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

USTC, CHINA
ADSLAB
先进数据系统实验室

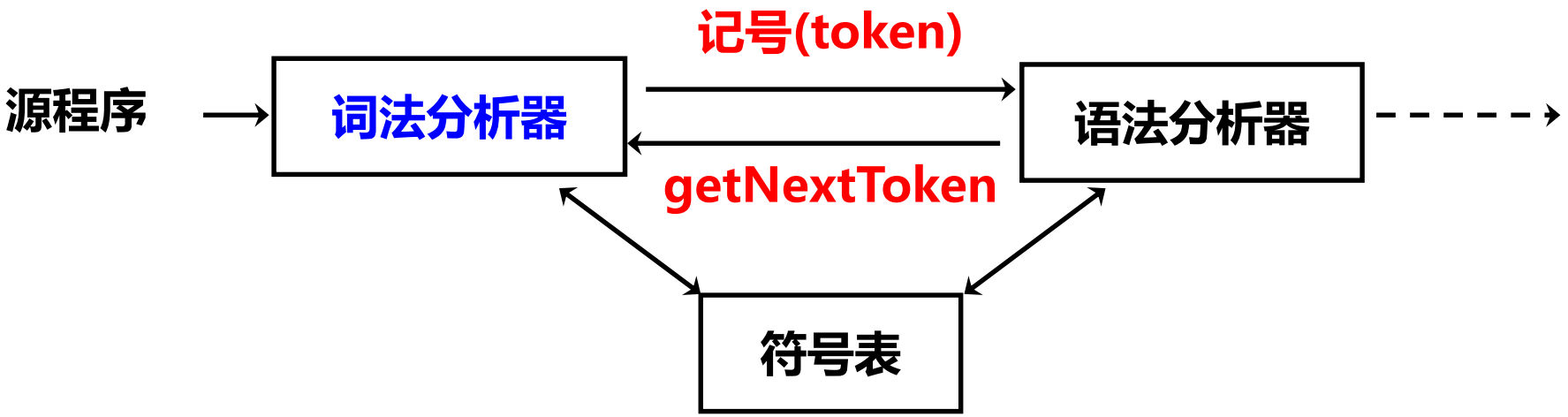
《编译原理与技术》
词法分析

计算机科学与技术学院

李 诚

06/09/2018

主要内容



- **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

此课件参考了陈意云、张昱老师及MIT、Stanford课程材料，特此感谢！

- **程序示例：**

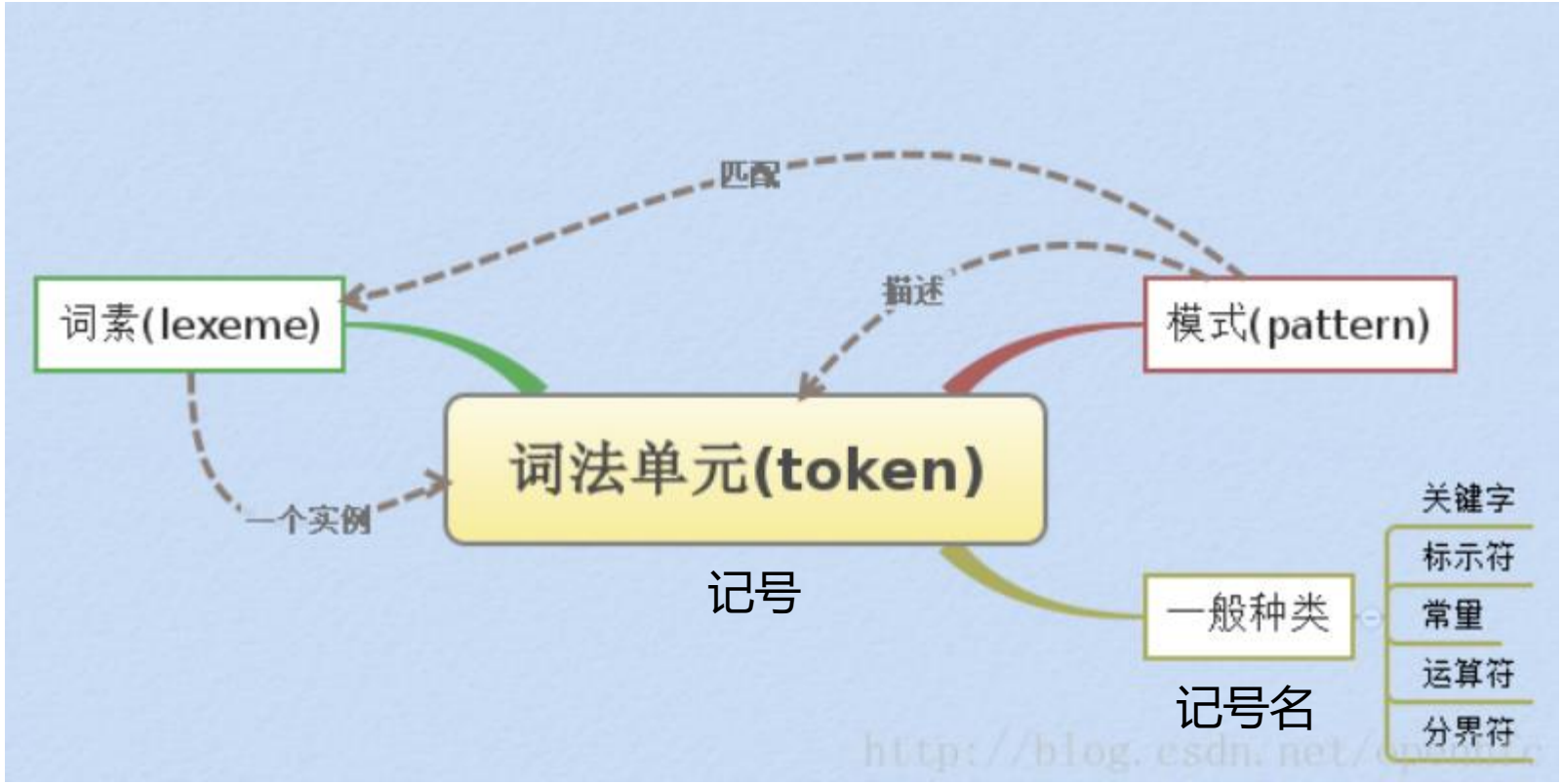
```
if (i == j)
  printf("equal!");
else
  num5 = 1;
```

- **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\telse\n\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子串的模式（Pattern）

