



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



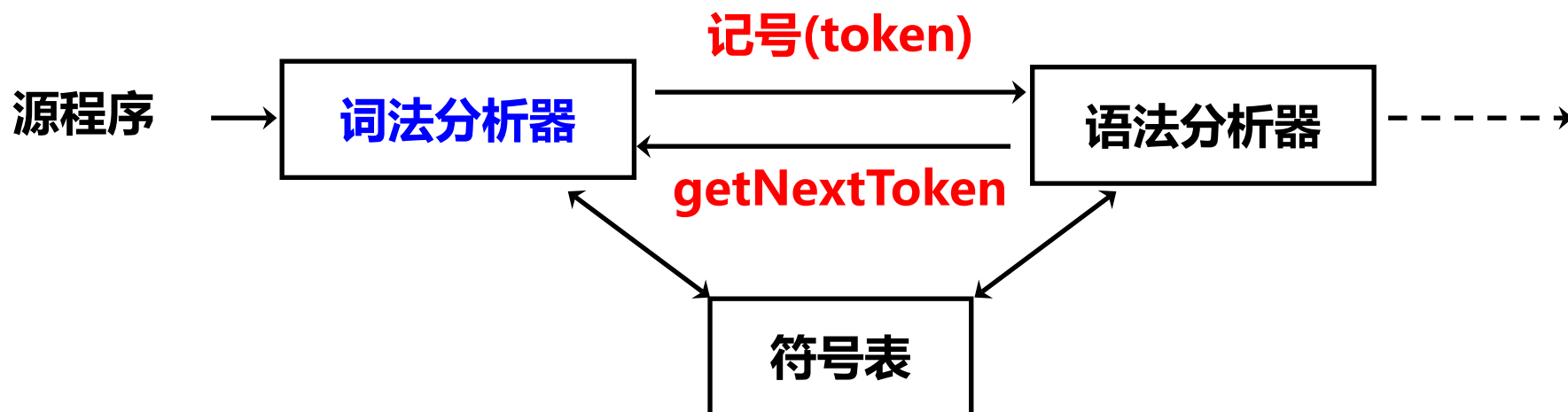
《编译原理与技术》

词法分析

计算机科学与技术学院

李 诚

2021-09-08



□ 词法分析概述及自动化需要解决的问题

□ 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
- ❖ 正则表达式 → NFA → DFA → 化简的DFA



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

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

- ❖ 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子串的模式 (Pattern)



四个关键术语



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

源程序中的
字符序列

词素
(lexeme)

匹配

描述

模式
(pattern)

一个实例

词法单元
(token)

记号

一般种类
记号名

关键字
标识符
常量
运算符
分界符



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

记号名	实例 (词素)	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

- 由一个记号名和一个可选的属性值组成
 - ❖ $\text{token} := \langle \text{token_name}, \text{attribute_value} \rangle$



□ 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
- ❖ 可以为空（optional）

□ 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

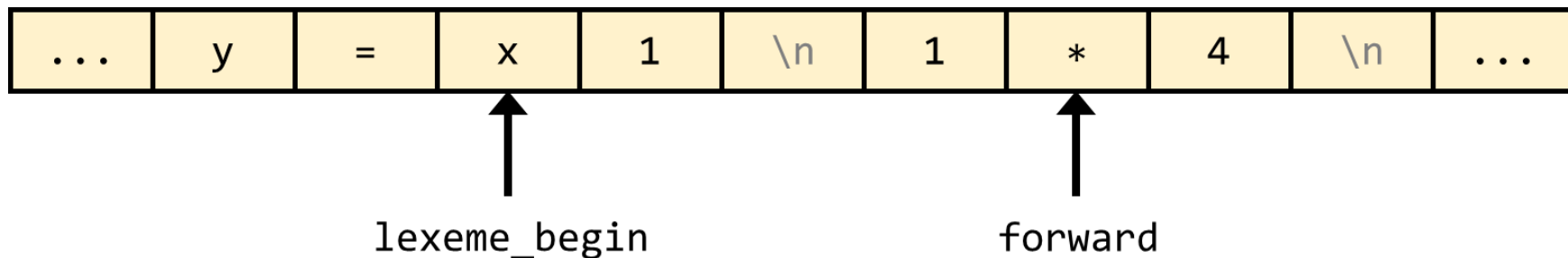
1	position	...
2	initial	...
3	rate	...

词素
(实例)



□ 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





□ 可否用空格分词?

❖ `a>b, if(expression)`

□ 如何应对任意的词素序列?

❖ `int a, int aaa, int aaaa, int aaaaaaa`

□ 可否处理可能出现的歧义?

❖ `i, f vs. if, = vs. ==`



□ 可否用空格分词？

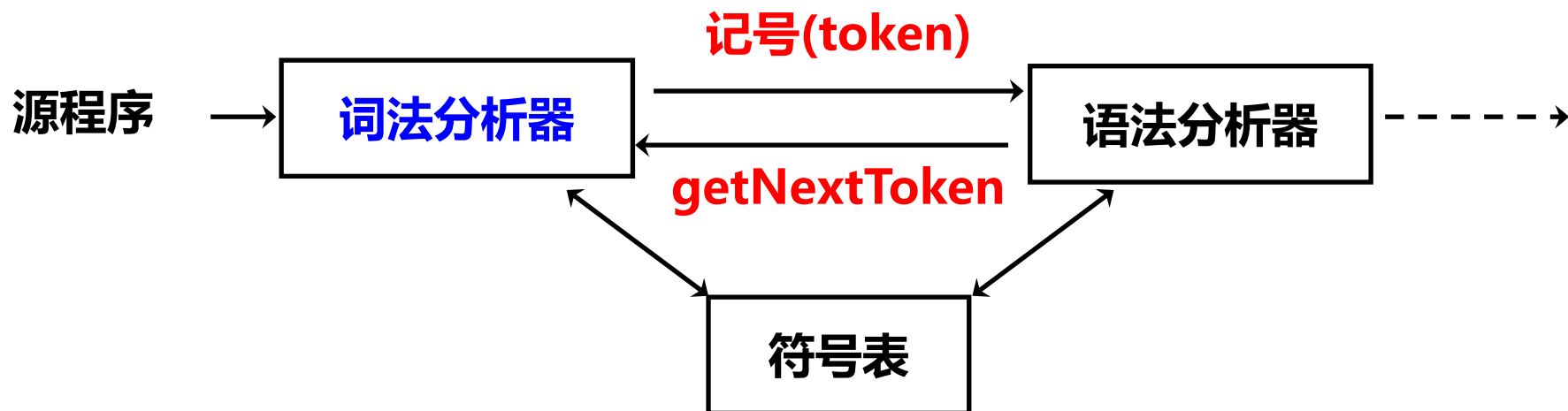
- ❖ `a>b, if(expression)`
- ❖ 不能，需要用扫描匹配或者回溯的方法

□ 如何应对任意的词素序列？

- ❖ `int a, int aaa, int aaaa, int aaaaaaa`
- ❖ 虽然程序可以任意，但是记号的类型数量可控，为每一种类型设计匹配模式
- ❖ 利用模式，进行模式匹配

□ 可否处理可能出现的歧义？

- ❖ `i, f vs. if, = vs. ==`
- ❖ 最长匹配原则 + 关键字保留原则



□ 词法分析概述及自动化需要解决的问题

□ 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
- ❖ 正则表达式 → NFA → DFA → 化简的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

注意区别：

$\varepsilon, \{\varepsilon\}, \emptyset$

□ 串的运算

- **连接（积）**： $xy, s\varepsilon = \varepsilon s = s$
- **指数（幂）**： s^0 为 ε , 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 $\{\epsilon\}$, 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

□ 示例

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



□ $\Sigma = \{a, b\}$

优先级:
闭包* > 连接 > 选择 |

❖ $a \mid b$

$\{a, b\}$

❖ $(a \mid b)(a \mid b)$

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

❖ $aa \mid ab \mid ba \mid bb$

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

❖ 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



C语言的标识符是字母、数字和下划线组成的串

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

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

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



□ 正则式用来表示简单的语言

正则式	定义的语言	备注
ε	$\{\varepsilon\}$	
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 \mid

□ 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，是新符号，not in Σ
- ❖ 每个 r_i 都是 $\Sigma \cup \{d_1, d_2, \dots, d_{i-1}\}$ 上的正则式



无符号数集合，例1946,11.28,63E8,1.99E-6



无符号数集合，例1946,11.28,63E8,1.99E-6

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

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

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

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

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



无符号数集合，例1946,11.28,63E8,1.99E-6

digit $\rightarrow 0 \mid 1 \mid \dots \mid 9$ [0-9]

digits \rightarrow digit digit*

optional_fraction \rightarrow . digits $\mid \epsilon$

optional_exponent \rightarrow (E (+ \mid - \mid ϵ) digits) $\mid \epsilon$

number \rightarrow digits optional_fraction optional_exponent

– 简化表示

number \rightarrow digit⁺ (.digit⁺)? (E[+-]? digit⁺)?

注意区分：
? 和 *



while \rightarrow while

do \rightarrow do

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

letter_ \rightarrow [A-Za-z_]

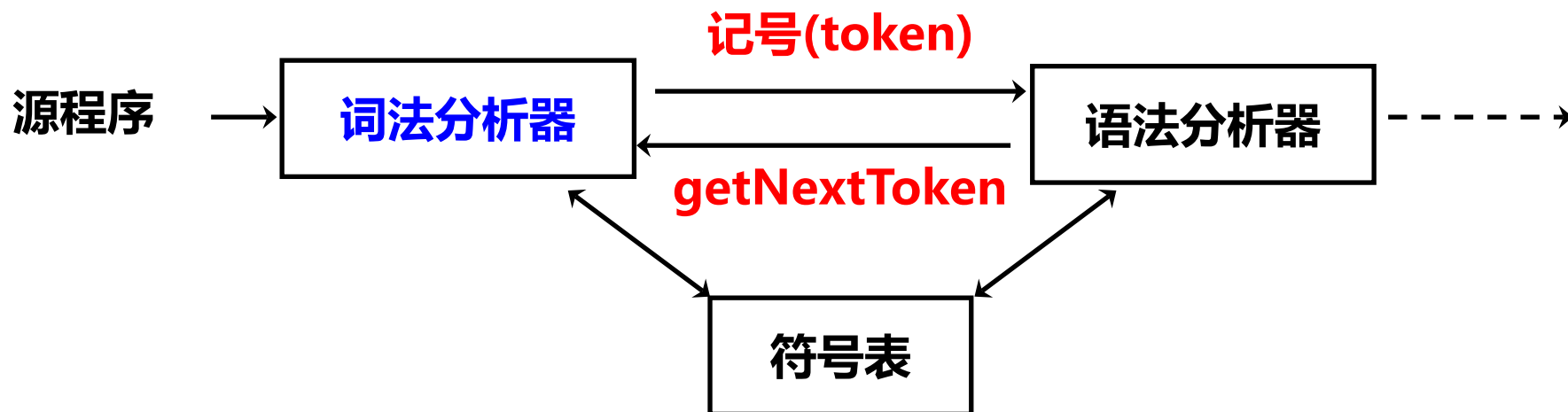
id \rightarrow letter_ (letter_ | digit)*

number \rightarrow digit⁺ (.digit⁺)? (E[+-]? digit⁺)?

delim \rightarrow blank | tab | newline

ws \rightarrow delim⁺

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？



□ 词法分析概述及自动化需要解决的问题

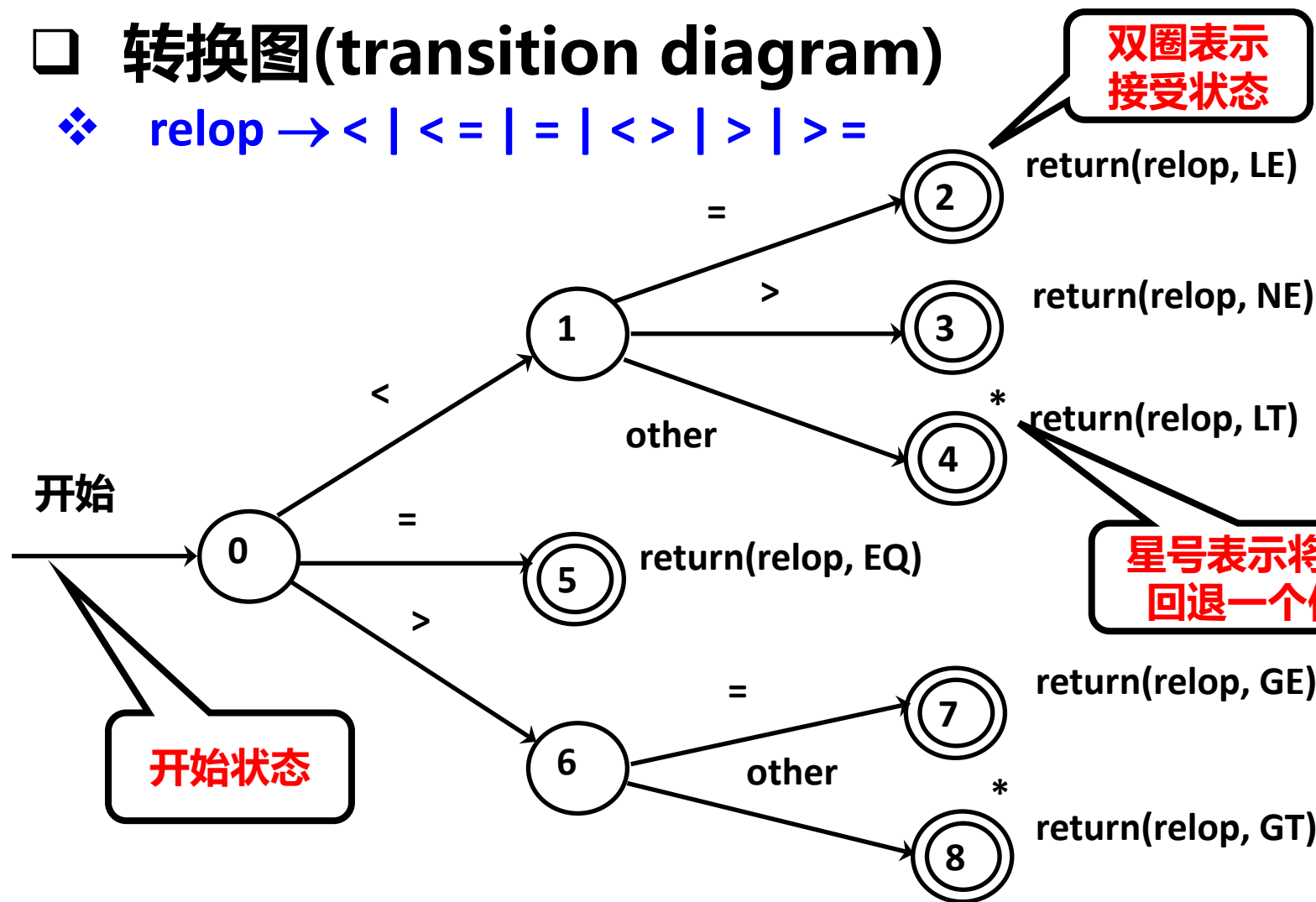
□ 词法分析器的自动生成

- ❖ 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
- ❖ 正则表达式 → NFA → DFA → 化简的DFA



❑ 转换图(transition diagram)

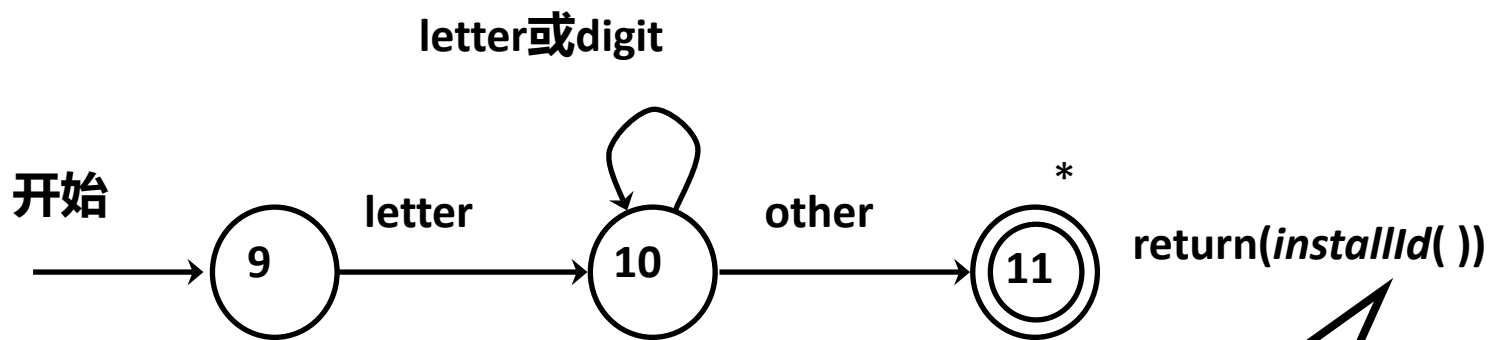
❖ $\text{relop} \rightarrow < | <= | = | <> | > | >=$





❑ 标识符和关键字的转换图

❖ $id \rightarrow letter (letter \mid digit)^*$

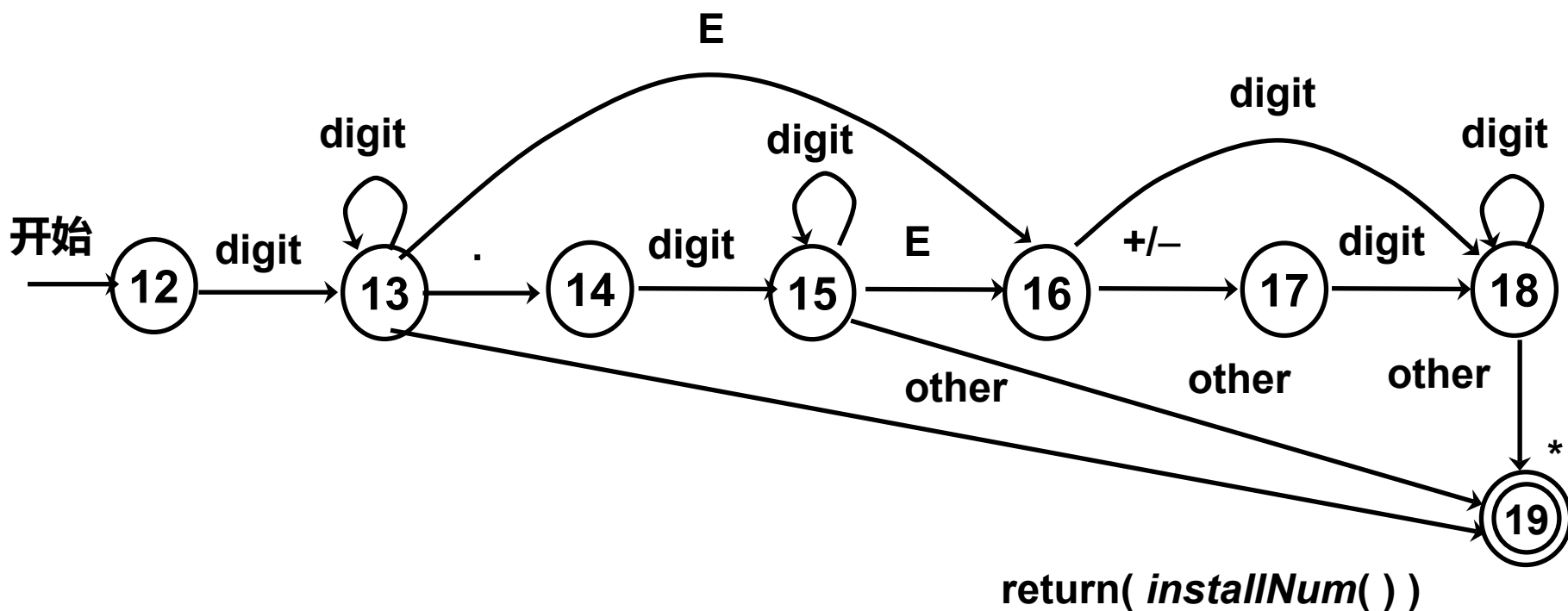


installId将该标识符放入符号表内，并返回符号表指针。如果是关键字则不需要！



□ 无符号数的转换图

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

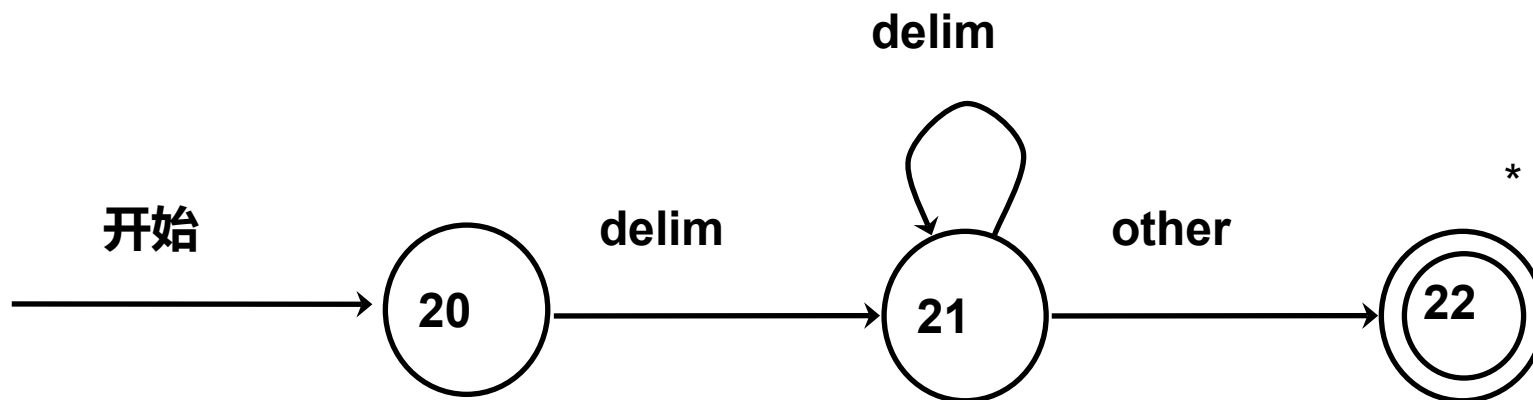




□ 空白的转换图

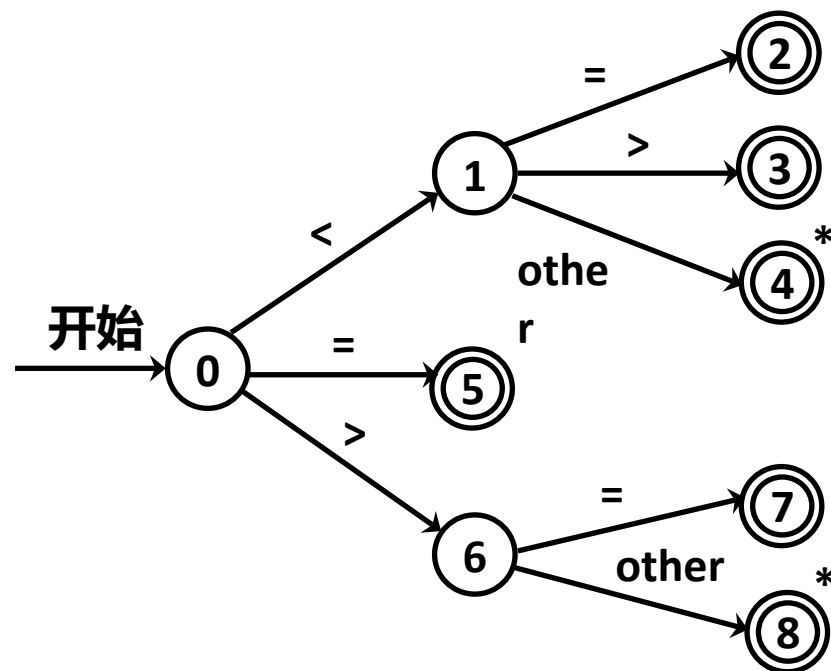
$\text{delim} \rightarrow \text{blank} \mid \text{tab} \mid \text{newline}$

