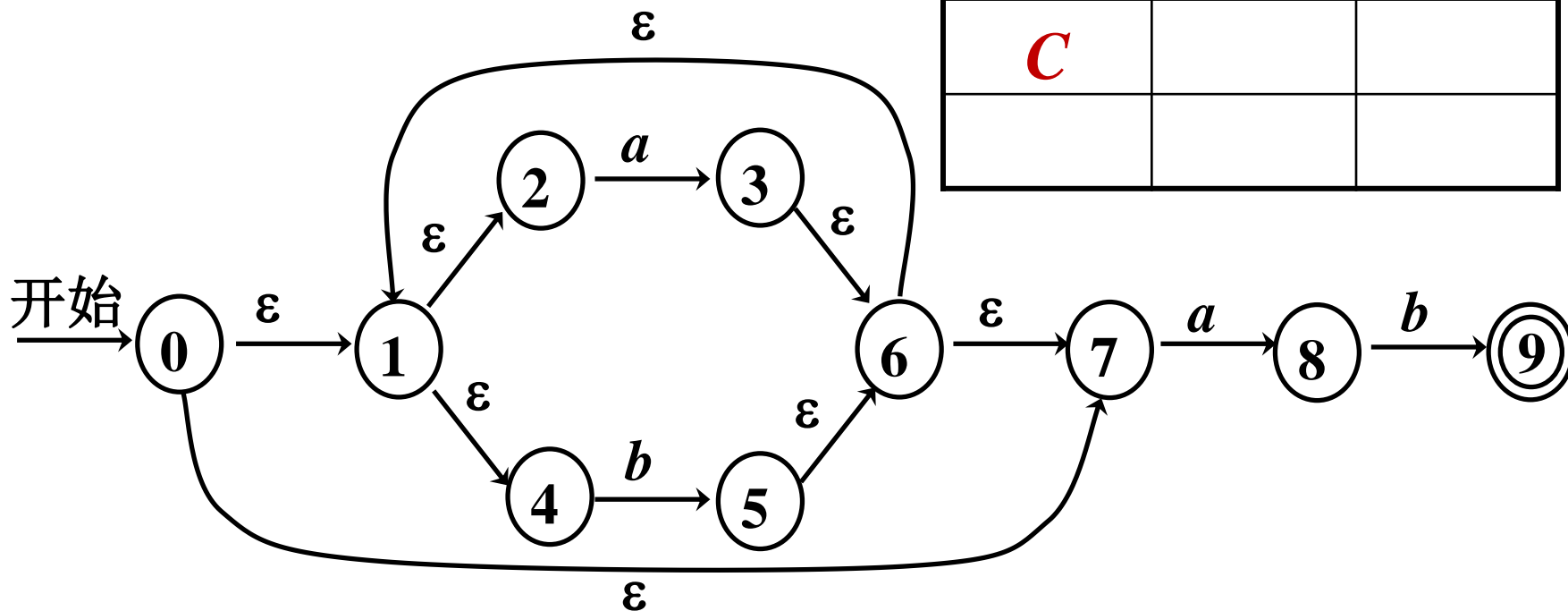
# 例题6

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

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

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

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







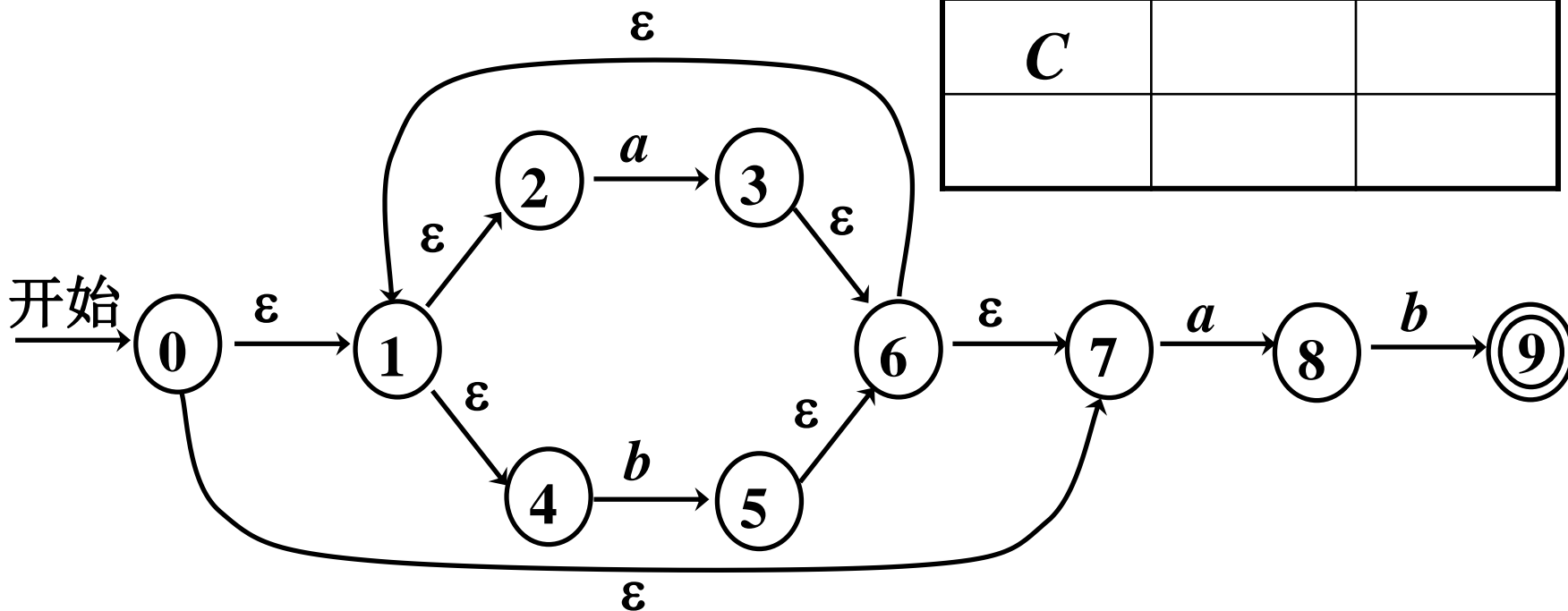
# 例题6

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

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