四个关键术语


匹配
描述
词素(lexeme)
模式(pattern)
词法单元(token)
一个实例
记号
一般种类
记号名
关键字
标示符
常量
运算符
分界符

词法单元(记号)、实例与模式


```
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

词法单元(记号)的属性

- **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
  - token := <token_name, attribute_value>

- 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）

词法定义中的一些问题

- 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 I = 3. 75　等同于　DO8I = 3. 75

  DO 8 I = 3, 75　其中　DO 是关键字，8是语句标号...

词法定义中的一些问题

- **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
- **关键字不保留**

  IF THEN THEN THEN=ELSE；ELSE …

❑ 词法分析

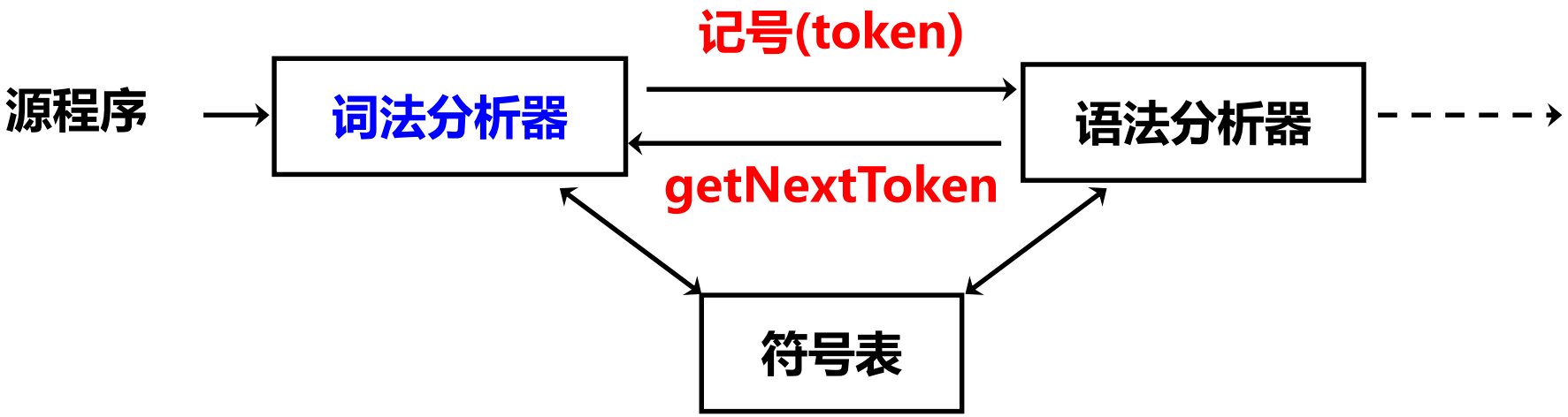
- ❖ 从左到右读取输入串，每次识别出一个token实例
- ❖ 可能需要“lookahead”来判断当前是否是一个token实例的结尾、下一个实例的开始（尤其是在Fortran语言中）

- 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

设计词法分析器的挑战是?

- 一种可以描述记号所对应的全部实例的方法
  - id => {a, b, cat, dog, …}
- 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

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*，ε
- 语言：字母表上的一个串集

  $\{\varepsilon, 0, 00, 000, \ldots\}$，$\{\varepsilon\}$，$\varnothing$
- 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$\{\varepsilon\}$, L^i是$L^{i-1}L$
- ❖ 闭包: $L^* = L^0 \cup L^1 \cup L^2 \cup \ldots$
- ❖ 正闭包: $L^+ = L^1 \cup L^2 \cup \ldots$

优先级: 幂〉连接〉并

❑ 示例

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^+

- 正则式用来表示简单的语言

正则式	定义的语言	备注
ε	{ε}	
a	{*a*}	$a \in \Sigma$
(*r*)	*L*(*r*)	*r*是正则式
(*r*) | (*s*)	*L*(*r*)∪*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*) $(b)^*$)| (*c*)可以写成ab^*| *c*

优先级：
闭包*〉连接〉选择 |

正则表达式例子

- $\Sigma = \{a, b\}$
  - $a \mid b$ $\{a, b\}$
  - $(a \mid b)(a \mid b)$ $\{aa, ab, ba, bb\}$
  - $aa \mid ab \mid ba \mid bb$ $\{aa, ab, ba, bb\}$
  - a^* 由字母a构成的所有串集
  - $(a \mid b)^*$ 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：0100110100001000001011001

❑ 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。

$$d_1 \rightarrow r_1$$

$$d_2 \rightarrow r_2$$

$$\ldots$$

$$d_n \rightarrow r_n$$

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

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)*

正则定义的例子

无符号数集合，例1946,11.28,63E8,1.99E–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(+|–)? $digit^+$)?

正则定义的例子

while → while

do → do

relop → < | < = | = | < > | > | > =

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

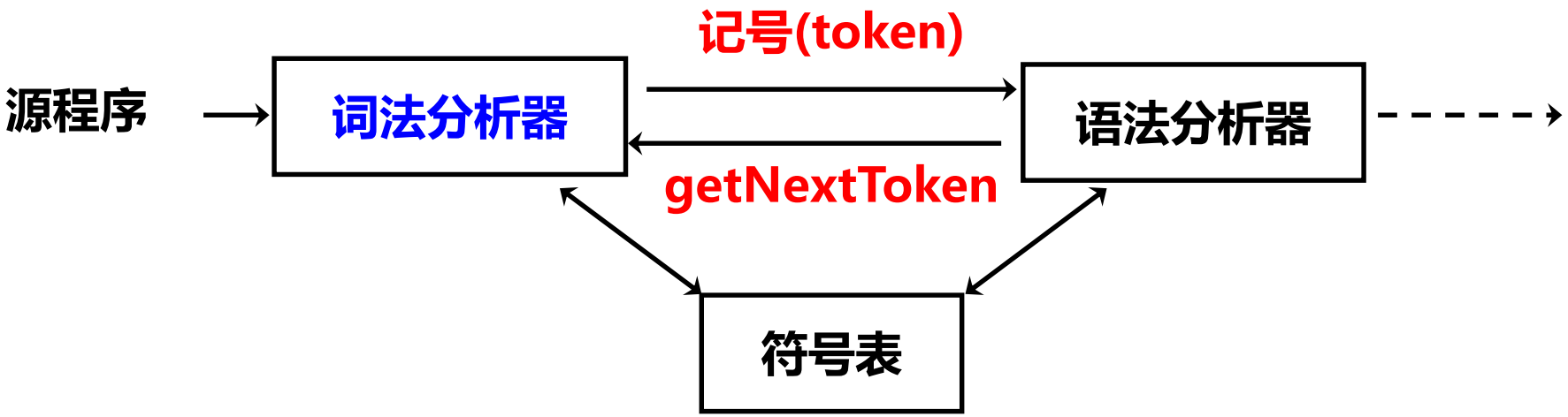
id → **letter** (**letter** | **digit**)*

number → **digit**+ (.**digit**+)? (E (+ | –)? **digit**+)?

delim → **blank** | **tab** | **newline**

ws → **delim**+

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？

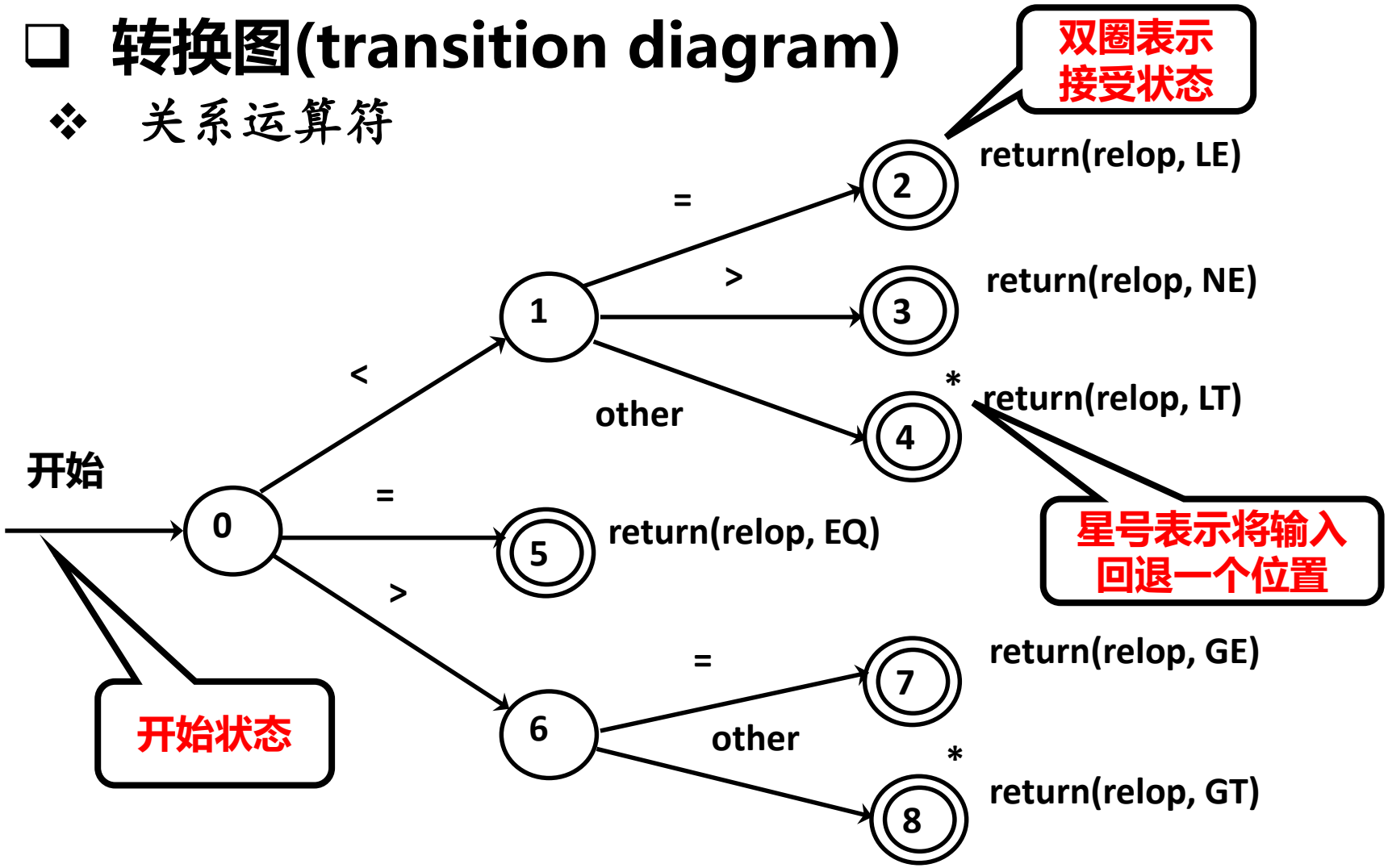


- 词法分析所面临的问题
  - 向前看 (Lookahead)、歧义 (Ambiguities)
- **词法分析器的自动生成**
  - 词法单元的描述：正则式
  - 词法单元的识别：转换图
  - 有限自动机：NFA、DFA

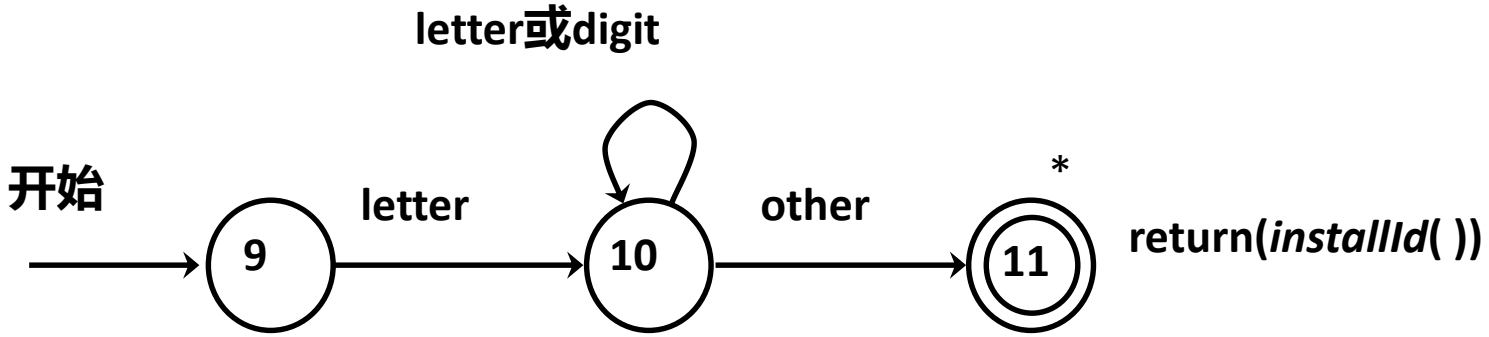
词法记号的识别：转换图

❑ 转换图(transition diagram)

- ❖ 关系运算符

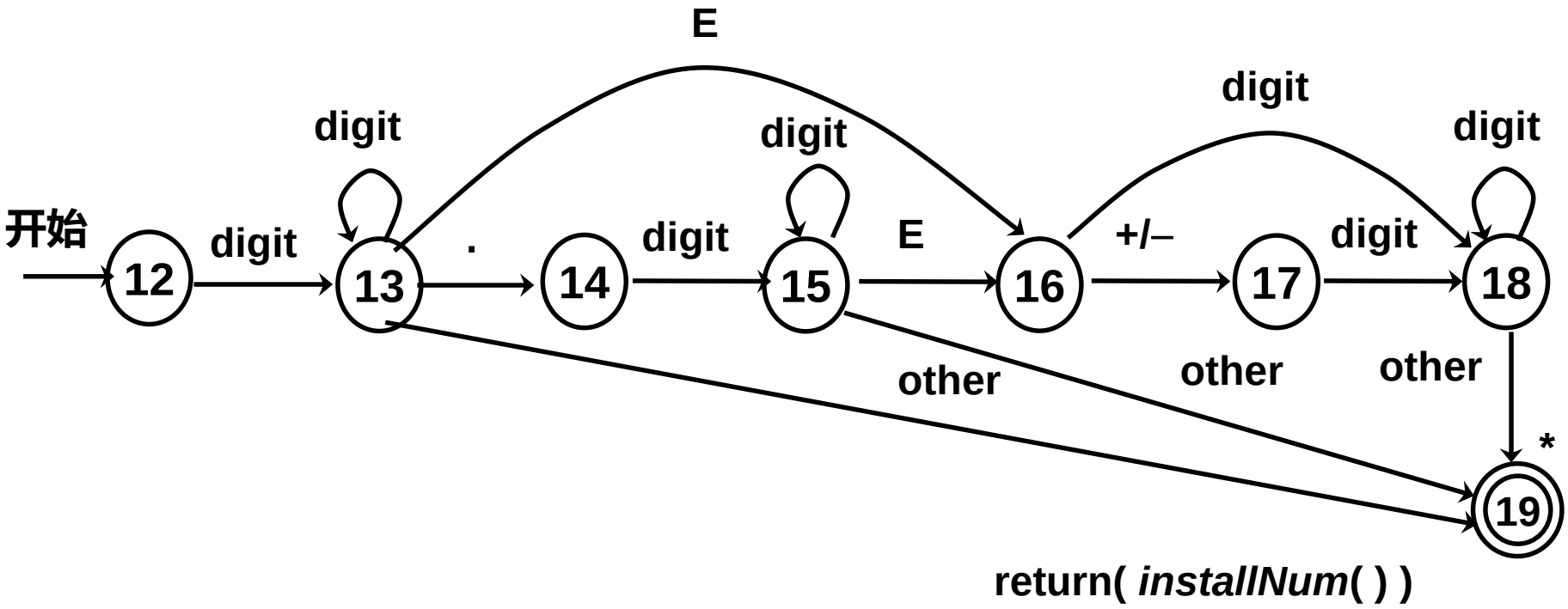


- 标识符和关键字的转换图



- 无符号数的转换图

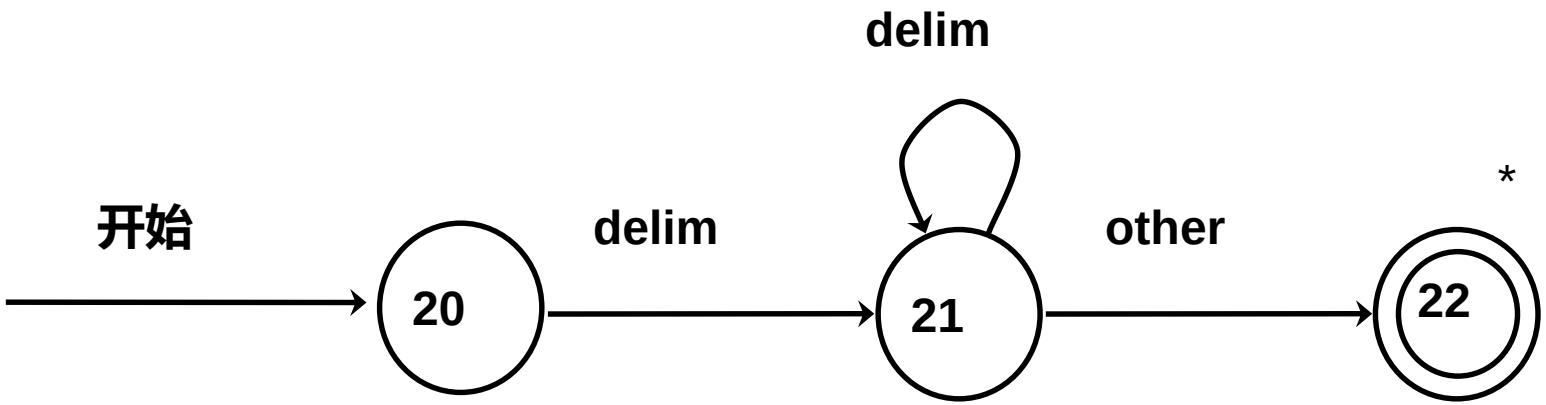
number → digit+ (.digit+)? (E (+ | –)? digit+)?



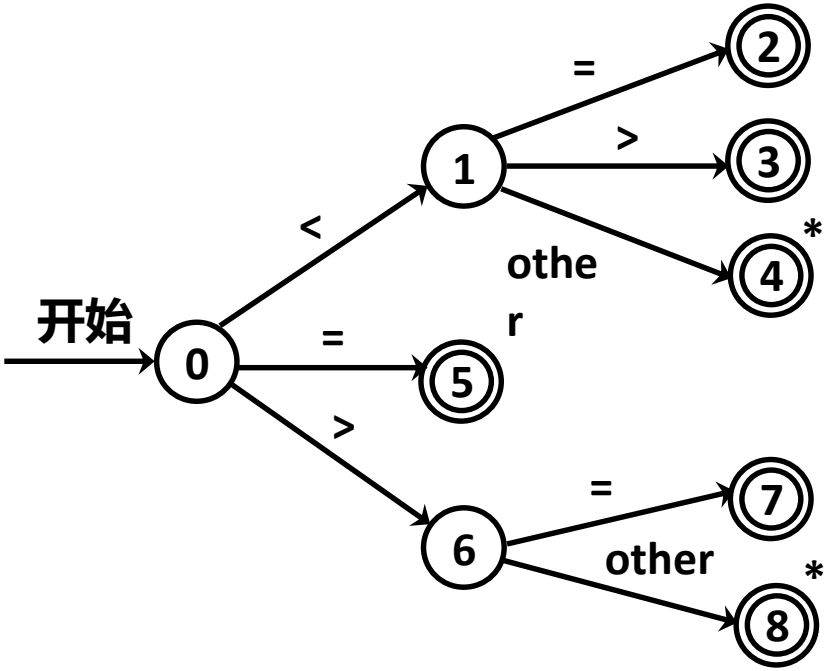
❑ 空白的转换图

delim → blank | tab | newline

ws → delim+

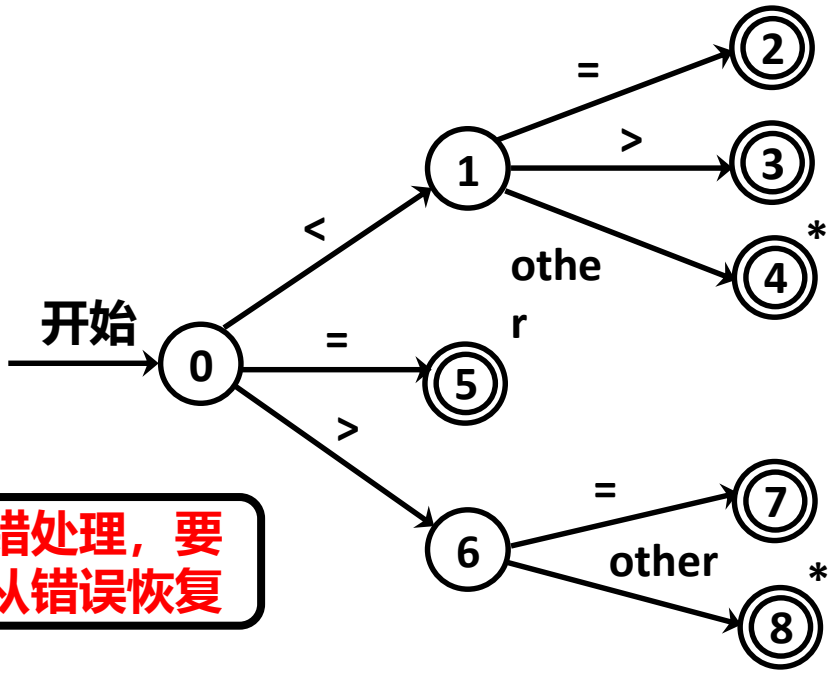


- 例：relop的转换图的概要实现



❑ 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

- $x_1 \ldots x_i \in L(R)$ 并且 $x_1 \ldots x_k \in L(R)$

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 = Whitespace| 'new' | Integer| 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

- $x_1 \ldots x_i \in L(R_j)$ 并且 $x_1 \ldots x_i \in L(R_k)$

优先 match 规则:

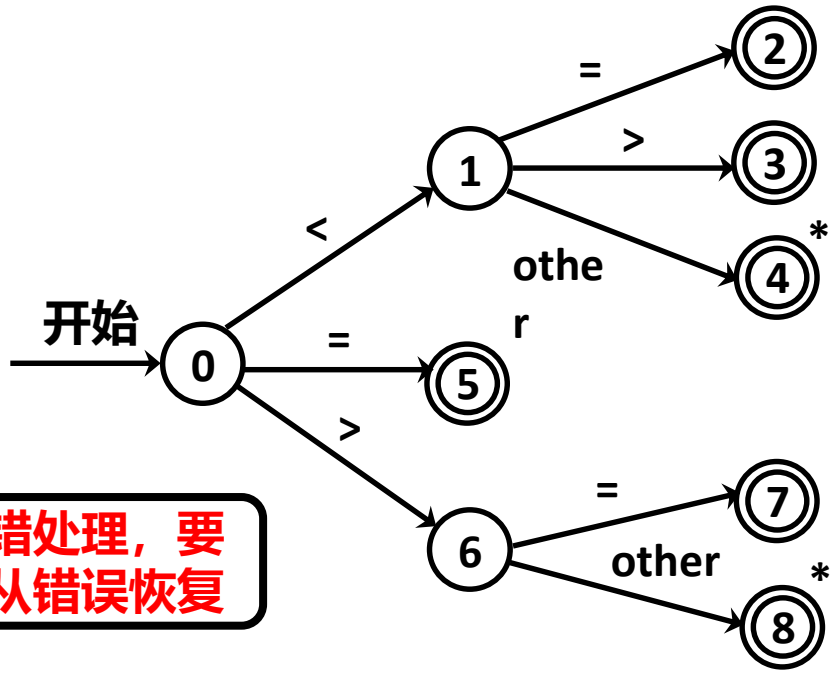
- 选择先列出的模式 (j 如果 j < k)
- 必须将 'new'列在 Identifier的前面

- 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

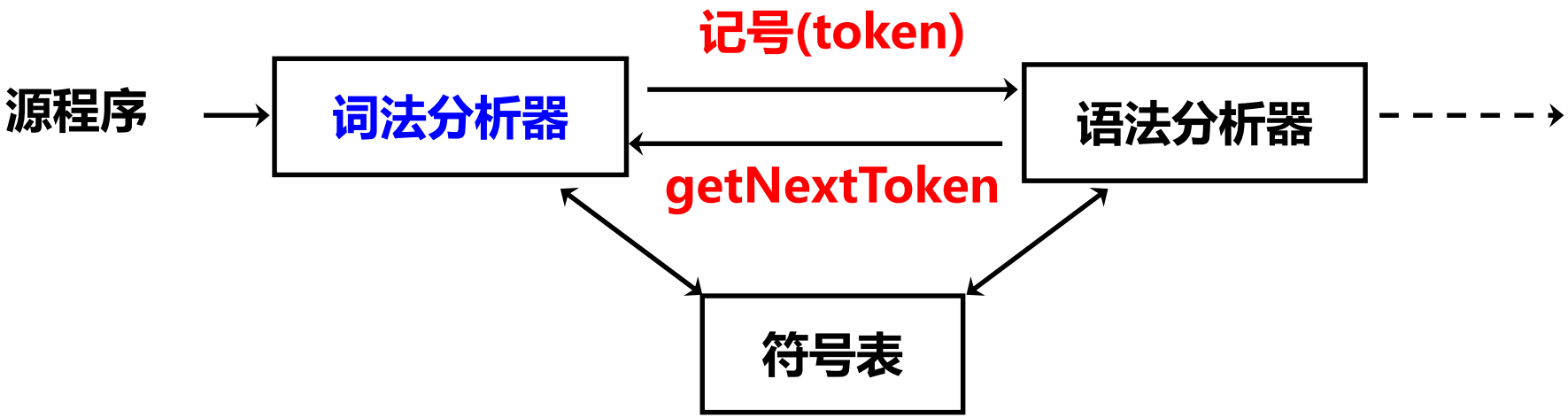
❑ 例：relop的转换图的概要实现