$\text{ws} \rightarrow \text{delim}^+$





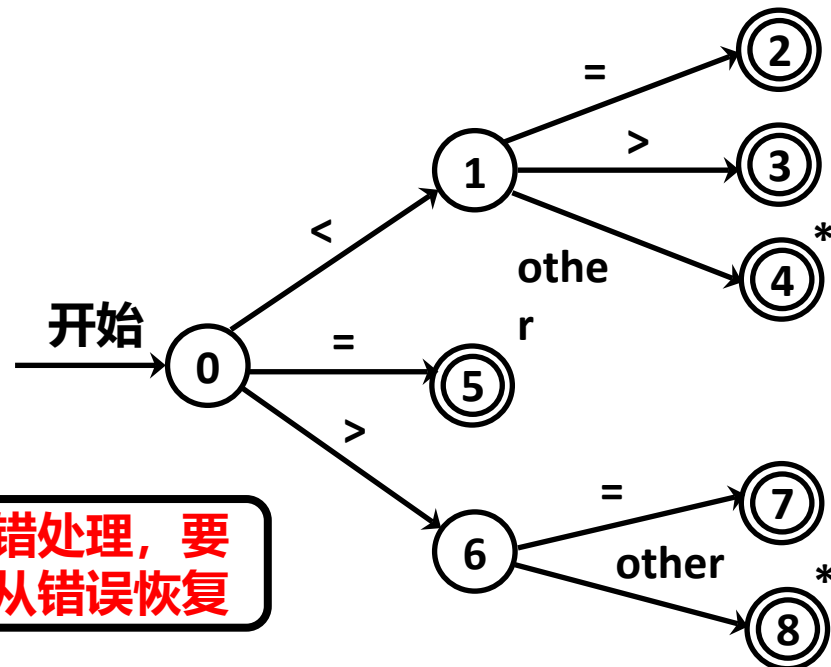
□ 例：relop的转换图的概要实现





□ 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



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

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 $x_1 \dots x_i \in L(R_k)$

优先 match 规则:

❖ 选择先列出的模式 (**j** 如果 $j < k$)

