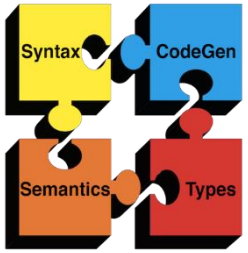
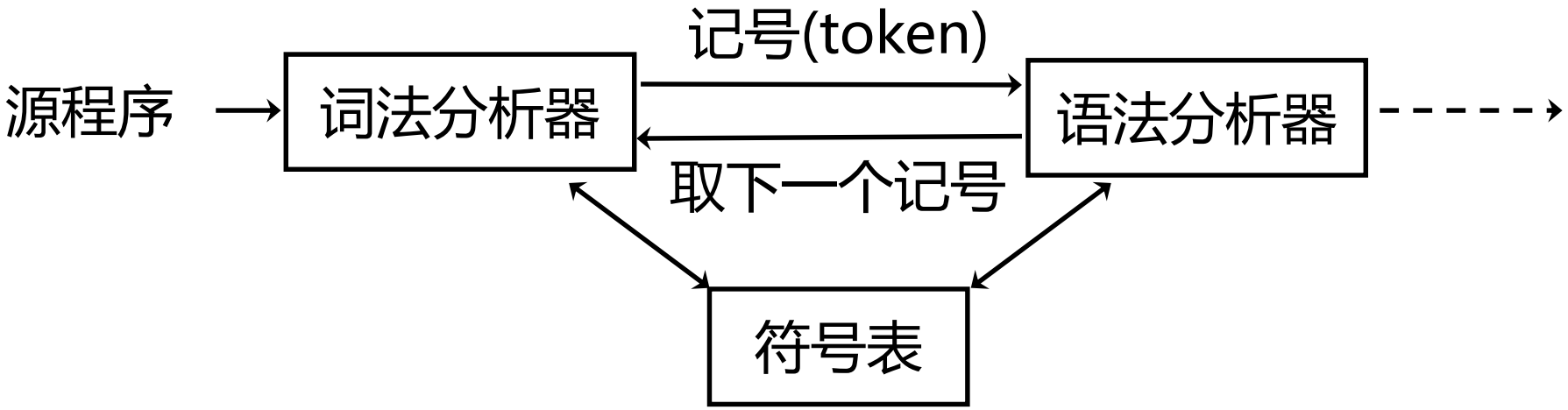
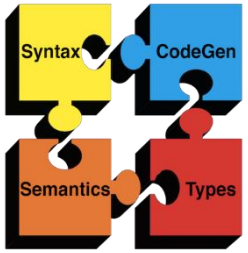


编译原理与技术

第2章 词法分析



- 词法分析器：把构成源程序的字符流翻译成记号流，还完成和用户接口的一些任务
- 围绕词法分析器的自动生成展开
- 介绍正规式、状态转换图和有限自动机概念



第2章 词法分析

2.1 词法记号及属性

2.1.1 词法记号、模式、词法单元

记号名	词法单元例举	模式的非形式描述
if	if	字符i, f
for	for	字符f, o, r
relation	< , <= , = , ...	< 或 <= 或 = 或 ...
id	sum, count, D5	由字母开头的字母数字串
number	3.1, 10, 2.8E12	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