```
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？



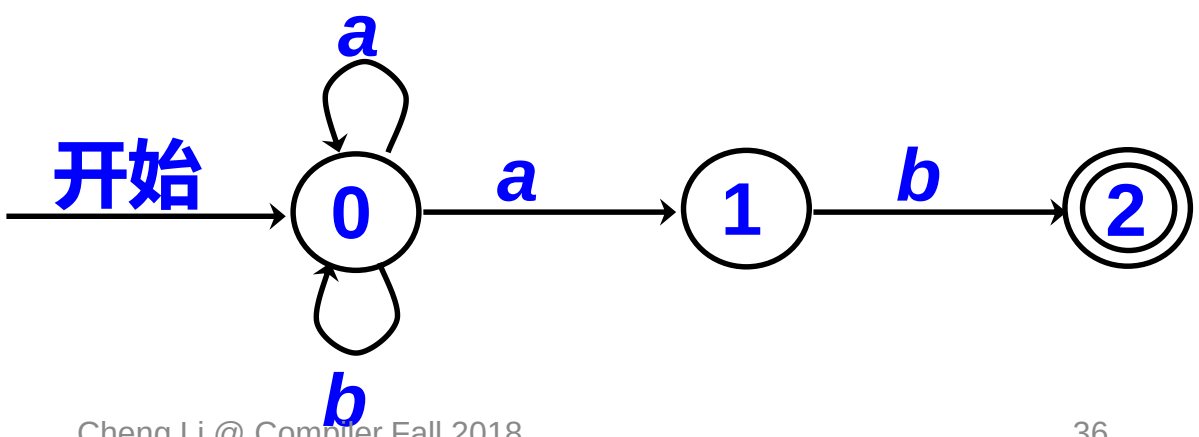
- 词法分析所面临的问题
  - 向前看 (Lookahead)、歧义 (Ambiguities)
- **词法分析器的自动生成**
  - 词法单元的描述：正则式
  - 词法单元的识别：转换图
  - 有限自动机：NFA、DFA

- 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$\sum$
  - 转换函数$move: S \times (\sum \cup \{\varepsilon\}) \rightarrow P(S)$
  - 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$\sum$
  - 转换函数$move: S \times (\sum \cup \{\varepsilon\}) \rightarrow P(S)$
  - 状态s_0是唯一的开始状态
  - $F \subseteq S$是接受状态集合

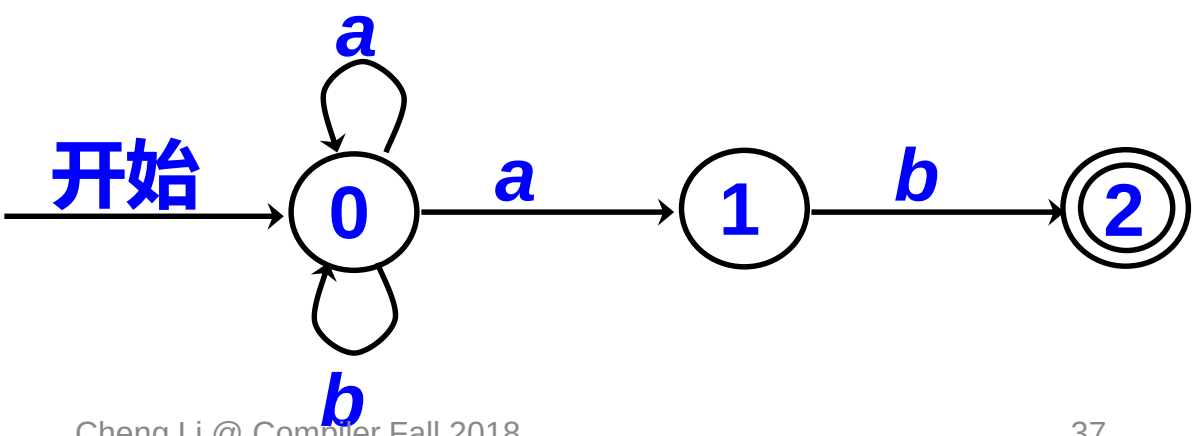
识别语言 $(a|b)^*ab$ 的NFA



❑ NFA的转换表

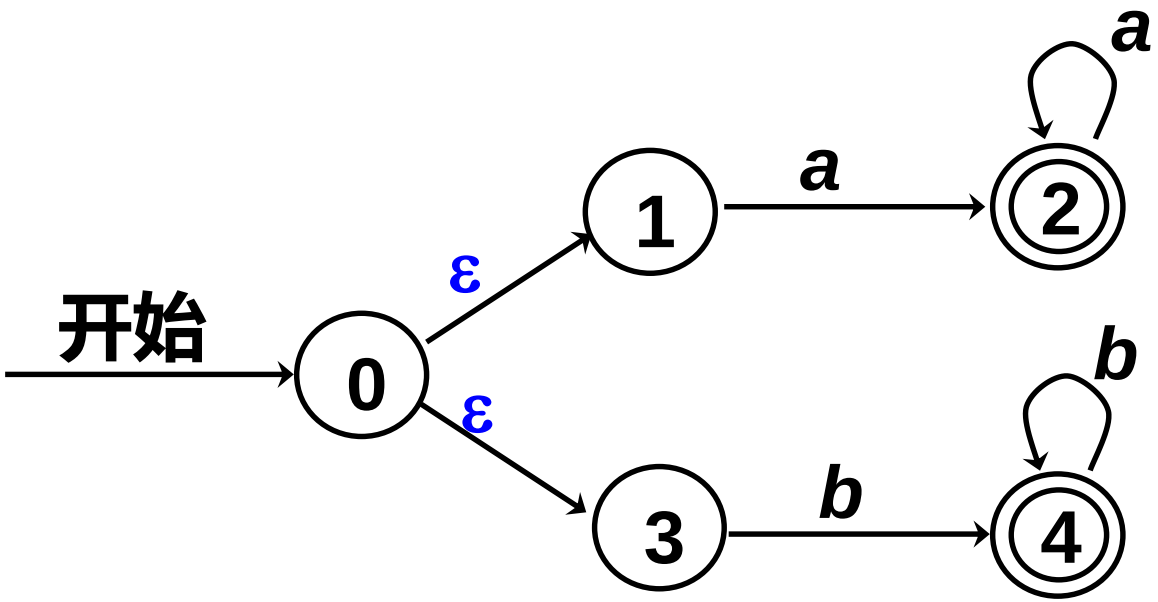
状 态	输 入 符 号	
	a	*b*
0	{0, 1}	{0}
1	∅	{2}
2	∅	∅

