



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



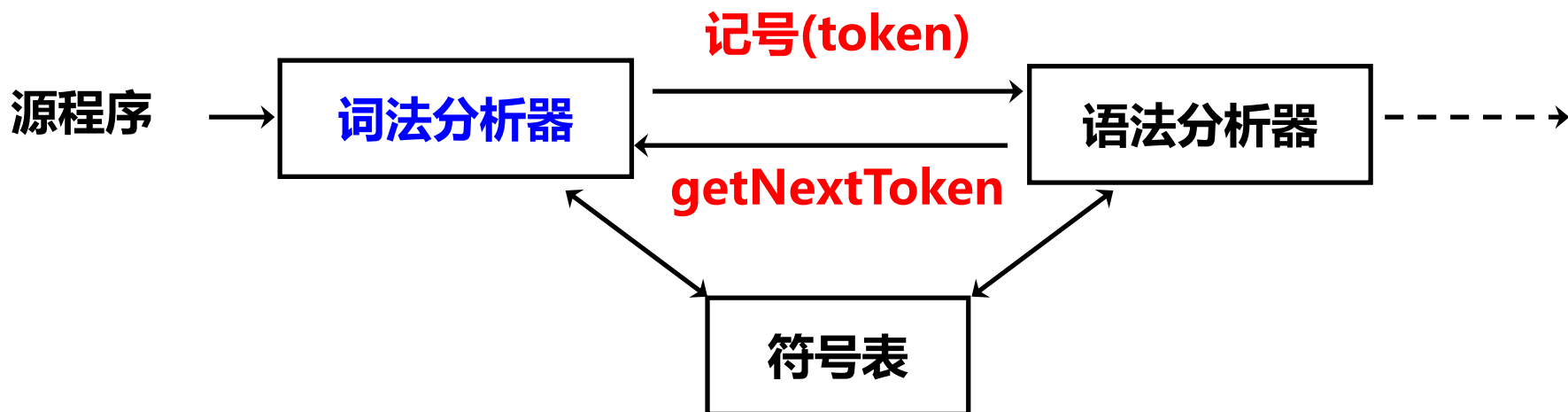
# 《编译原理与技术》

## 词法分析

计算机科学与技术学院

李诚

06/09/2018



## □ 词法分析所面临的问题

- ❖ 向前看 (Lookahead)、歧义 (Ambiguities)

## □ 词法分析器的自动生成

- ❖ 词法单元的描述：正则式
- ❖ 词法单元的识别：转换图
- ❖ 有限自动机：NFA、DFA

## □ 程序示例:

```
if (i == j)
    printf("equal!");
else
    num5 = 1;
```

## □ 程序是以字符串的形式传递给编译器的

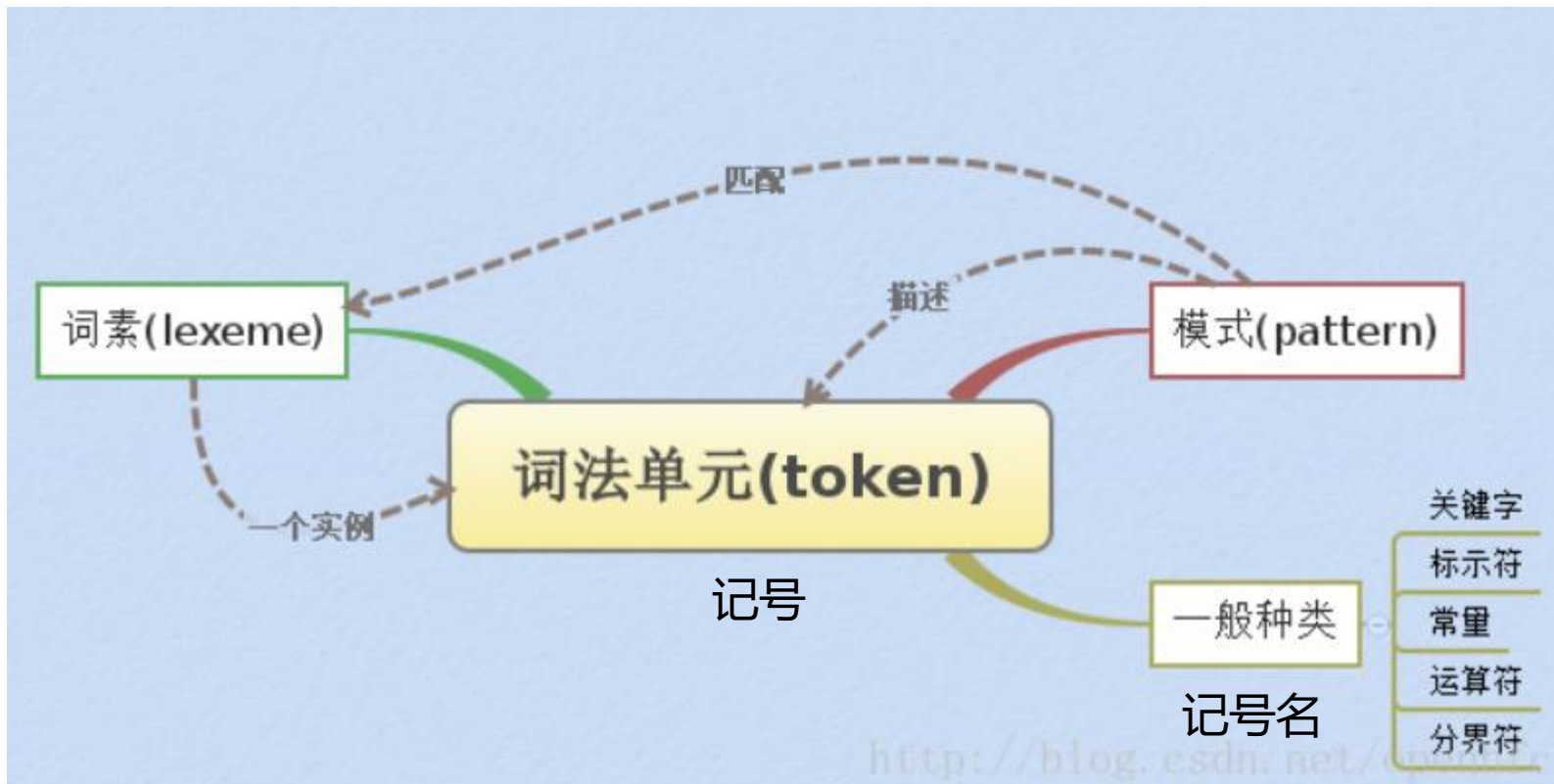
```
\tif (i == j)\n\t\tprintf("equal!");\n\t\telse\n\t\t\tnum5 = 1;
```

## □ 目的: 将输入字符串识别为**有意义的子串**

- ❖ 子串的种类 (Name)
- ❖ 可帮助解释和理解该子串的属性 (Attribute)
- ❖ 可描述具有相同特征的字串的模式 (Pattern)



# 四个关键术语





```
if (i == j) printf("equal!");  
else num5 = 1;
```

记号名	实例	模式的非形式描述
if	if	字符i, f
else	else	字符e, l, s, e
relation	==, <, <=, ...	== 或 < 或 <= 或 ...
id	i, j, num5	由字母开头的字母数字串
number	1, 3.1, 10, 2.8 E12	任何数值常数
literal	"equal!"	引号“和”之间任意不含引号本身的字符串



## □ 属性记录词法单元的附加属性，影响语法分析 对该词法单元的翻译

- ❖ 例：标识符id的属性会包括词法单元实例、类型、第一次出现的位置等
- ❖ 保存在符号表（Symbol table）中，以便编译的各个阶段取用

- 由一个记号名和一个可选的属性值组成
  - ❖ `token := <token_name, attribute_value>`

□ 由一个记号名和一个可选的属性值组成

❖  $\text{token} := \langle \text{token\_name}, \text{attribute\_value} \rangle$

□ 示例:

**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**⟩

## 符号表

1	<b>position</b>	...
2	<b>initial</b>	...
3	<b>rate</b>	...

词素  
(实例)



## □ 怎么识别相近的实例?

❖ **i vs. if** 或者 **= vs. ==**

## □ 忽略空格带来的困难

**DO 8 | = 3. 75** 等同于 **DO8| = 3. 75**

**DO 8 | = 3, 75** 其中 **DO** 是关键字, **8**是语句标号...

## □ 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

## □ 关键字不保留

**IF THEN THEN=ELSE; ELSE ...**

## □ 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** 实例的结尾、下一个实例的开始（尤其是在 **Fortran** 语言中）



## □ 词法分析的目标

- ❖ 将输入字符流识别成记号的实例（词素）
- ❖ 识别出每个实例对应的记号（词法单元）

## □ 从左到右的扫描 => 需要向前看 (lookahead) 这样的方式



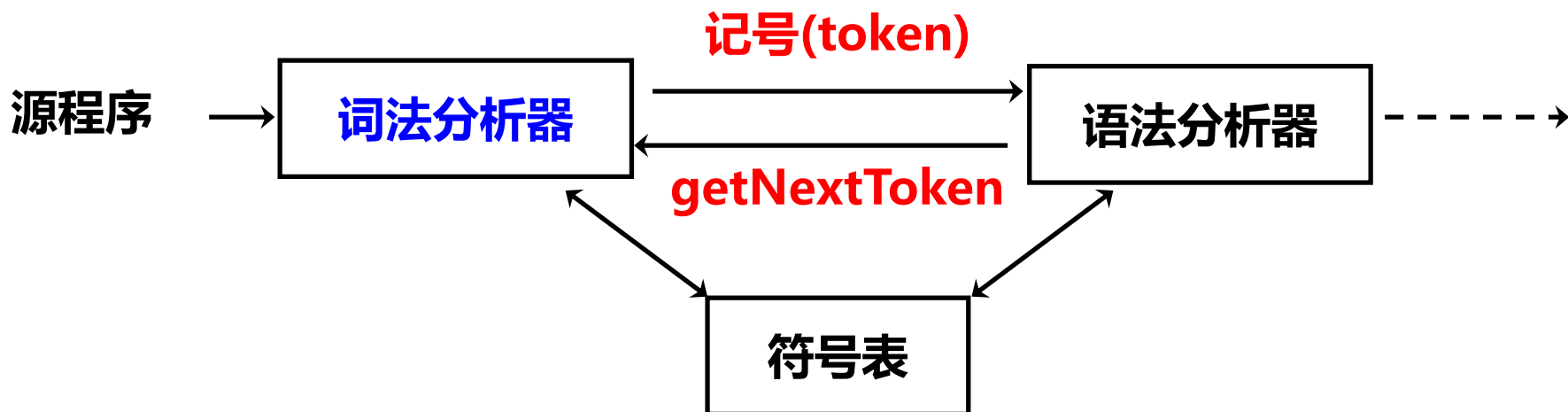
□ 一种可以描述记号所对应的全部实例的方法

❖  $id \Rightarrow \{a, b, cat, dog, \dots\}$

□ 一种可以消除歧义的方法

❖  $if$  是关键字  $if$  还是两个变量  $i, f$ ?

❖  $==$  是两个等号  $= =$ ?