❖ 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

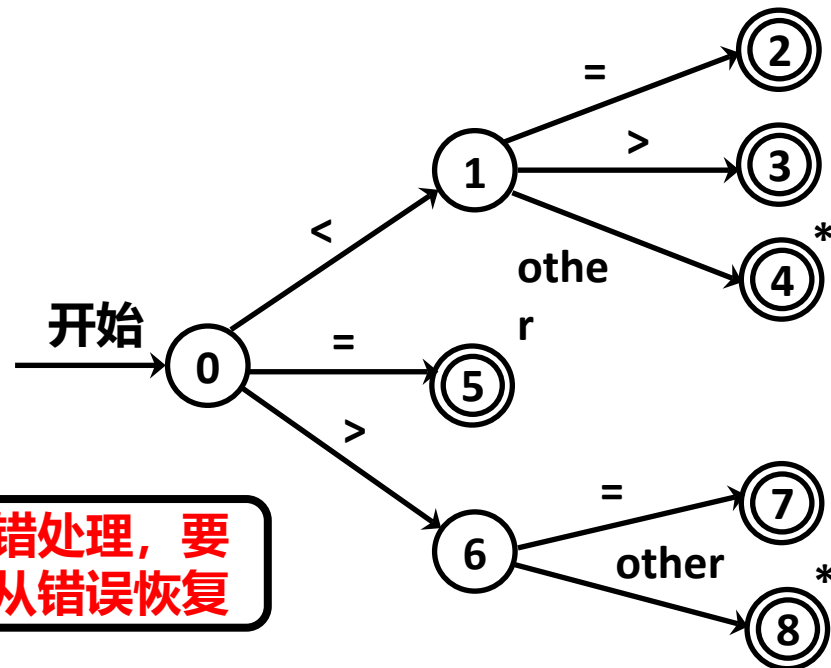
❖ 进行增、删、替换和交换字符的尝试

❖ 变换代价太高，不值得



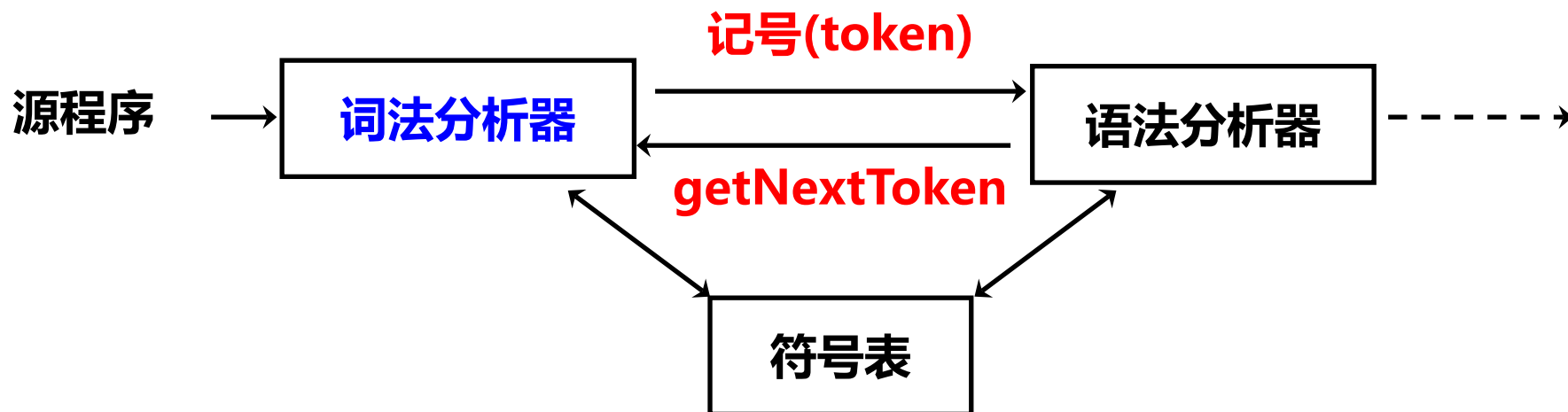
□ 例：relop的转换图的概要实现

```
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？**



□ 词法分析概述及自动化需要解决的问题

□ 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
- ❖ 正则表达式 → NFA → DFA → 化简的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 Σ
- ❖ 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

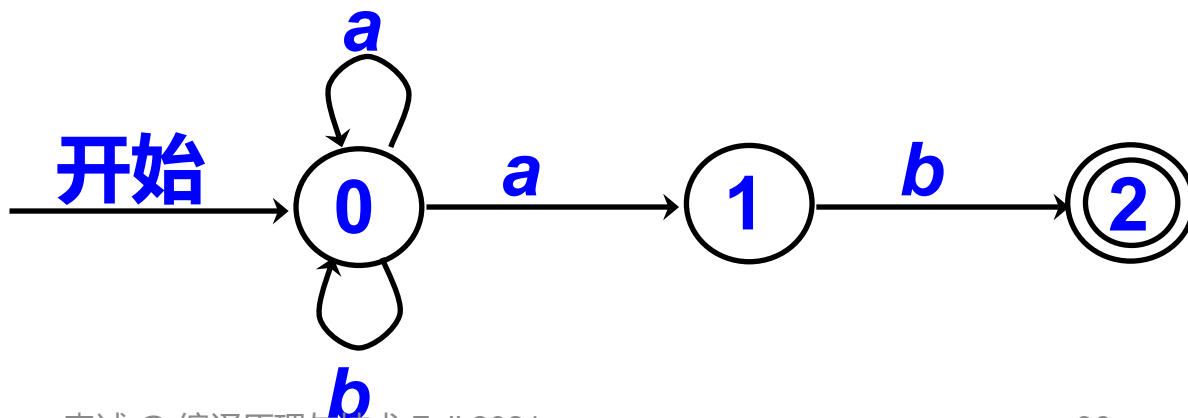
幂集



□ 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 Σ
- ❖ 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

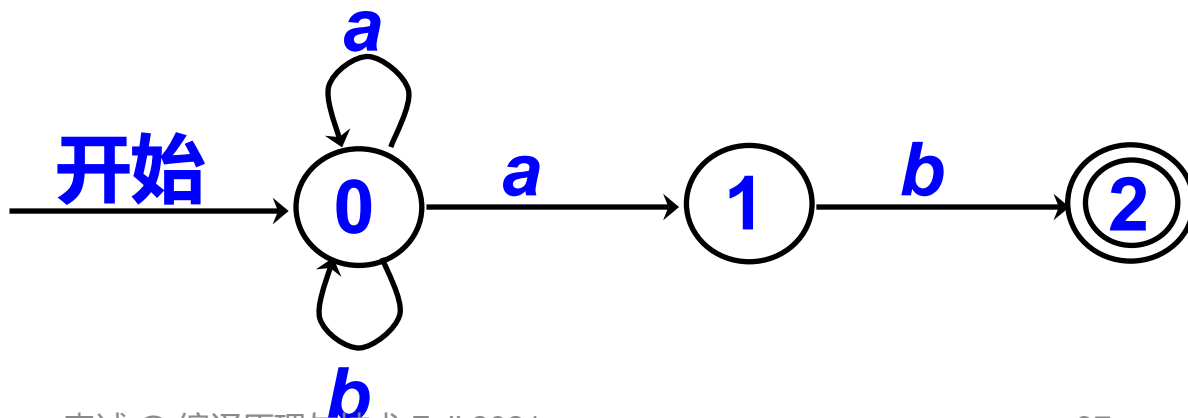




□ NFA的转换表

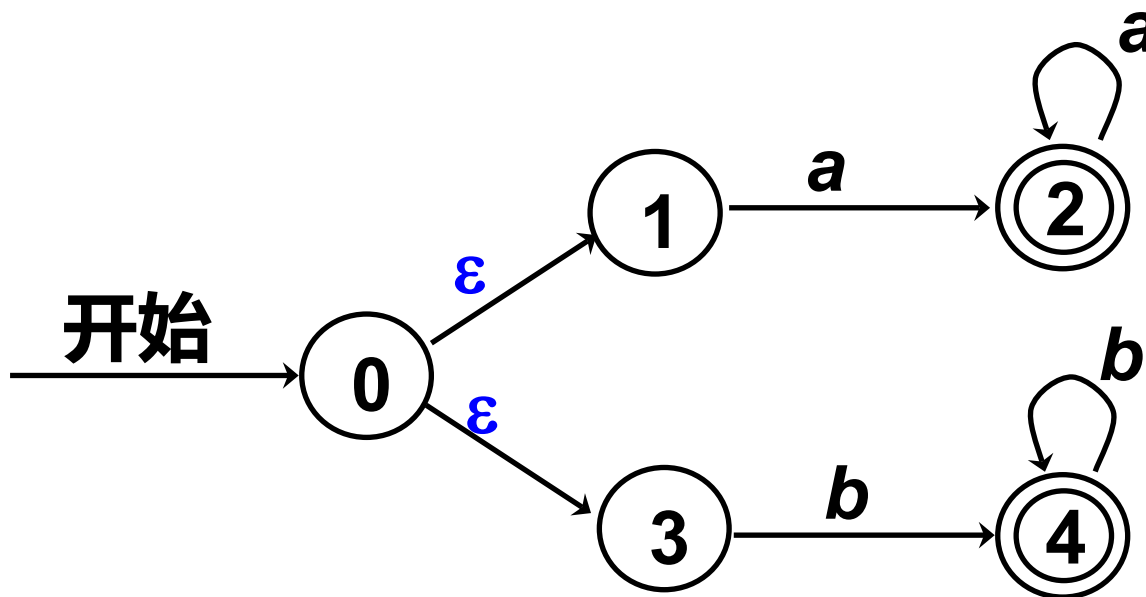
状 态	输 入 符 号	
	a	b
0	$\{0, 1\}$	$\{0\}$
1	\emptyset	$\{2\}$
2	\emptyset	\emptyset

识别语言
 $(a|b)^*ab$
的NFA



□ 例 识别 $aa^*|bb^*$ 的NFA

□ 例 识别 $aa^*|bb^*$ 的NFA





□ 转换函数 $\text{move} : S \times (\Sigma \cup \{\varepsilon\}) \rightarrow P(S)$



□ 转换函数move : $S \times (\Sigma \cup \{\epsilon\}) \rightarrow P(S)$

□ 对于一个token,

- ❖ 有可能要尝试很多不同的路径,
- ❖ 大部分路径都是白费功夫
- ❖ 尝试+回退的方式 \Rightarrow 效率很低
- ❖ 考虑很多project, 百万行代码+

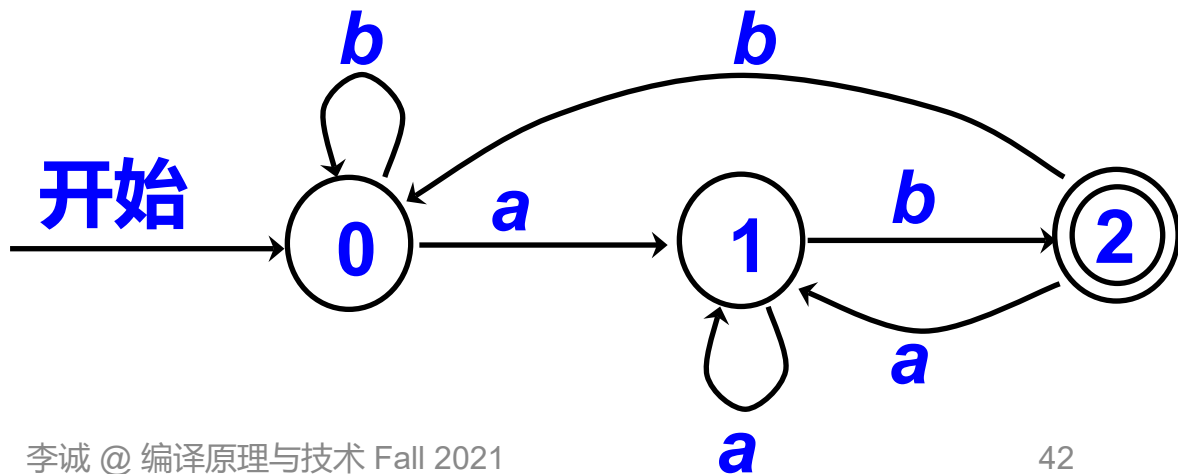
□ 思考: 有没有一种确定的形式化描述,
对于输入的一个符号, 只有唯一的跳转?



□ 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 Σ
- ❖ 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

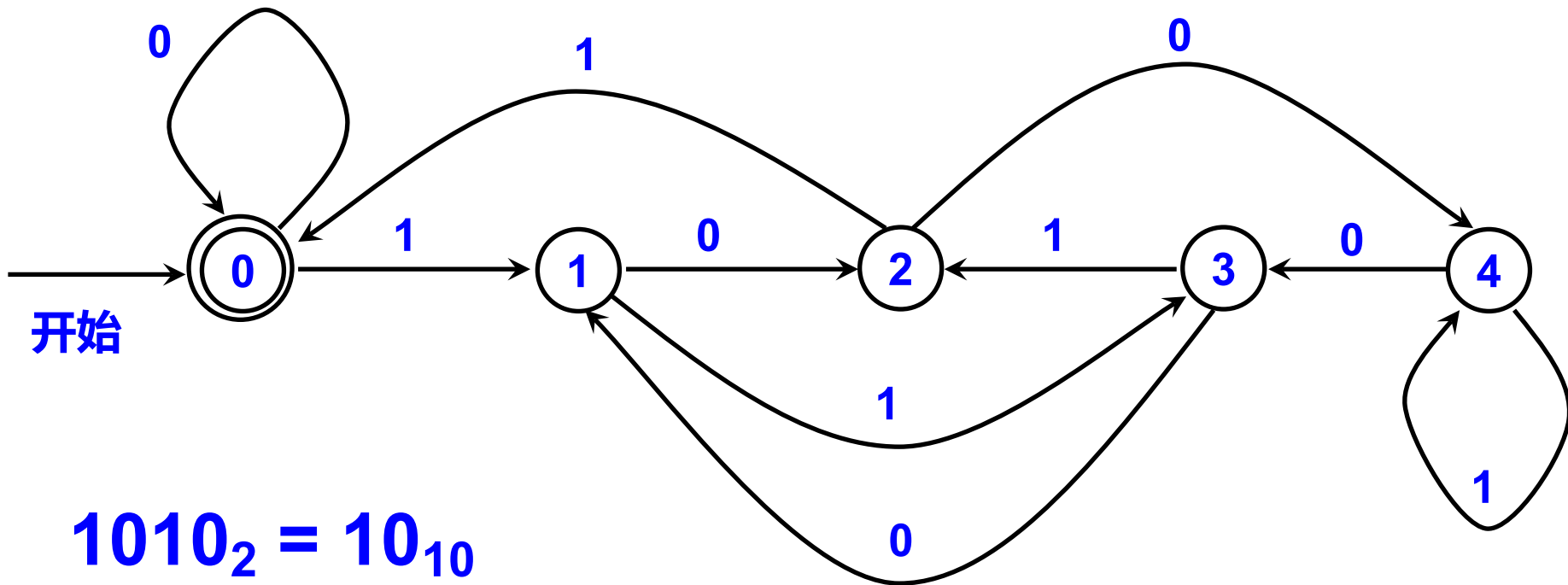
识别语言
 $(a|b)^*ab$
的DFA



□ 例 DFA, 识别 $\{0,1\}$ 上能被5整除的二进制数



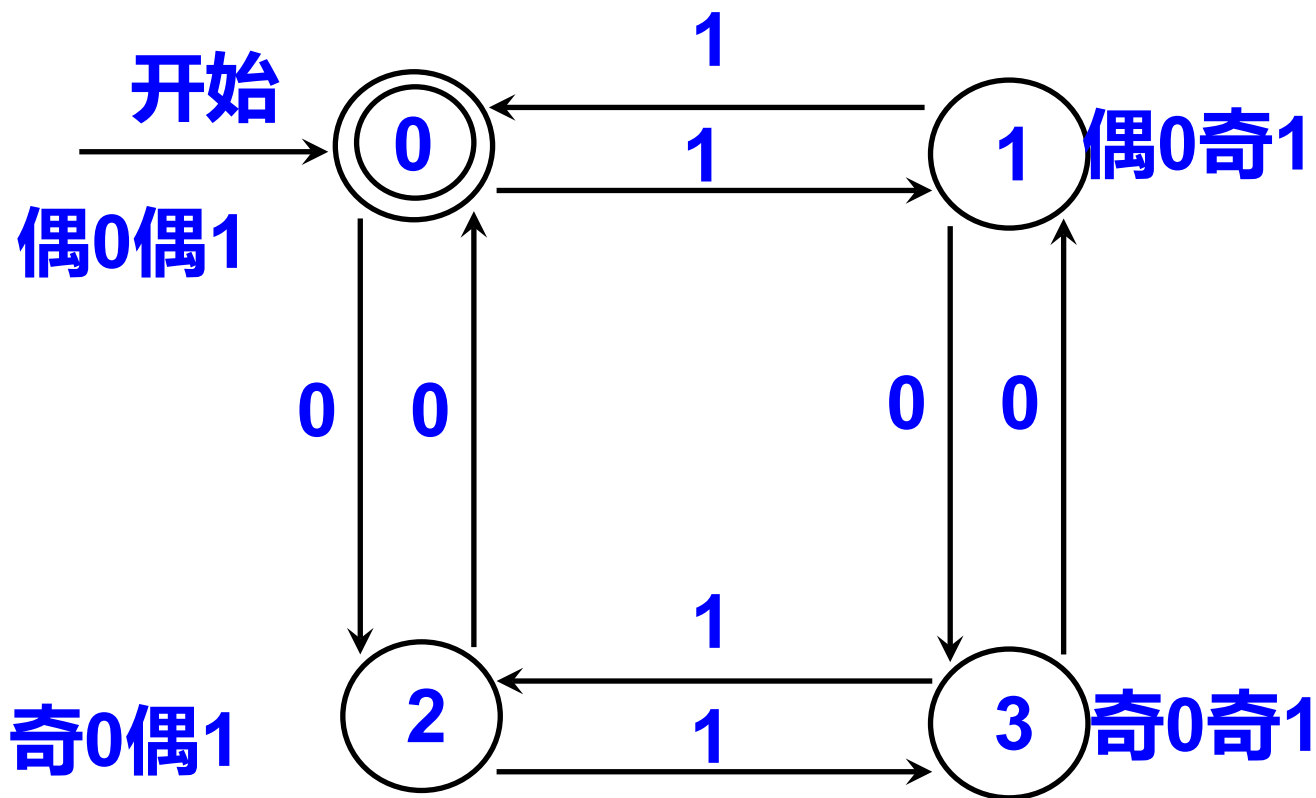