识别语言 $(a|b)^*ab$ 的NFA



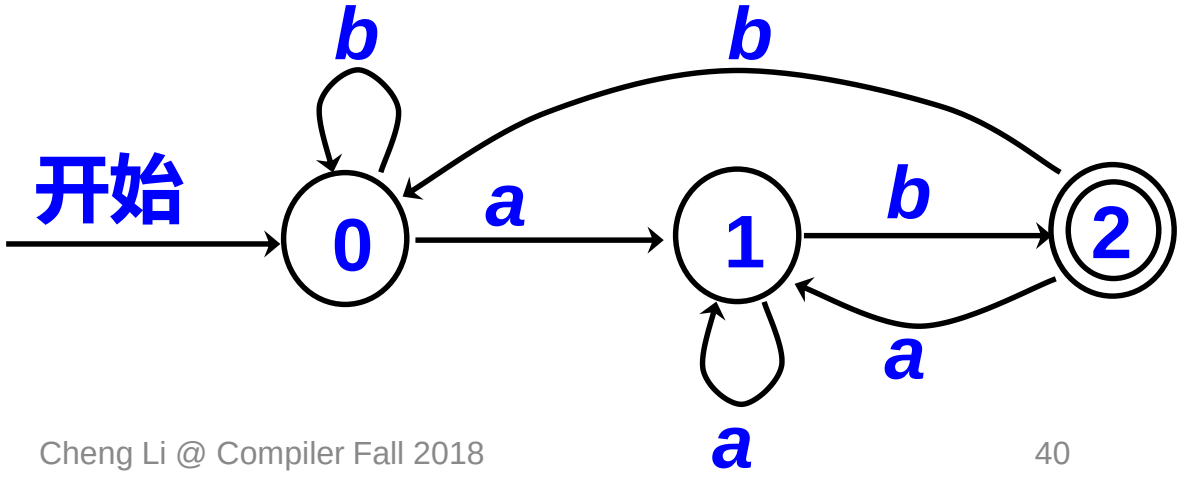
- 例 识别aa*|bb*的NFA

- 例 识别aa*|bb*的NFA



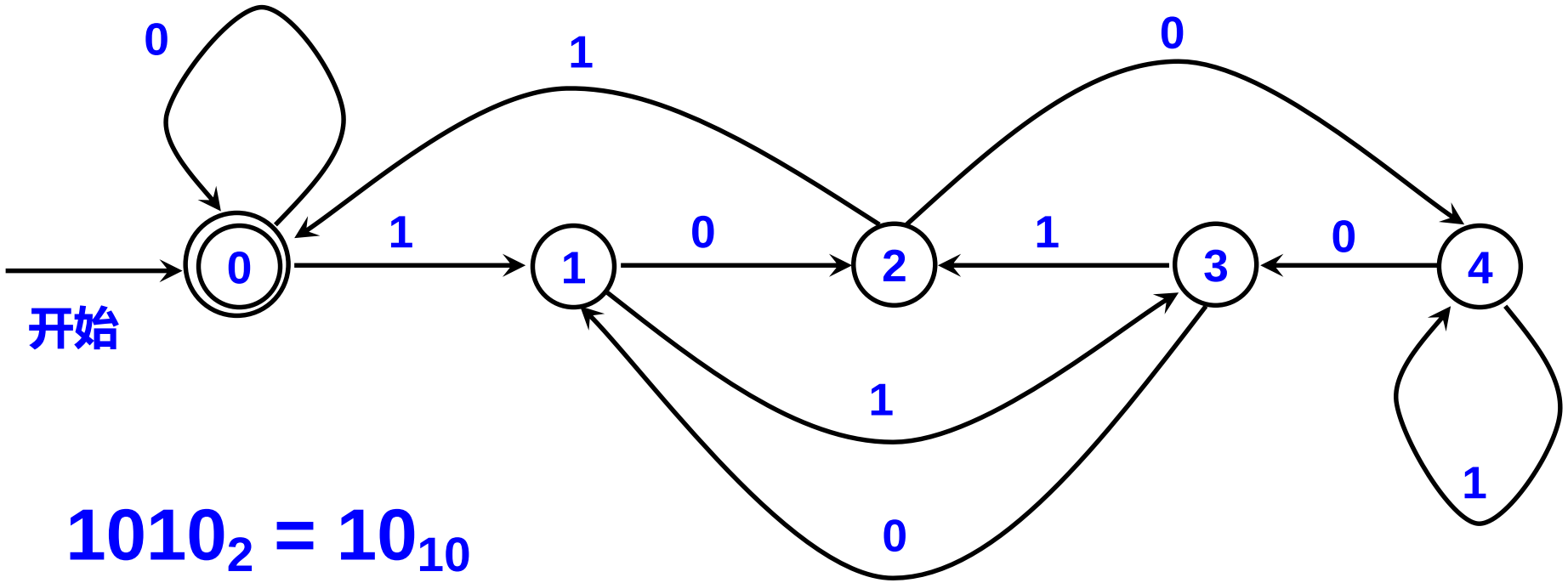
- 确定的有限自动机（简称DFA)也是一个数学模型，包括：
  - 有限的状态集合S
  - 输入符号集合$\sum$
  - 转换函数$move: S \times \sum \to S$，且可以是部分函数
  - 状态s_0是唯一的开始状态
  - $F \subseteq S$是接受状态集合

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

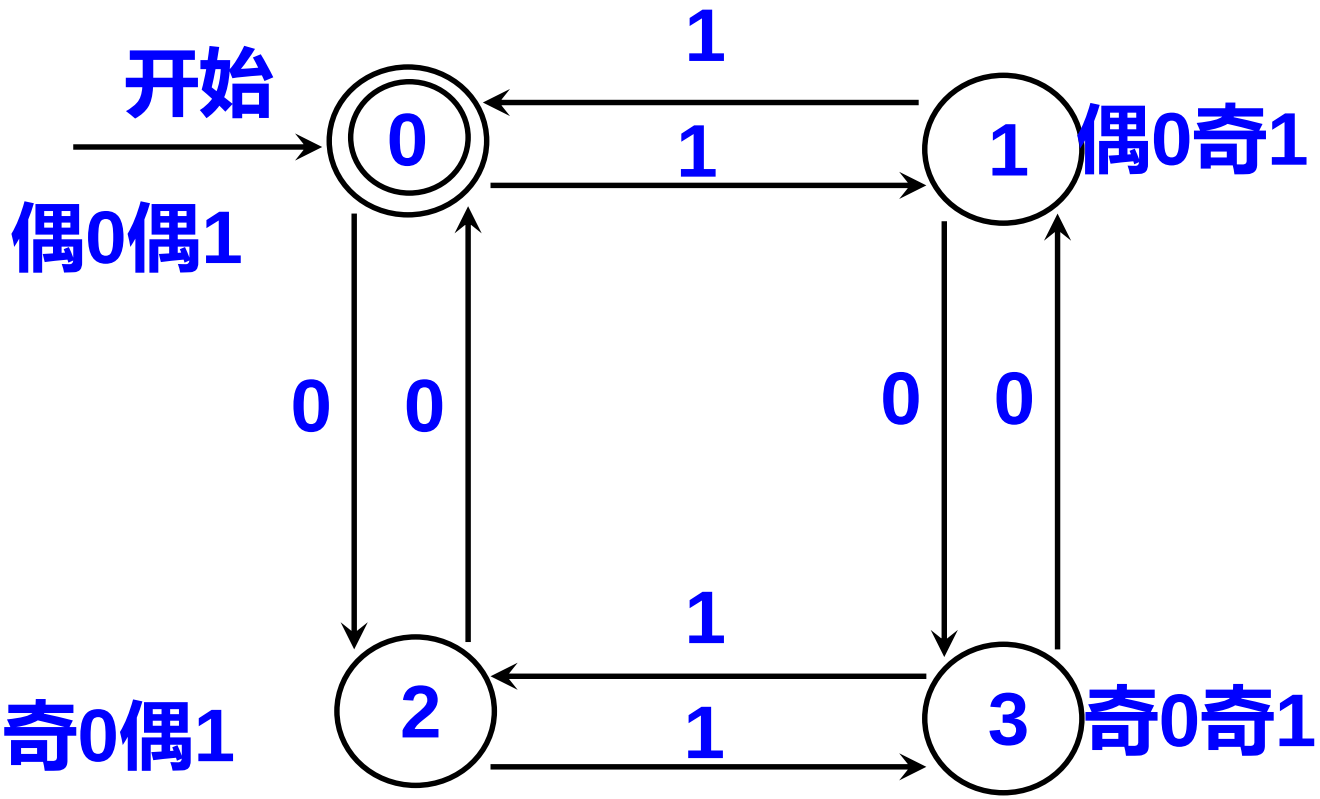
❑ 例 DFA，识别{0,1}上能被5整除的二进制数



$1010_2 = 10_{10}$

$111_2 = 7_{10}$

- 例DFA,接受 0和1的个数都是偶数的字符串

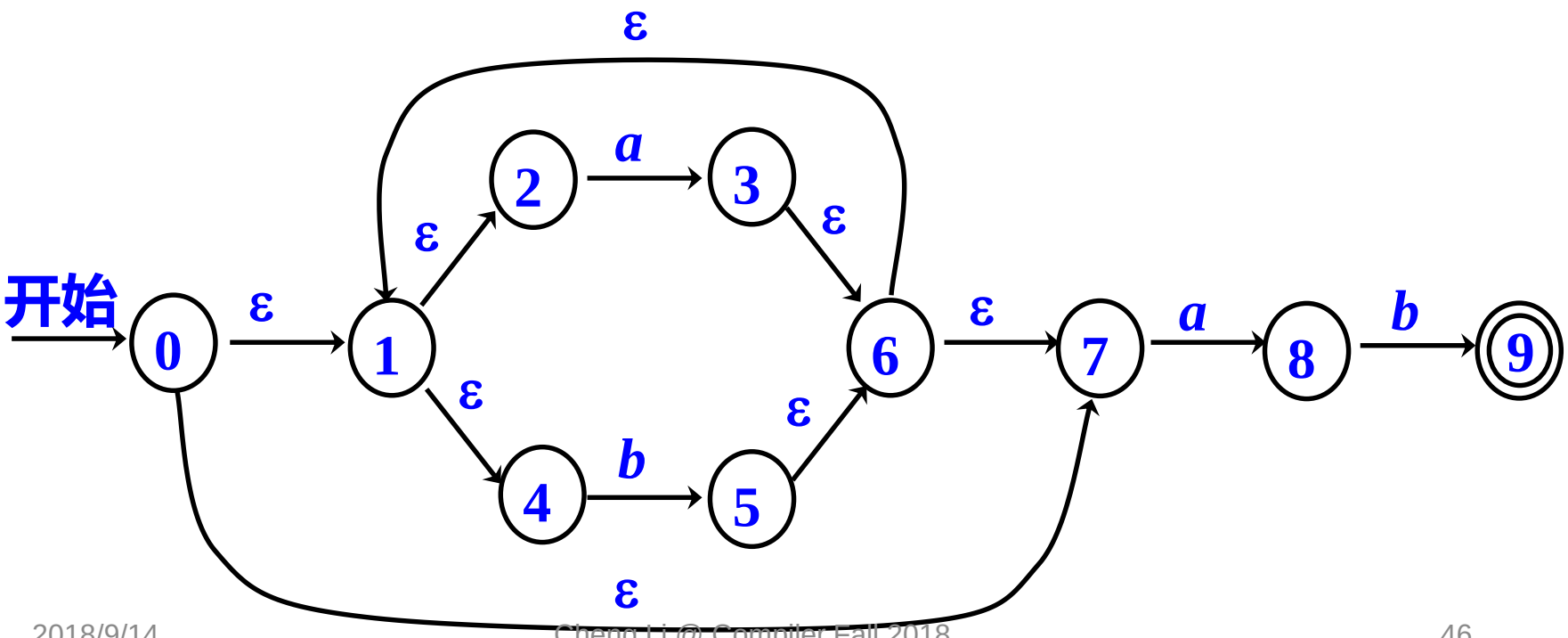


- NFA到DFA的变换
- 子集构造法
  - DFA的一个状态是NFA的一个状态集合
  - 读了输入$a_1\ a_2 \ldots a_n$后，NFA能到达的所有状态：$s_1, s_2, \ldots, s_k$，则DFA到达状态$\{s_1, s_2, \ldots, s_k\}$