## □ 词法分析所面临的问题

- ❖ 向前看 (Lookahead)、歧义 (Ambiguities)

## □ 词法分析器的自动生成

- ❖ 词法单元的描述：正则式
- ❖ 词法单元的识别：转换图
- ❖ 有限自动机：NFA、DFA

## □ 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）**： $xy, s\varepsilon = \varepsilon s = s$
- **指数（幂）**： $s^0$ 为 $\varepsilon$ ， $s^i$ 为 $s^{i-1}s$  ( $i > 0$ )

## □ 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:  $L^+ = L^1 \cup L^2 \cup \dots$

优先级:  
幂) 连接) 并

## □ 示例

$L: \{A, B, \dots, Z, a, b, \dots, z\}$ ,  $D: \{0, 1, \dots, 9\}$   
 $L \cup D$ ,  $LD$ ,  $L^6$ ,  $L^*$ ,  $L(L \cup D)^*$ ,  $D^+$





## □ 正则式用来表示简单的语言

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

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

优先级:  
闭包\* > 连接 > 选择 |

□  $\Sigma = \{a, b\}$

- ❖  $a \mid b$   $\{a, b\}$
- ❖  $(a \mid b)(a \mid b)$   $\{aa, ab, ba, bb\}$
- ❖  $aa \mid ab \mid ba \mid bb$   $\{aa, ab, ba, bb\}$
- ❖  $a^*$  由字母  $a$  构成的所有串集
- ❖  $(a \mid b)^*$  由  $a$  和  $b$  构成的所有串集

□ 复杂的例子

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

句子: **01001101000010000010111001**

## □ bottom-up方法

- ❖ 对于比较复杂的语言，为了构造简洁的正则式，可先构造简单的正则式，再将这些正则式组合起来，形成一个与该语言匹配的正则序列。

$$d_1 \rightarrow r_1$$

$$d_2 \rightarrow r_2$$

...

$$d_n \rightarrow r_n$$

- ❖ 各个 $d_i$ 的名字都不同，是符号
- ❖ 每个 $r_i$ 都是  $\Sigma \cup \{d_1, d_2, \dots, d_{i-1}\}$  上的正则式

## 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**)\*

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

### – 简化表示

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



**while** → while

**do** → do

**relop** → < | <= | = | <> | > | >=

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

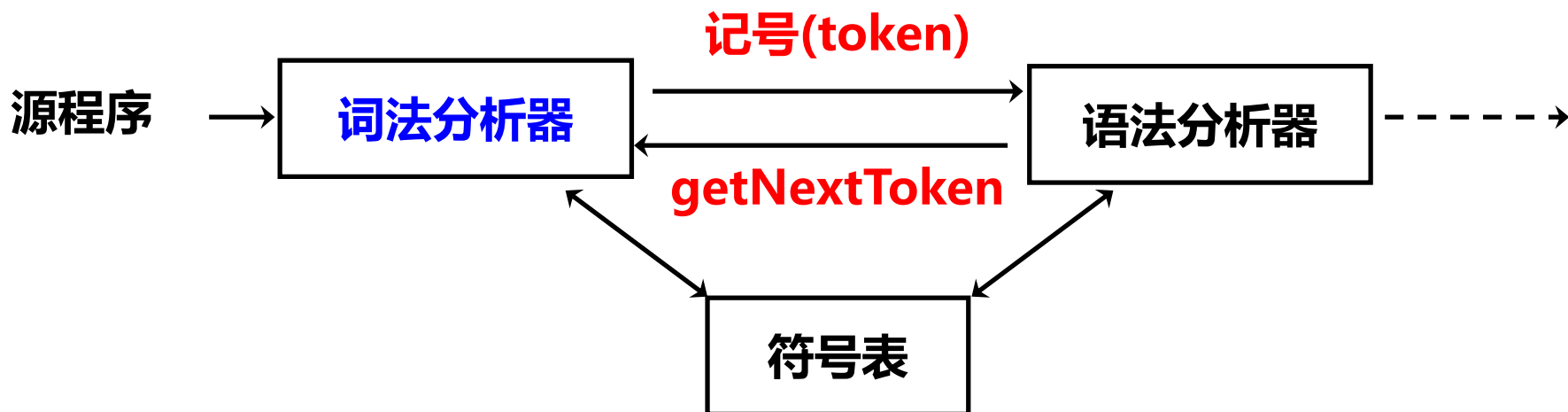
**id** → letter (letter | digit)\*

**number** → digit<sup>+</sup> (.digit<sup>+</sup>)? (E (+ | -)? digit<sup>+</sup>)?

**delim** → blank | tab | newline

**ws** → delim<sup>+</sup>

**问题：正则式是静态的定义，如何通过正则式动态识别输入串？**



## □ 词法分析所面临的问题

- ❖ 向前看 (Lookahead)、歧义 (Ambiguities)

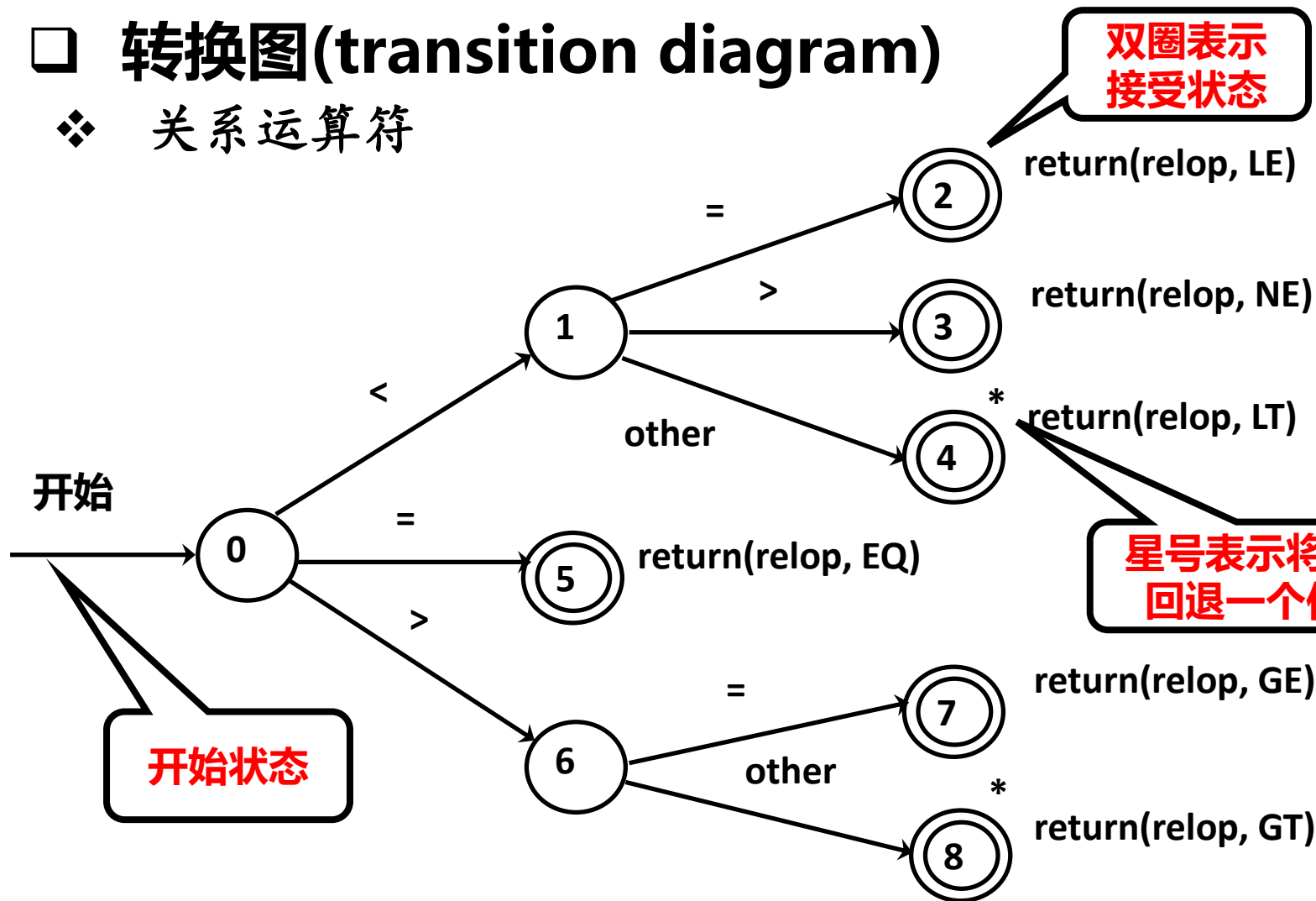
## □ 词法分析器的自动生成

- ❖ 词法单元的描述：正则式
- ❖ 词法单元的识别：转换图
- ❖ 有限自动机：NFA、DFA



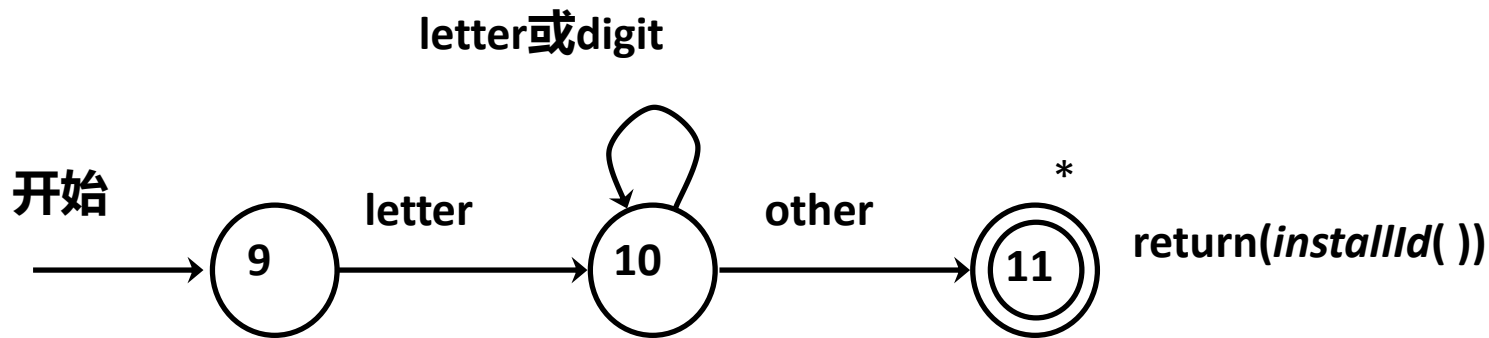
## 转换图(transition diagram)

❖ 关系运算符



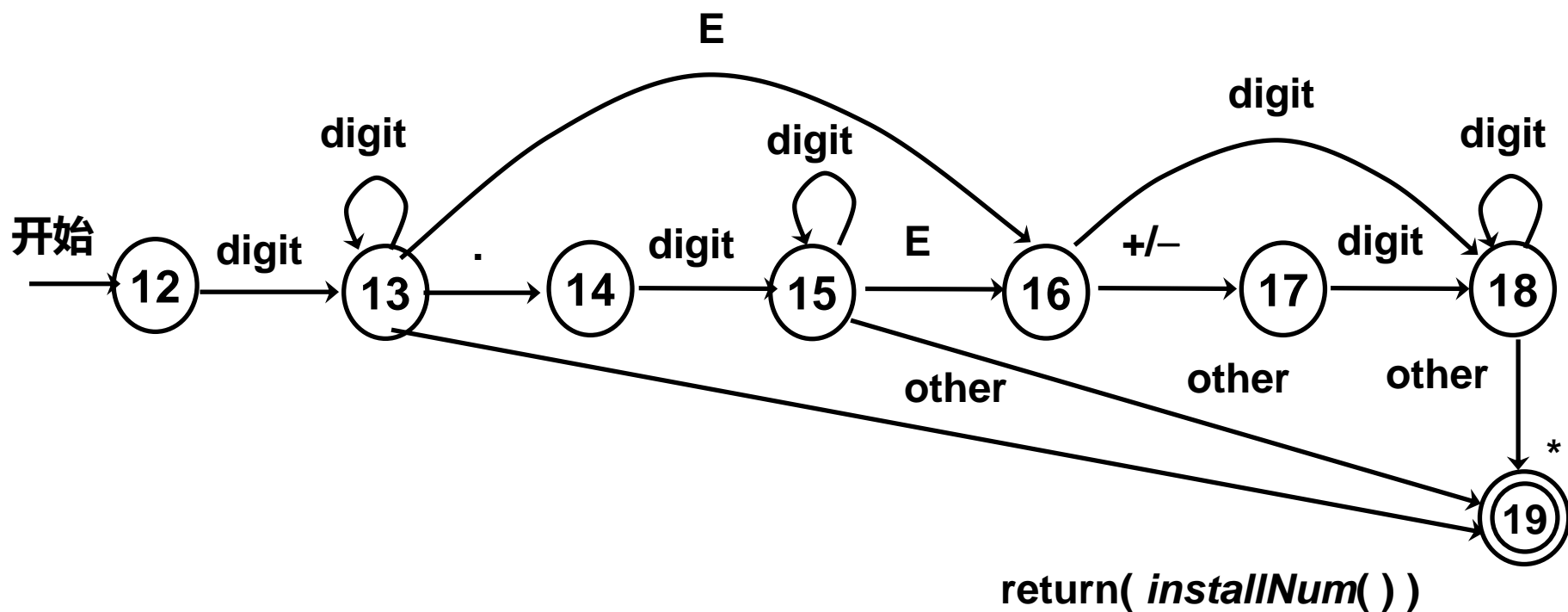


## □ 标识符和关键字的转换图



## □ 无符号数的转换图

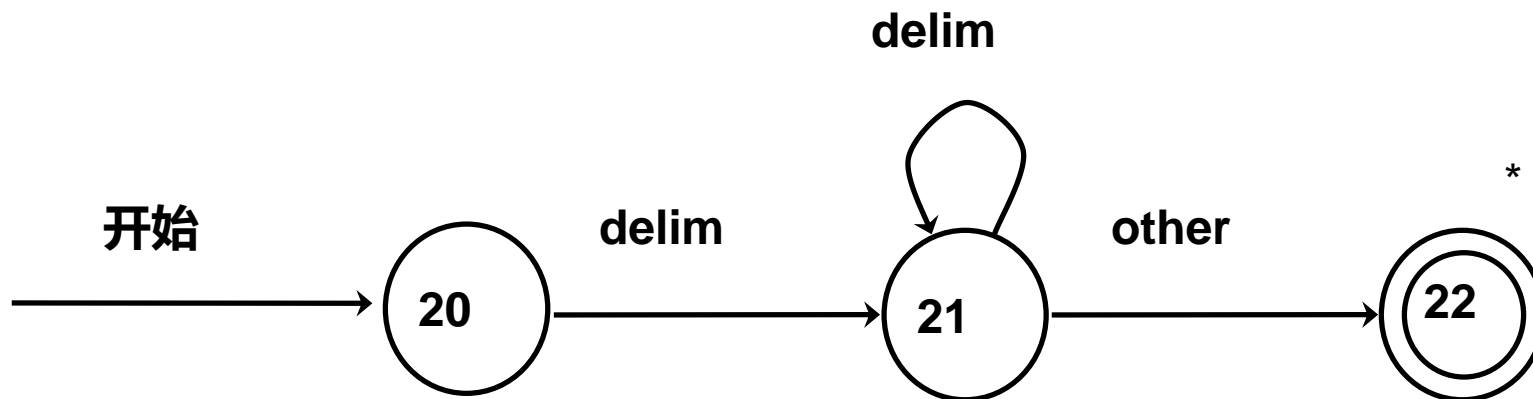
$\text{number} \rightarrow \text{digit}^+ (. \text{digit}^+)? (E (+ | -)? \text{digit}^+)?$