□ 例 DFA, 识别{0,1}上能被5整除的二进制数



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

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

□ 例DFA,接受 0和1的个数都是偶数的字符串

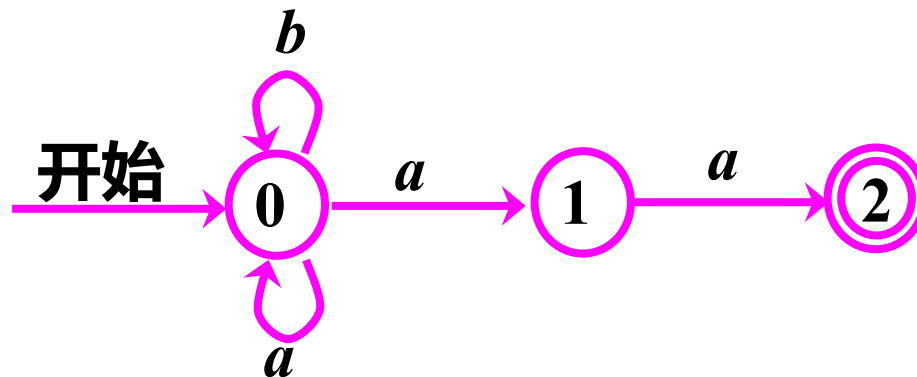




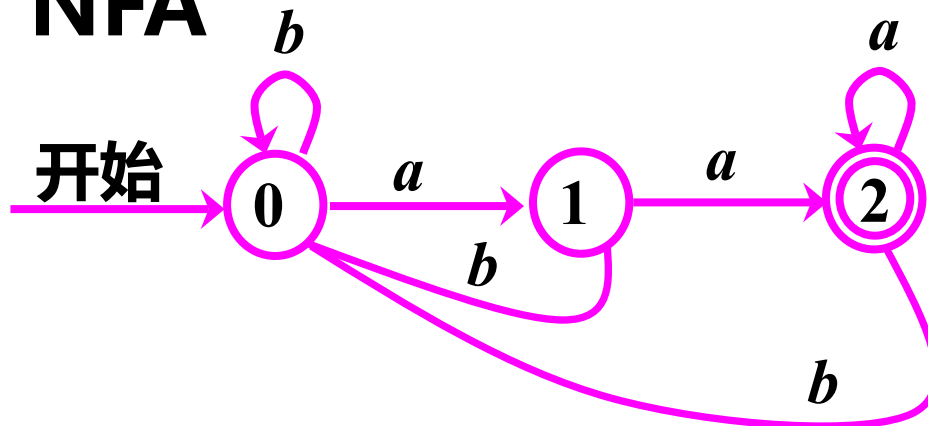
- ❑ NFAs and DFAs recognize the same set of languages (regular languages)
- ❑ Major differences:
 - ❖ Move function
 - ❖ $S \times (\Sigma \cup \{\epsilon\}) \rightarrow P(S)$ NFA
 - ❖ $S \times \Sigma \rightarrow S$ DFA
 - ❖ DFA does not accept ϵ as input
- ❑ DFAs are faster to execute
 - ❖ There are no choices to consider

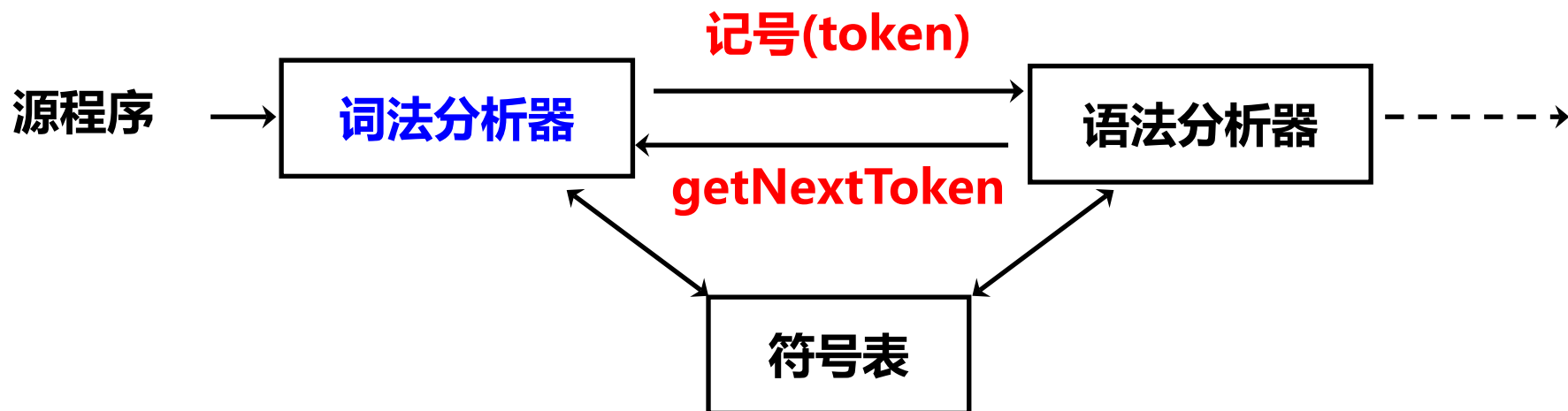


- For a given language NFA can be simpler than DFA



- DFA can be exponentially larger than NFA





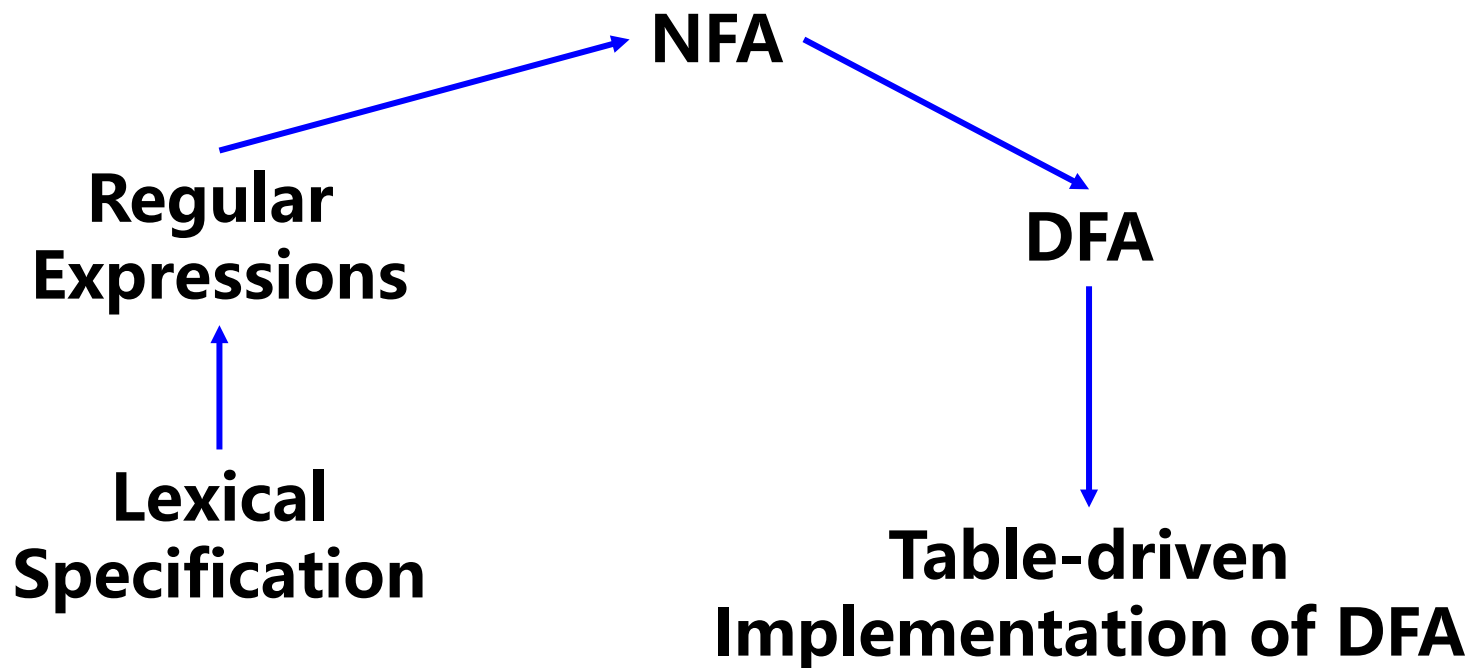
□ 词法分析概述及自动化需要解决的问题

□ 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
- ❖ 正则表达式 → NFA → DFA → 化简的DFA



- ❑ 正则表达式 = Specification
- ❑ 有限自动机 = Implementation





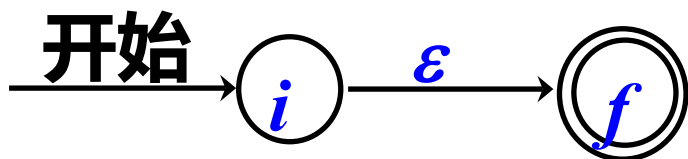
- ❑ 正则表达式 = Specification
- ❑ 有限自动机 = Implementation

- ❑ 二者之间的转换:
 - ❖ 用语法制导的算法，它用正则表达式的语法结构来指导有限自动机的构造过程

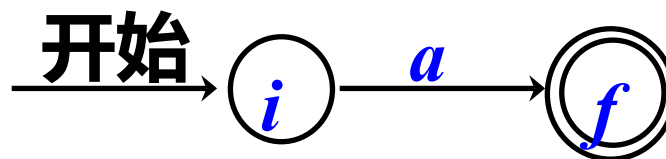


□ 首先构造识别 ε 和字母表中一个符号 a 的NFA

❖ 重要特点：仅一个接受状态，它没有向外的转换



识别正则表达式 ε 的
NFA



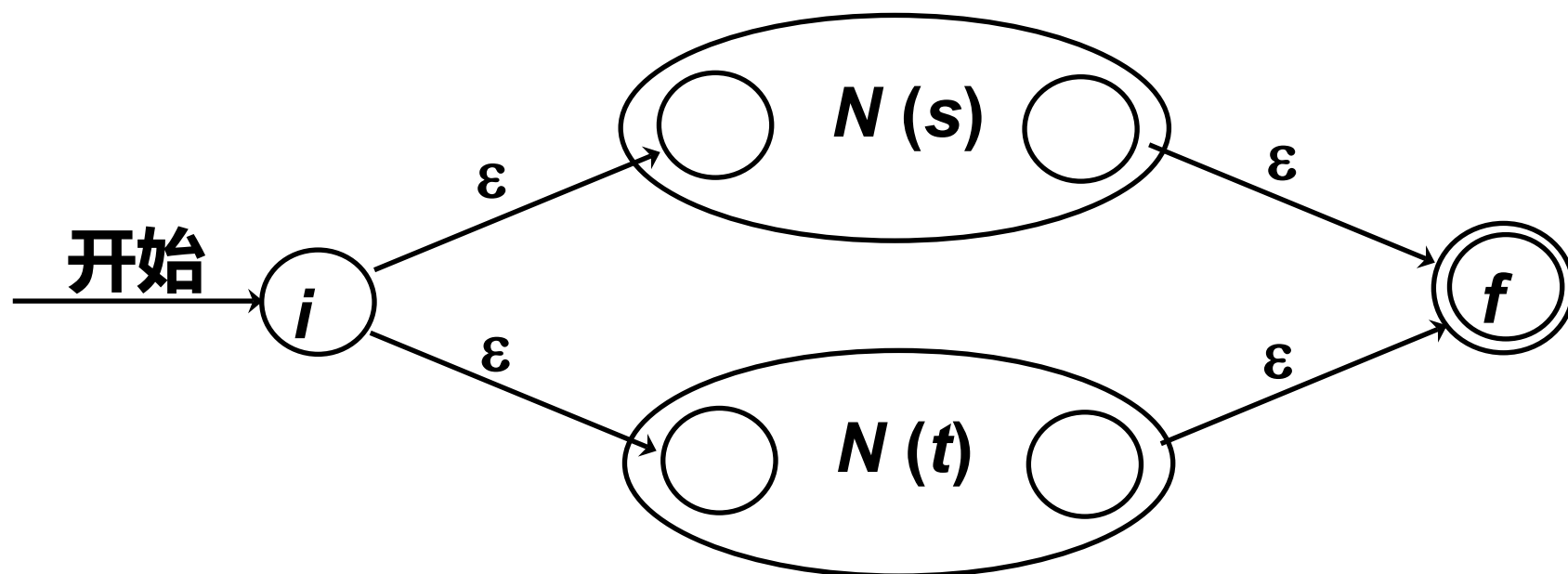
识别正则表达式 a 的
NFA

□ 对于加括号的正则表达式(s)，其NFA可用 s 的NFA（用 $N(s)$ 表示）代替



□ 构造识别主算符为选择的正则表达式的NFA

❖ 重要特点：仅一个接受状态，它没有向外的转换

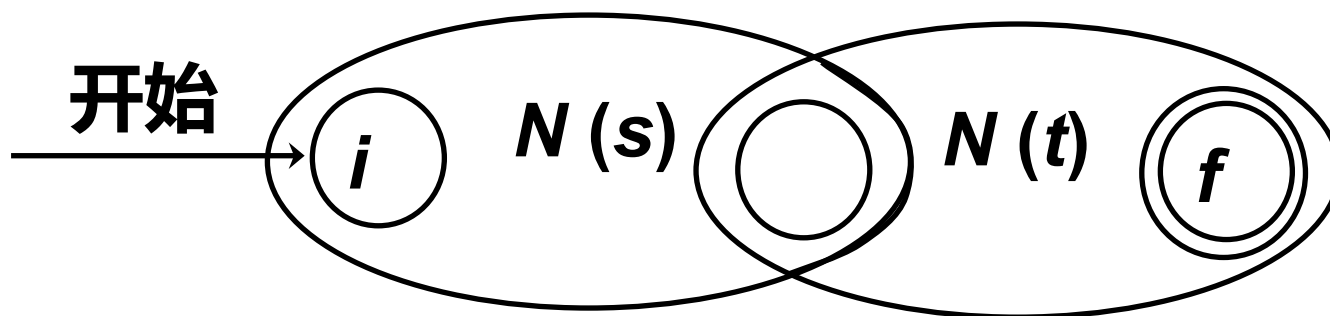


识别正则表达式 $s \mid t$ 的NFA



□ 构造识别主算符为连接的正则表达式的NFA

❖ 重要特点：仅一个接受状态，它没有向外的转换

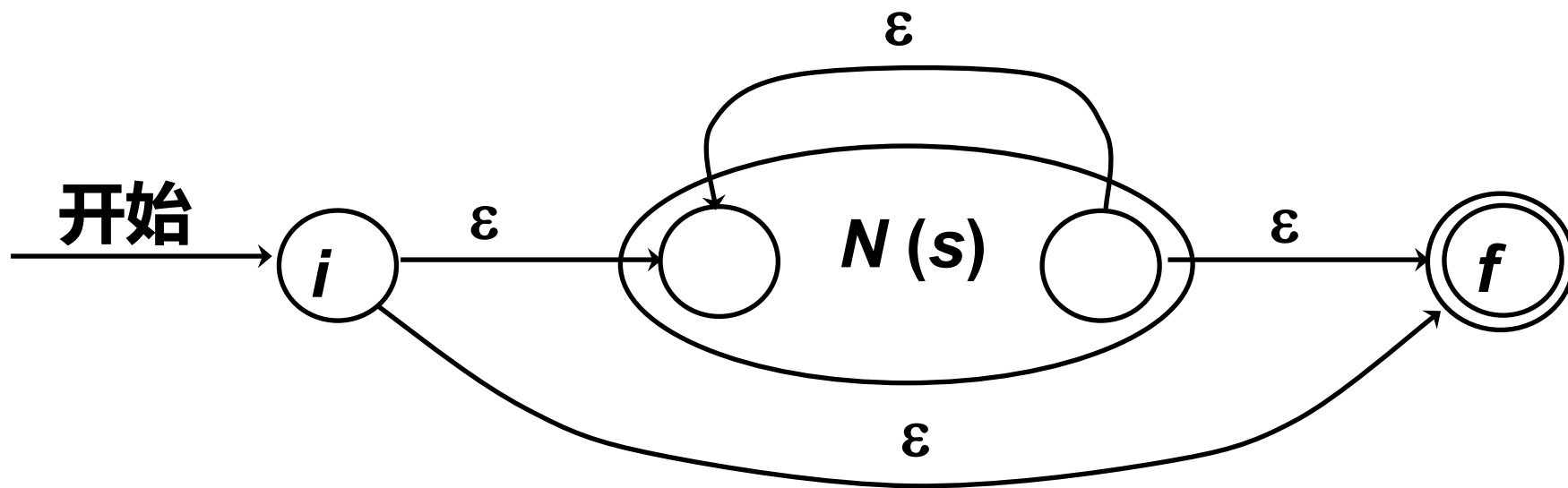