❑ 子集构造法(subset construction)

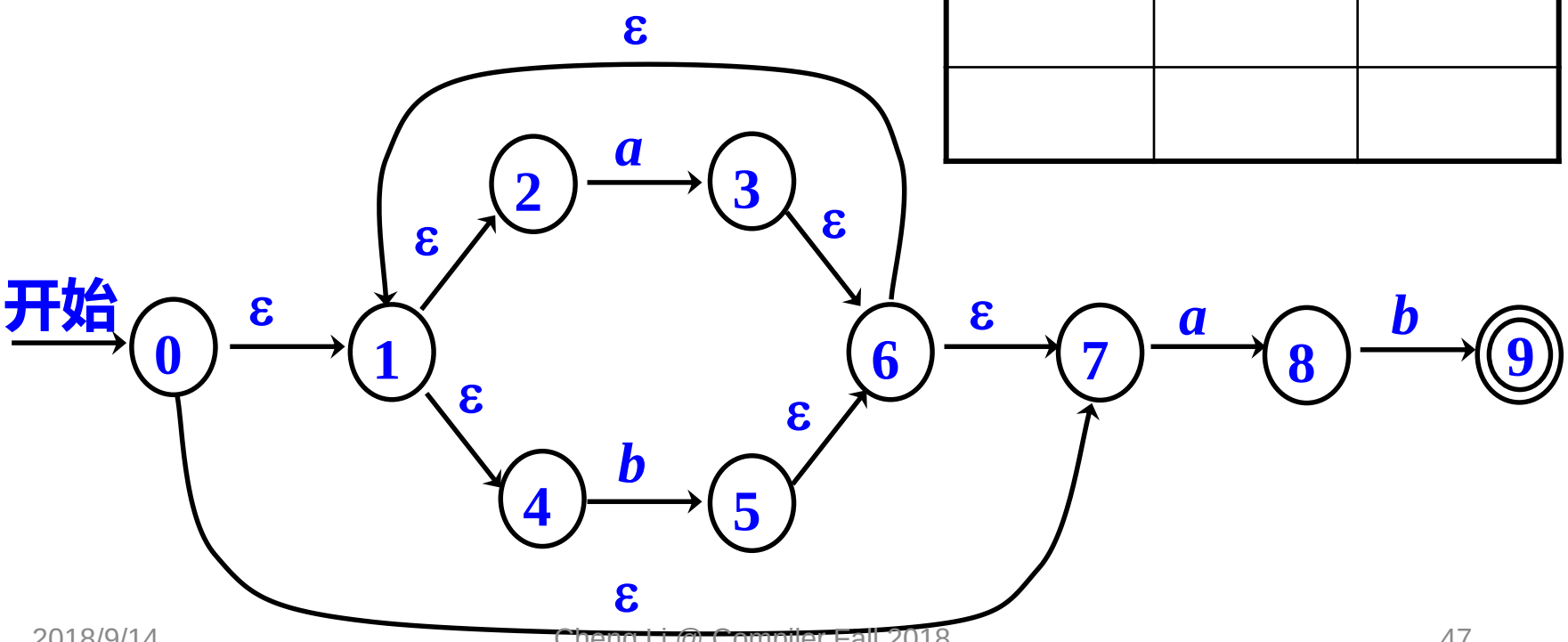
- ε-闭包（ε- closure）：状态s 的ε-闭包是 s 经 ε转换所能到达的状态集合
- NFA的初始状态的 ε-闭包对应于DFA的初始状态
- 针对每个DFA 状态 – NFA状态子集A，求输入每个a_i 后能到达的NFA状态的ε-闭包并集（ $\varepsilon\text{- closure}(move(A, a_i))$），该集合对应于DFA中的一个已有状态，或者是一个要新加的DFA状态