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

状态	输入符号	
	$a$	$b$
$A$	$B$	$C$
$B$	<b><math>B</math></b>	
$C$		





# 例题6

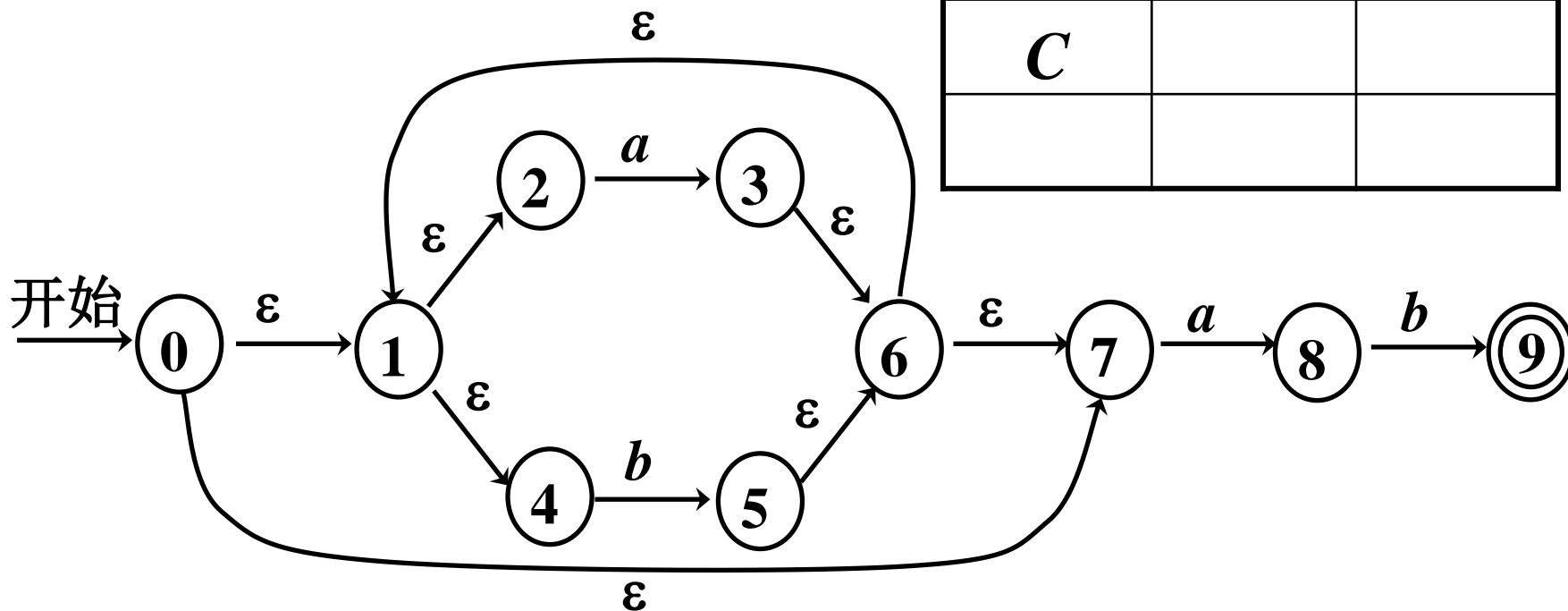
$A = \{0, 1, 2, 4, 7\}$

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

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

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

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