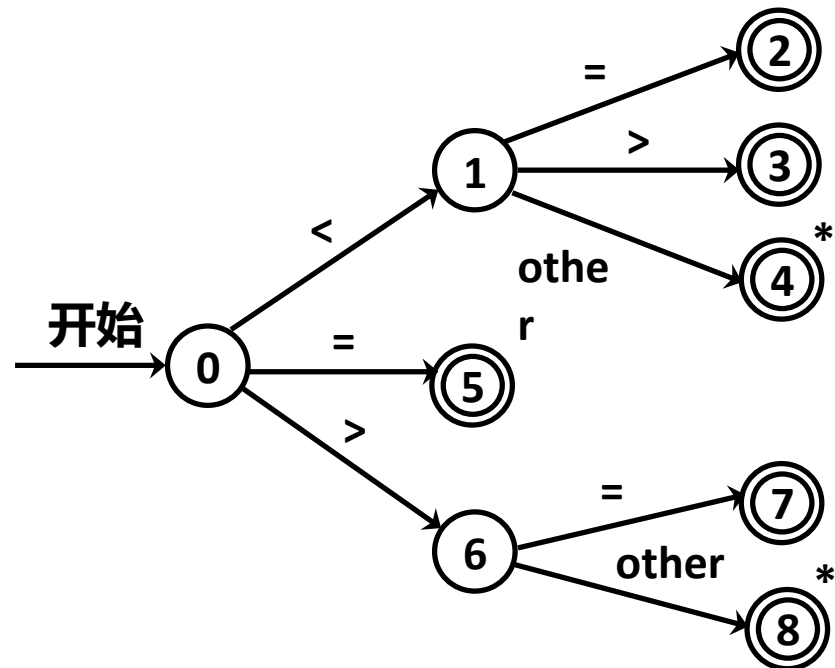
## □ 空白的转换图

delim  $\rightarrow$  blank | tab | newline

ws  $\rightarrow$  delim+



## □ 例：relop的转换图的概要实现



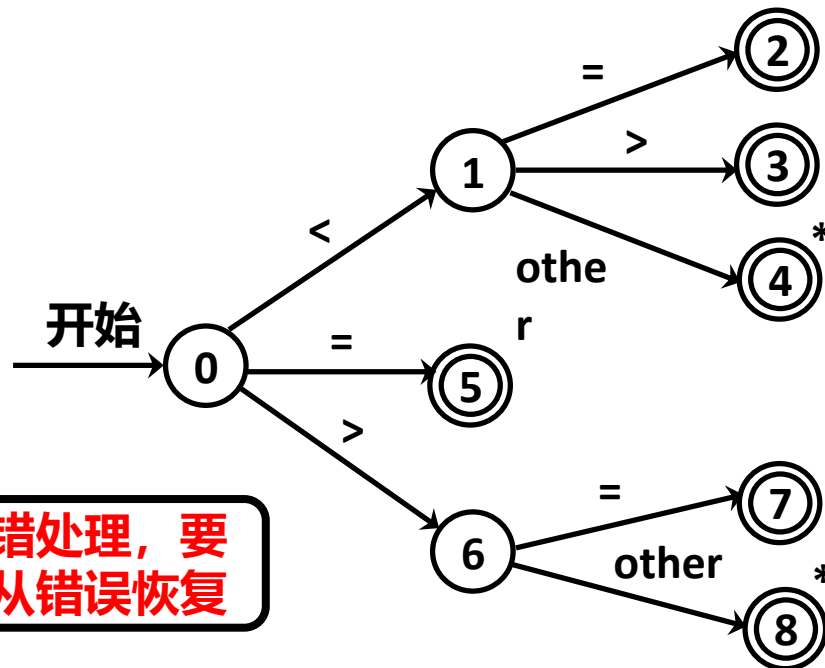


## □ 例：relop的转换图的概要实现

```

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**

**R = Whitespace | Integer | Identifier | '+'**

识别 “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  $x_1 \dots x_k \in L(R)$

Maximal match 规则:

❖ 选择匹配 R 的最长前缀

**最长匹配规则在实现时: lookahead, 不符合则回退**

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

识别 “new foo”

❖ “new” 匹配  $R$ , 更精确地说是 ‘new’

❖ 但是 “new” 也匹配 Identifier

如何处理输入? 如果

❖  $x_1 \dots x_i \in L(R_j)$  并且  $x_1 \dots x_k \in L(R_m)$

优先 match 规则:

❖ 选择先列出的模式 ( $j$  如果  $j < m$ )

❖ 必须将 ‘new’ 列在 Identifier 的前面

## □ 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 `123.x` 中的错误

## □ 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

❖ 会给语法分析器带来混乱

## □ 错误修补

❖ 进行增、删、替换和交换字符的尝试

❖ 变换代价太高，不值得



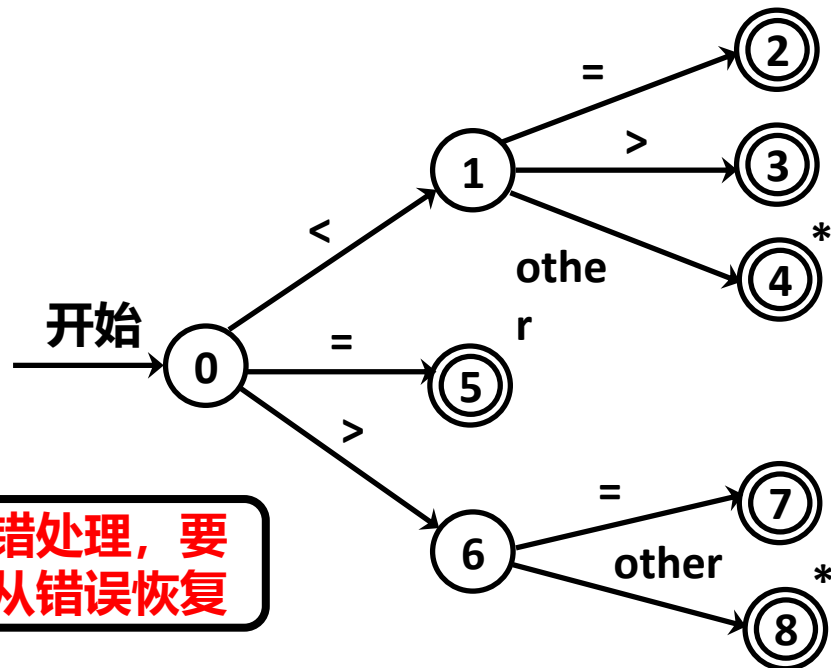


## □ 例：relop的转换图的概要实现