- 历史上词法定义中的一些问题
  - 忽略空格带来的困难

    DO 8 I = 3. 75 等同于 DO8I=3. 75

    DO 8 I = 3, 75

  - 关键字不保留

    IF THEN THEN THEN=ELSE; ELSE …

- 关键字、保留字和标准标识符的区别
  - 保留字是语言预先确定了含义的词法单元
  - 标准标识符也是预先确定了含义的标识符，但程序可以重新声明它的含义

2.1.2 词法记号的属性

position = initial + rate * 60的记号和属性值：

<**id**，指向符号表中position条目的指针>

<**assign_op**>

<**id**，指向符号表中initial条目的指针>

<**add_op**>

<**id**，指向符号表中rate条目的指针>

<**mul_ op**>

<**number**，整数值60>

2.1.3 词法错误

- 词法分析器对源程序采取非常局部的观点
  - 例：难以发现下面的错误

    fi (a == f (x)) ...
- 在实数是“数字串.数字串”格式下，可以发现下面的错误

  123.x
- 紧急方式的错误恢复
  - 删掉当前若干个字符，直至能读出正确的记号
- 错误修补
  - 进行增、删、替换和交换字符的尝试

下面C语言编译器编译下面的函数时，报告

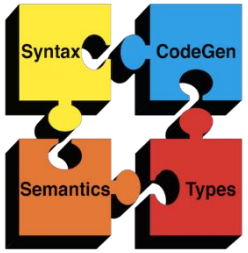
parse error before 'else'

```
long gcd(p,q)
long p,q;
{
        if (p%q == 0)
                /* then part */
                return q
        else
                /* else part */
                return gcd(q, p%q);
}
```