识别正则表达式 st 的NFA



□ 构造识别主算符为闭包的正则表达式的NFA

❖ 重要特点：仅一个接受状态，它没有向外的转换

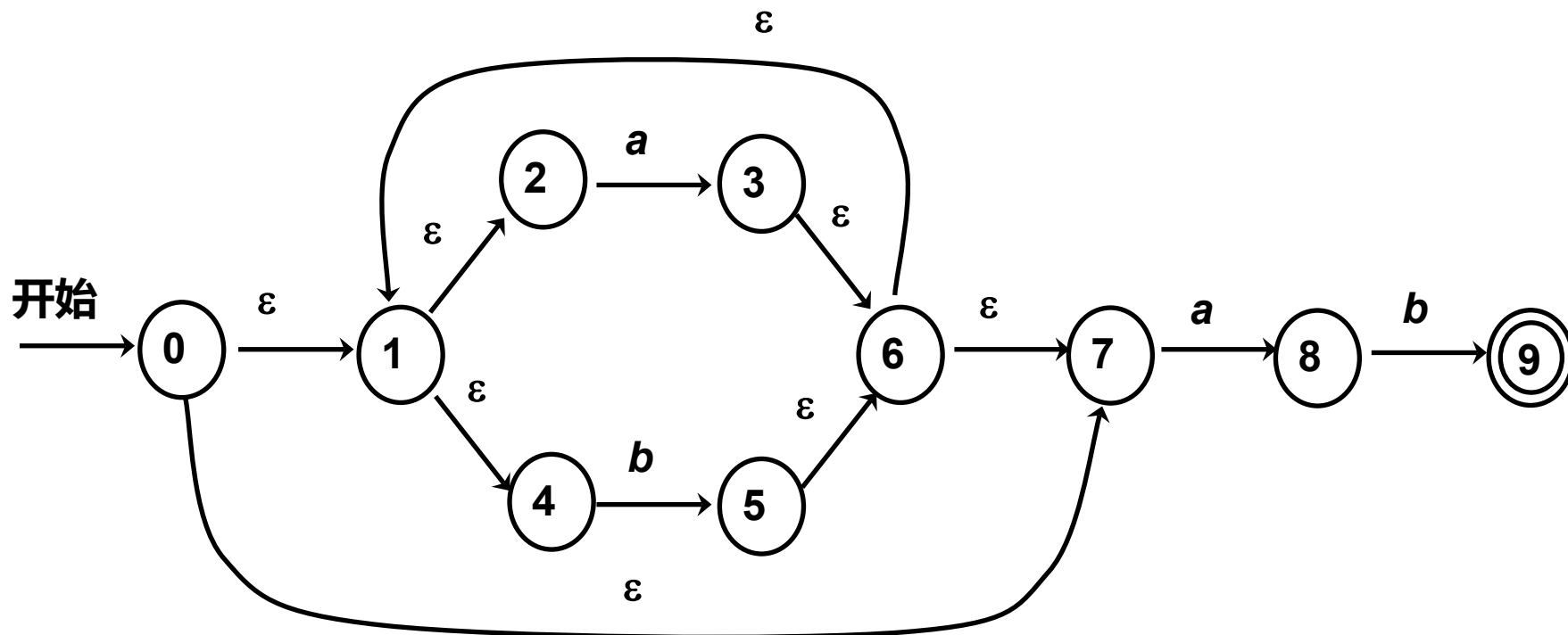


识别正则表达式 s^* 的NFA



□ 由本方法产生的NFA具有下列性质：

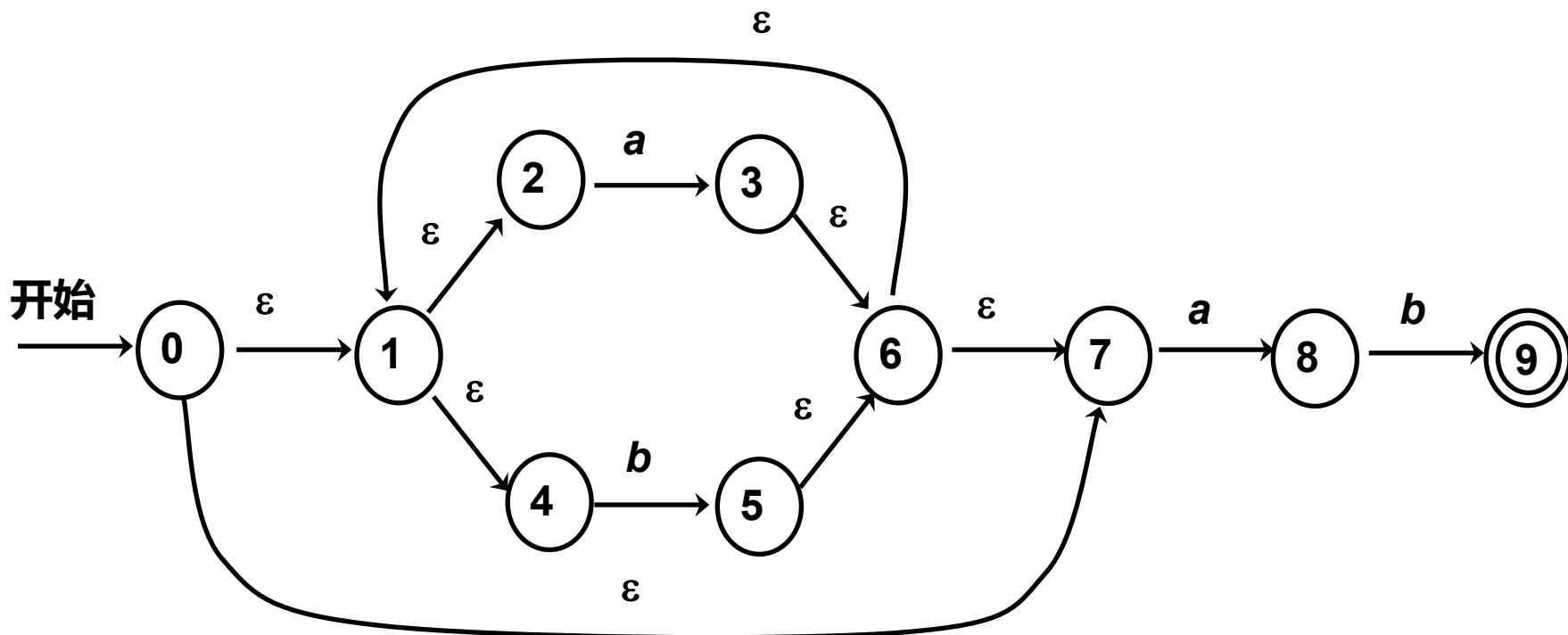
- ❖ $N(r)$ 的状态数最多是 r 中符号和算符总数的两倍
- ❖ $N(r)$ 只有一个接受状态，接受状态没有向外的转换





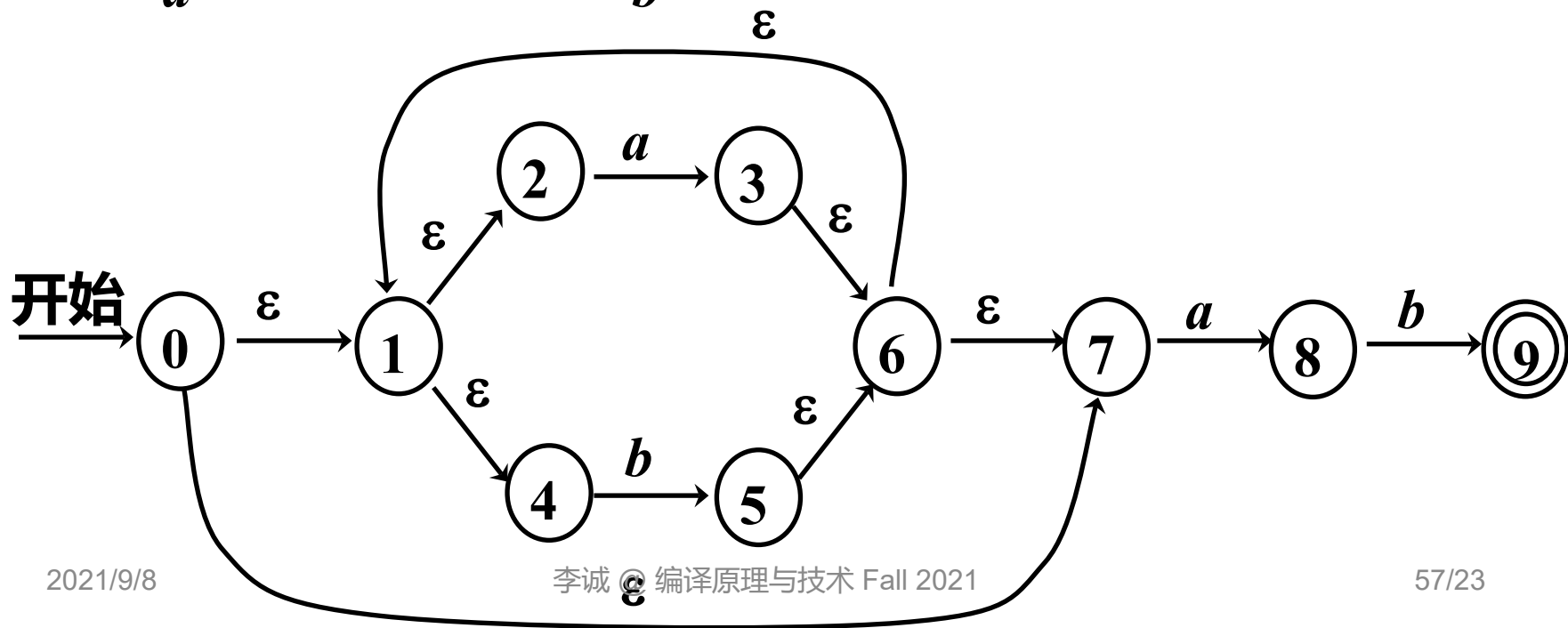
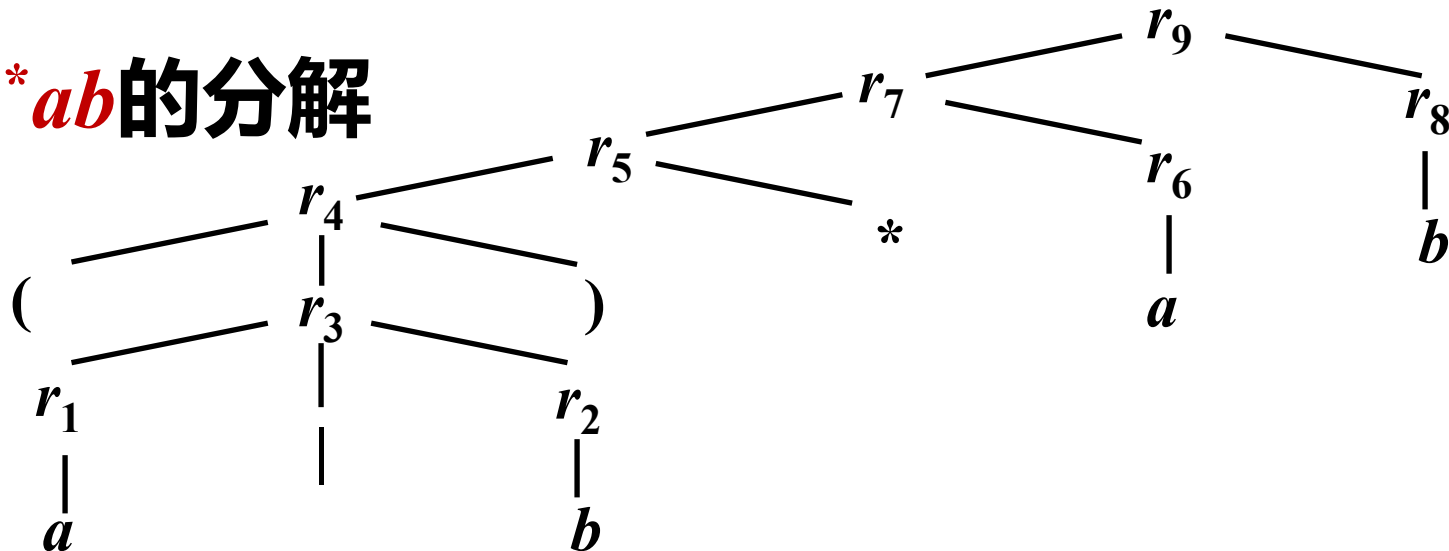
□ 由本方法产生的NFA具有下列性质:

- ❖ $N(r)$ 的每个状态有 (1) 一个其标号为 Σ 中符号的指向其它状态的转换, 或者 (2) 最多两个指向其它状态的 ϵ 转换



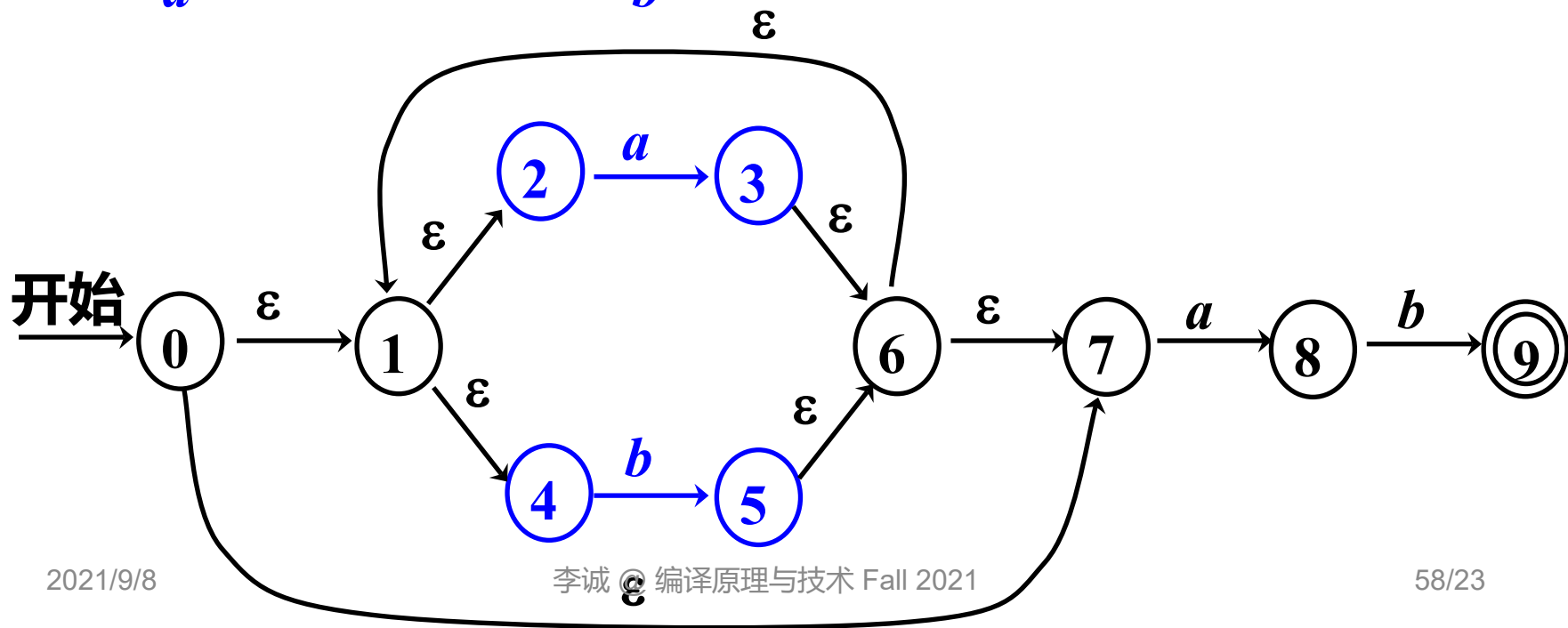
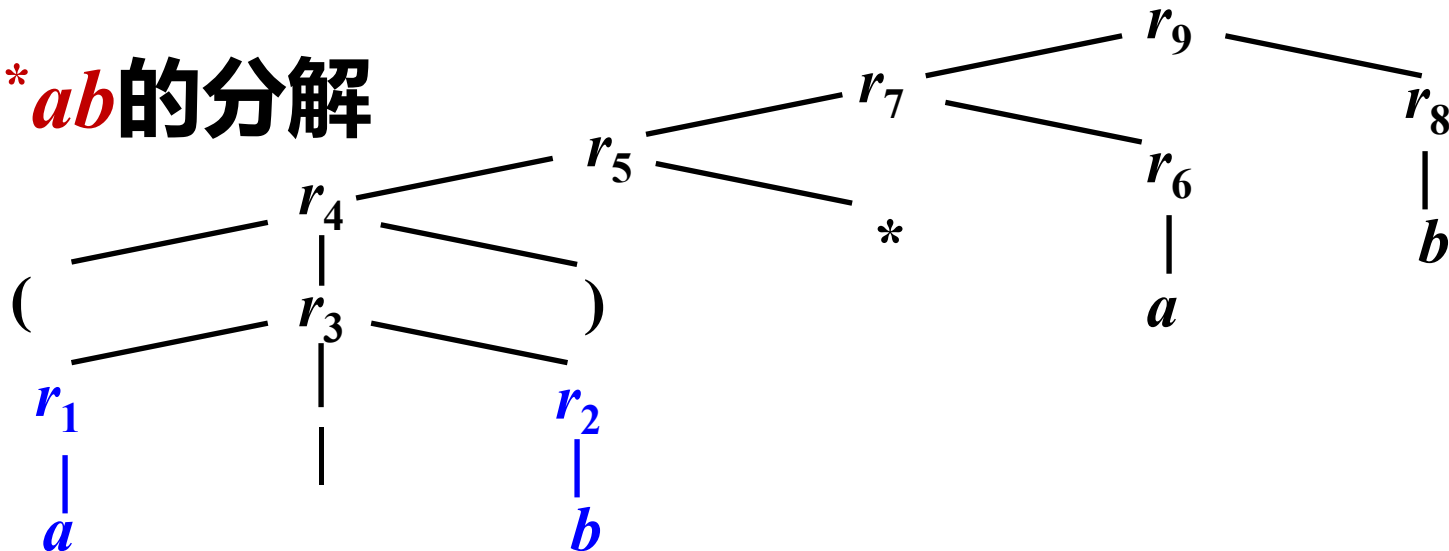


$(a|b)^*ab$ 的分解



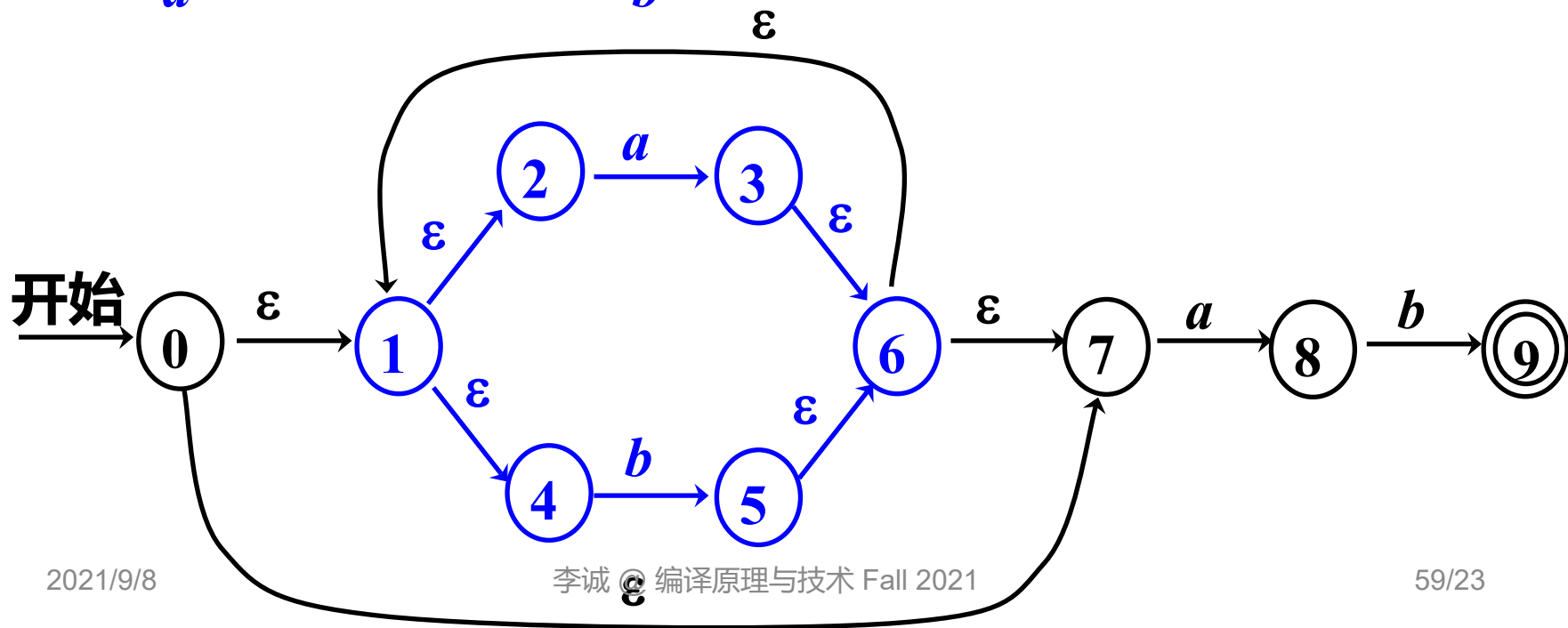
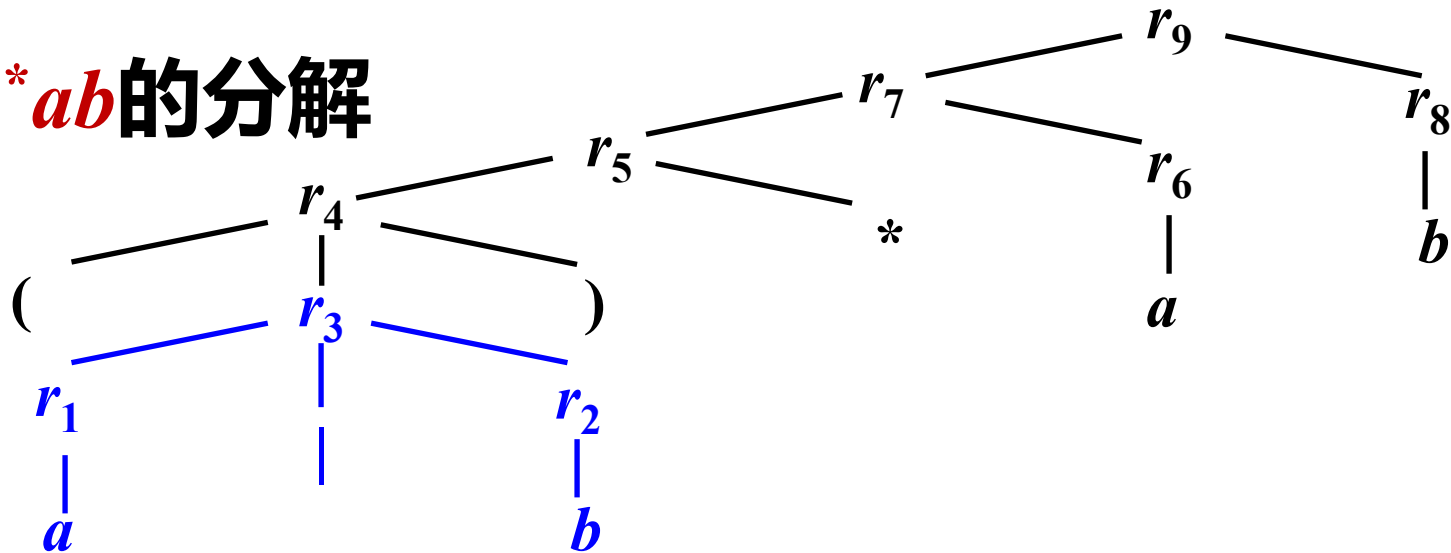


$(a|b)^*ab$ 的分解



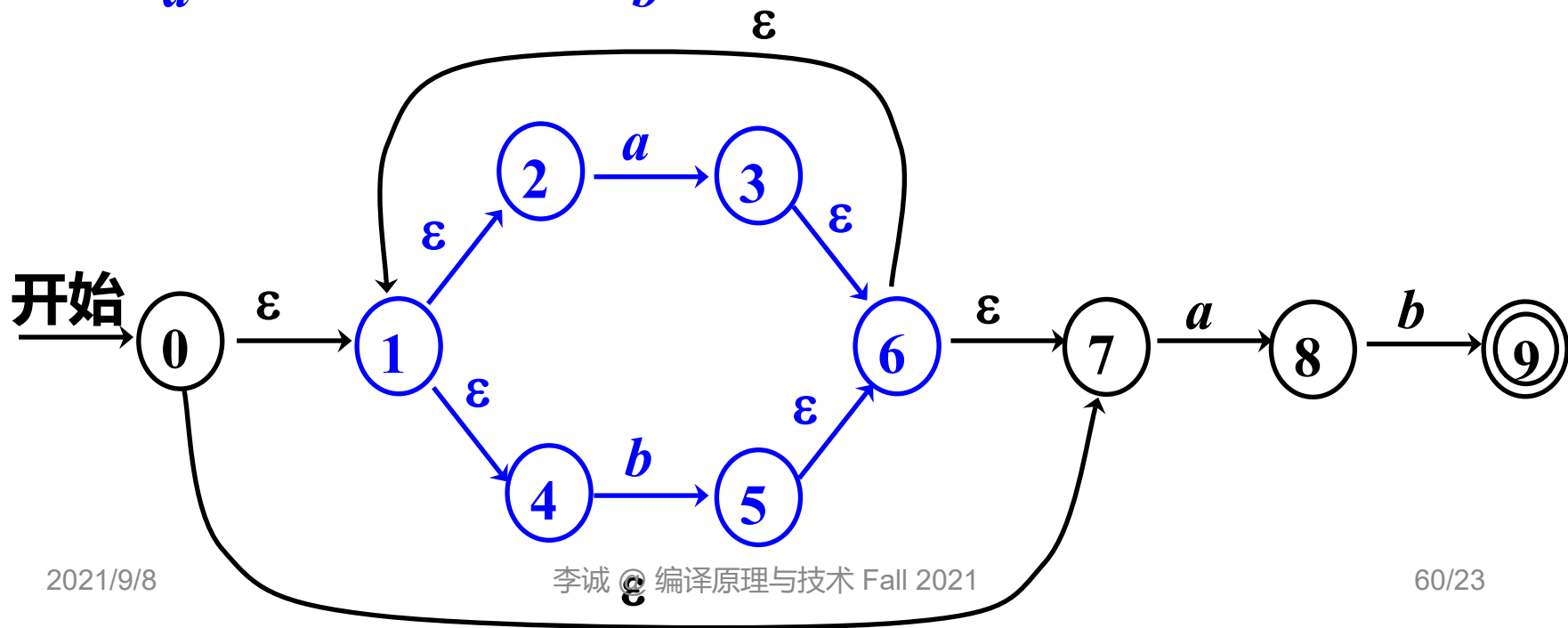
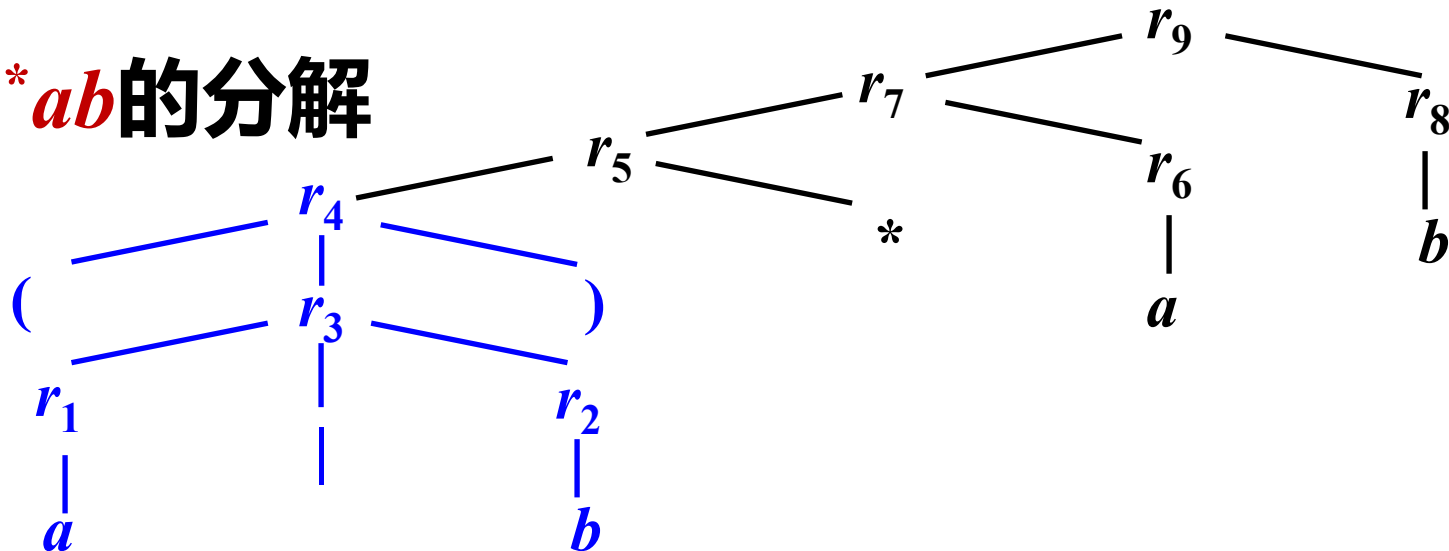


$(a|b)^*ab$ 的分解



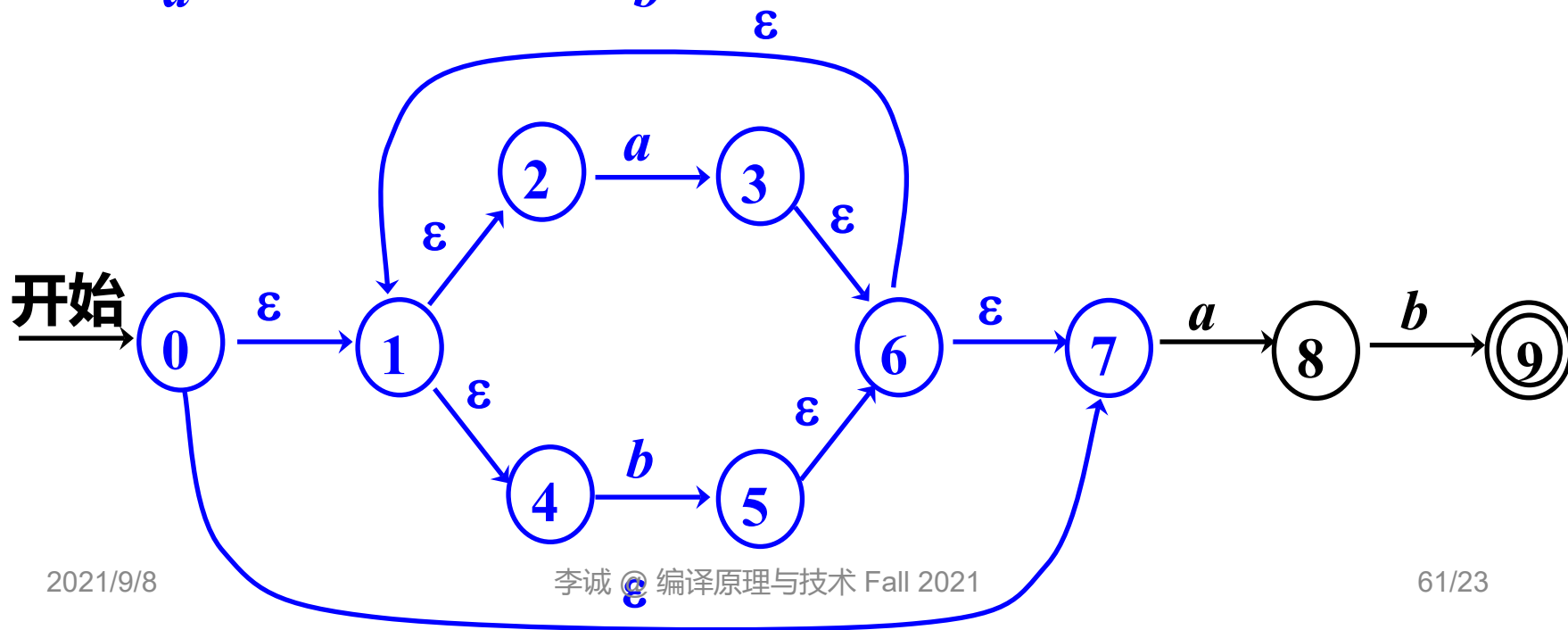
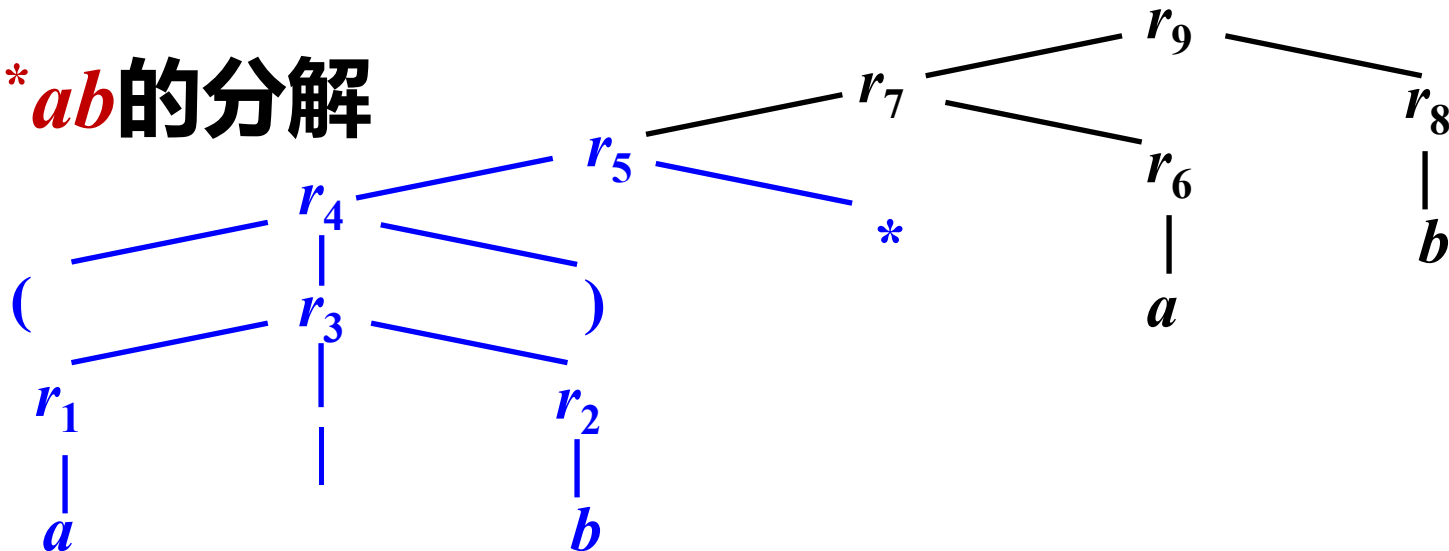


$(a|b)^*ab$ 的分解



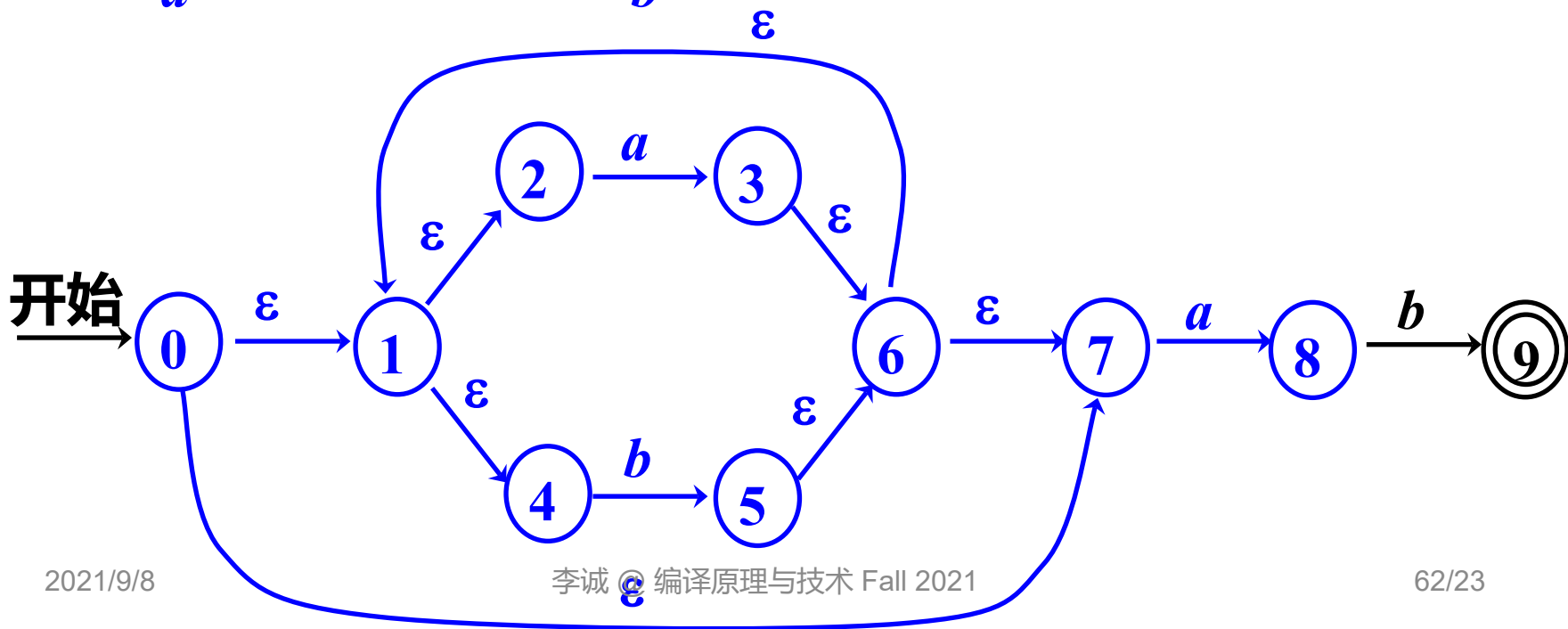
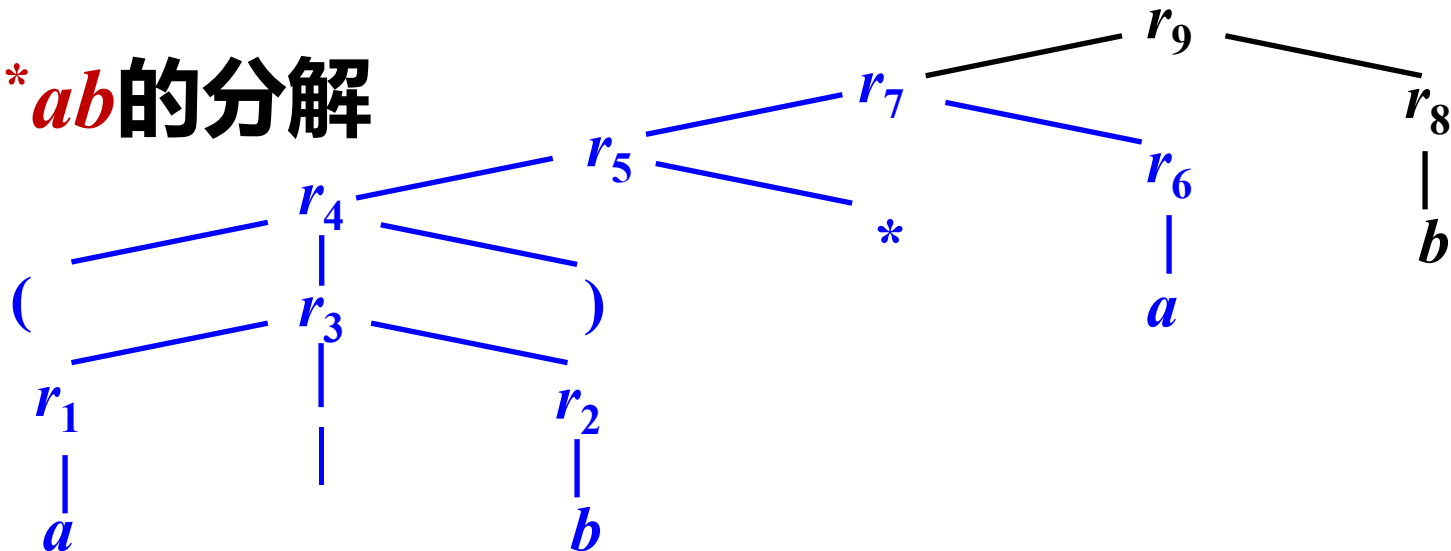


$(a|b)^*ab$ 的分解



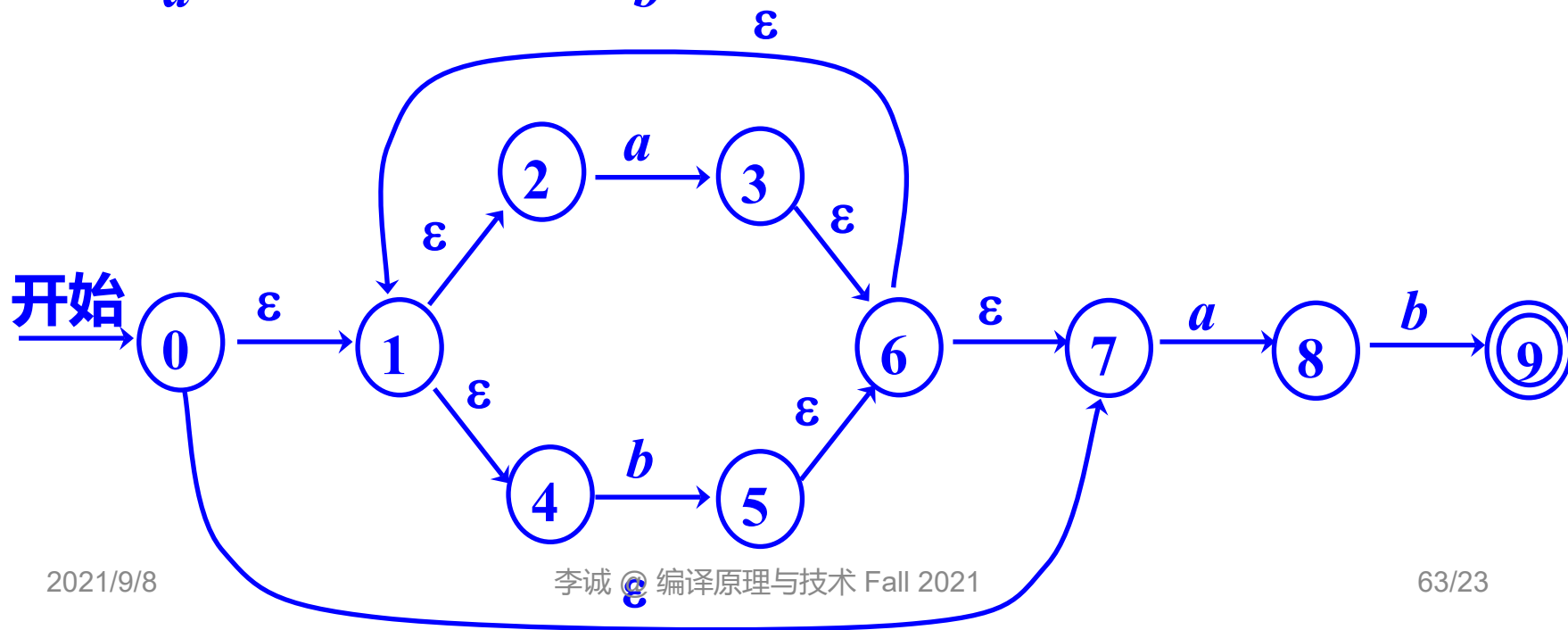
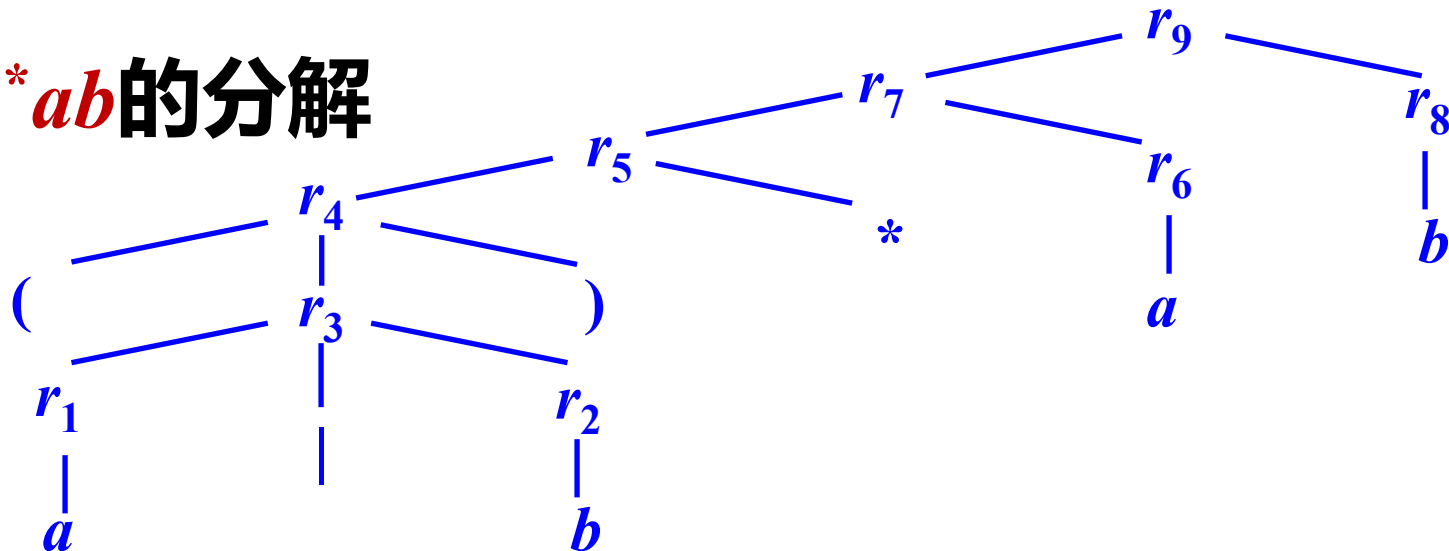


$(a|b)^*ab$ 的分解





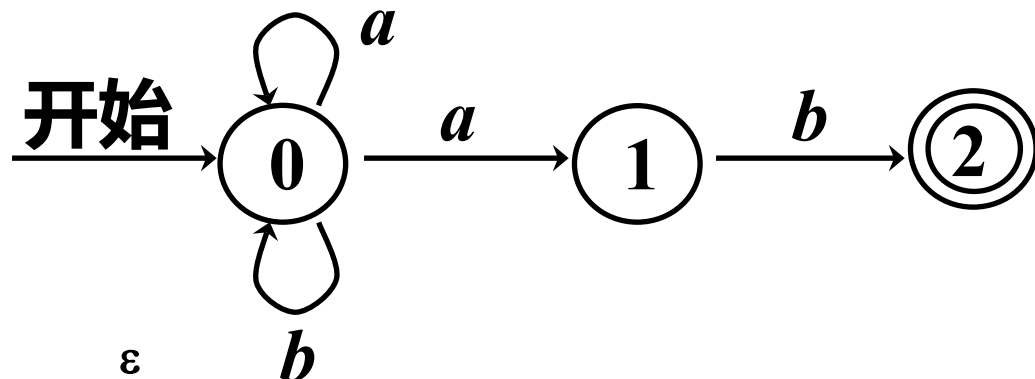
$(a|b)^*ab$ 的分解



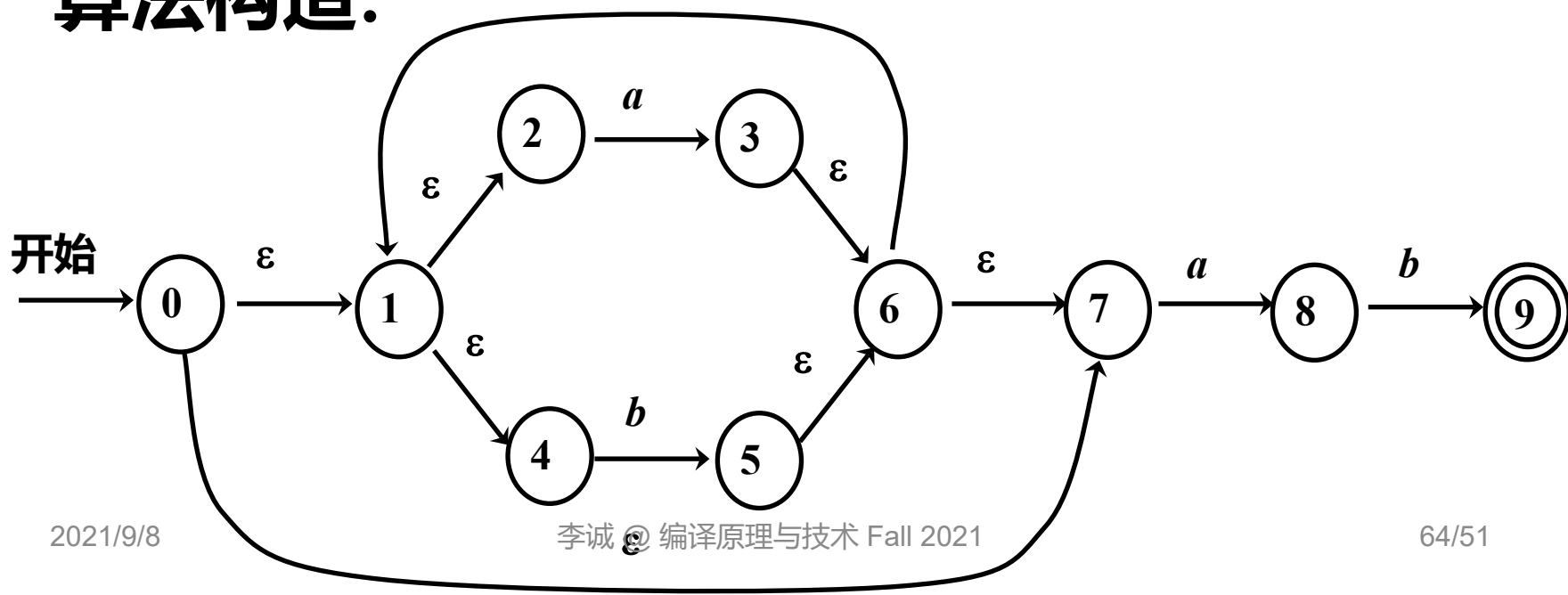


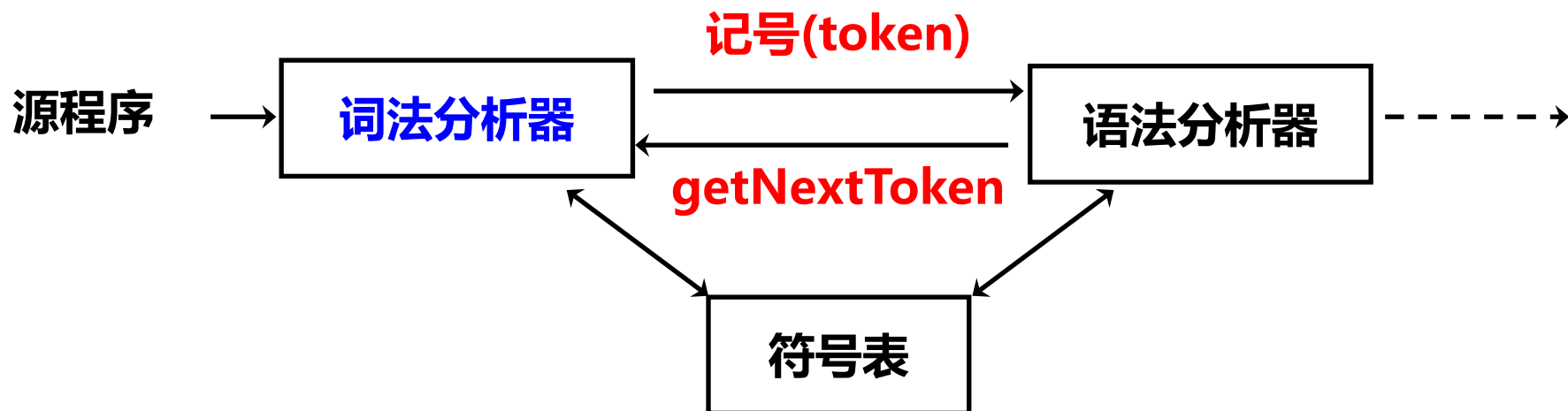
□ $(a|b)^*ab$ 的两个NFA的比较

手工构造:



算法构造:





□ 词法分析概述及自动化需要解决的问题

□ 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
- ❖ 正则表达式 → **NFA** → **DFA** → 化简的DFA

□ 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 $\{s_1, s_2, \dots, s_k\}$

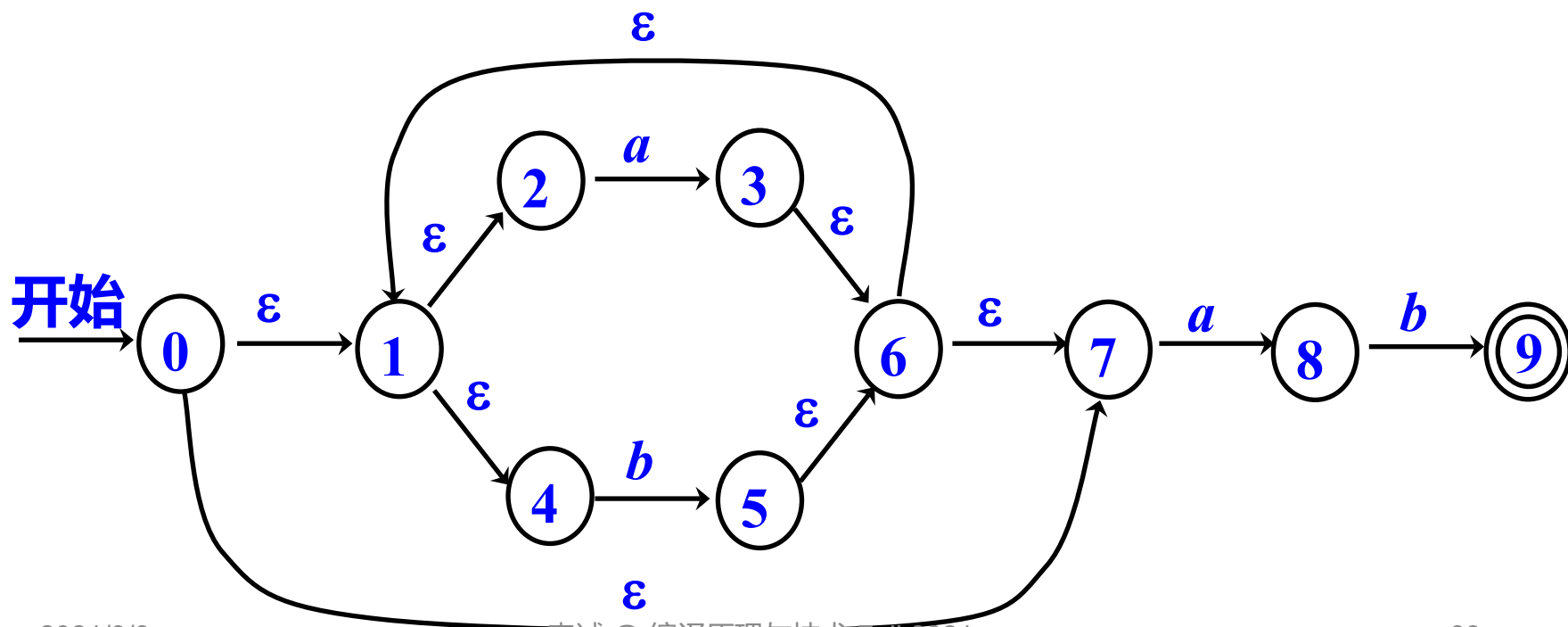


□ 子集构造法(subset construction)