# 例题6

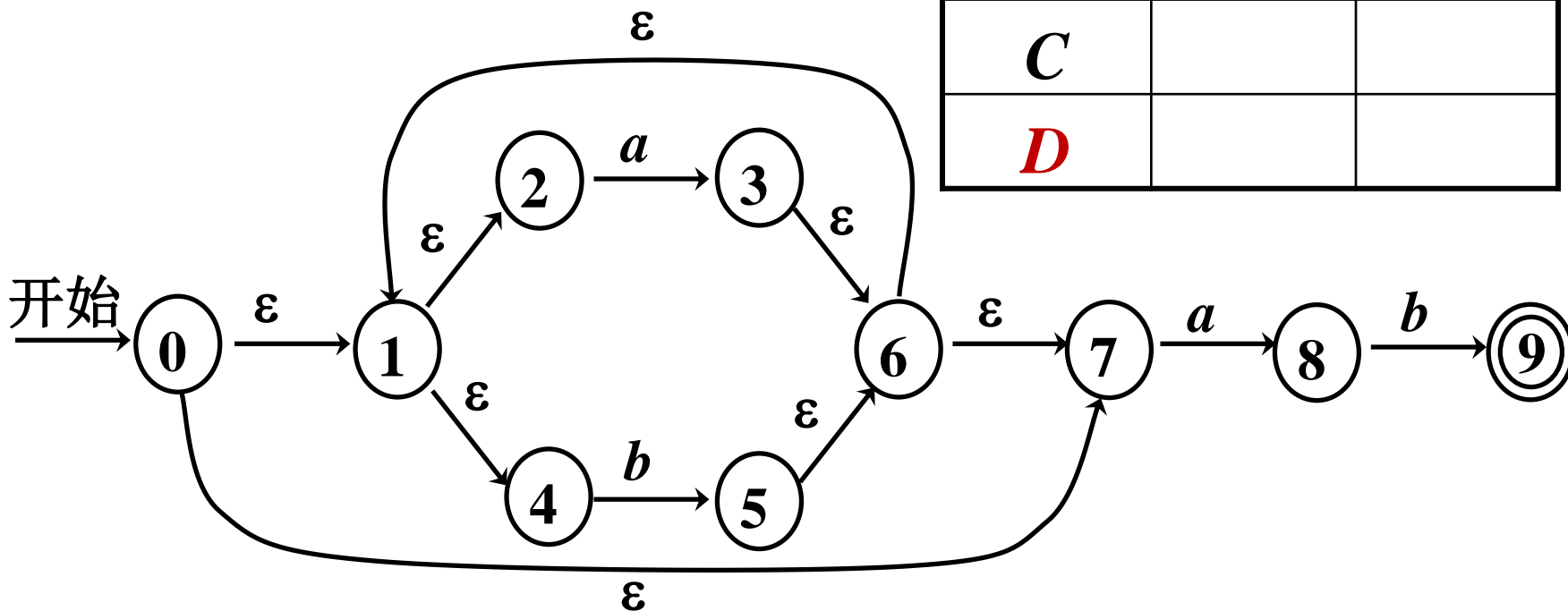
$A = \{0, 1, 2, 4, 7\}$

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

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

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

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





# 例题6

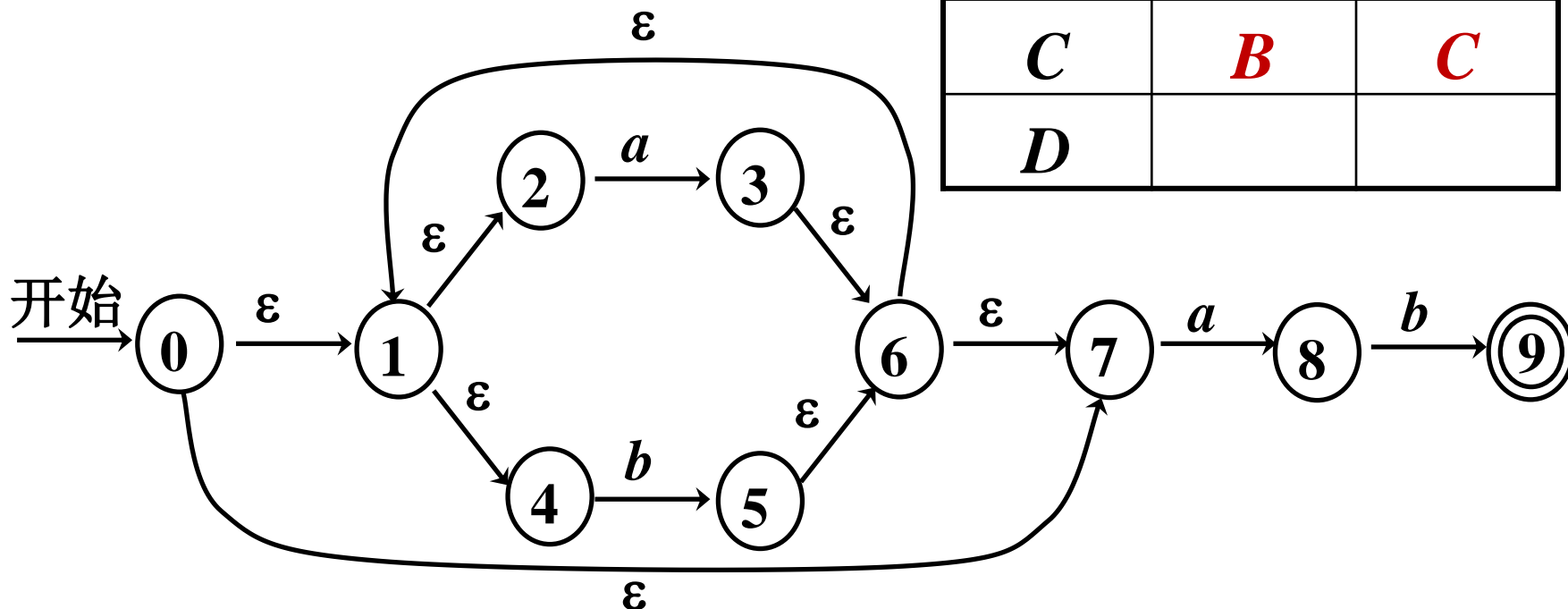
$A = \{0, 1, 2, 4, 7\}$