```

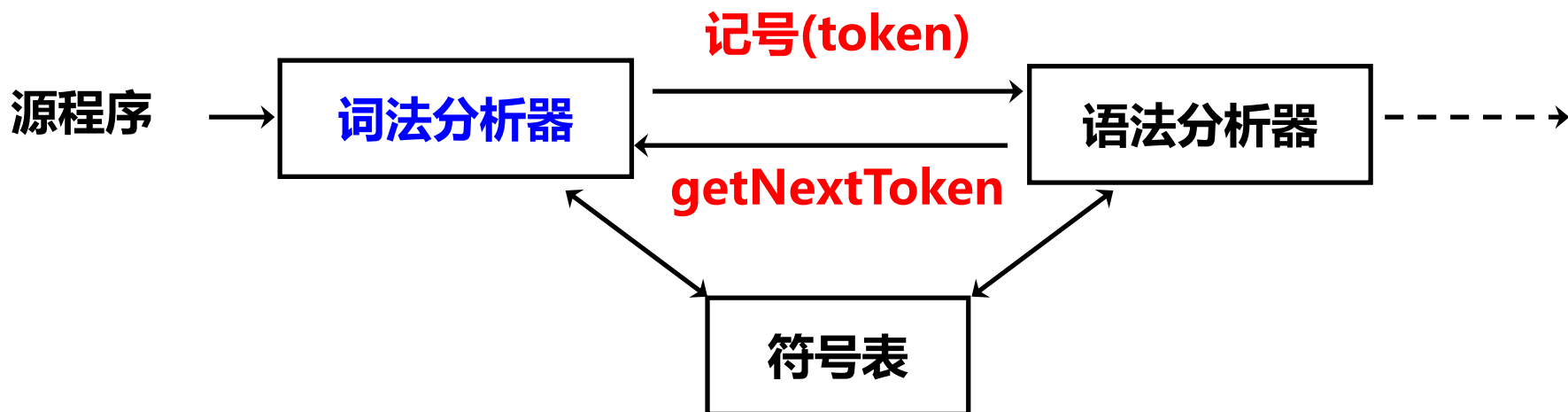
TOKEN getRelop() {
  TOKEN retToken = new(RELOP);
  while (1) {
    switch (state) {
      case 0: c = nextChar();
              if (c == '<') state = 1;
              else if (c == '=') state = 5;
              else if (c == '>') state = 6;
              else fail();
              break;
      case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

```



**出错处理，要  
能从错误恢复**

**问题：怎么为每一个正则定义  
自动找到一个状态转换图？**



## □ 词法分析所面临的问题

- ❖ 向前看 (Lookahead)、歧义 (Ambiguities)

## □ 词法分析器的自动生成

- ❖ 词法单元的描述：正则式
- ❖ 词法单元的识别：转换图
- ❖ 有限自动机：NFA、DFA

□ 不确定的有限自动机（简称NFA）是一个数学模型，它包括：

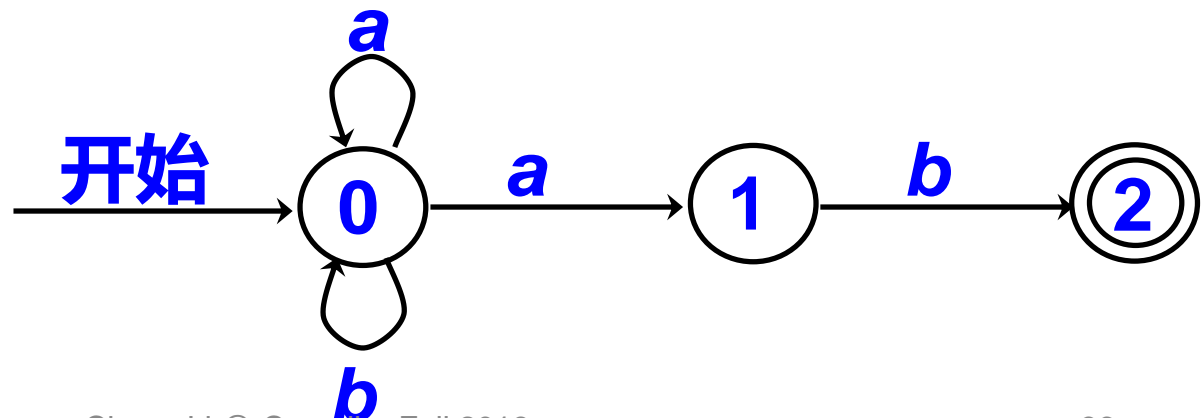
- ❖ 有限的状态集合 $S$
- ❖ 输入符号集合 $\Sigma$
- ❖ 转换函数 $move : S \times (\Sigma \cup \{\epsilon\}) \rightarrow P(S)$
- ❖ 状态 $s_0$ 是唯一的开始状态
- ❖  $F \subseteq S$ 是接受状态集合

幂集

□ 不确定的有限自动机（简称NFA）是一个数学模型，它包括：

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

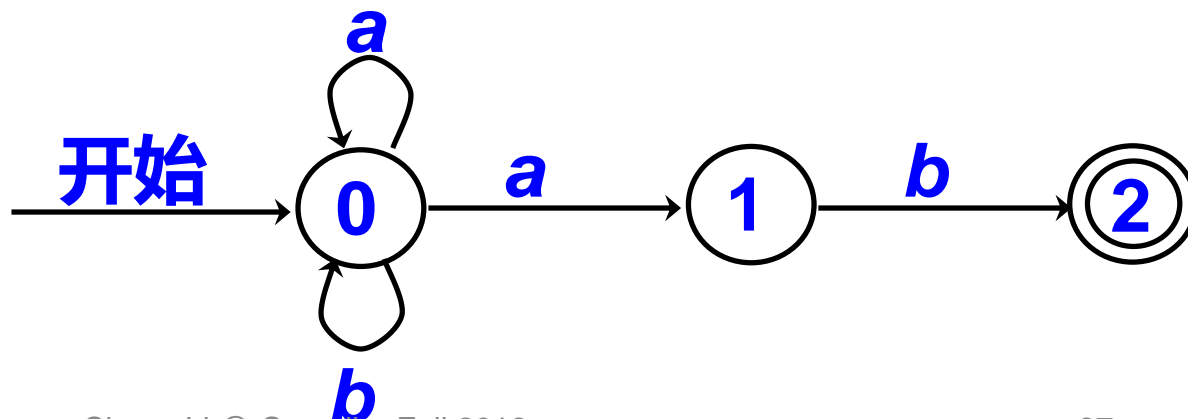
识别语言  
 $(a|b)^* ab$   
的NFA



## □ NFA的转换表

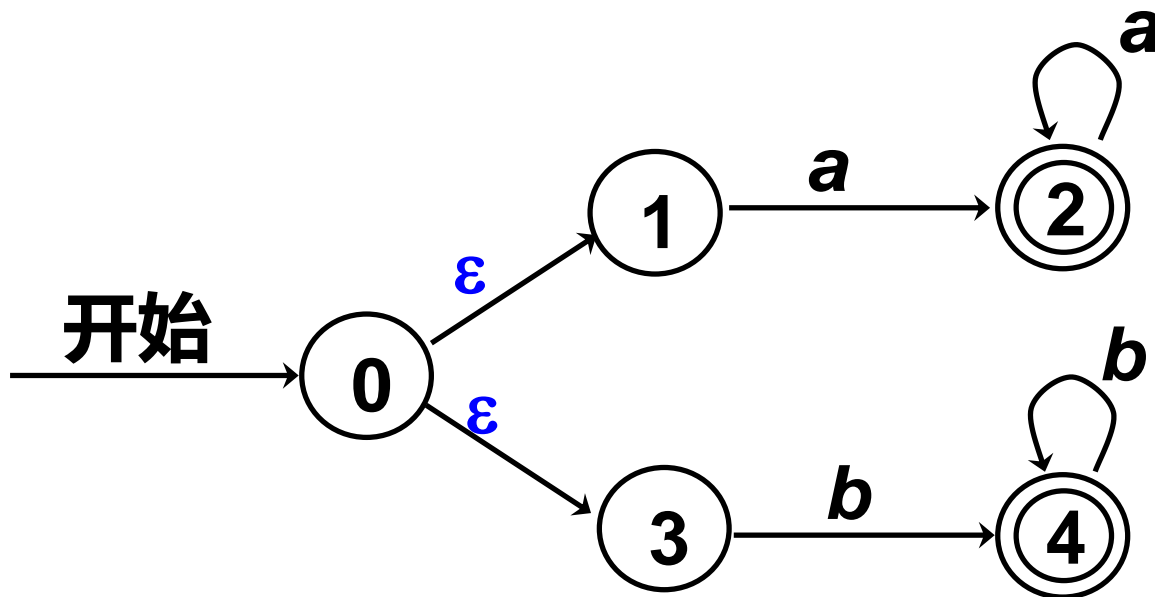
状 态	输 入 符 号	
	<i>a</i>	<i>b</i>
0	{0, 1}	{0}
1	∅	{2}
2	∅	∅

识别语言  
 $(a|b)^* ab$   
的NFA