- ❖ ϵ -闭包 (ϵ -closure) : 状态 s 的 ϵ -闭包是 s 经 ϵ 转换所能到达的状态集合
- ❖ NFA的初始状态的 ϵ -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 ϵ -闭包并集 (ϵ -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

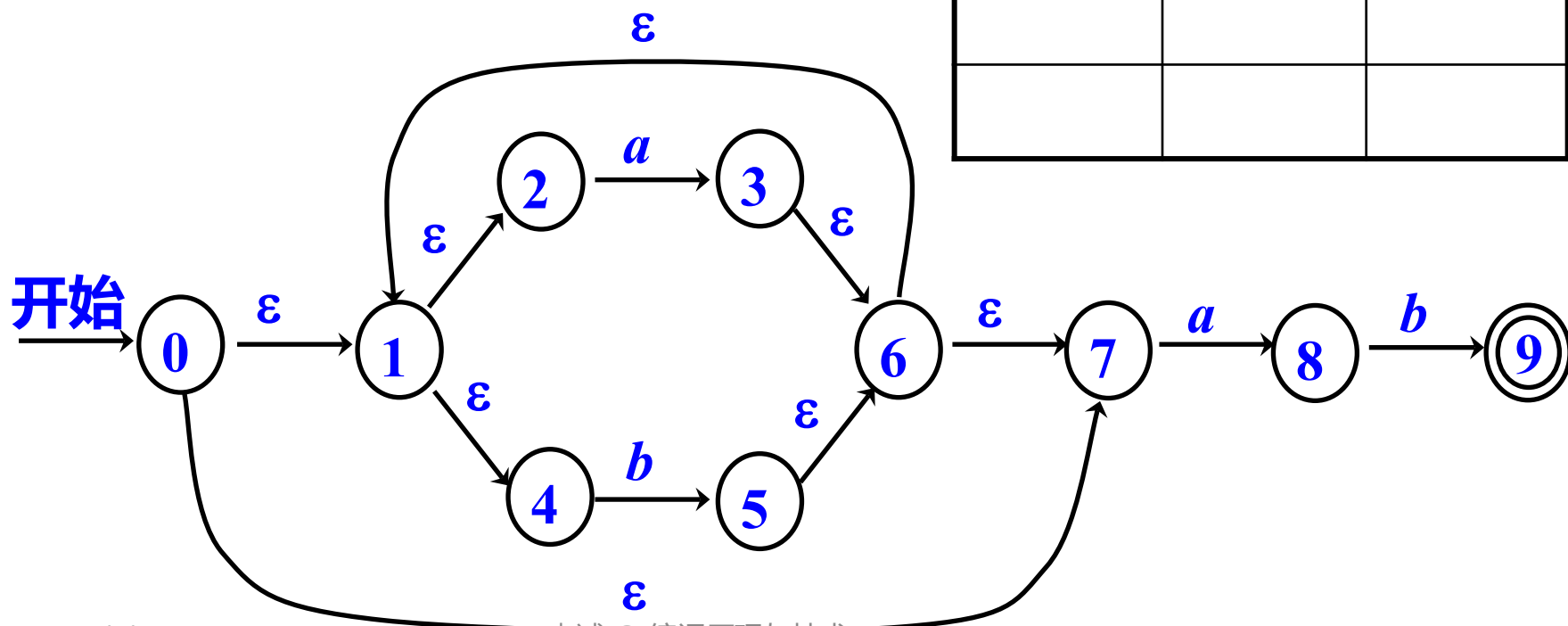


□ 例 $(a|b)^*ab$, NFA如下, 把它变换为DFA





NFA到DFA的变换



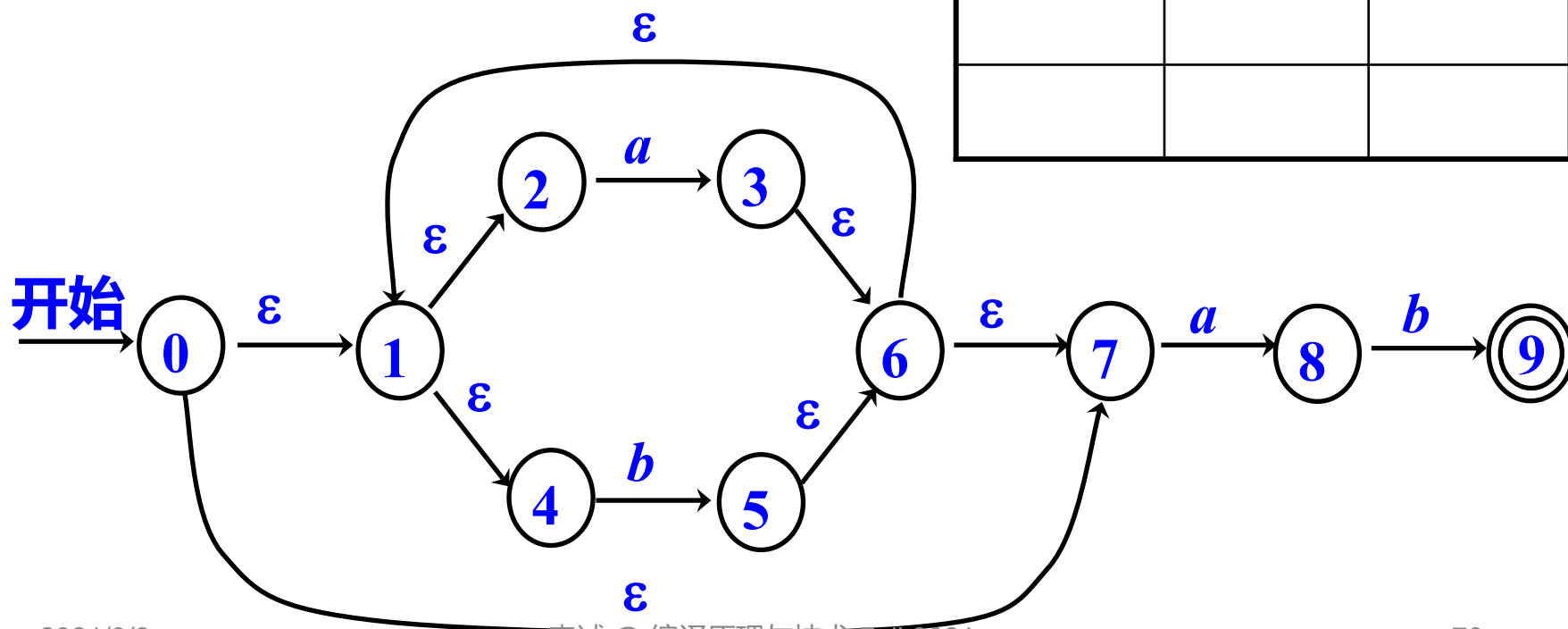


NFA到DFA的变换



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

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





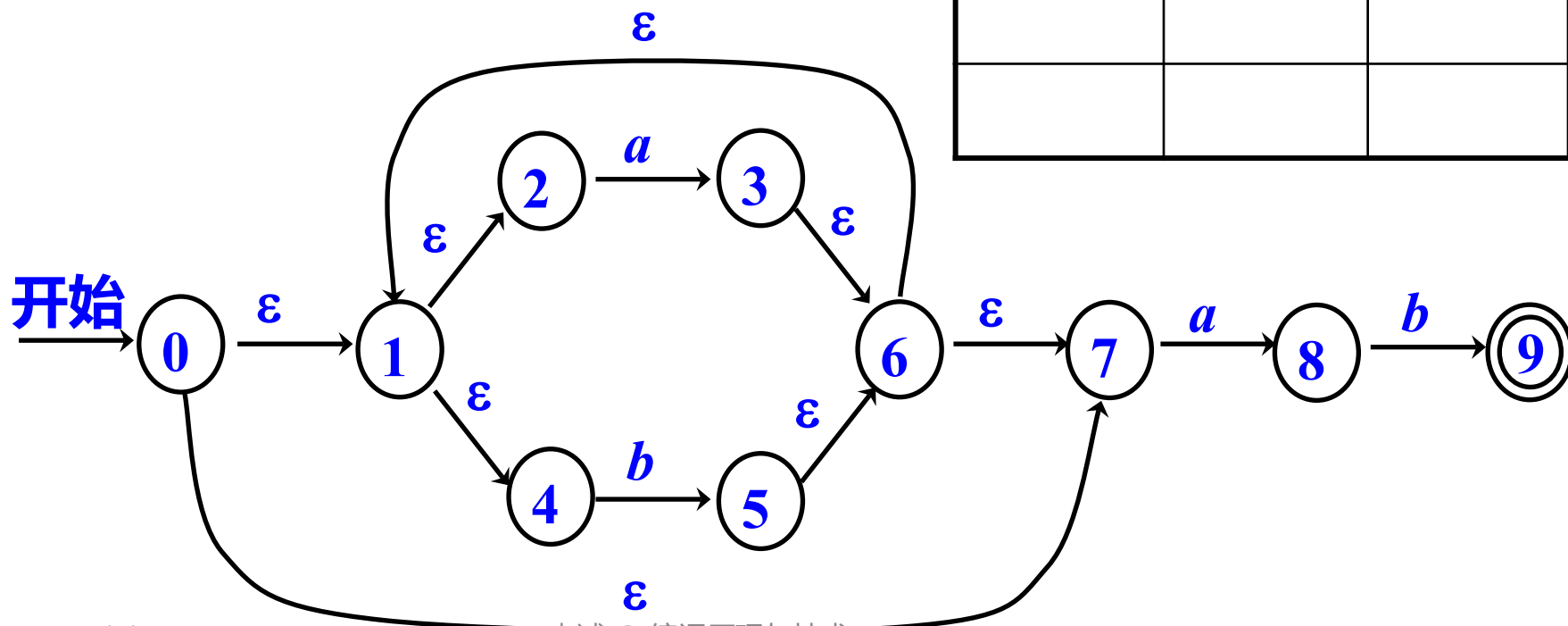
NFA到DFA的变换



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

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

状态	输入符号	
	a	b
A	B	





NFA到DFA的变换

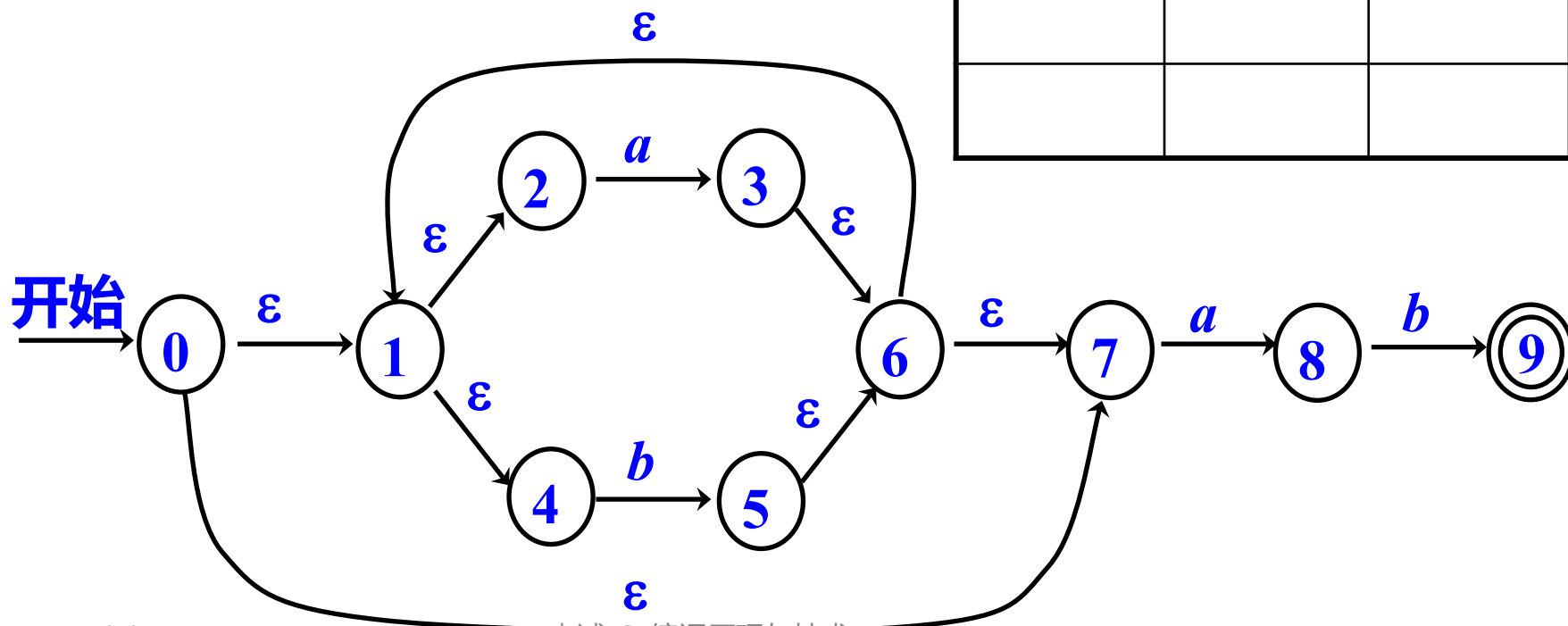


$A = \{0, 1, 2, 4, 7\}$

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

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

状态	输入符号	
	a	b
A	B	C





NFA到DFA的变换

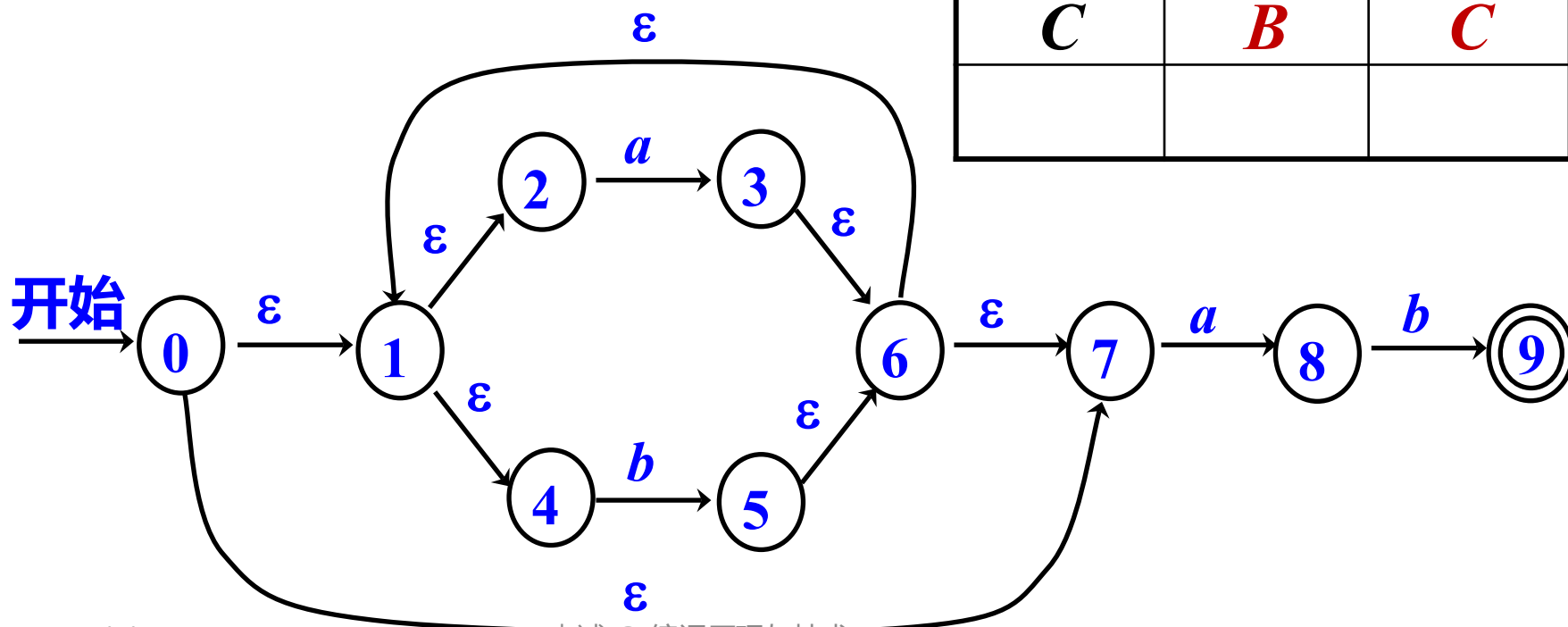


$$A = \{0, 1, 2, 4, 7\}$$

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

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

状态	输入符号	
	<i>a</i>	<i>b</i>
<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



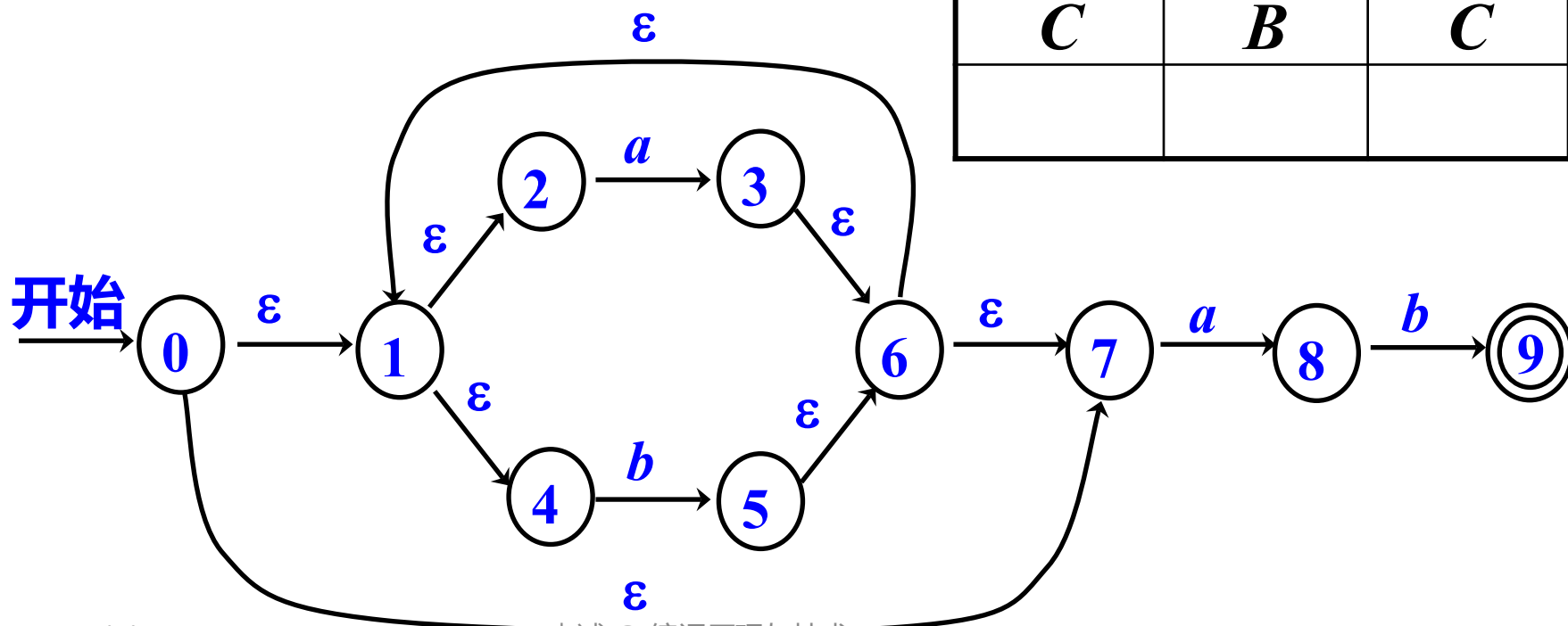
$$A = \{0, 1, 2, 4, 7\}$$

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

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

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

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>C</i>	<i>B</i>	<i>C</i>





NFA到DFA的变换



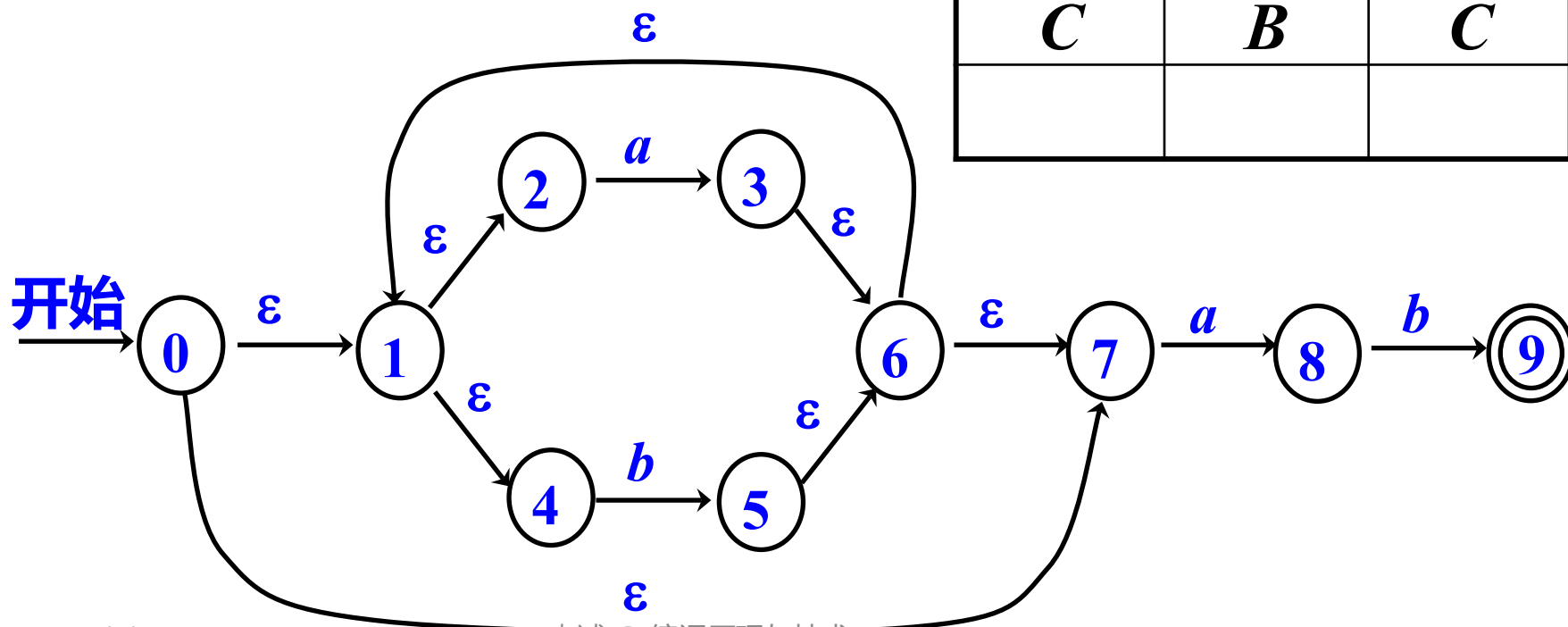