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

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

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

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





# 例题6

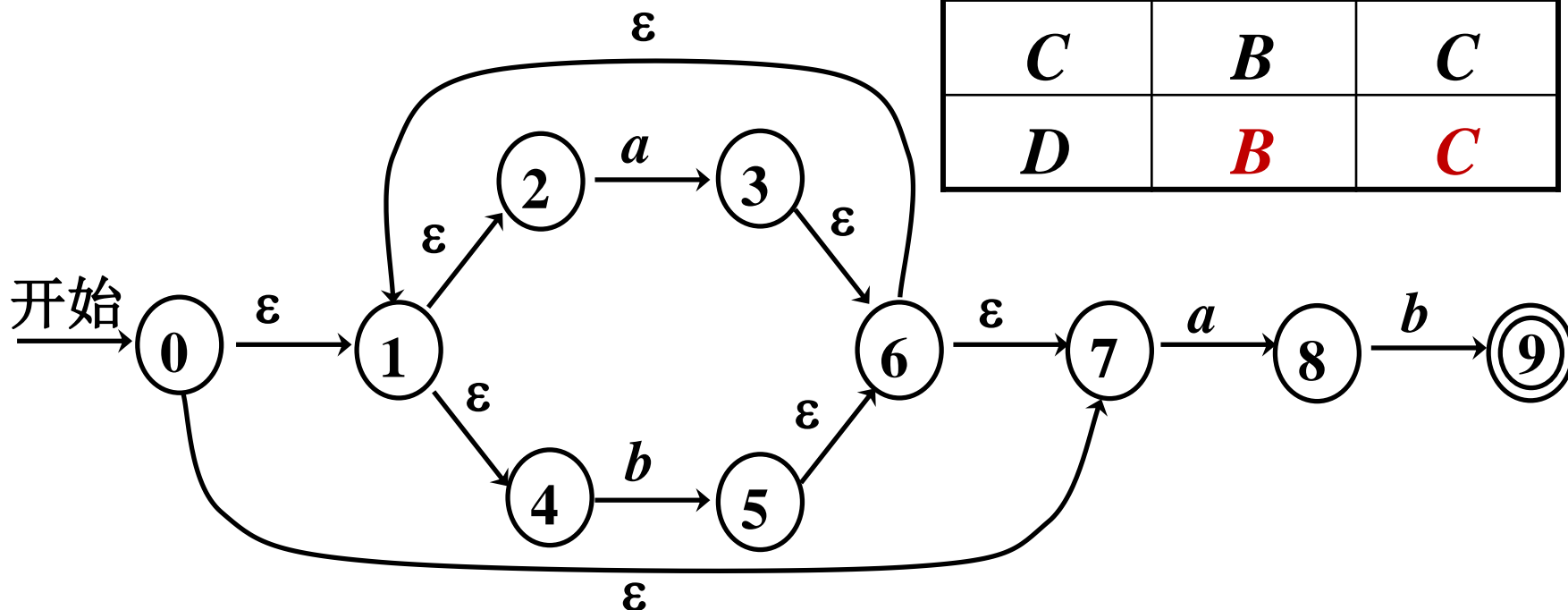
$A = \{0, 1, 2, 4, 7\}$

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

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

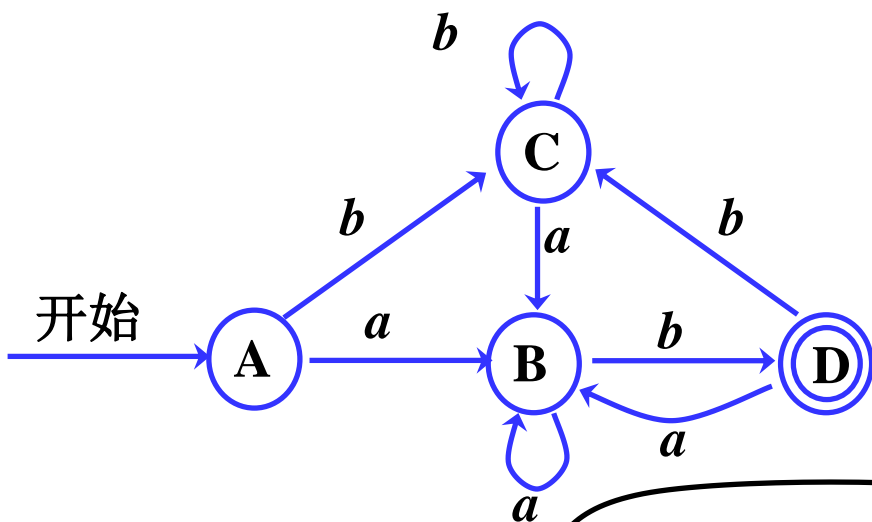