- 例(a|b)*ab，NFA如下，把它变换为DFA



NFA到DFA的变换

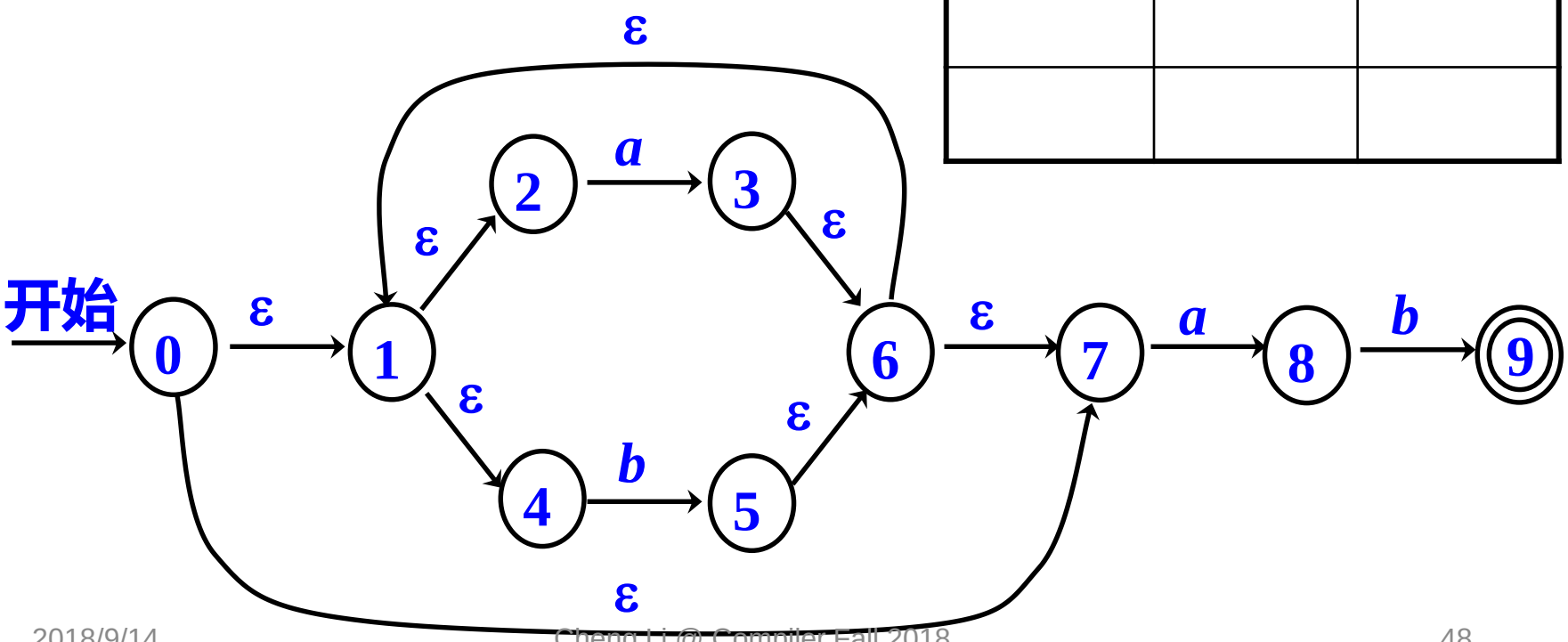
状态	输入符号	
	a	*b*



NFA到DFA的变换

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

状态	输入符号	
	a	*b*
A		

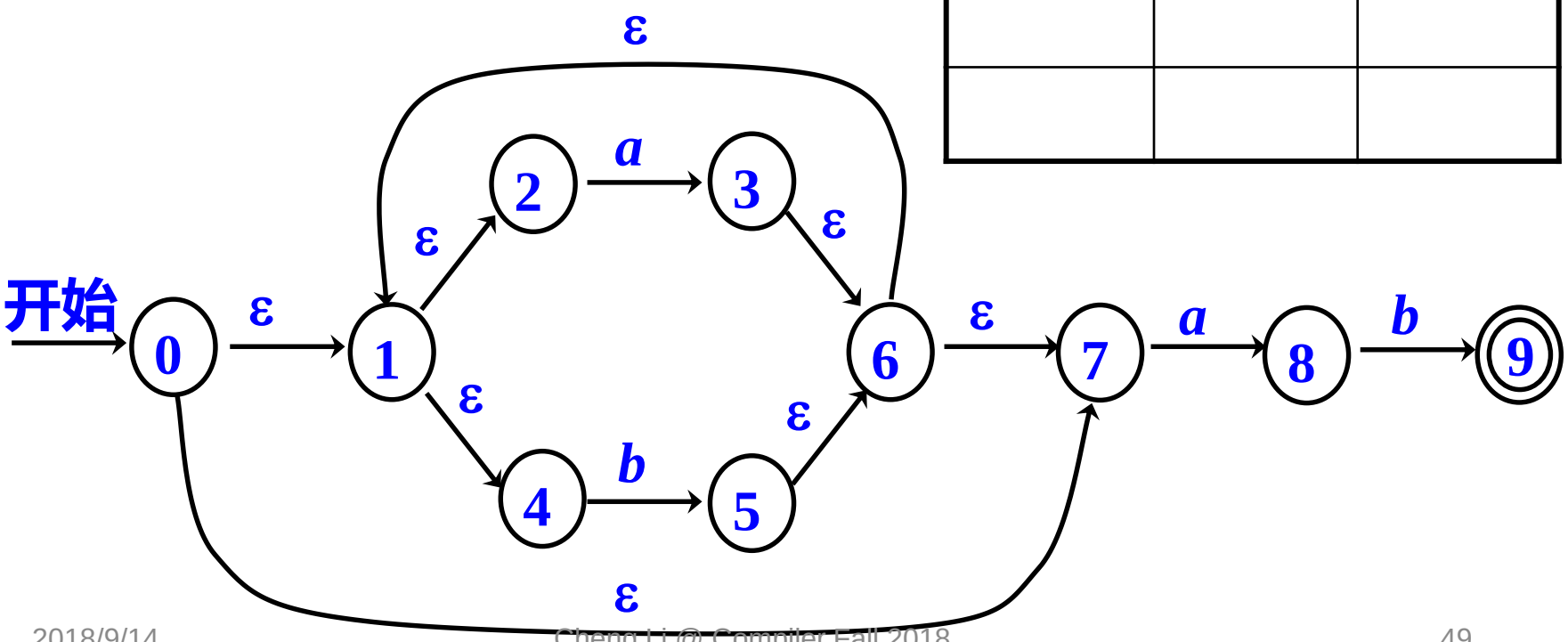


NFA到DFA的变换

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

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

状态	输入符号	
	a	b
A	B	



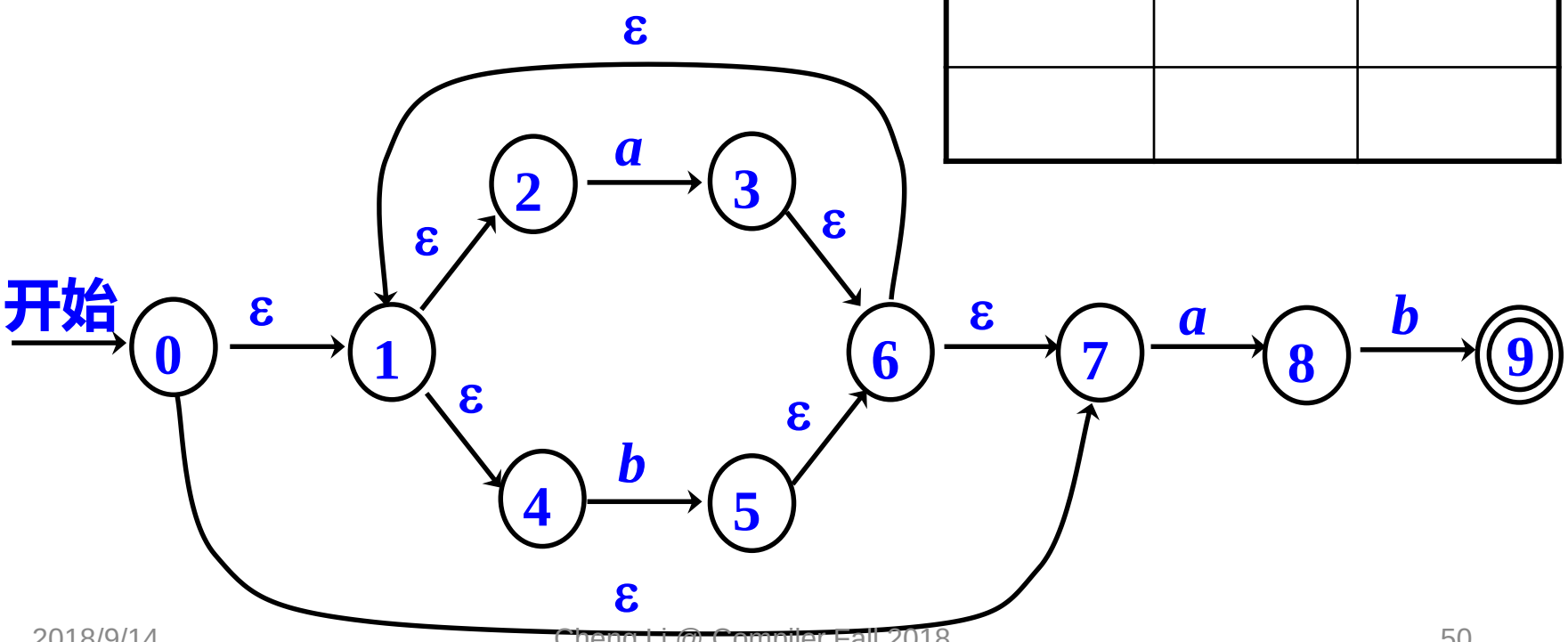
NFA到DFA的变换

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

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

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

状态	输入符号	
	a	b
A	B	C



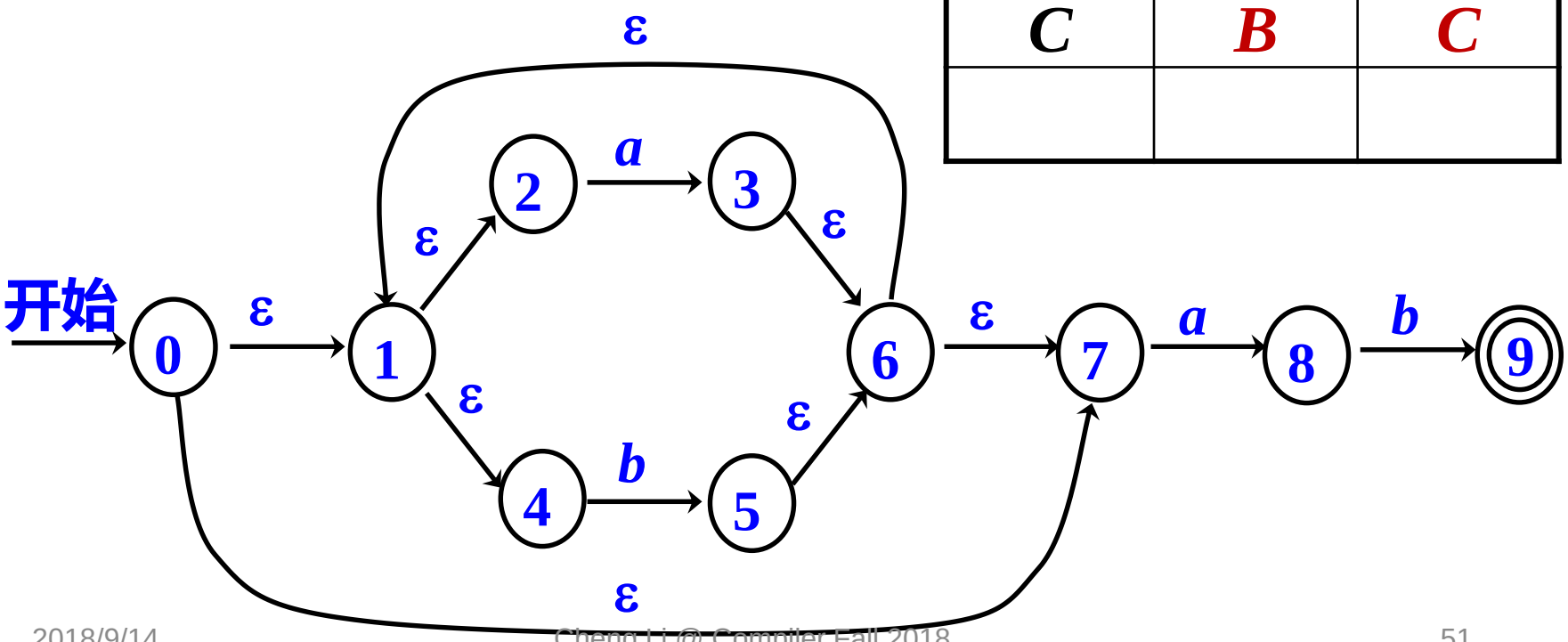
NFA到DFA的变换

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

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

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

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



NFA到DFA的变换

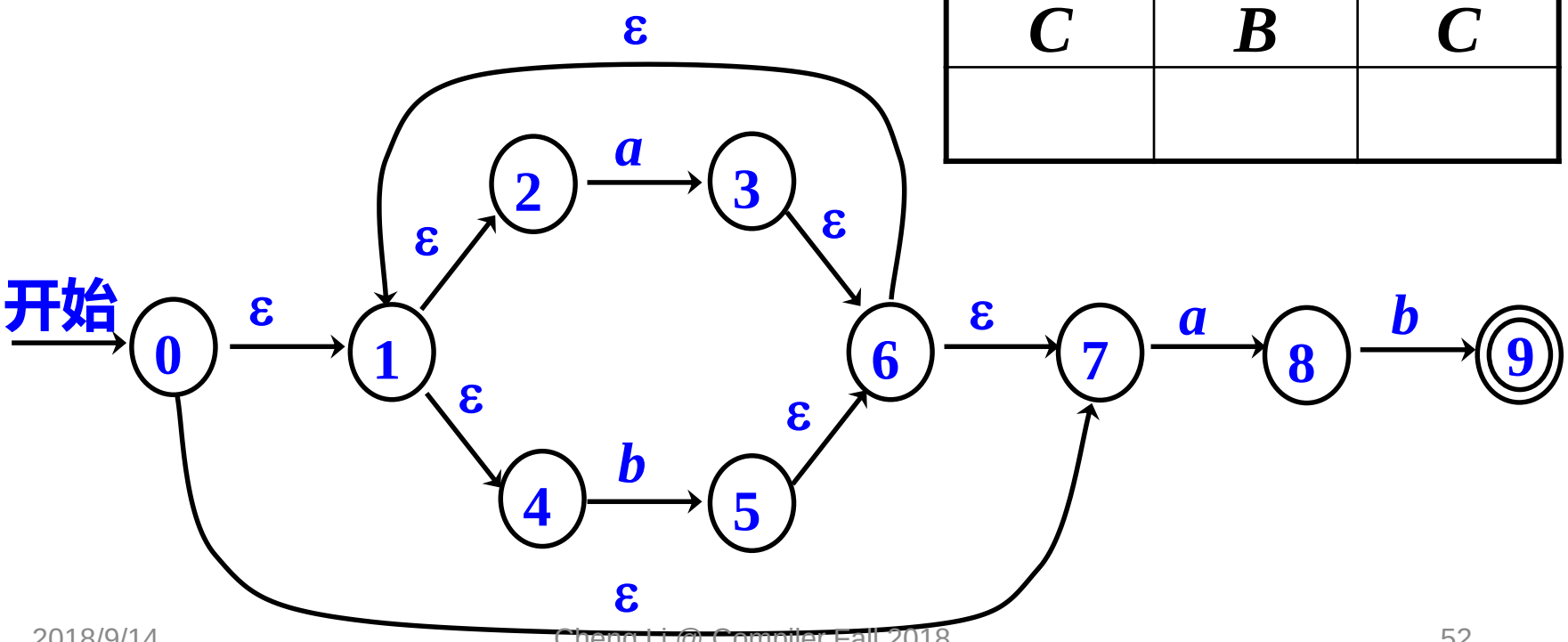
$A = \{0, 1, 2, 4, 7\}$

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

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