此处遗漏分号

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);
}
```



第2章 词法分析

2.2 词法记号的描述与识别

2.2.1 串和语言

- 字母表：符号的有限集合，例：Σ = { 0, 1}
- 串：符号的有穷序列，例：0110, ε
- 语言：字母表上的一个串集
  {ε, 0, 00, 000, ...}，{ε}，∅
- 句子：属于语言的串

- 串的运算
  - 连接（积）　xy，sε = εs = s
  - 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{ε}，L^i是$L^{i-1}L$
  - 闭包: $L^* = L^0 \cup L^1 \cup L^2 \cup \ldots$
  - 正闭包: $L^+ = L^1 \cup L^2 \cup \ldots$

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

2.2.2 正规式

正规式用来表示简单的语言，叫做正规语言或正规集

正规式	定义的语言	备注
ε	{ε}	
a	{a}	$a \in \Sigma$
(r) | (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
(r)	L(r)	r是正规式

((a) $(b)^*$)| (c)可以写成ab^*| c

- 正规式的例子 Σ = {a, b}
  - a | b {a, b}
  - (a | b) (a | b) {aa, ab, ba, bb}
  - aa | ab | ba | bb {aa, ab, ba, bb}
  - a* 由字母a构成的所有串集
  - (a | b)* 由a和b构成的所有串集

- 复杂的例子
  - (00 | 11 | ((01 | 10) (00 | 11) * (01 | 10))) *
  - 句子：01001101000010000010111001
  - 描述：0和1的个数都是偶数的01串

2.2.3 正规定义

- 对正规式命名，使表示简洁

$d_1 \rightarrow r_1$

$d_2 \rightarrow r_2$

. . .

$d_n \rightarrow r_n$

各个 d_i 的名字都不同

每个 r_i 都是 $\Sigma \cup \{d_1, d_2, \ldots, d_{i-1}\}$上的正规式

- 正规定义的例子

  - 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)*