$$A = \{0, 1, 2, 4, 7\}$$

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

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

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

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



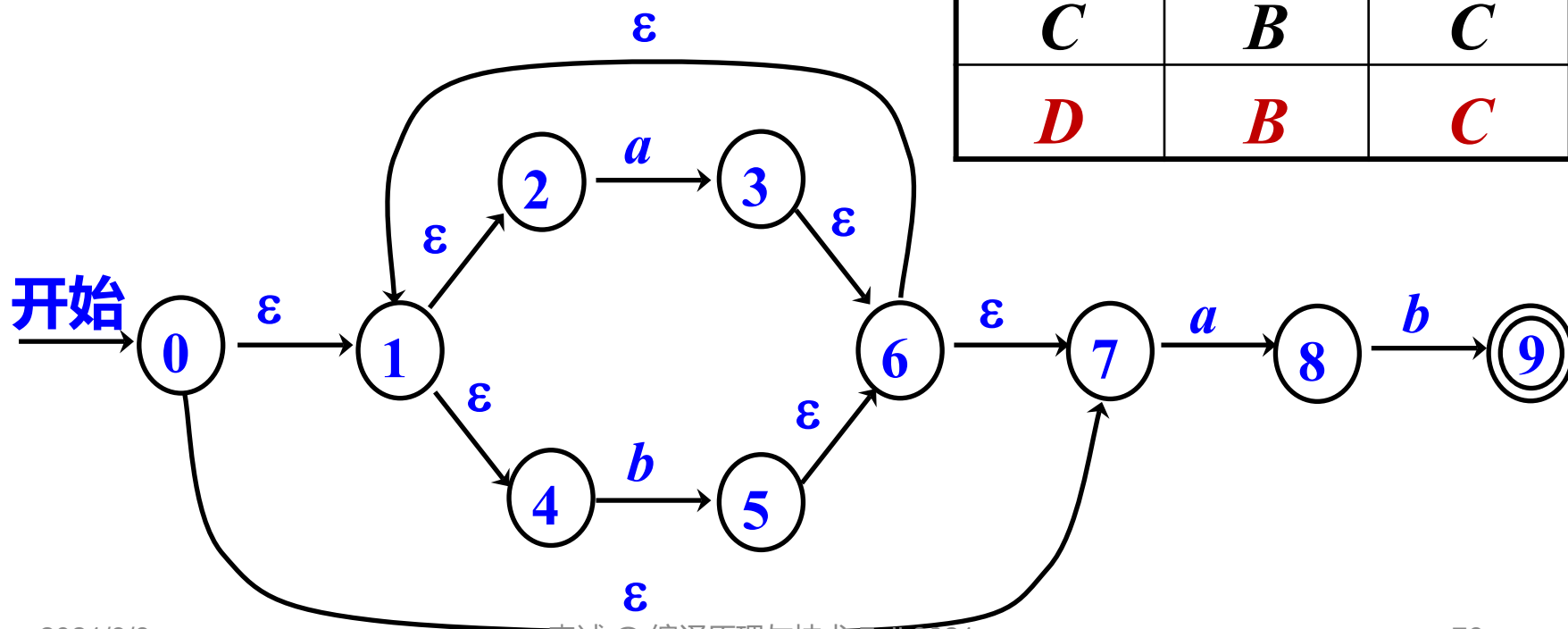
$$A = \{0, 1, 2, 4, 7\}$$

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

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

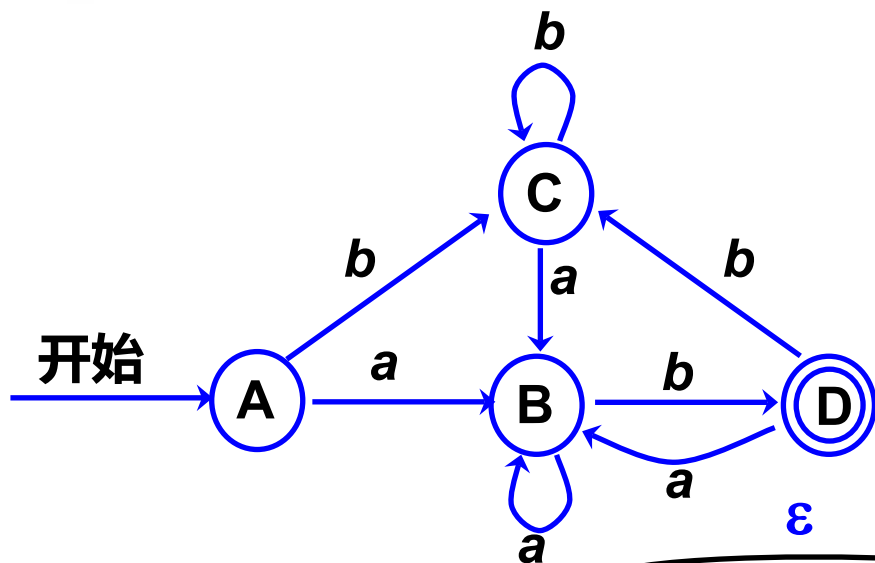
$$D = \{1, 2, 4, 5, 6, 7, 9\}$$

状态	输入符号	
	<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>

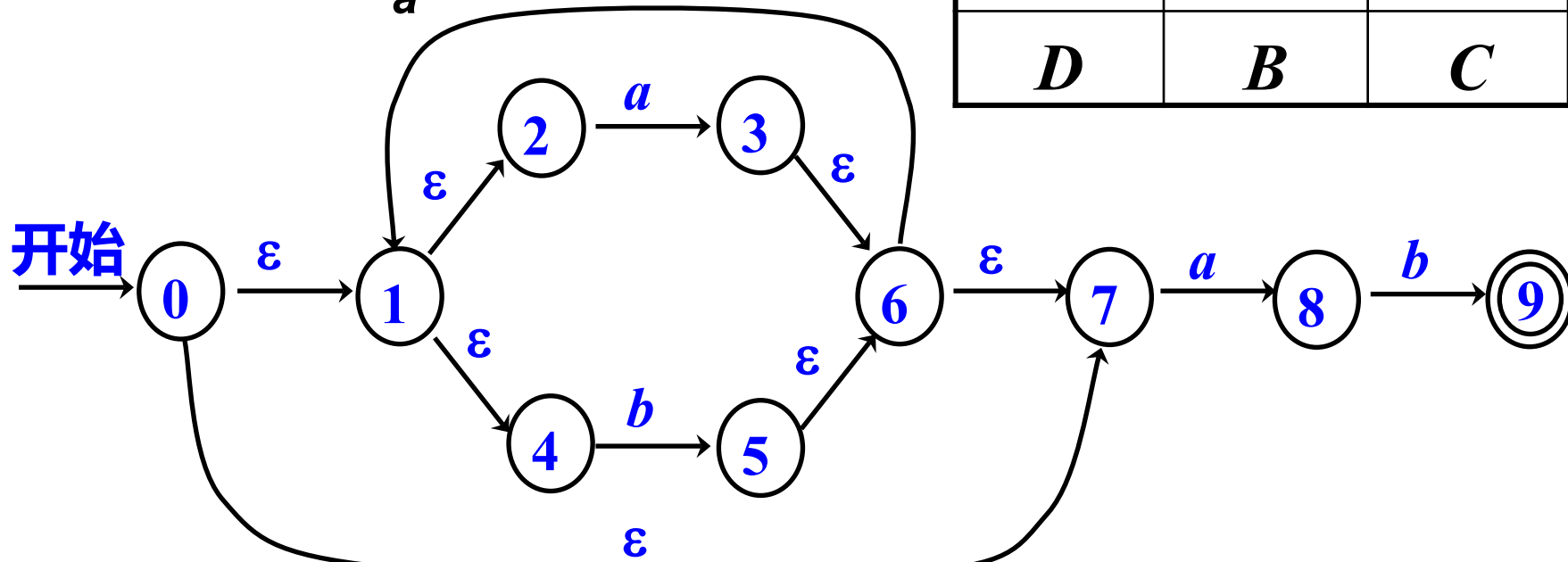




NFA到DFA的变换



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>

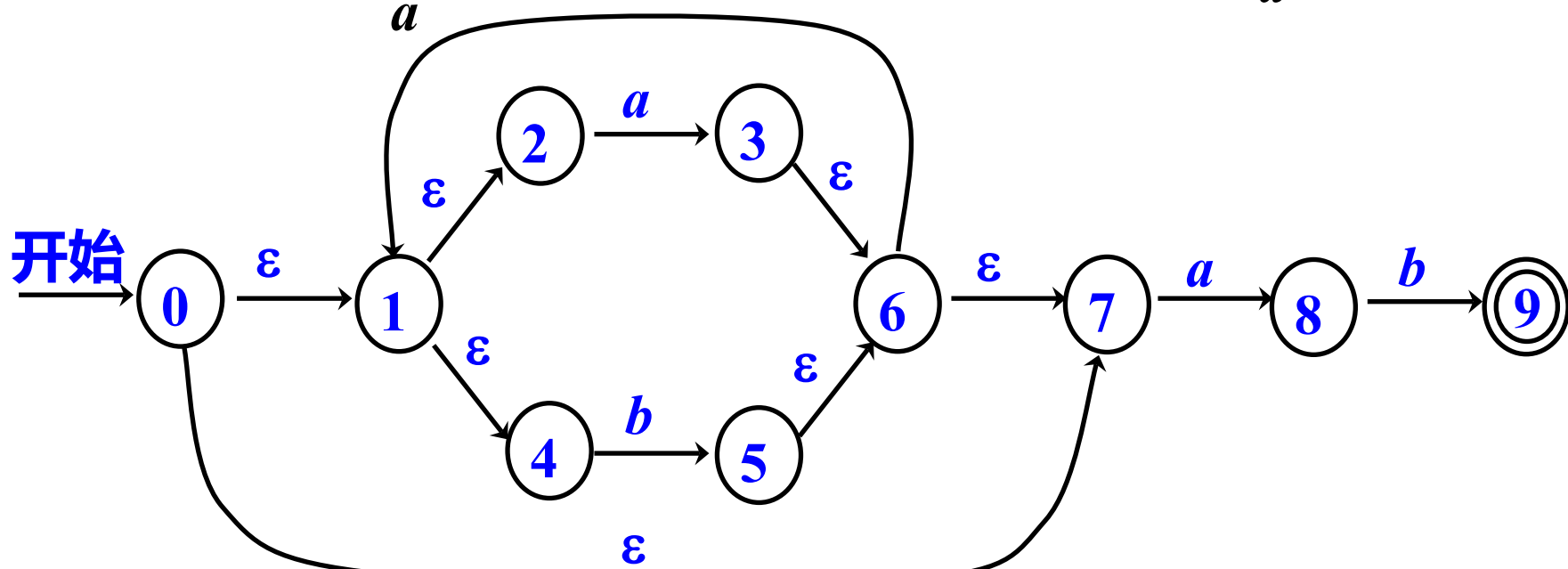
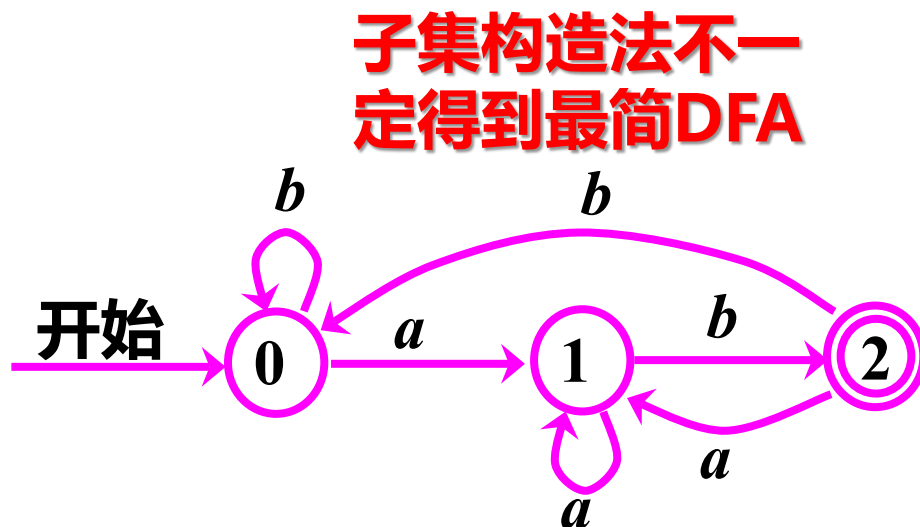
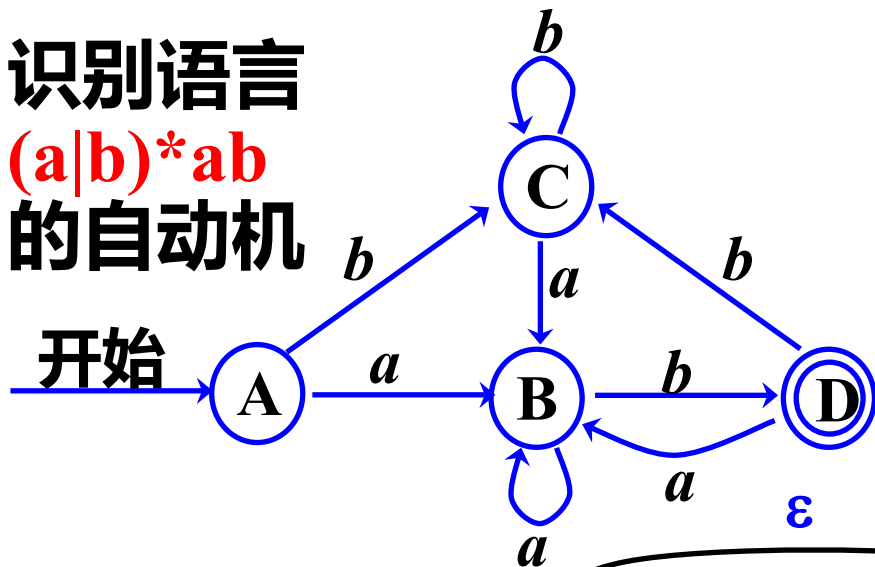


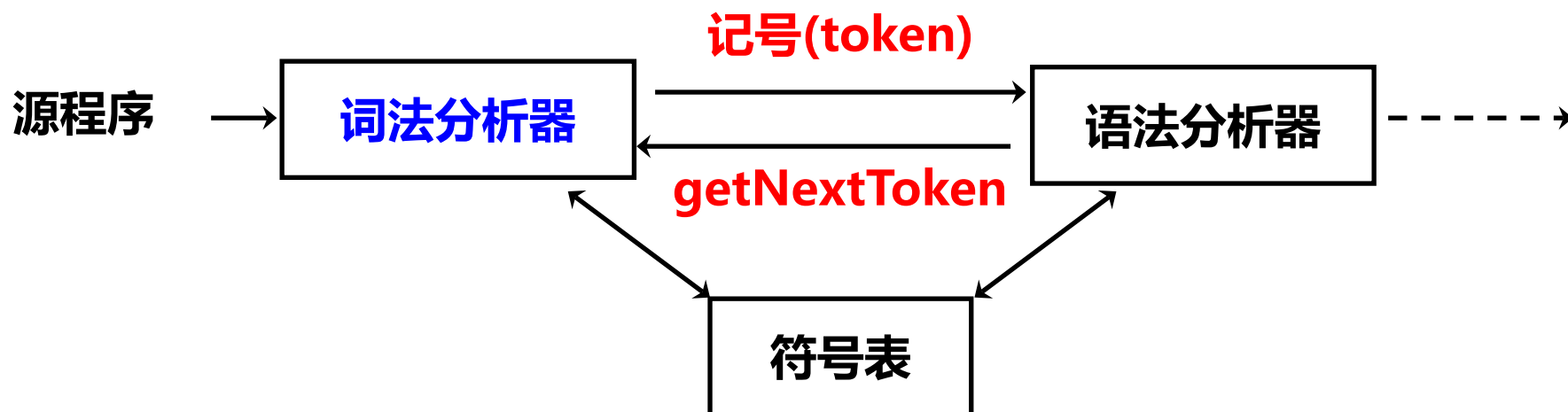


NFA到DFA的变换



识别语言
 $(a|b)^*ab$
的自动机





□ 词法分析概述及自动化需要解决的问题

□ 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
- ❖ 正则表达式 → NFA → **DFA** → 化简的**DFA**



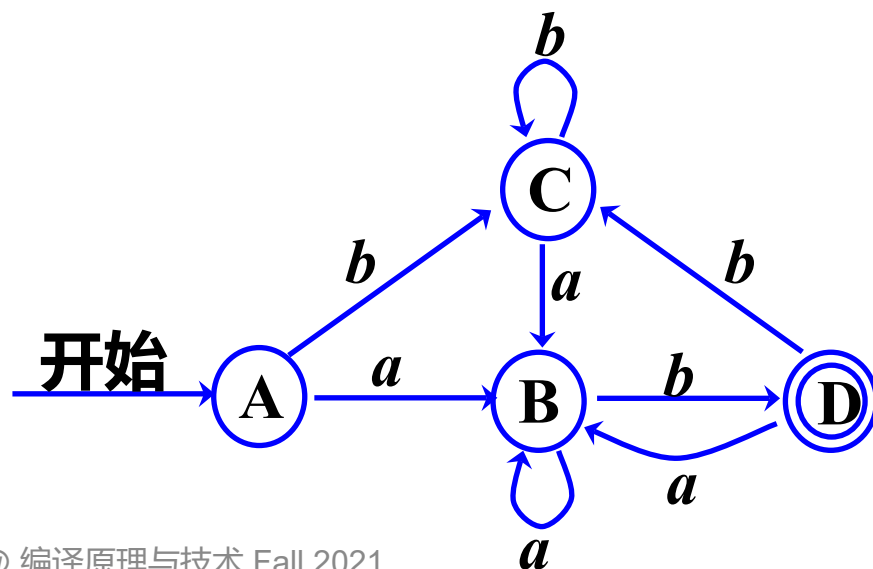
□ A和B是可区别的状态

- ❖ 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



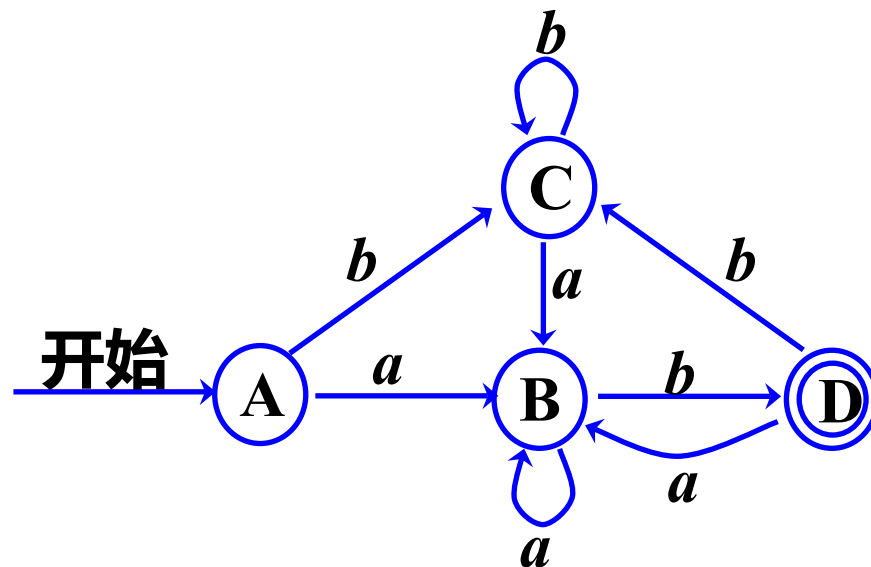


1. 按是否是接受状态来区分

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

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

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





1. 按是否是接受状态来区分

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

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

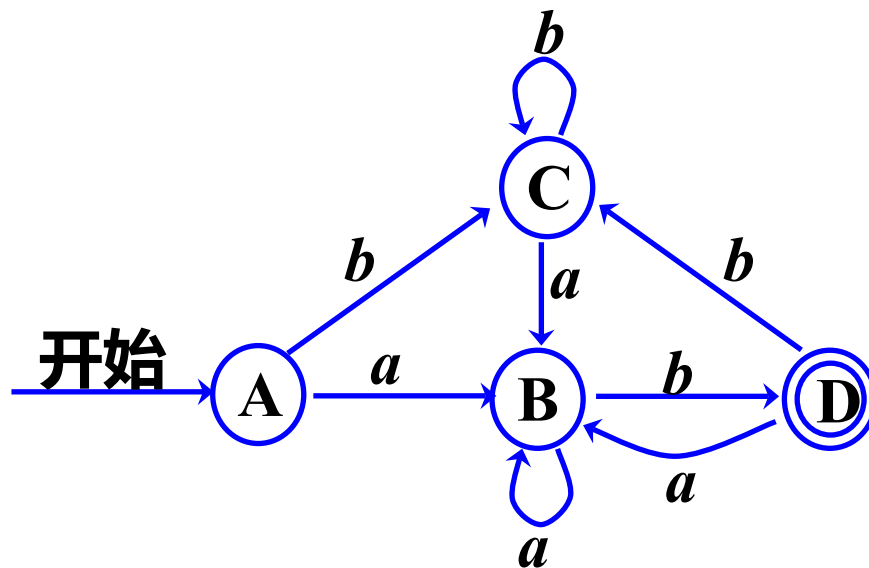