## □ 例 识别 $aa^*|bb^*$ 的NFA

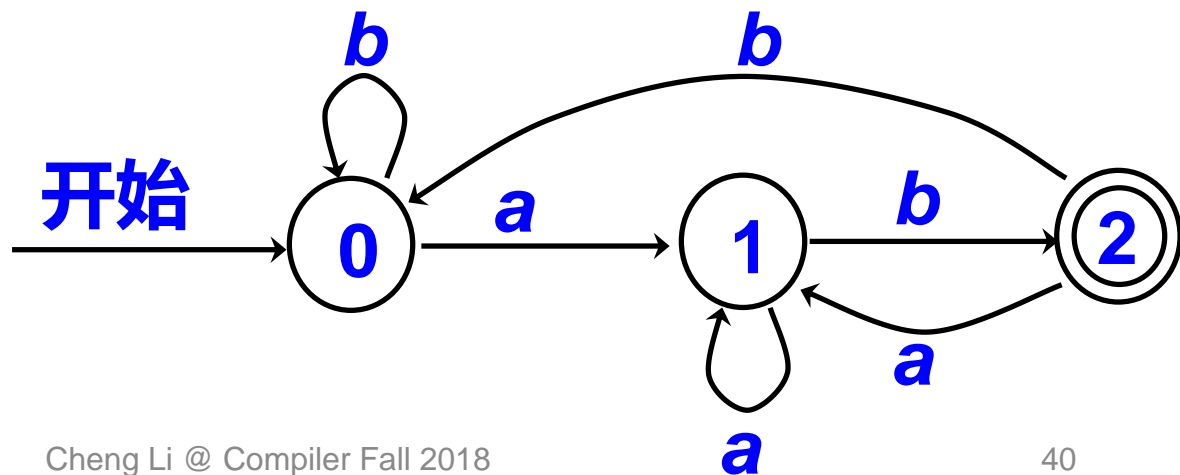
## □ 例 识别 $aa^*|bb^*$ 的NFA



□ 确定的有限自动机 (简称DFA)也是一个数学模型, 包括:

- ❖ 有限的状态集合 $S$
- ❖ 输入符号集合 $\Sigma$
- ❖ 转换函数 $move : S \times \Sigma \rightarrow S$ , 且可以是部分函数
- ❖ 状态 $s_0$ 是唯一的开始状态
- ❖  $F \subseteq S$ 是接受状态集合

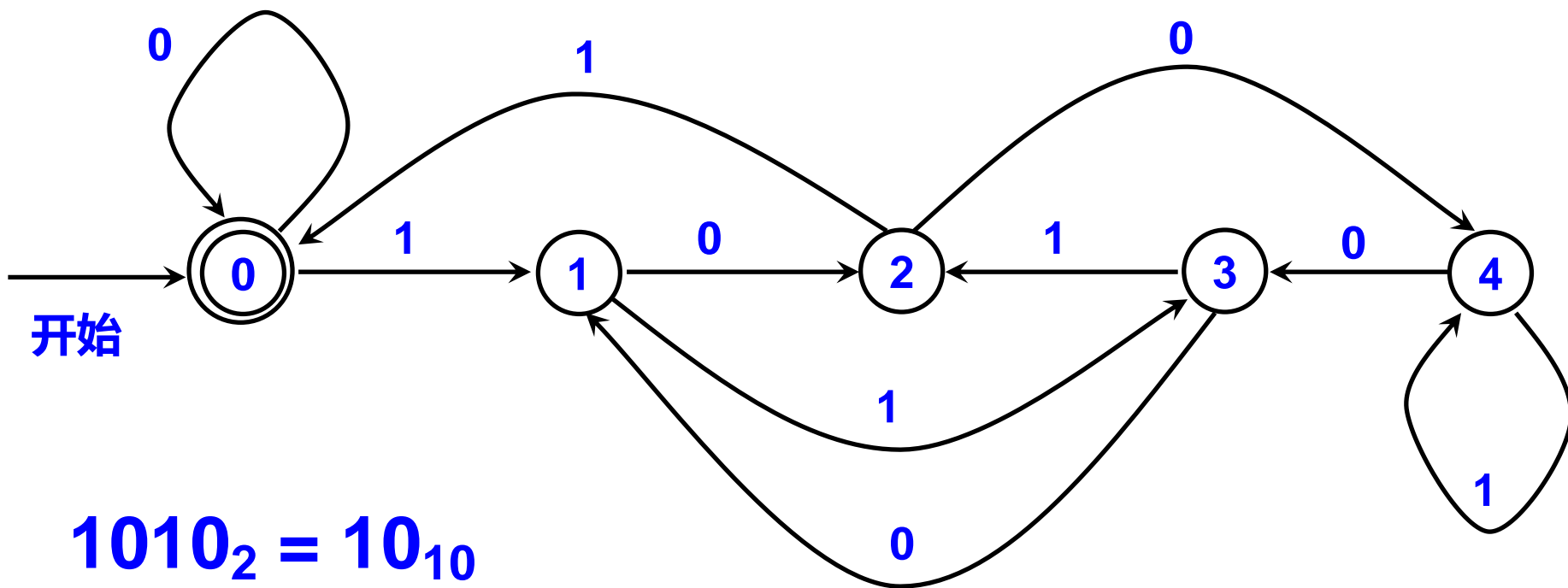
识别语言  
 $(a|b)^*ab$   
的DFA





## □ 例 DFA, 识别 $\{0,1\}$ 上能被5整除的二进制数

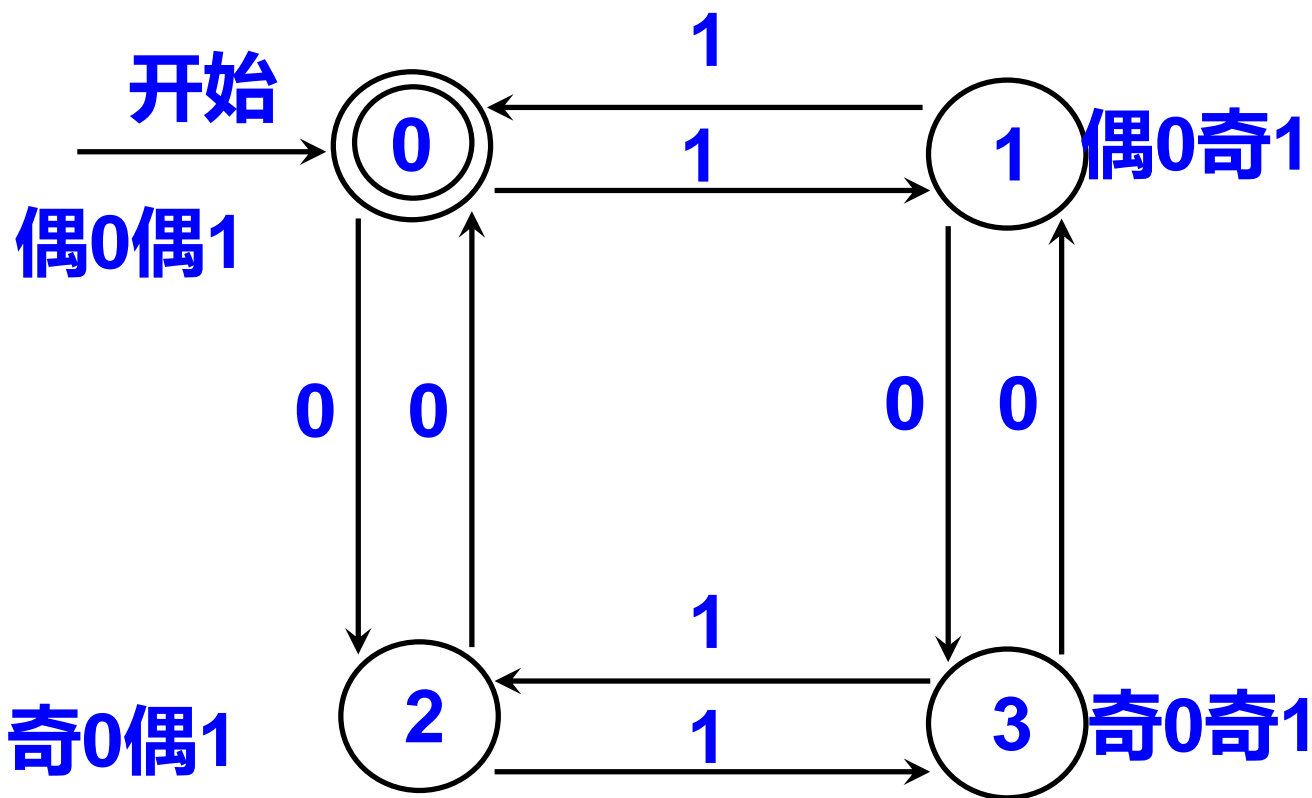
## □ 例 DFA, 识别{0,1}上能被5整除的二进制数



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

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

## □ 例DFA,接受 0和1的个数都是偶数的字符串



## □ NFA到DFA的变换

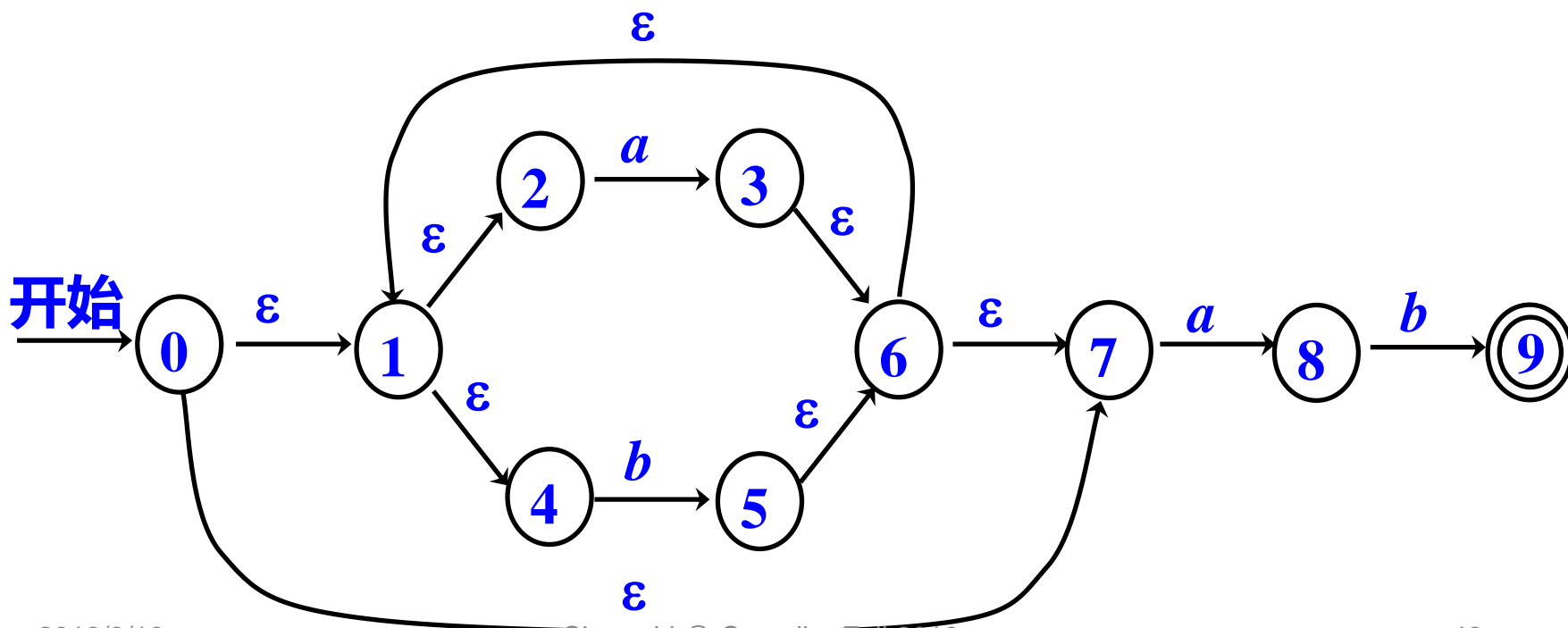
### □ 子集构造法

- ❖ DFA的一个状态是NFA的一个状态集合
- ❖ 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 $\{s_1, s_2, \dots, s_k\}$

## □ 子集构造法(subset construction)

- ❖  $\epsilon$ -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
- ❖ NFA的初始状态的  $\epsilon$ -闭包对应于DFA的初始状态
- ❖ 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 ( $\epsilon$ -closure(move( $A, a_i$ ))), 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  $(a|b)^*ab$ , NFA如下, 把它变换为DFA

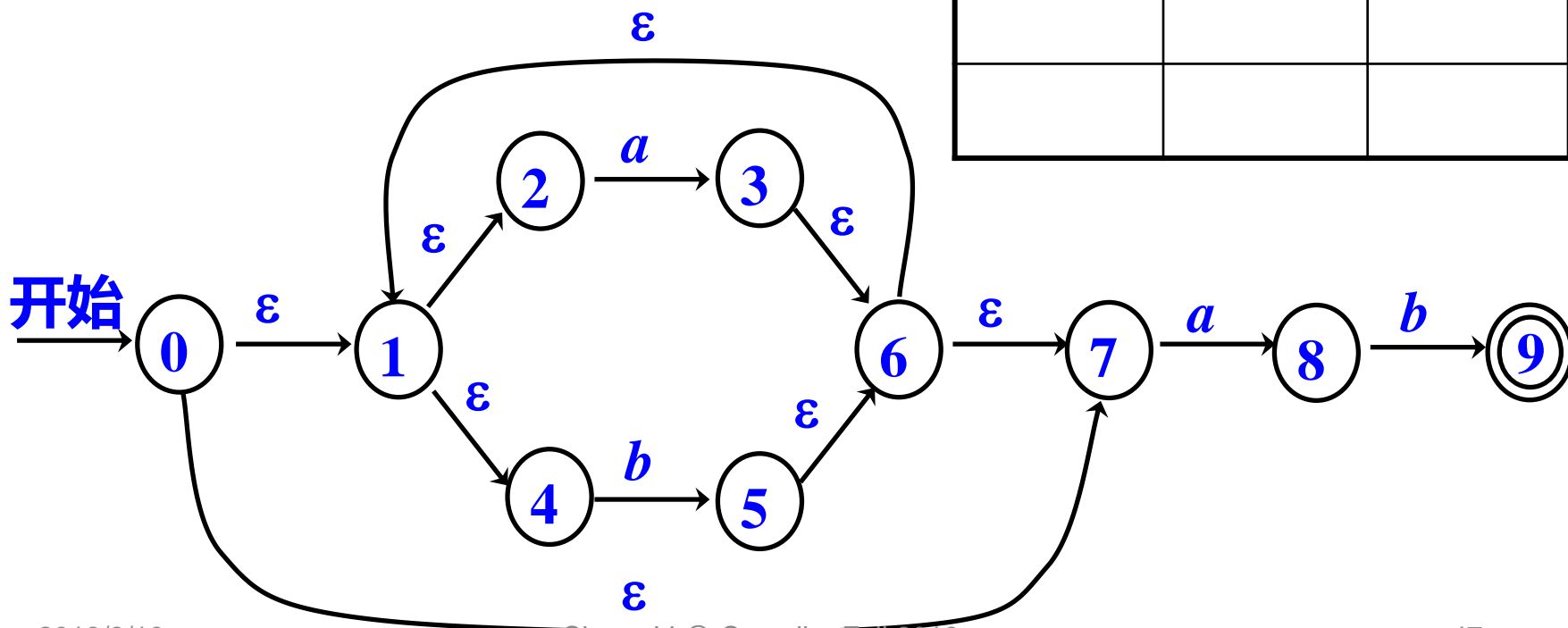




# NFA到DFA的变换

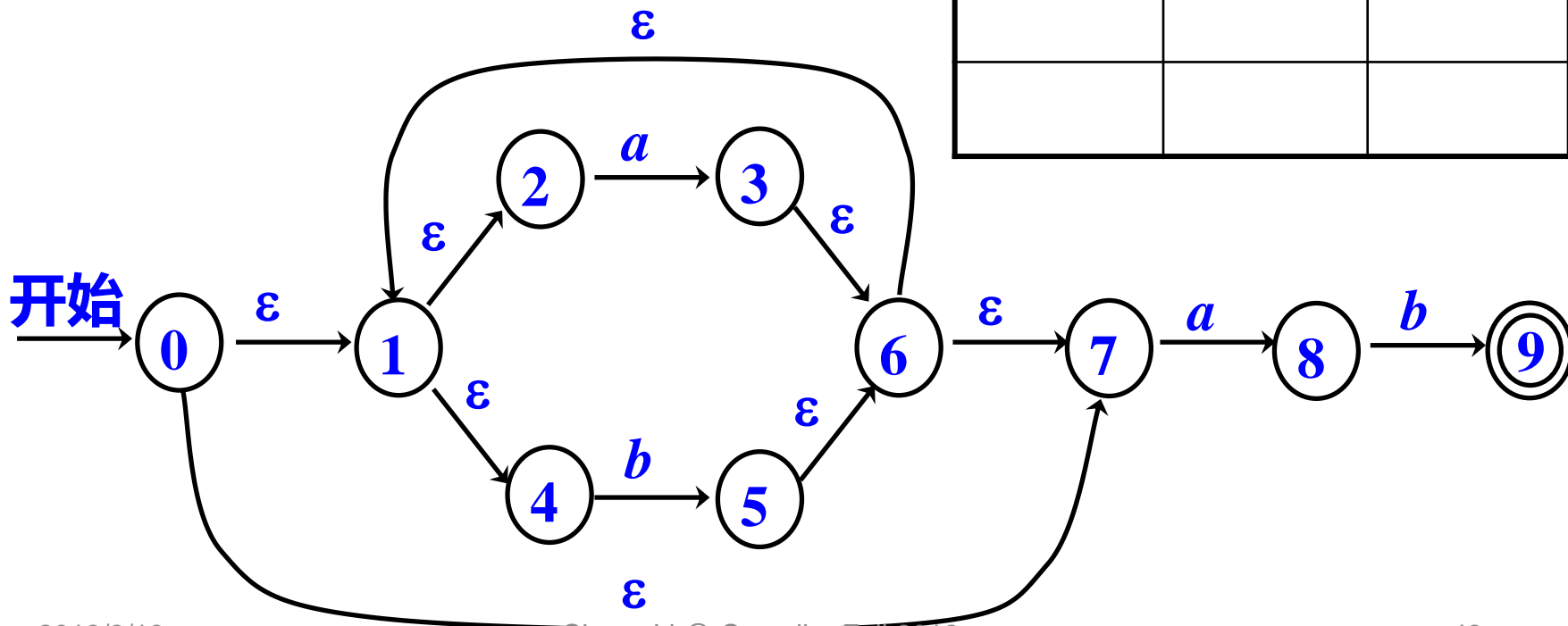