$D = \{1, 2, 4, 5, 6, 7, 9\}$

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

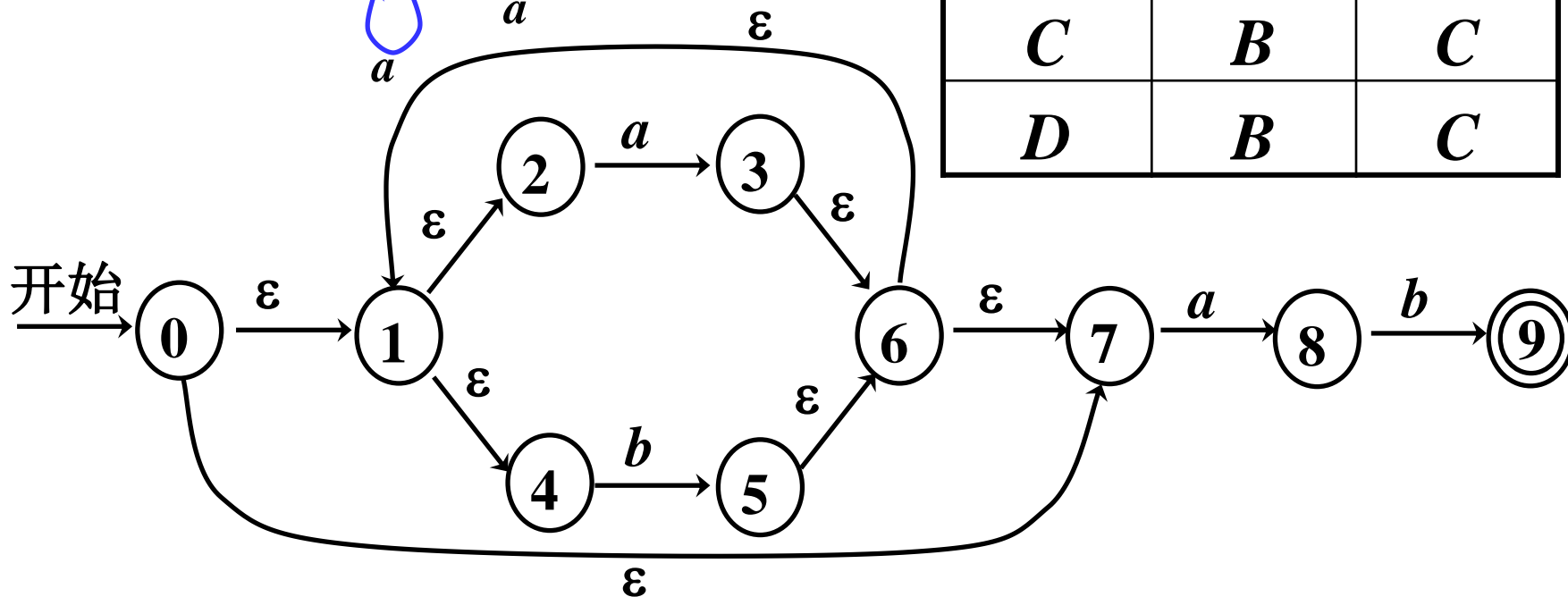




# 例题6



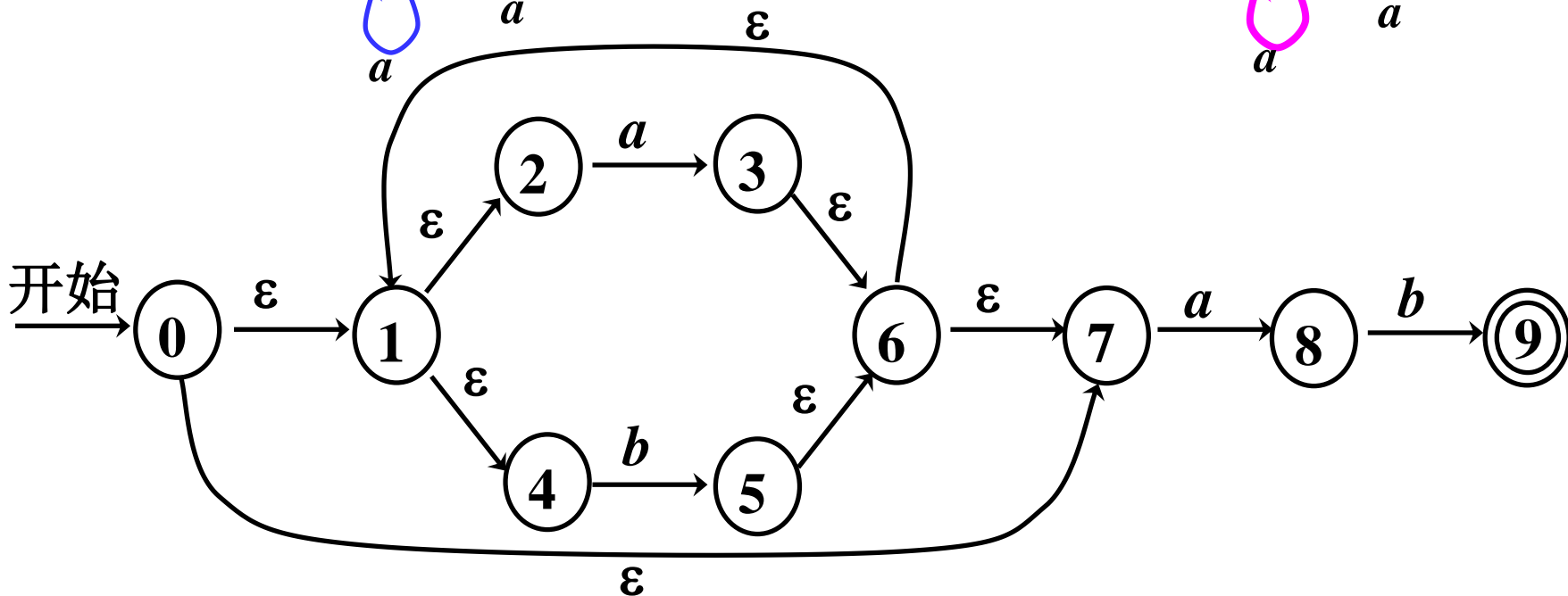
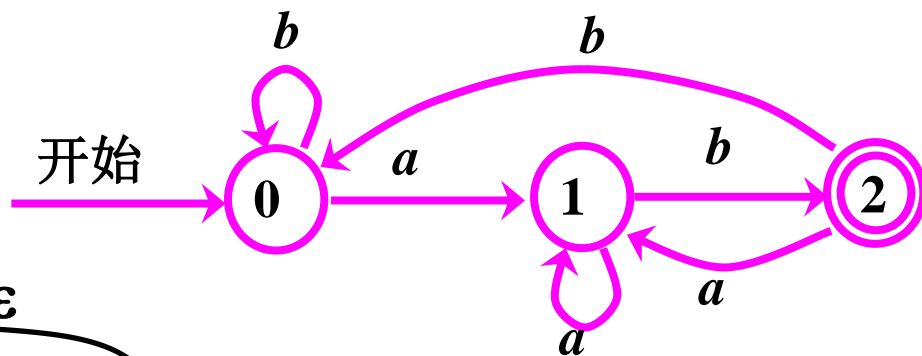
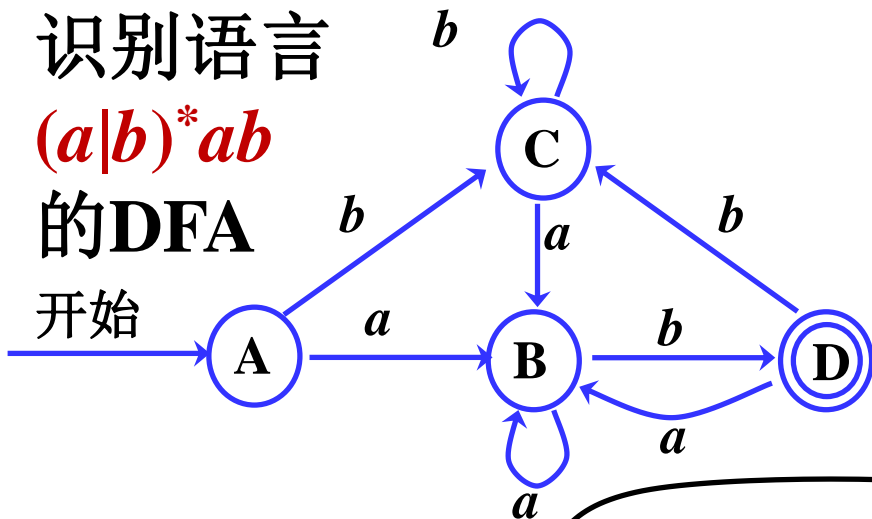
状态	输入符号	
	<i>a</i>	<i>b</i>
<i>A</i>	<i>B</i>	<i>C</i>
<i>B</i>	<i>B</i>	<i>D</i>
<i>C</i>	<i>B</i>	<i>C</i>
<i>D</i>	<i>B</i>	<i>C</i>