$\text{move}(\{A, B, C\}, b) = \{C, D\}$

2. 继续分解

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

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

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



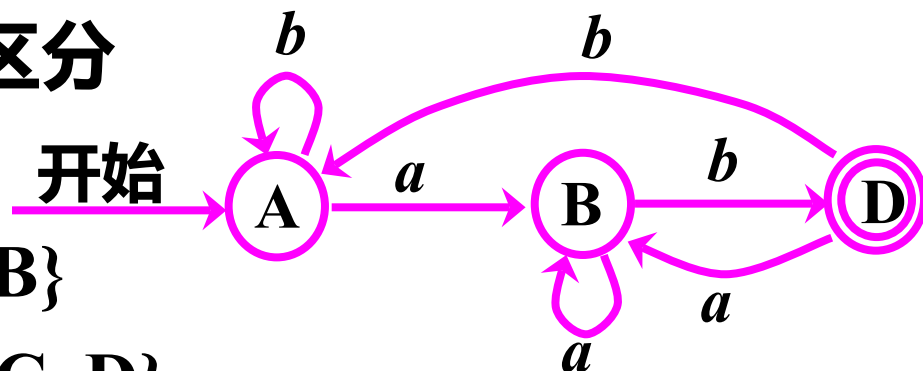


1. 按是否是接受状态来区分

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

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

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

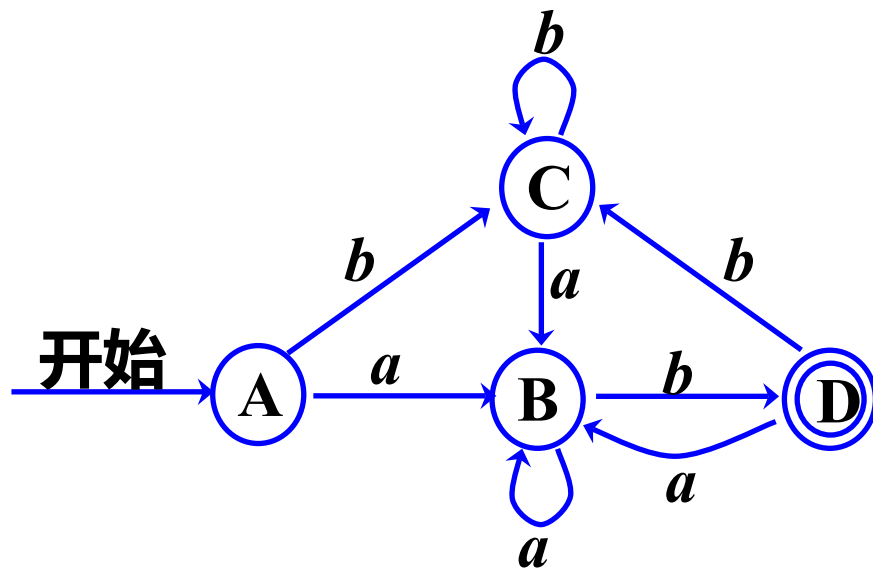


2. 继续分解

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

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

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





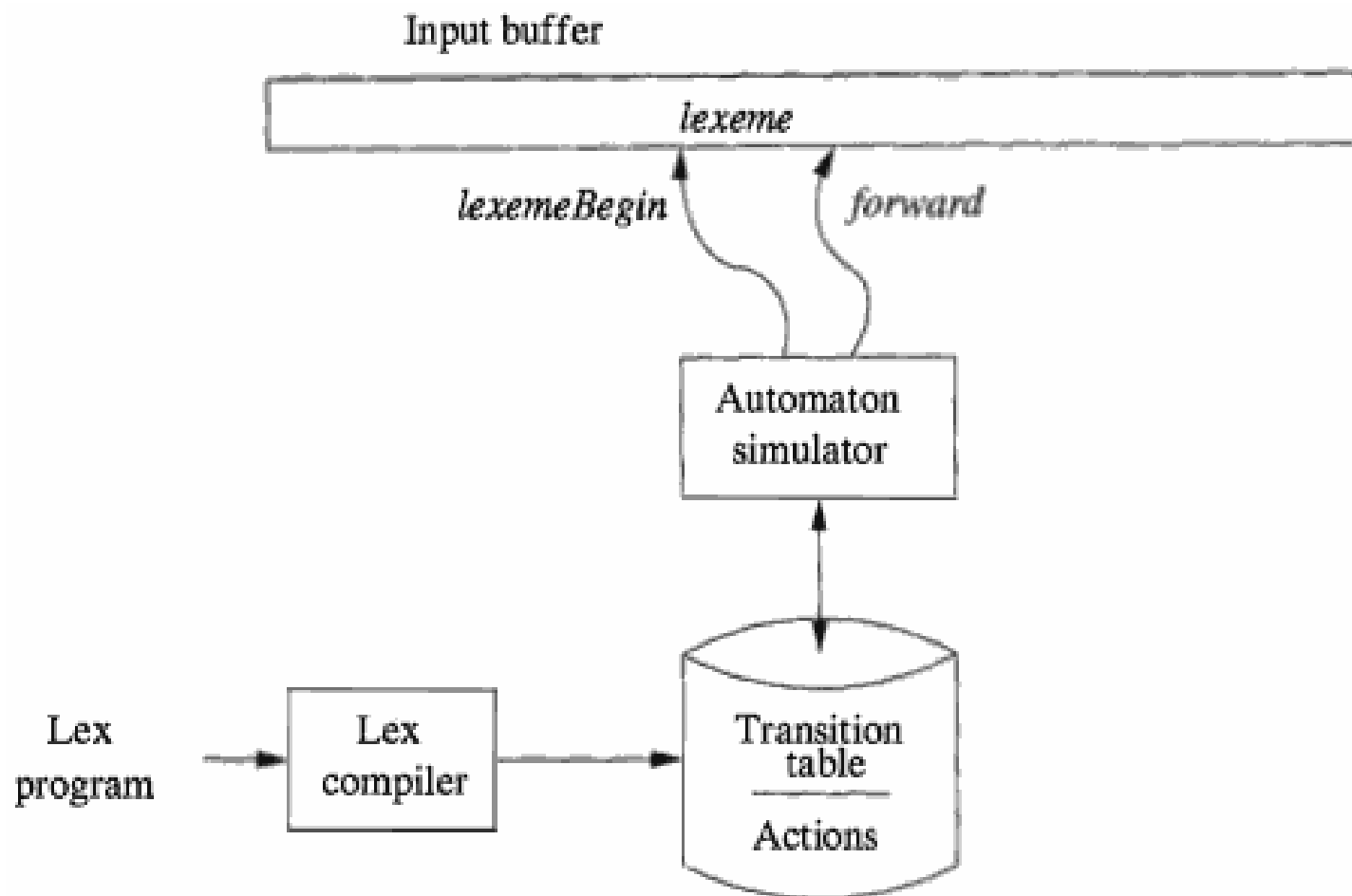
□ 正则表达式 $(a|b)^*$ 与 $(a^*|b^*)^*$ 是否等价？

❖ 提示：可利用其最简化DFA的

□ 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

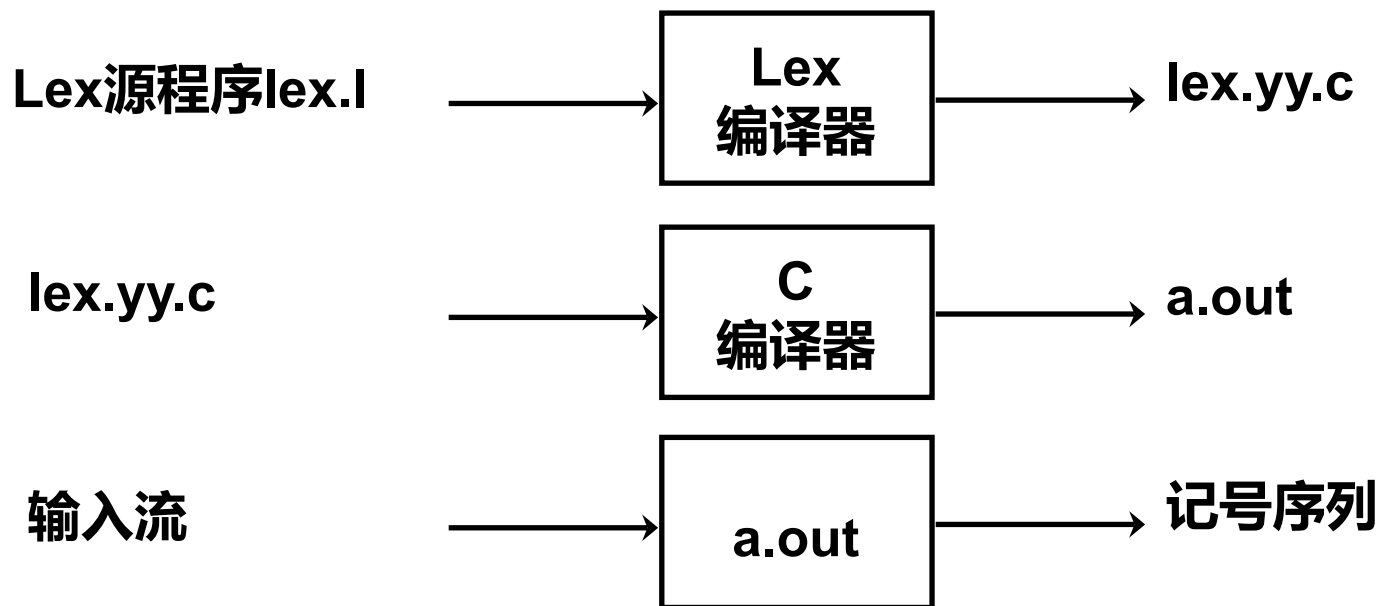
□ 掌握下面涉及的一些概念，它们之间转换的技巧、方法或算法

- ❖ 非形式描述的语言 \leftrightarrow 正则表达式
- ❖ 正则表达式 \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）





□ 用Lex建立词法分析器的步骤



□ 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}



```
%{  
/* 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}+)?
```



{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);}



```
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 */
```

```
}
```



□ FLEX tutorial

❖ <http://alumni.cs.ucr.edu/~lgao/teaching/flex.html>



《编译原理与技术》

词法分析

KISS (Keep It Simple, Stupid!)

It is easier to modify a working system than to get a system working.

——Wisdom