状态	输入符号	
	<i>a</i>	<i>b</i>



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

状态	输入符号	
	<i>a</i>	<i>b</i>
<b>A</b>		





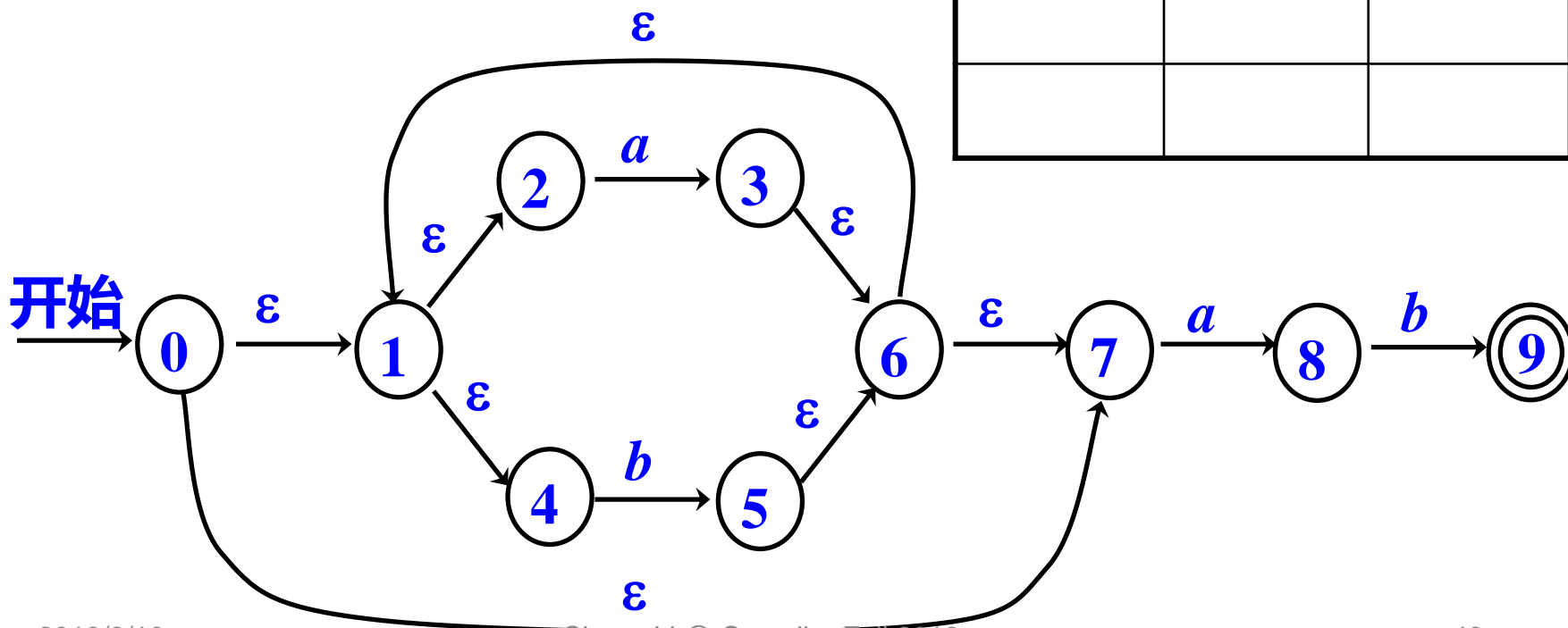
# NFA到DFA的变换



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

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

状态	输入符号	
	$a$	$b$
$A$	$B$	



# NFA到DFA的变换

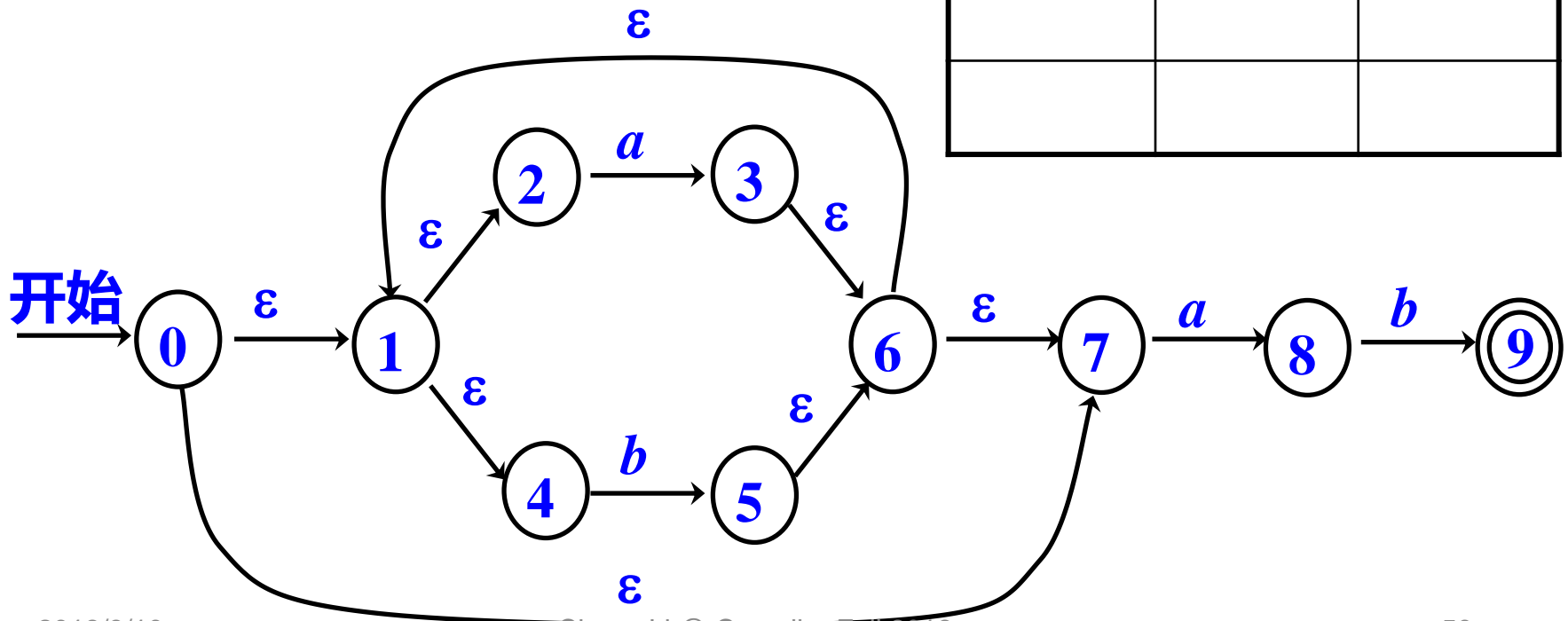


$A = \{0, 1, 2, 4, 7\}$

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

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

状态	输入符号	
	$a$	$b$
$A$	$B$	$C$



# NFA到DFA的变换

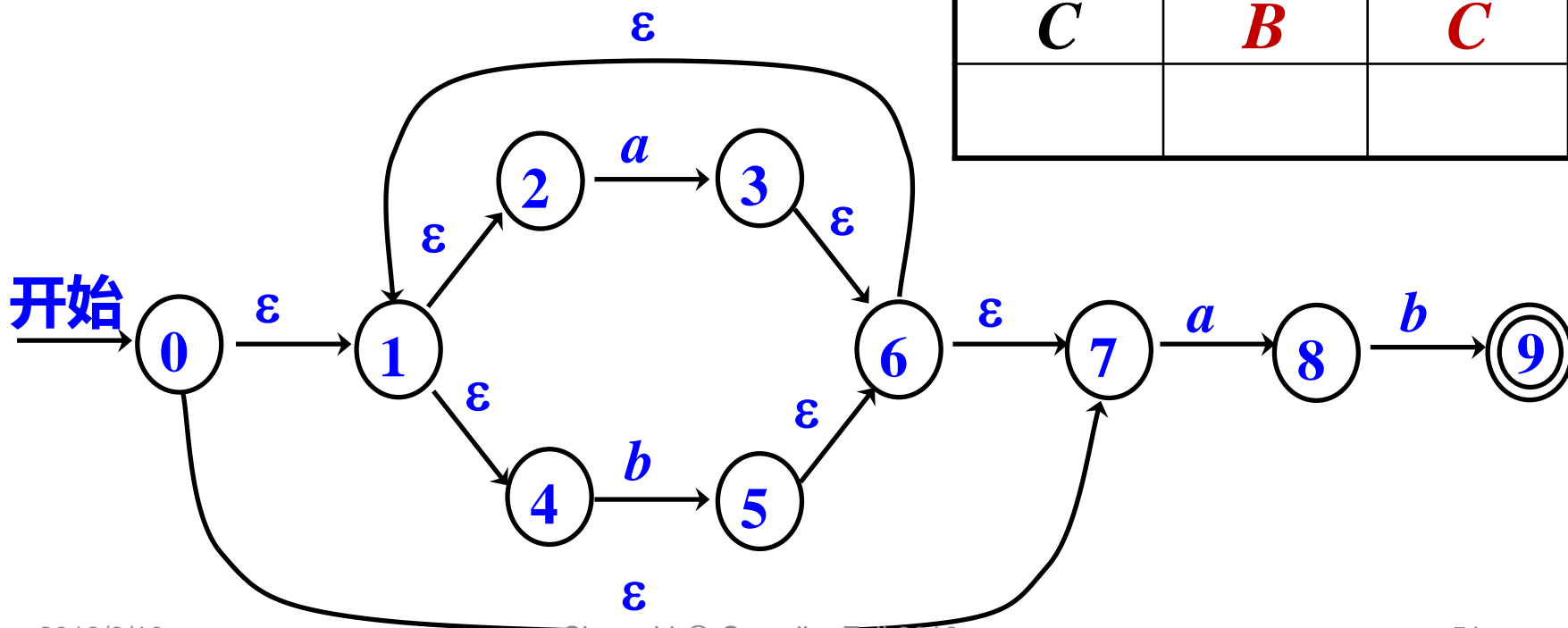


$A = \{0, 1, 2, 4, 7\}$

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

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

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



# NFA到DFA的变换



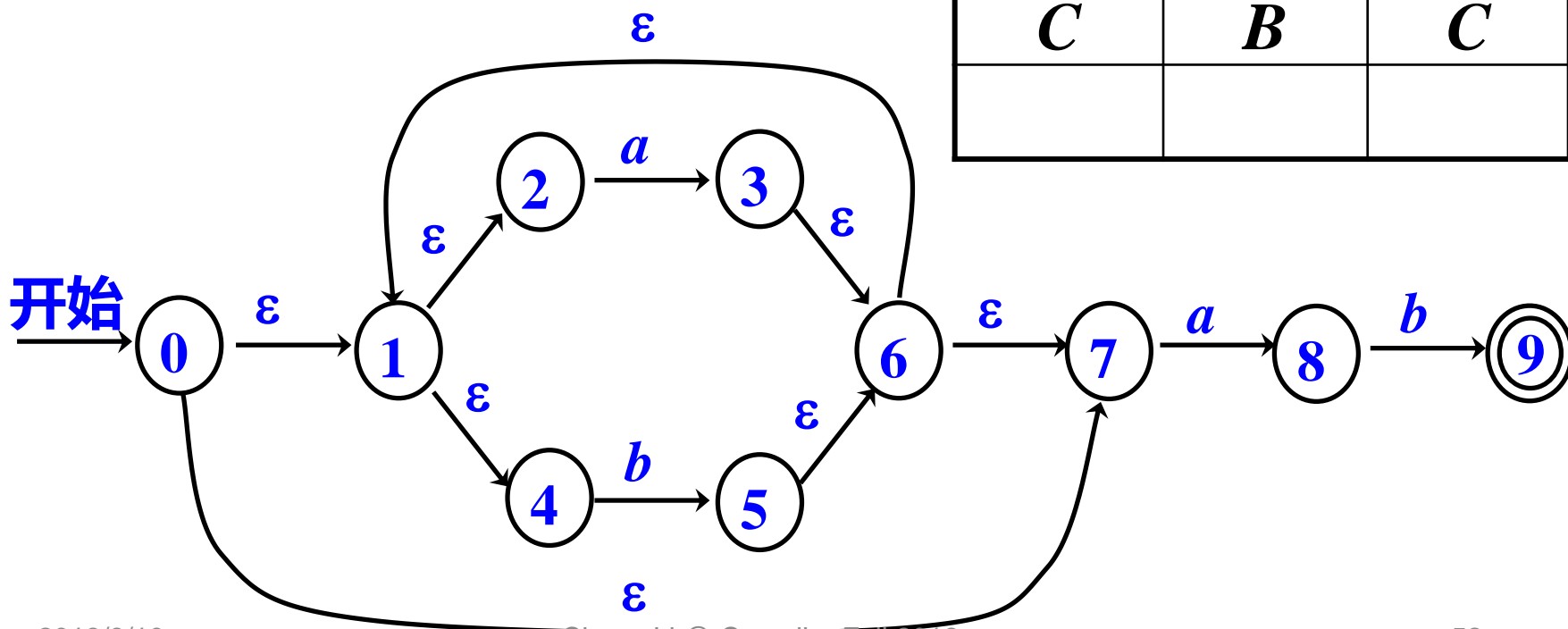
$A = \{0, 1, 2, 4, 7\}$

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

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

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

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





# NFA到DFA的变换



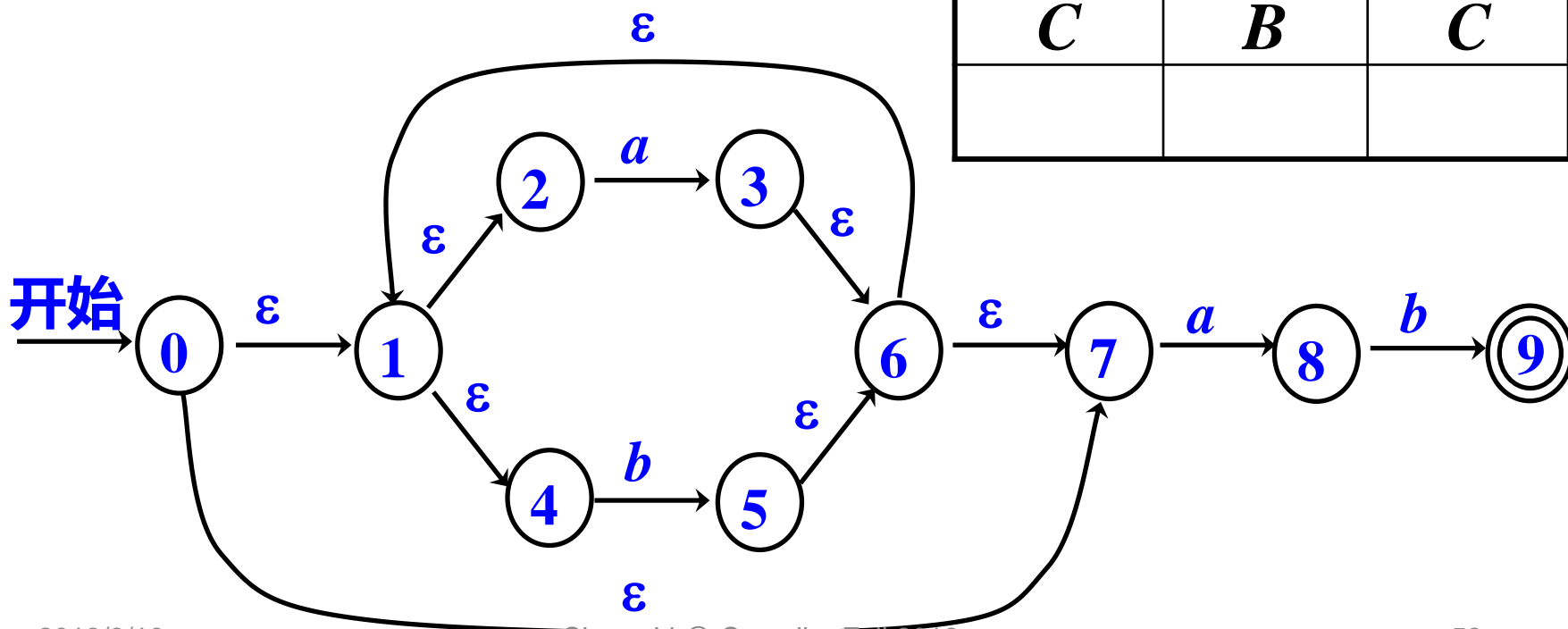