# 例题6

识别语言  
 $(a|b)^*ab$   
的DFA

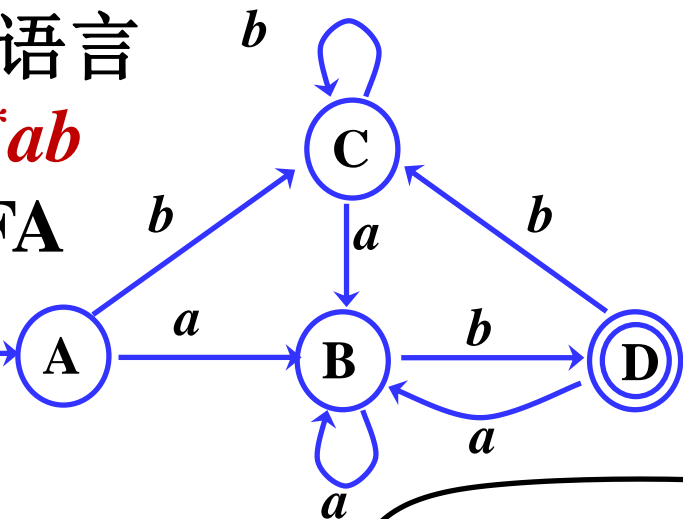




# 例题6

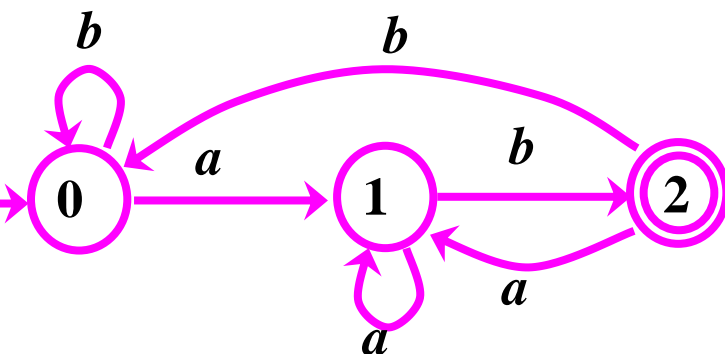
识别语言  
 $(a|b)^*ab$   
的DFA

开始

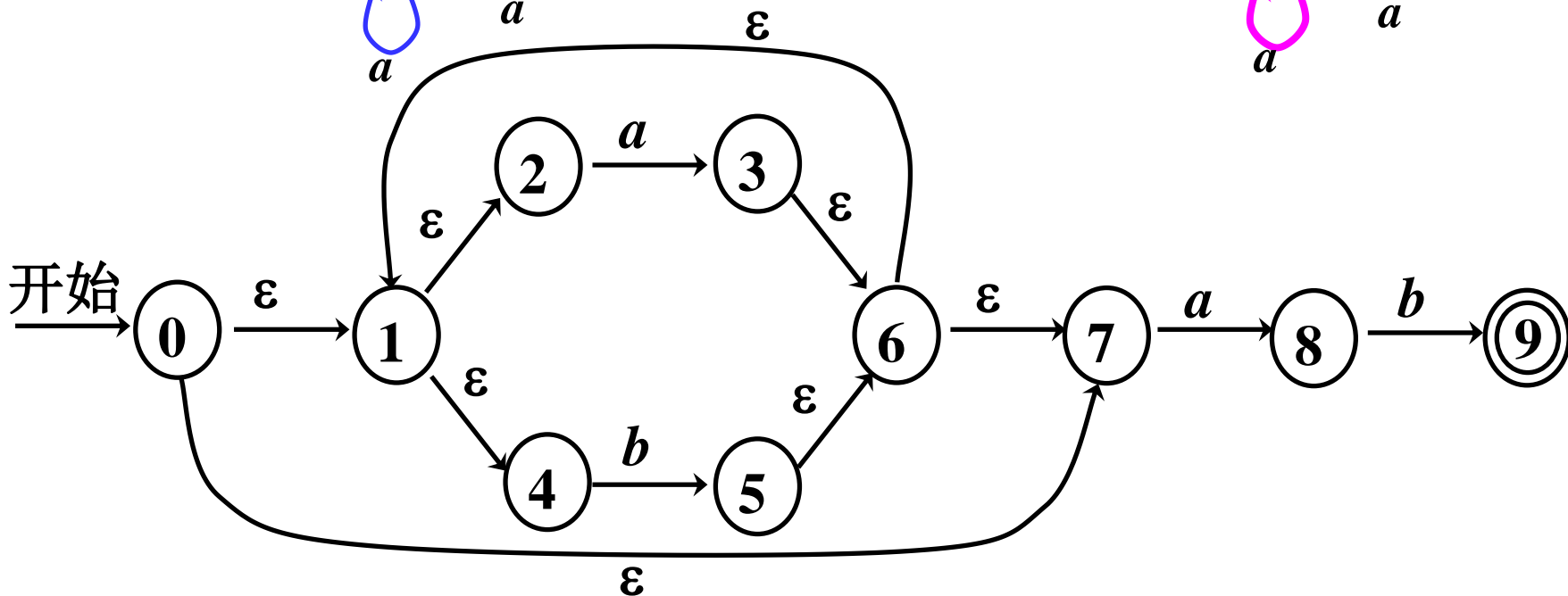


开始

子集构造法不一定  
得到最简DFA



开始







# 词法分析器的自动生成技术

- 正规式：描述语言的词法
- 有限自动机：刻画词法分析的实现
  
- 词法分析器自动生成的主要过程
  - 正规式→NFA（语法制导的构造算法）
  - NFA→DFA（子集构造法）
  - DFA化简
  - 根据DFA构造词法分析器源码

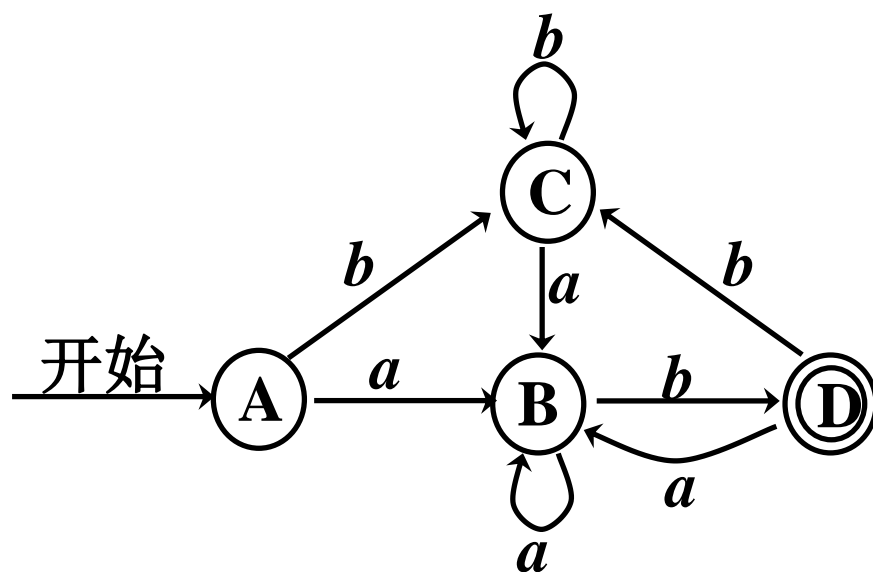
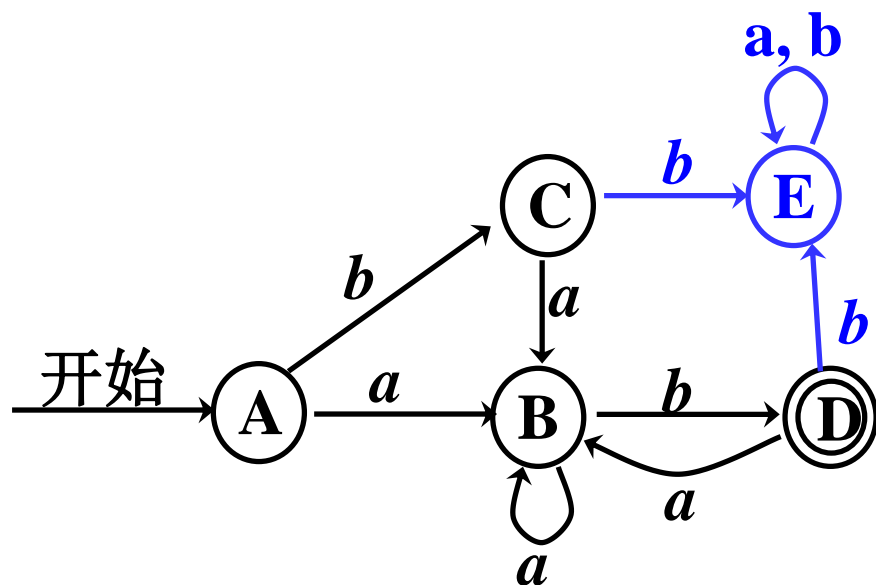