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

状态	输入符号	
	a	b
A	B	C
B	B	
C	B	C



NFA到DFA的变换

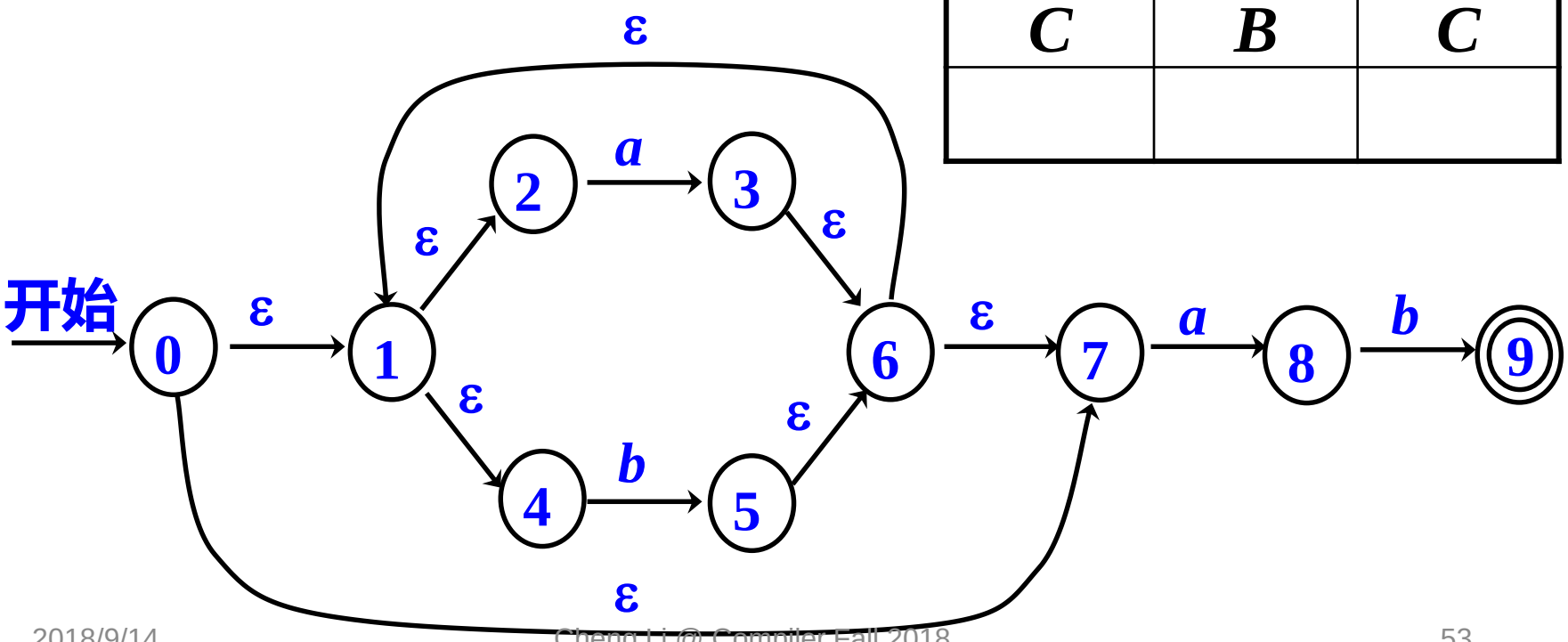
$A = \{0, 1, 2, 4, 7\}$

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

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

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

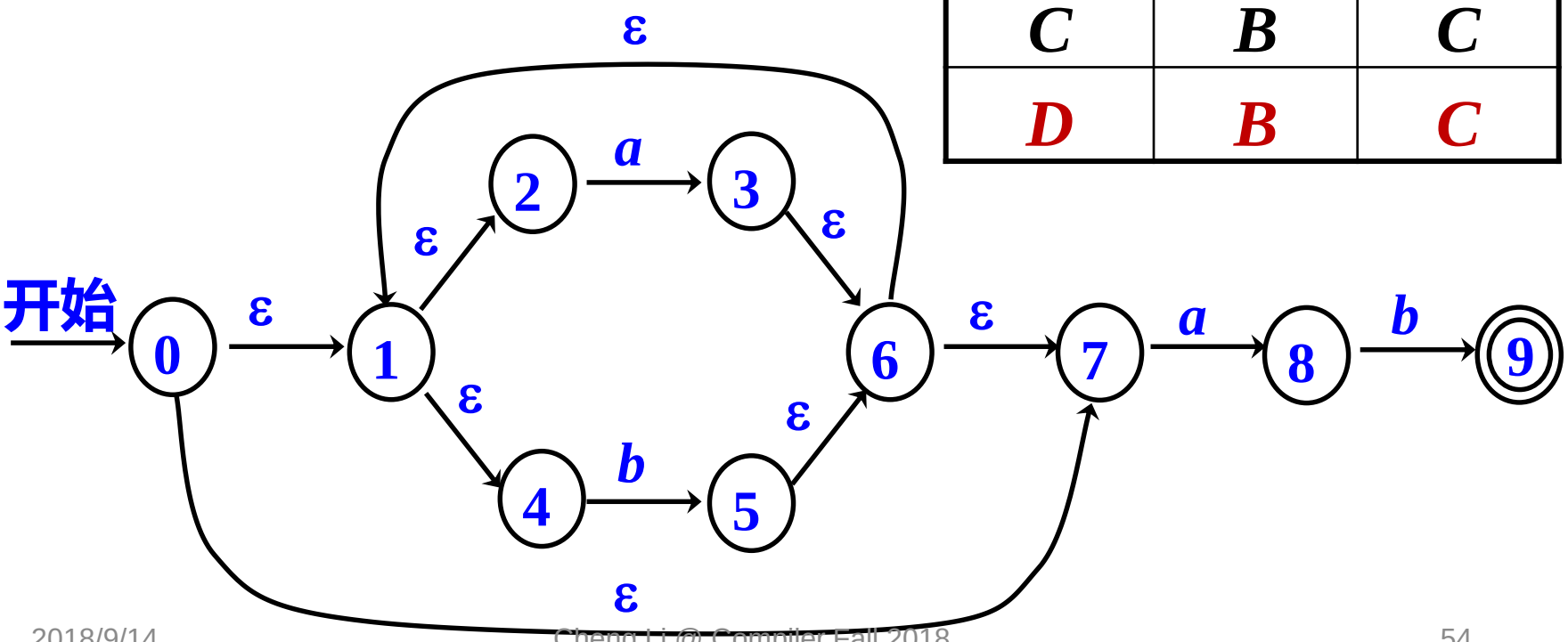
状态	输入符号	
	a	b
A	B	C
B	B	D
C	B	C



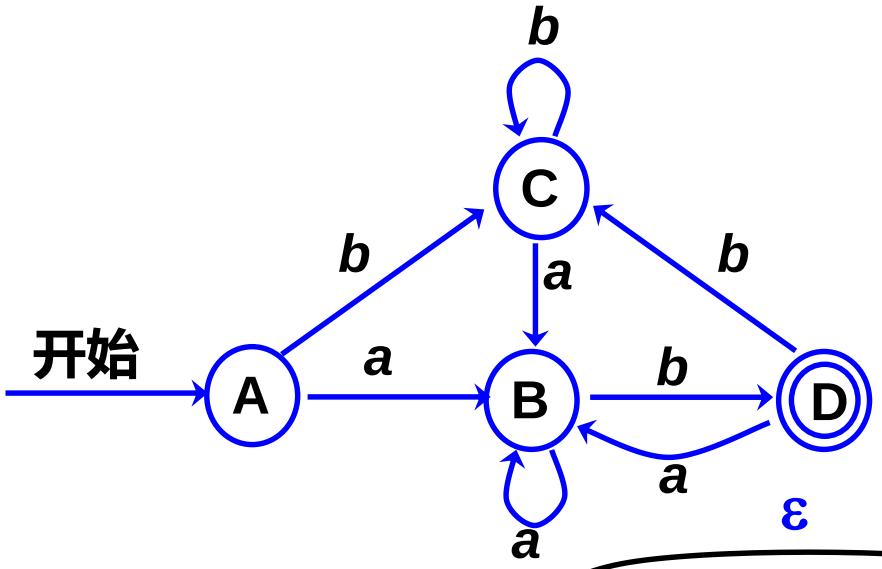
NFA到DFA的变换

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

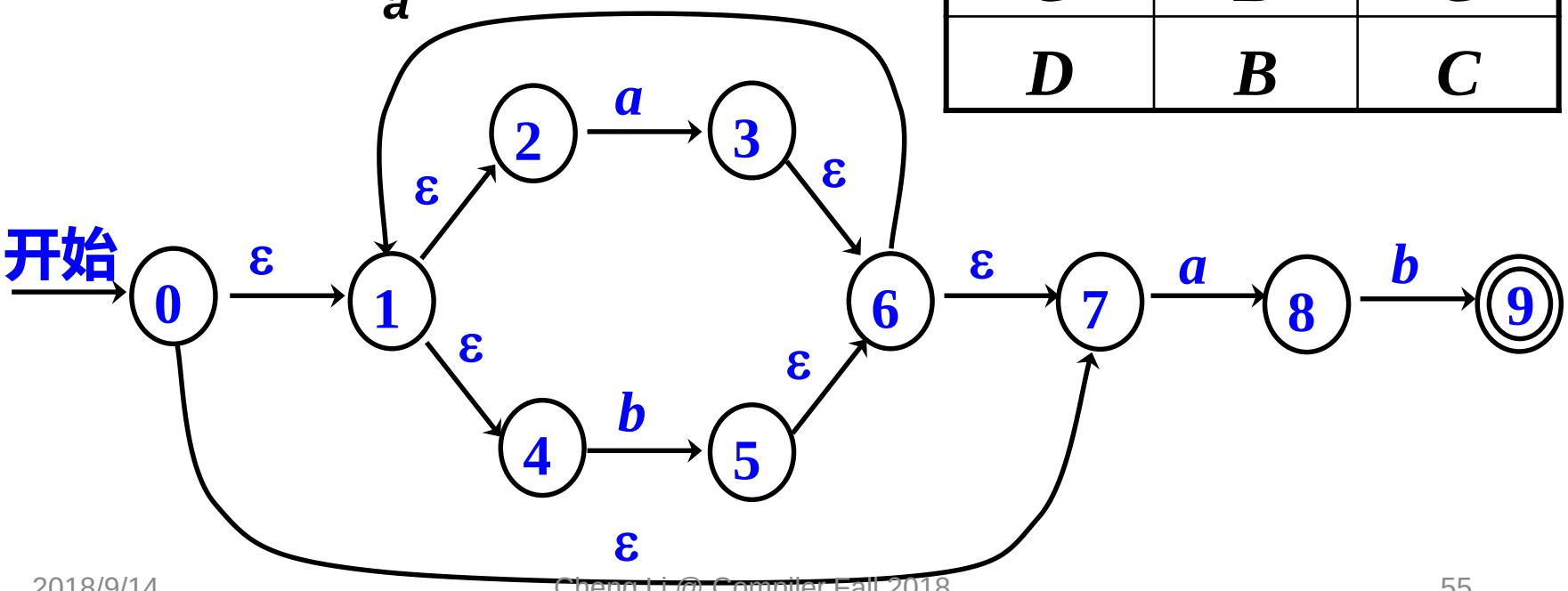
状态	输入符号	
	a	b
A	B	C
B	B	D
C	B	C
D	B	C



NFA到DFA的变换

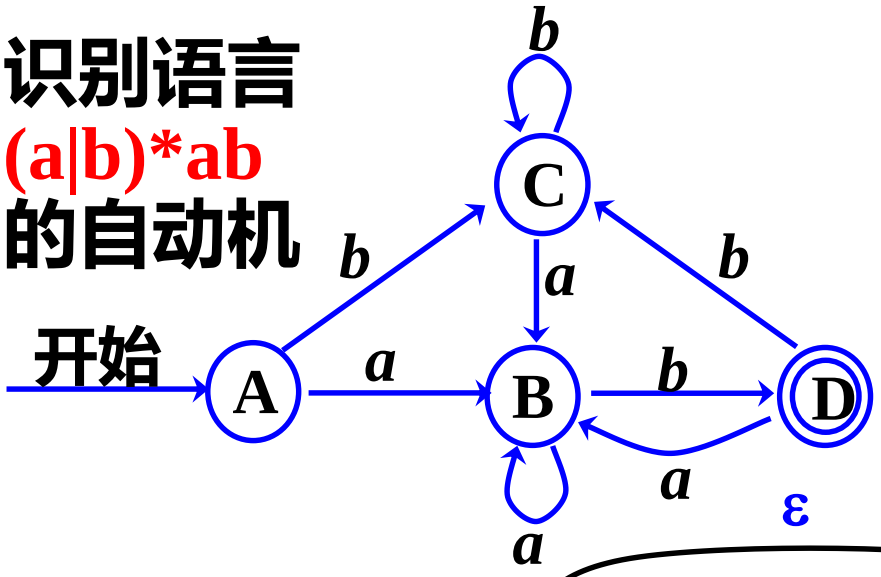


状态	输入符号	
	a	*b*
A	*B*	*C*
B	*B*	*D*
C	*B*	*C*
D	*B*	*C*

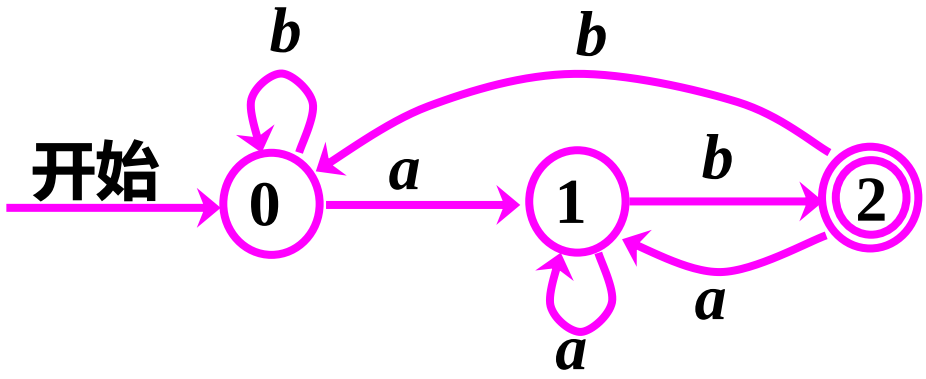


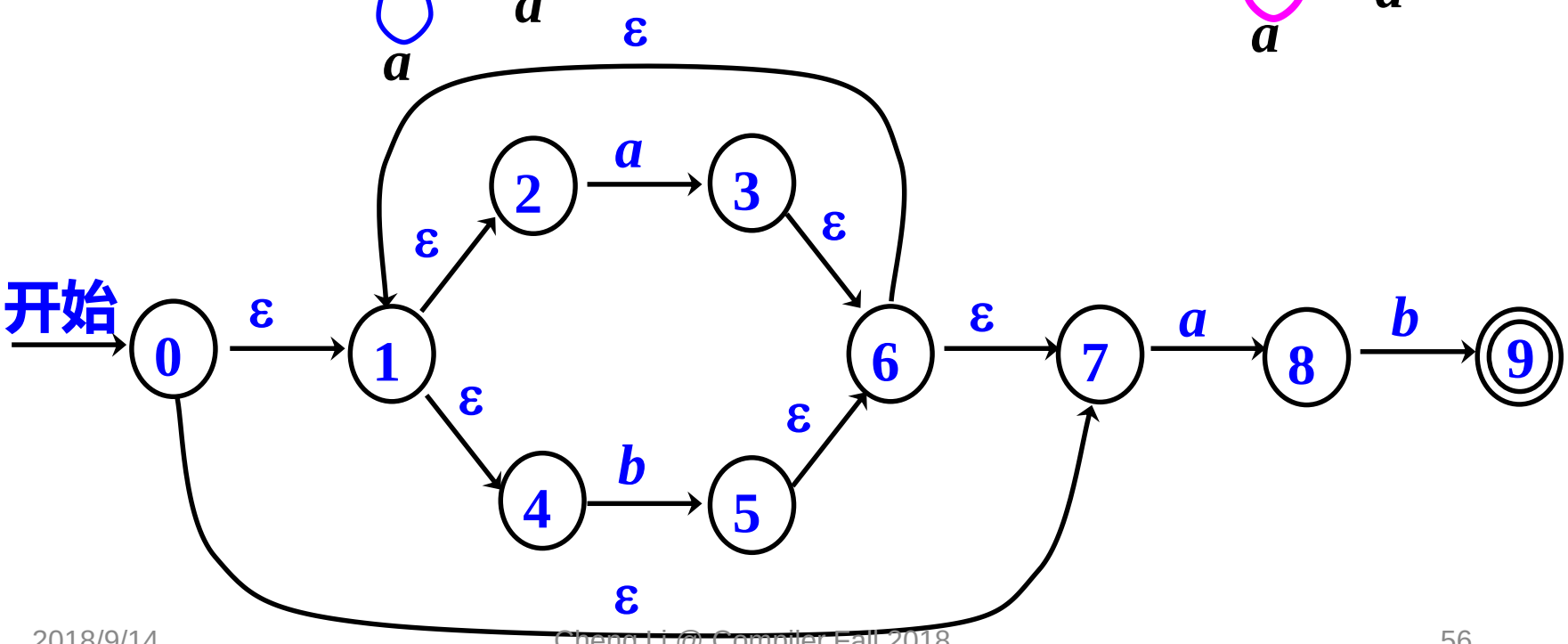
NFA到DFA的变换

识别语言
(a|b)*ab
的自动机



子集构造法不一定得到最简DFA

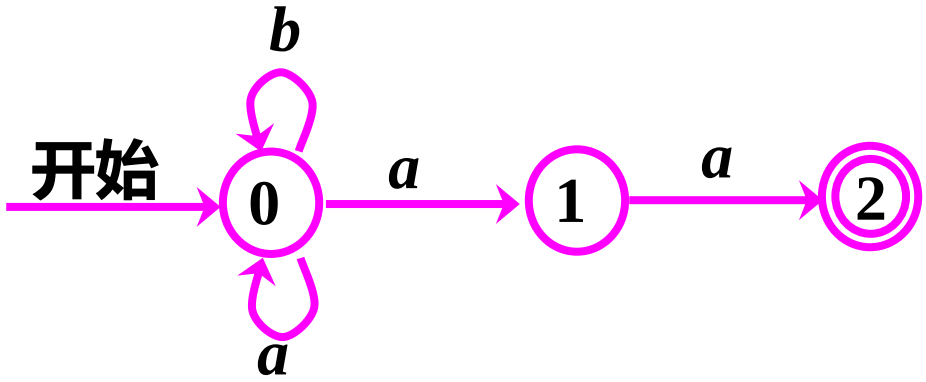




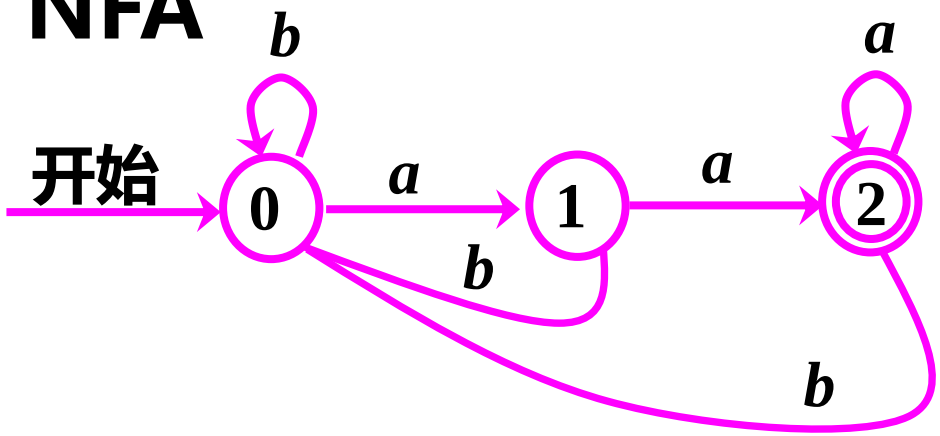
NFA vs. DFA

- NFAs and DFAs recognize the same set of languages (regular languages)
- DFAs are faster to execute
  - There are no choices to consider

- For a given language NFA can be simpler than DFA