$$A = \{0, 1, 2, 4, 7\}$$

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

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

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

状态	输入符号	
	<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>



# NFA到DFA的变换



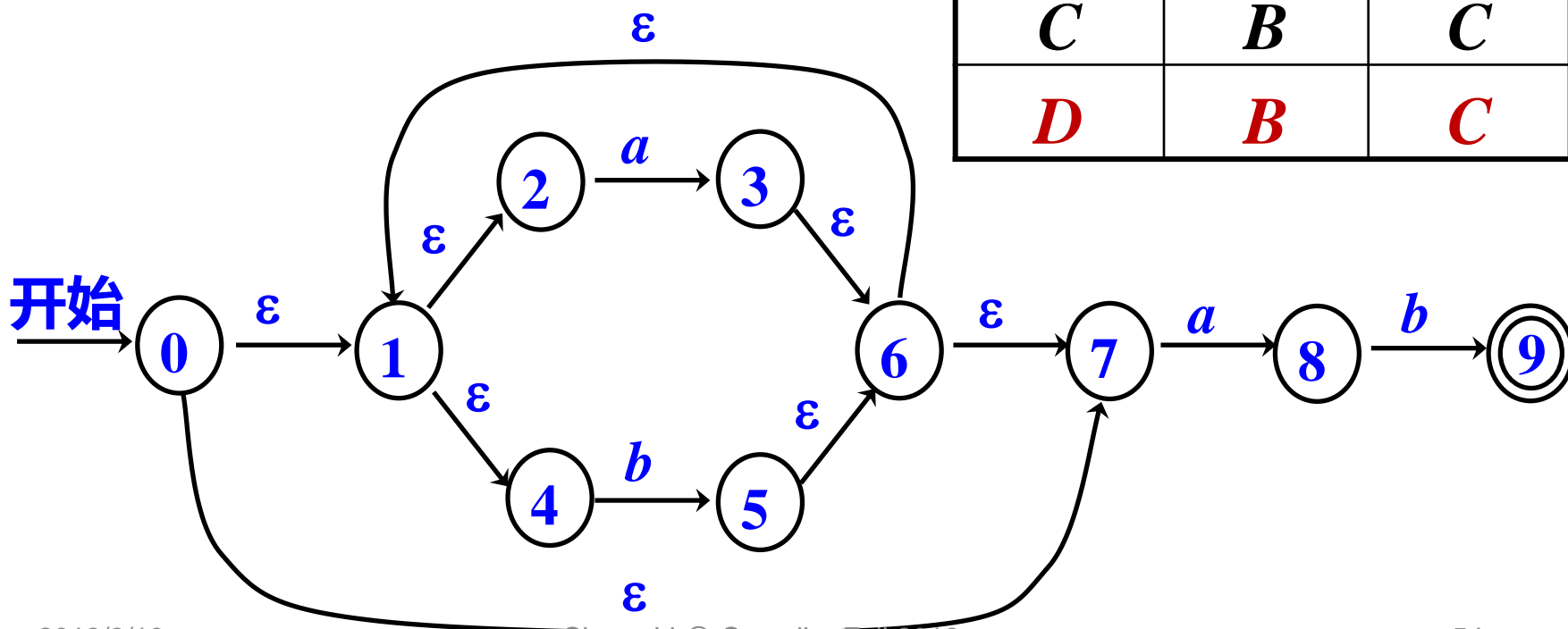
$A = \{0, 1, 2, 4, 7\}$

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

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

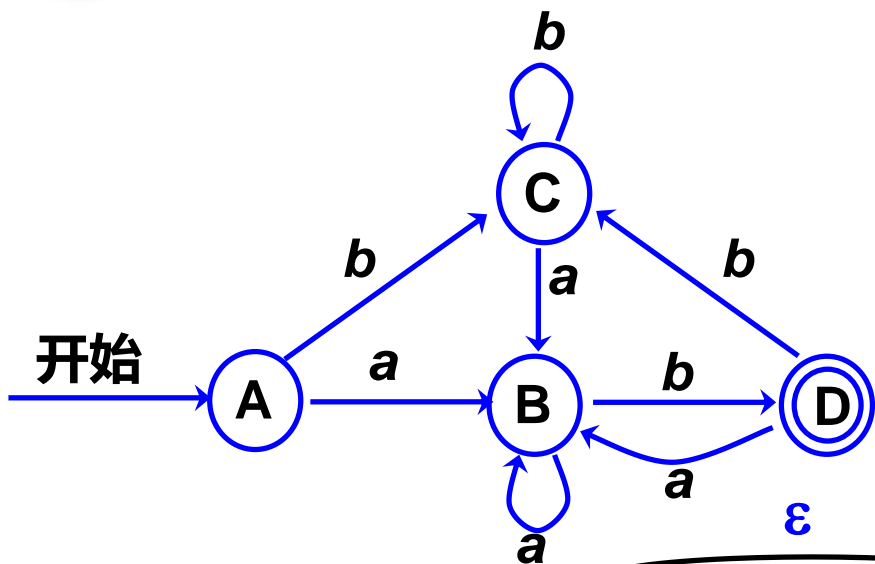
$D = \{1, 2, 4, 5, 6, 7, 9\}$

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

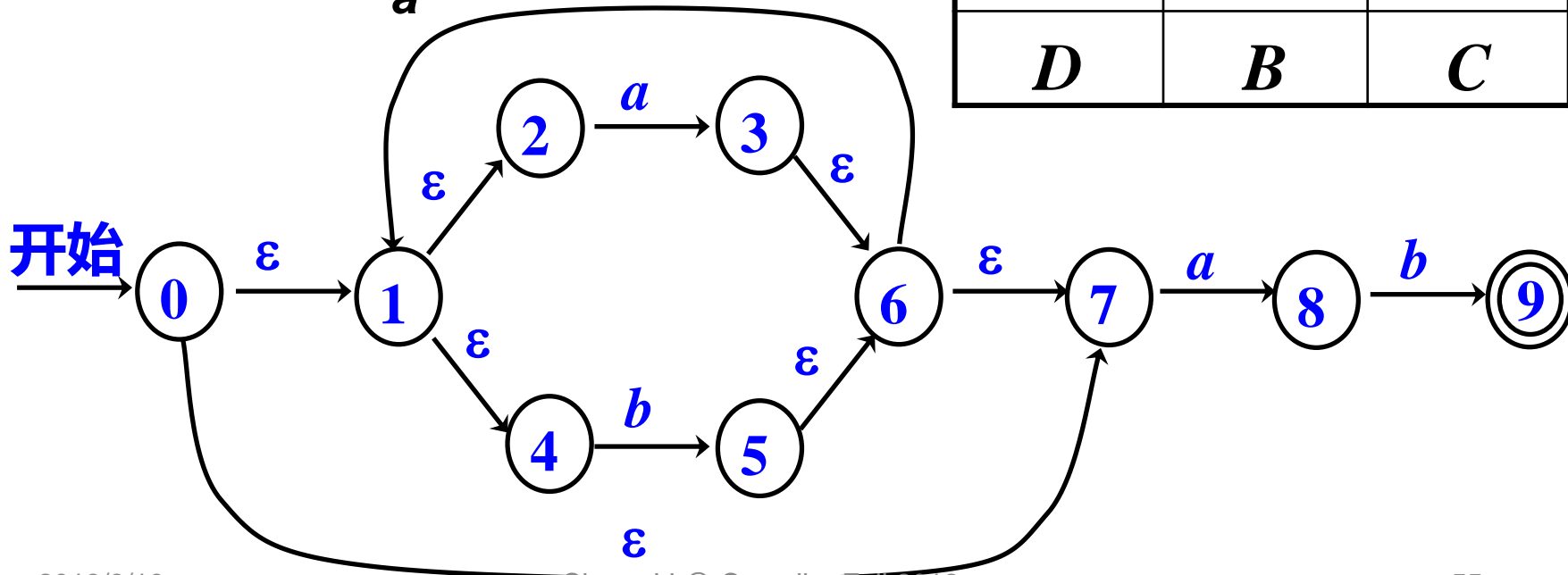




# NFA到DFA的变换

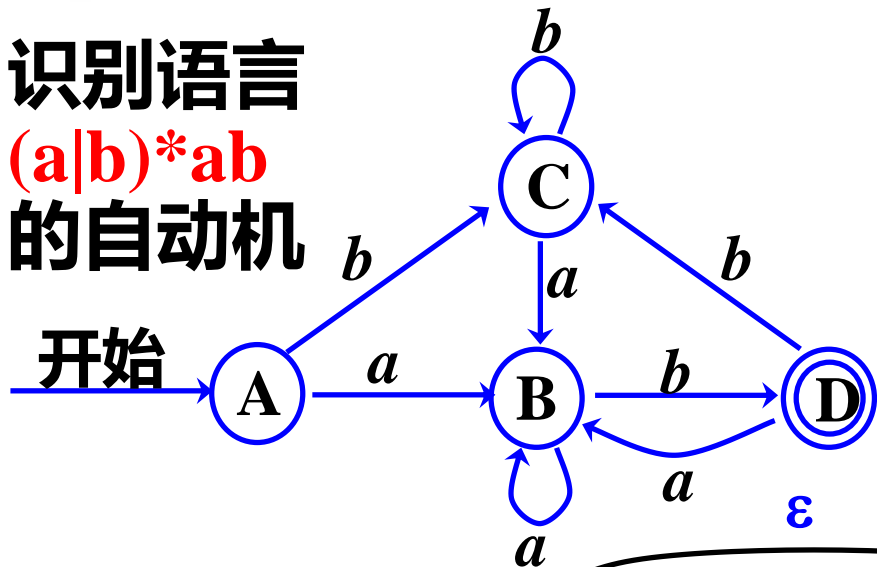


状态	输入符号	
	<i>a</i>	<i>b</i>
<i>A</i>	<i>B</i>	<i>C</i>
<i>B</i>	<i>B</i>	<i>D</i>
<i>C</i>	<i>B</i>	<i>C</i>
<i>D</i>	<i>B</i>	<i>C</i>

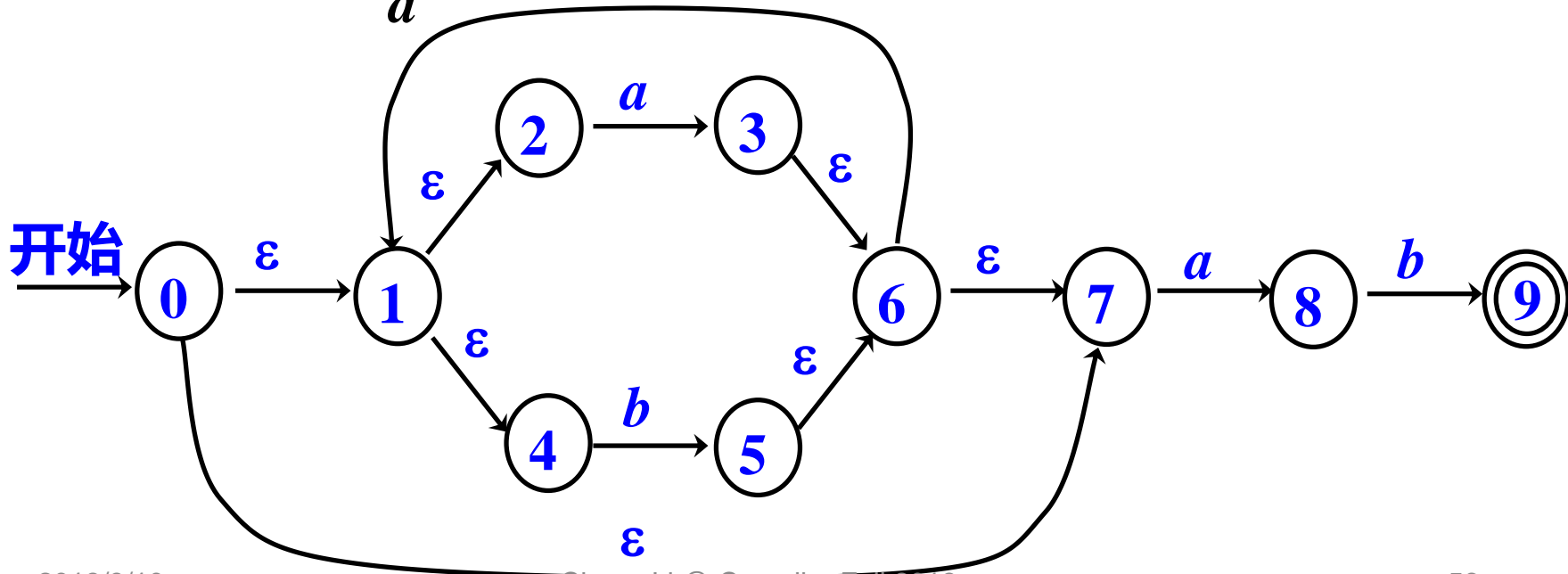
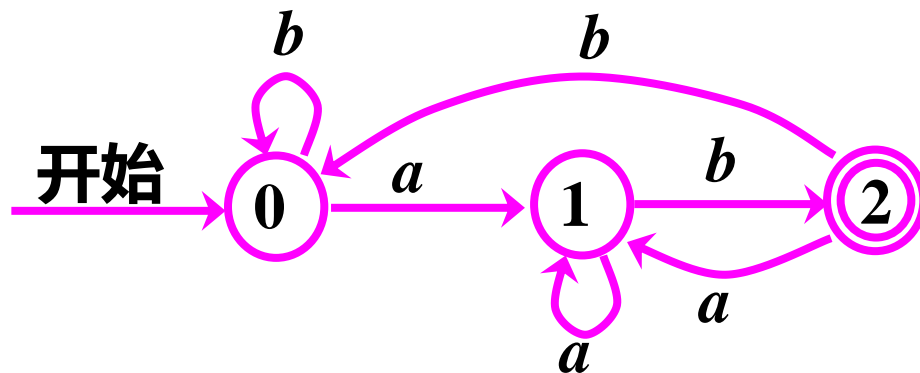


# NFA到DFA的变换

识别语言  
 $(a|b)^*ab$   
 的自动机



子集构造法不一定得到最简DFA





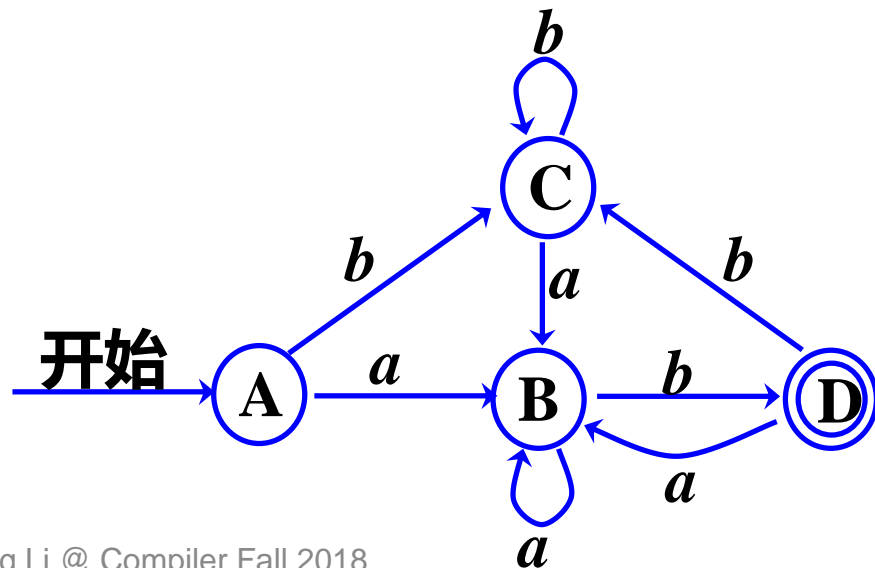
## □ A和B是可区别的状态

- ❖ 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

可区别的状态要  
分开对待



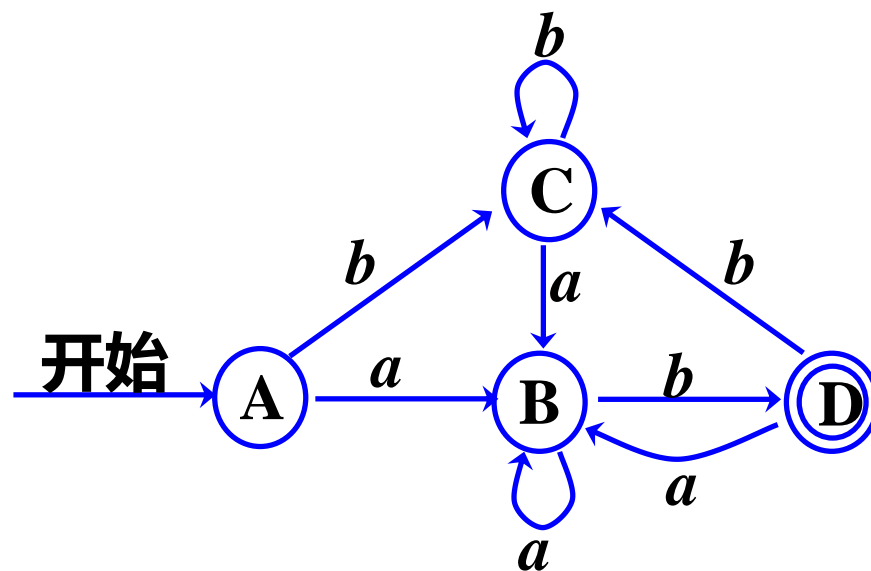