# DFA的化简

□ 该方法用于化简转换函数是**全函数**的DFA

□ 死状态 (dead state)

■ 当转换函数由部分函数改成全函数表示时，要在左图引入**死状态E**





# DFA的化简

## □ 可区别的状态(distinguishable states) $s$ 和 $t$

分别从 $s$ 、 $t$ 出发，存在一个输入符号 $w$ ，使得一个到达接受状态，另一个到达非接受状态。

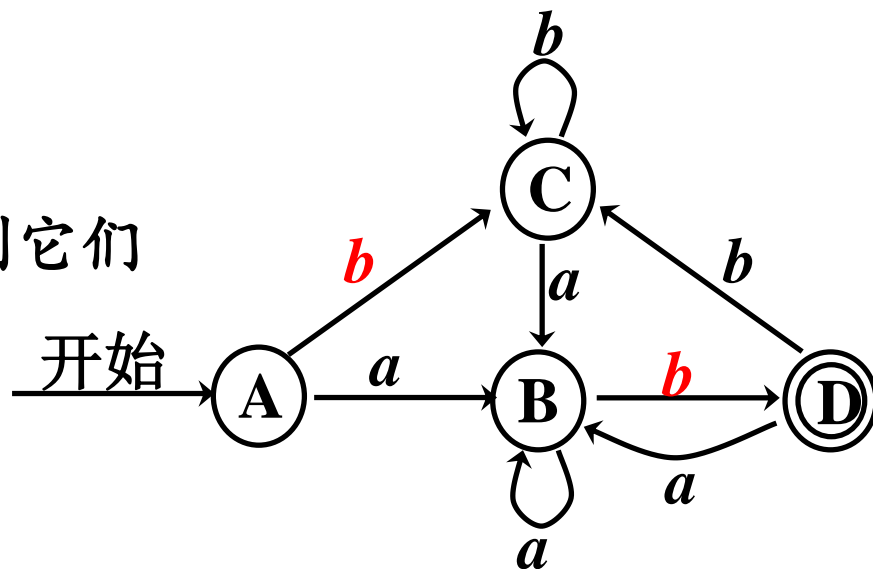
### ■ A 和 B 是可区别的状态：

从A出发，读入  $b$  后到达非接受状态C；从B出发，读过  $b$  后到达接受状态D

### ■ A 和 C 是不可区别的状态：

无任何输入符号可用来区别它们

可区别的状态  
要分开对待





# DFA的化简

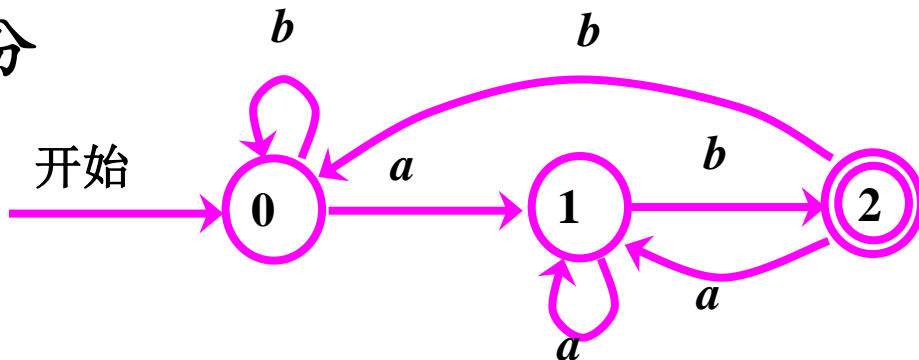
## □ 方法

### 1. 按是否是接受状态来区分

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

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

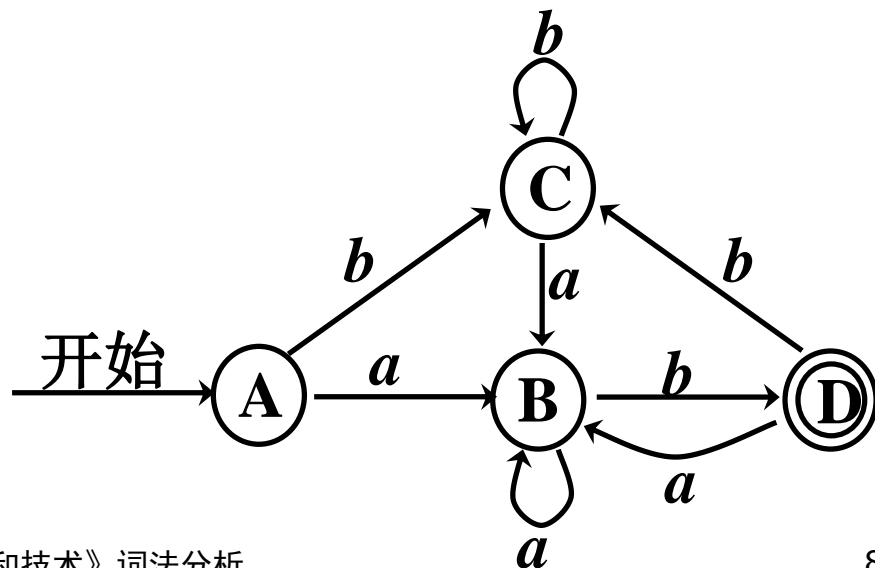