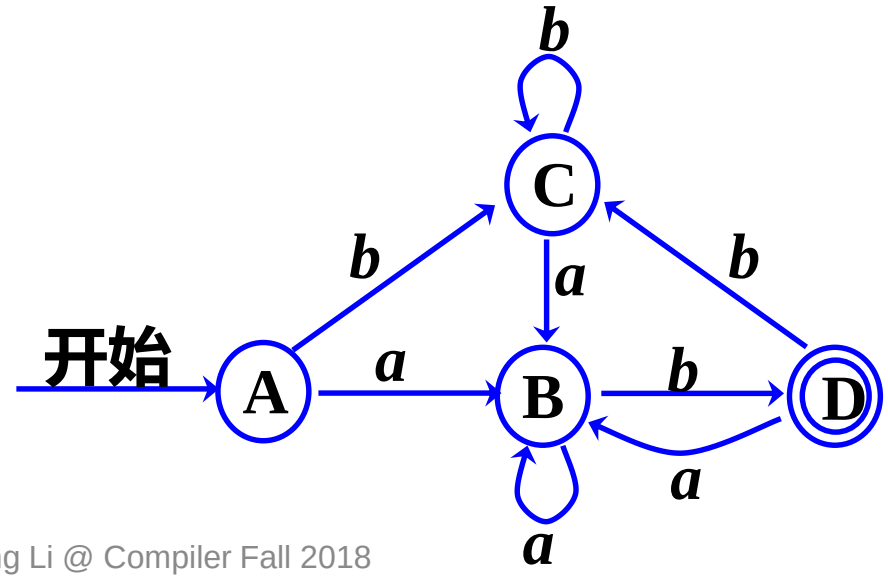


- DFA can be exponentially larger than NFA



- 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

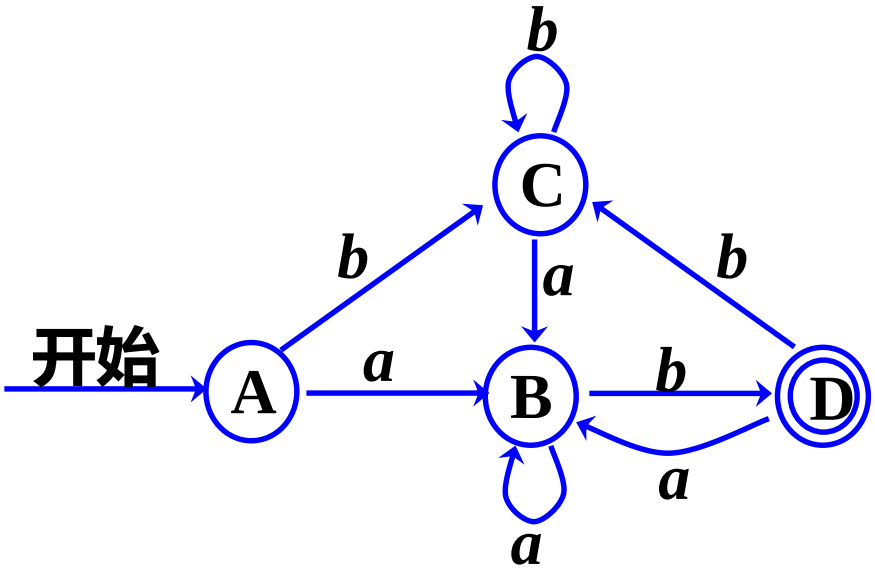


1. 按是否是接受状态来区分

   {A, B, C}, {D}

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

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



1. 按是否是接受状态来区分

   {A, B, C}, {D}

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

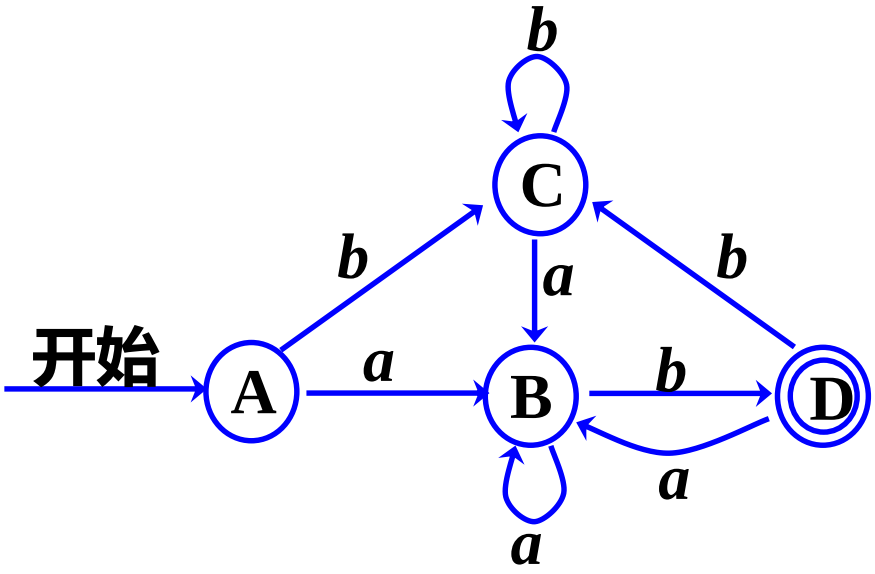
   move({A, B, C}, b) = {C, D}

2. 继续分解

   {A, C}, {B}, {D}

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

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



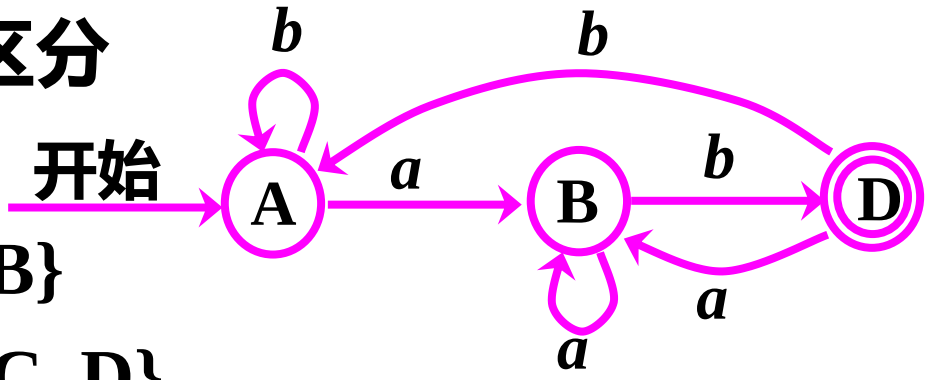
DFA的化简

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}

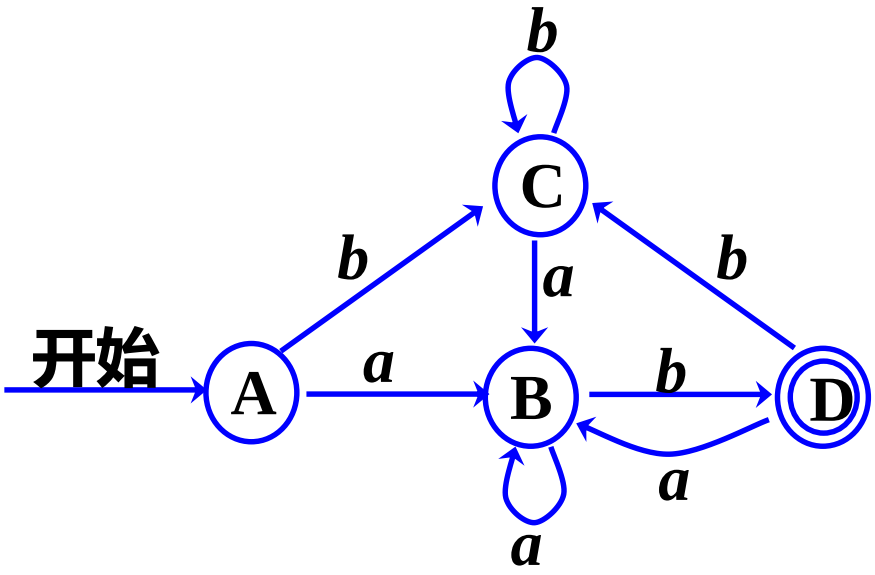


2. 继续分解

   {A, C}, {B}, {D}

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

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



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

USTC, CHINA
ADSLAB
先进数据系统实验室

《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