## 1. 按是否是接受状态来区分

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

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

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





## 1. 按是否是接受状态来区分

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

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

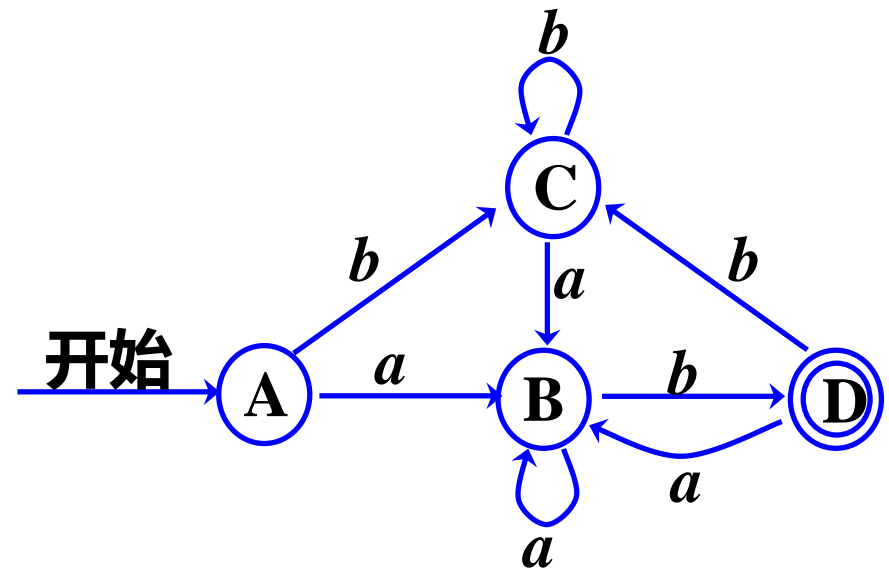
$\text{move}(\{A, B, C\}, b) = \{C, D\}$

## 2. 继续分解

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

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

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



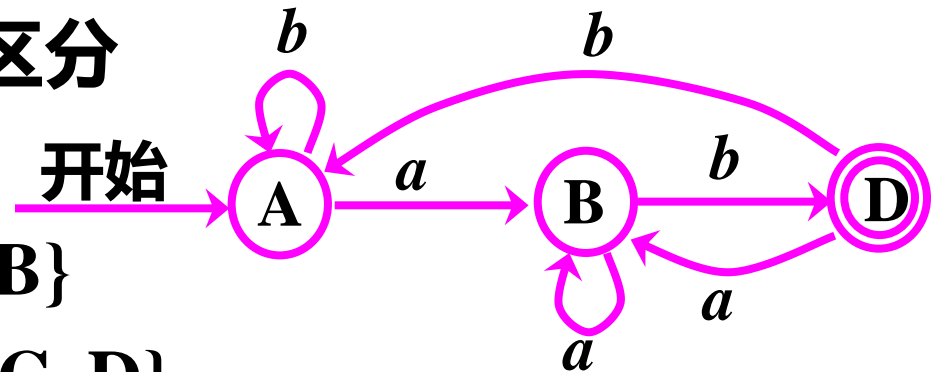


## 1. 按是否是接受状态来区分

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

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

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

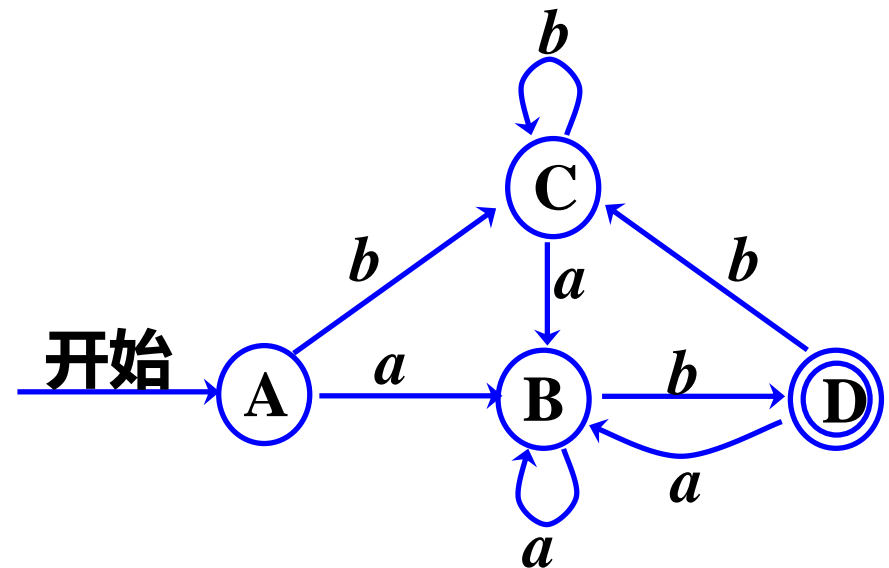


## 2. 继续分解

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

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

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





# 《编译原理与技术》

## 词法分析

**Computer science is no more  
about computers than  
astronomy is about telescopes.**

**——Edsger Dijkstra**