$move(\{A, B, C\}, b) = \{C, D\}$



### 2. 继续分解 $\{A, C\}, \{B\}, \{D\}$

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

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





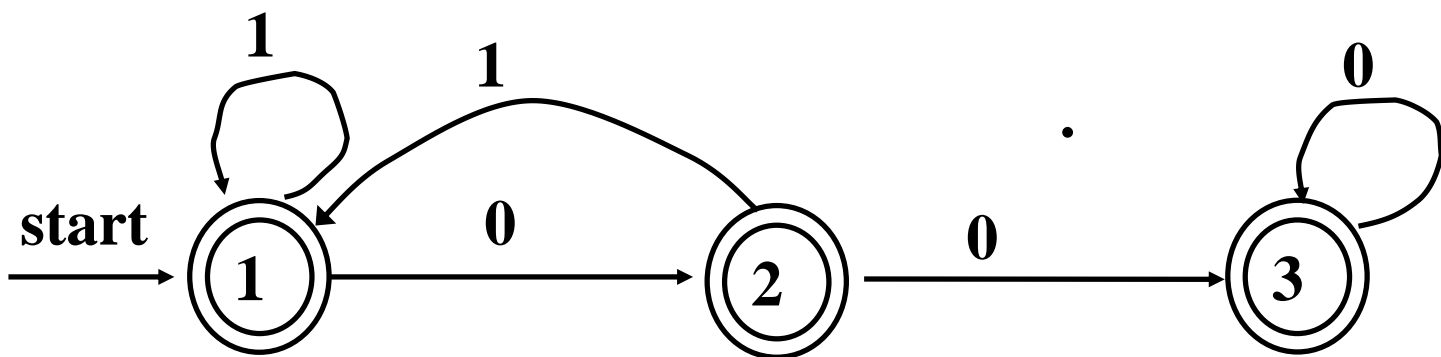
# 例题7

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

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

## 解答

描述的语言是，所有不含子串001的、由0和1组成的串



刚读过的不是0

连续读过一个0

连续读过  
不少于两个0

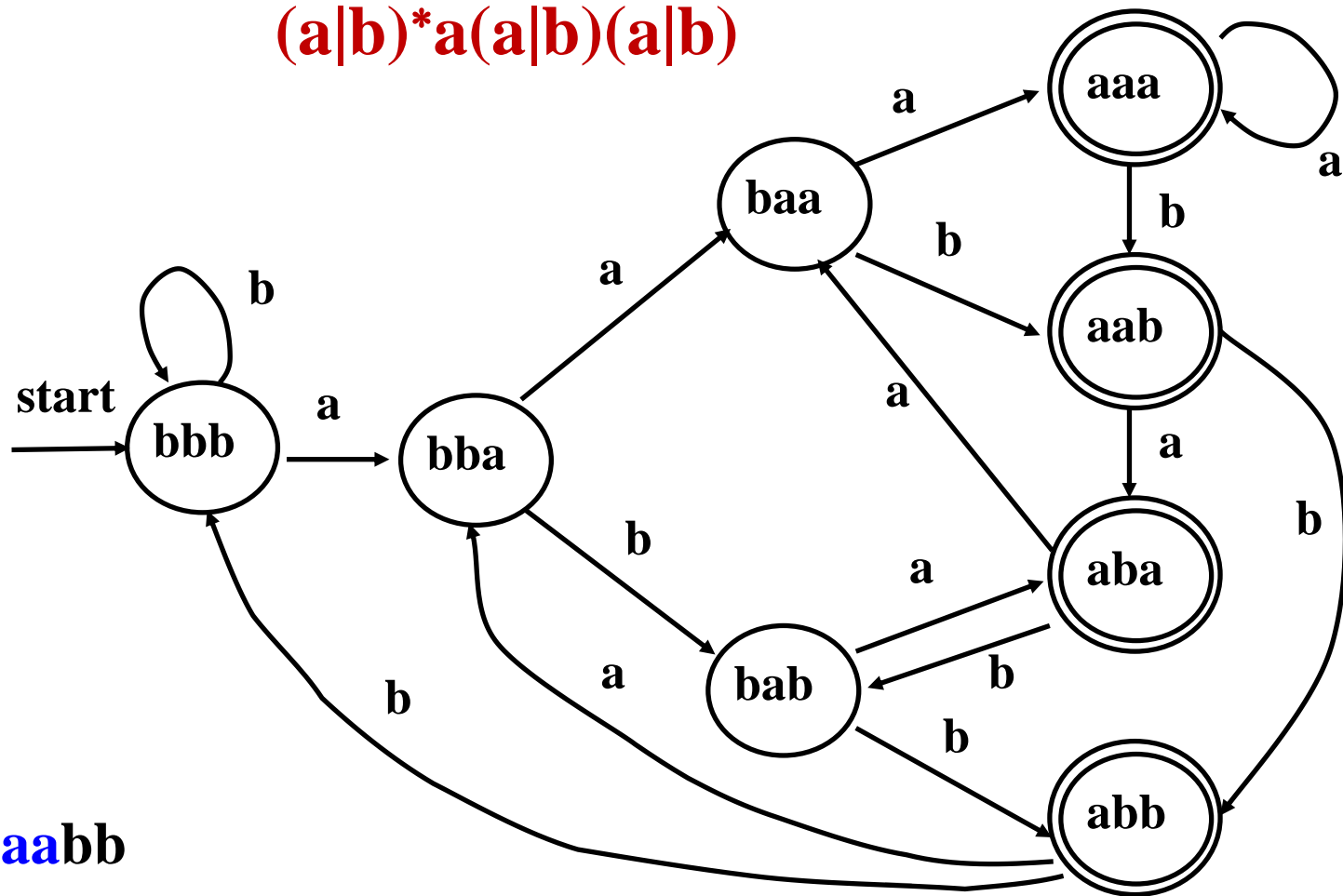


# 例题8

用状态转换图表示接受如下正规式的DFA

$(a|b)^*a(a|b)(a|b)$

解答





# 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**
- **掌握下面的相关概念，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）**