- 正规定义的例子

  - 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(+|-)? $digit^{+}$)?

- 正规定义的例子（进行下一步讨论的例子）

  while → while

  do → do

  relop → < | < = | = | < > | > | > =

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

  id → letter (letter | digit)*

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

  delim → blank | tab | newline

  ws → delim+

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

1230313571067980357901 23

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

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

上式包含空串，不符合定义

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

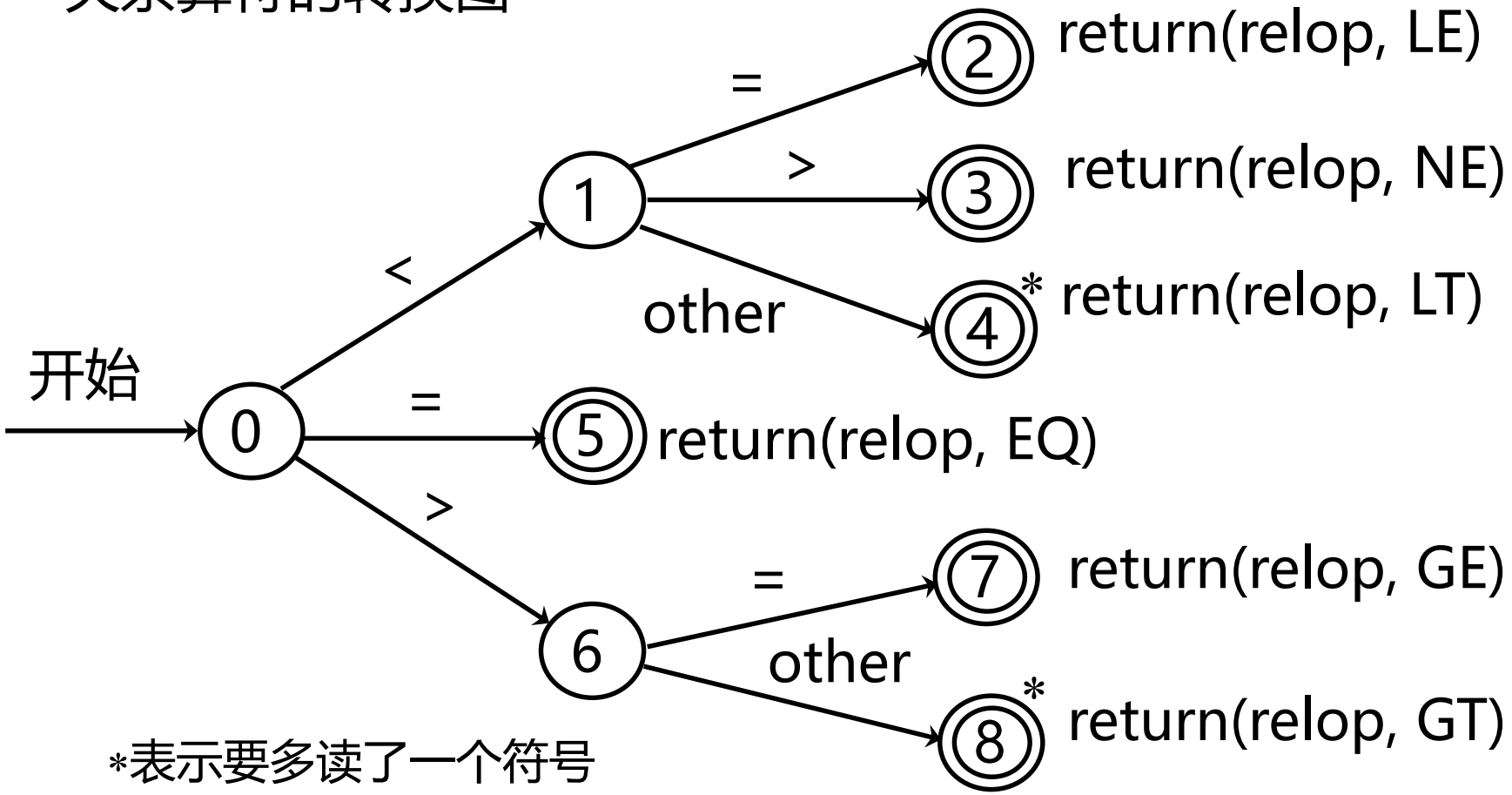
. . .

no_0-8 → 9

将这些正规定义逆序排列就是答案

2.2.4 转换图

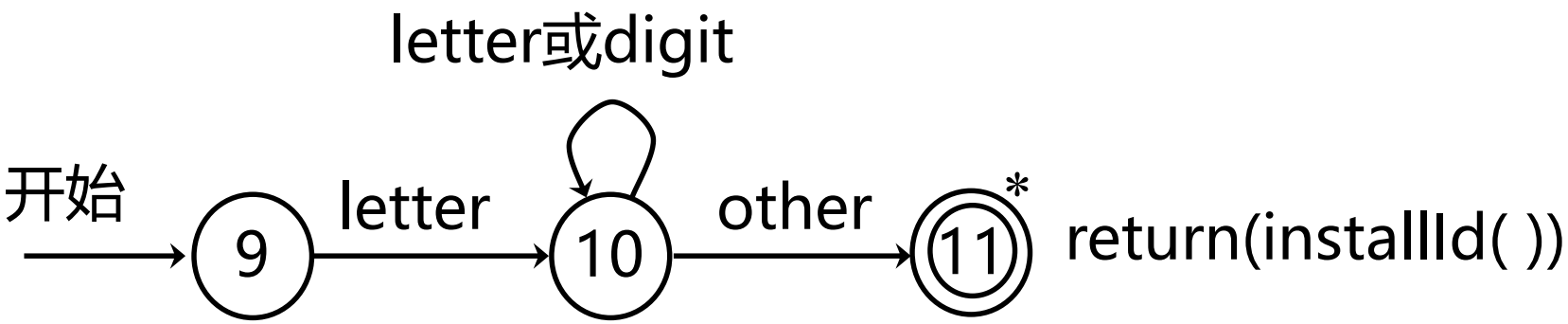
- 关系算符的转换图



*表示要多读了一个符号

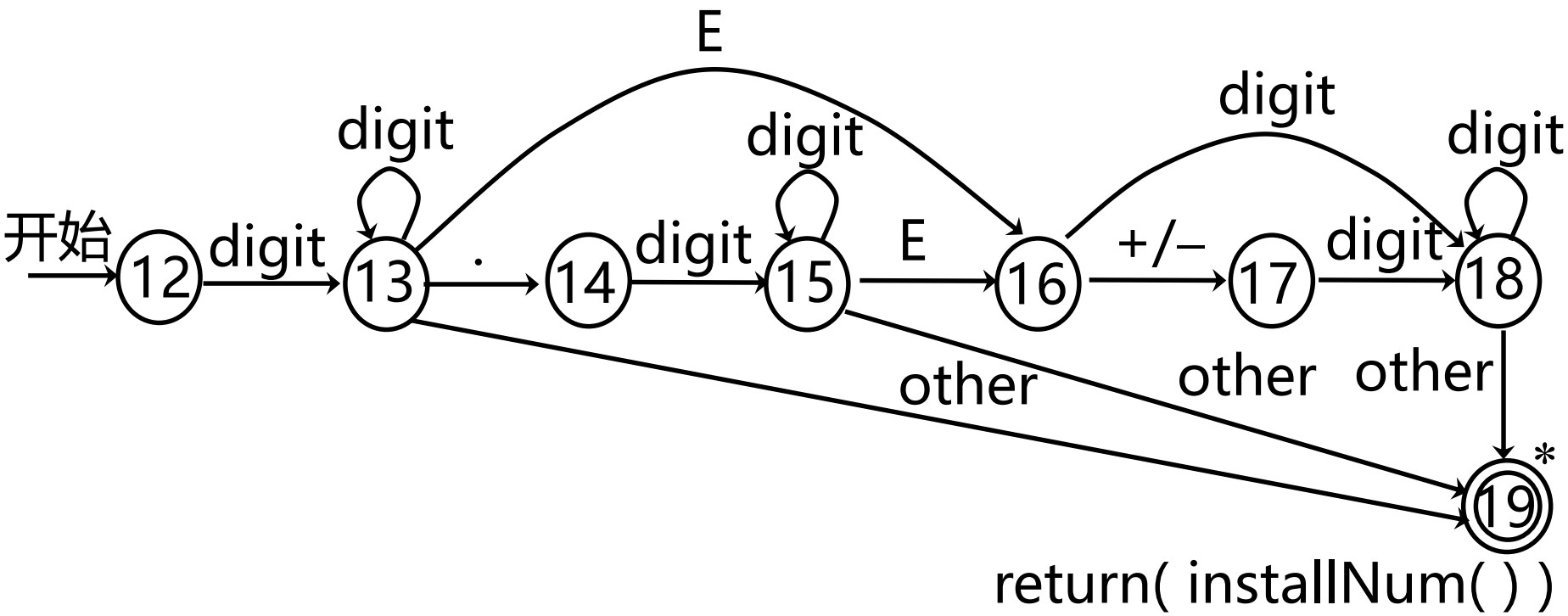
- 标识符和关键字的转换图

id → letter (letter | digit)*



- 无符号数的转换图

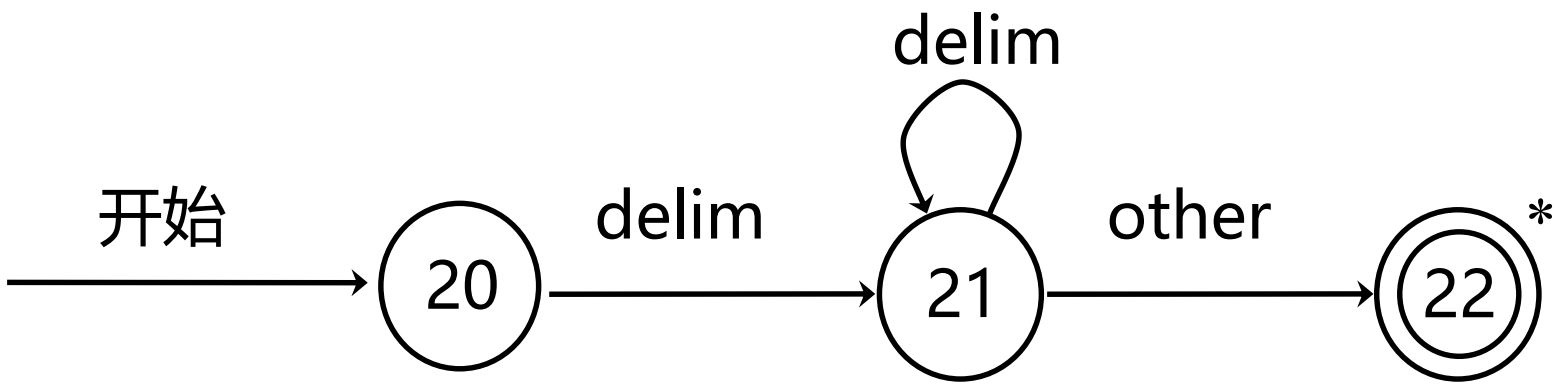
number → digit⁺ (.digit⁺)? (E (+|-)? digit⁺)?

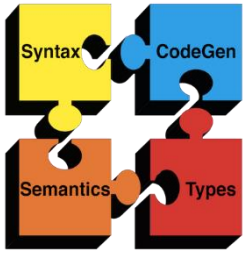


- 空白的转换图

  delim → blank | tab | newline

  ws → $delim^+$





第2章 词法分析

2.3 有限自动机

2.3.1 不确定的有限自动机（简称NFA）

一个数学模型，包括：

1、有限的状态集合S

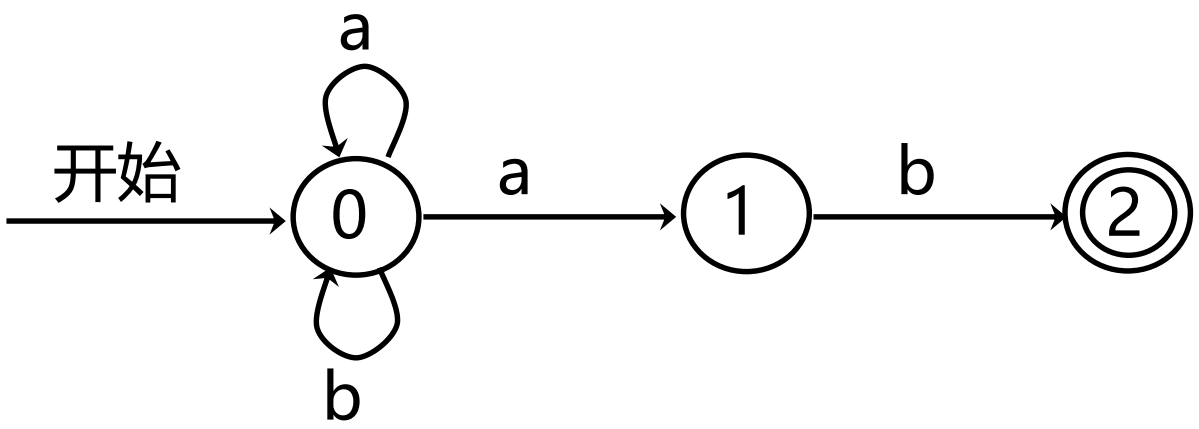
2、输入符号集合Σ

3、转换函数move：$S \times (\Sigma \cup \{\varepsilon\}) \rightarrow P(S)$，P(S)是S的幂集

4、状态s_0是唯一的开始状态

5、$F \subseteq S$是接受状态集合

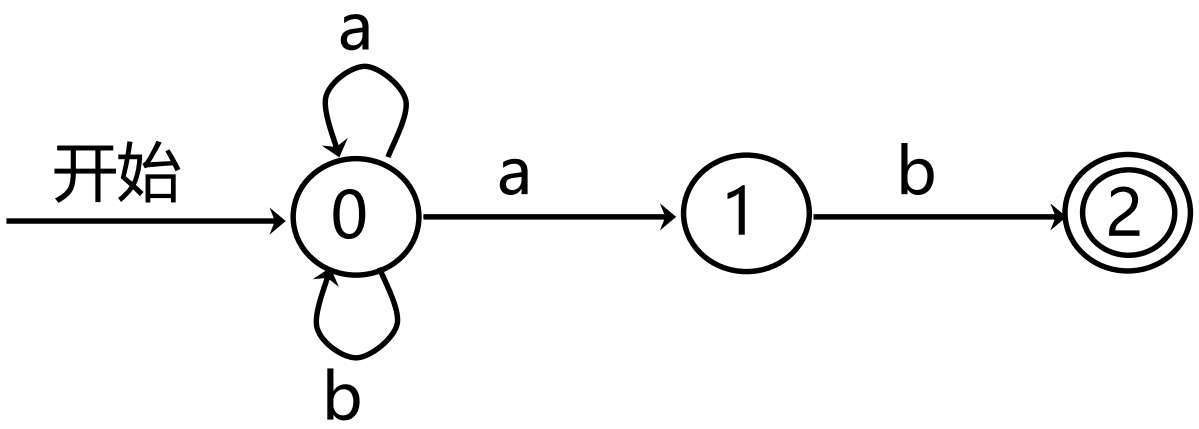
识别语言
$(a|b)^*ab$
的NFA



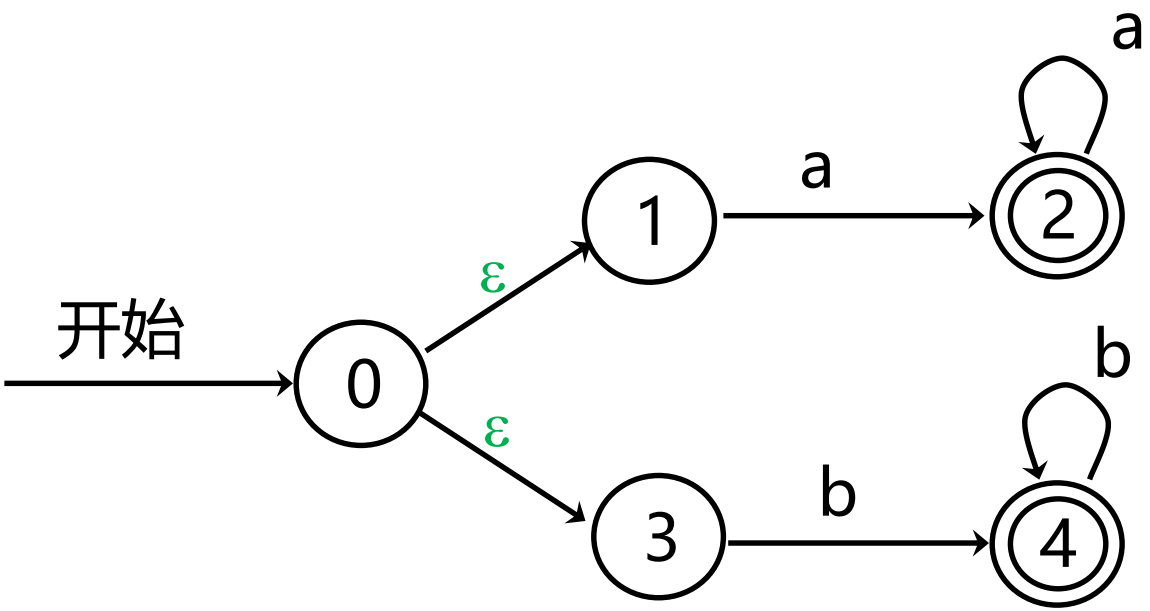
- NFA的转换表
  - 转换函数的一种表示方式

	输 入 符 号	
	a	b
0	{0, 1}	{0}
1	Ø	{2}
2	Ø	Ø

识别语言
$(a|b)^*ab$
的NFA



- 识别$aa^*|bb^*$的NFA



- 2.3.2 确定的有限自动机（简称DFA)

一个数学模型，包括：

1、有限的状态集合S

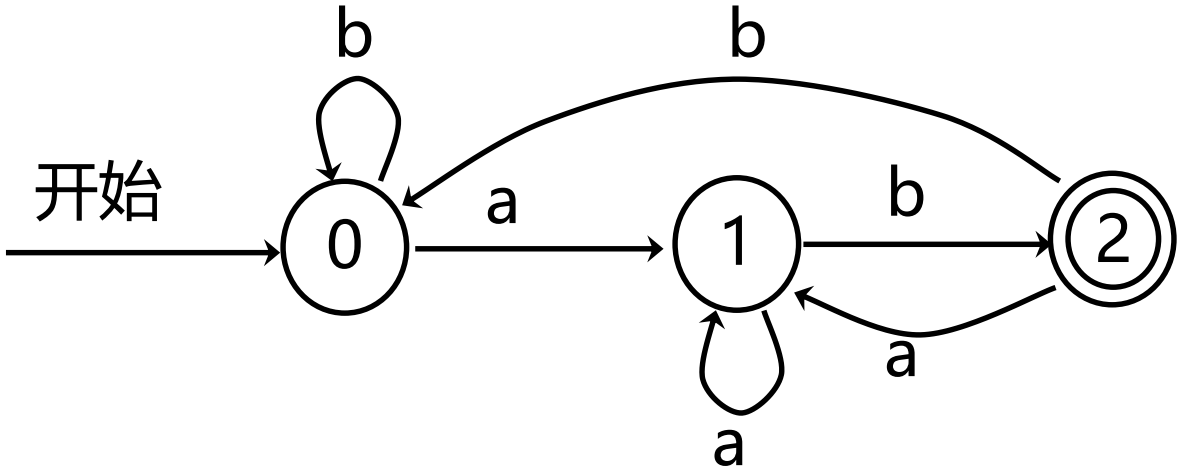
2、输入字母集合Σ

3、转换函数move：S × Σ → S，且可以是部分函数

4、状态s_0是唯一的开始状态

5、F ⊆ S是接受状态集合

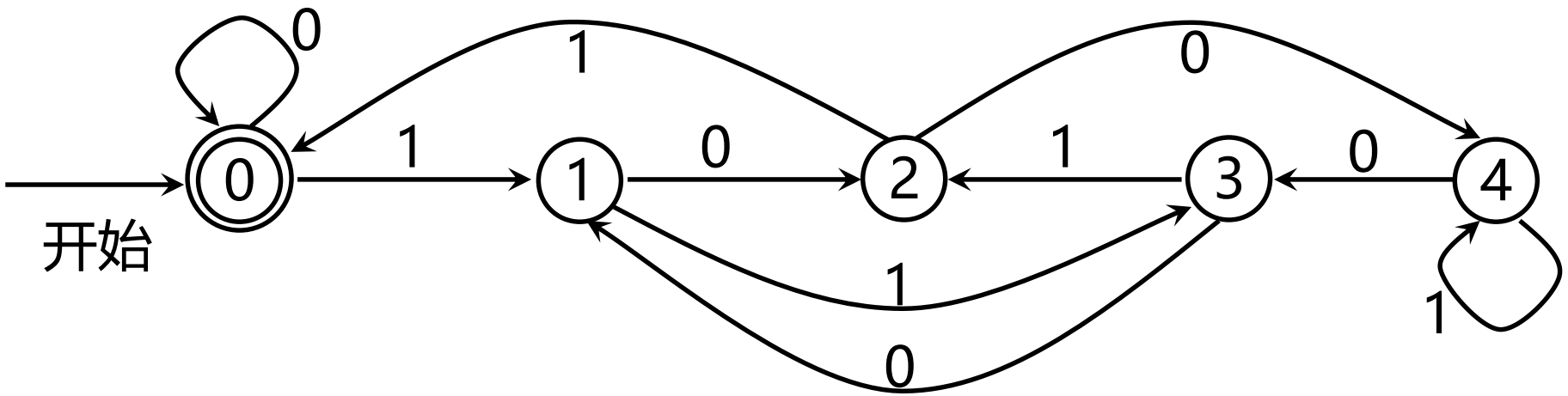
识别语言
(a|b)*ab
的DFA



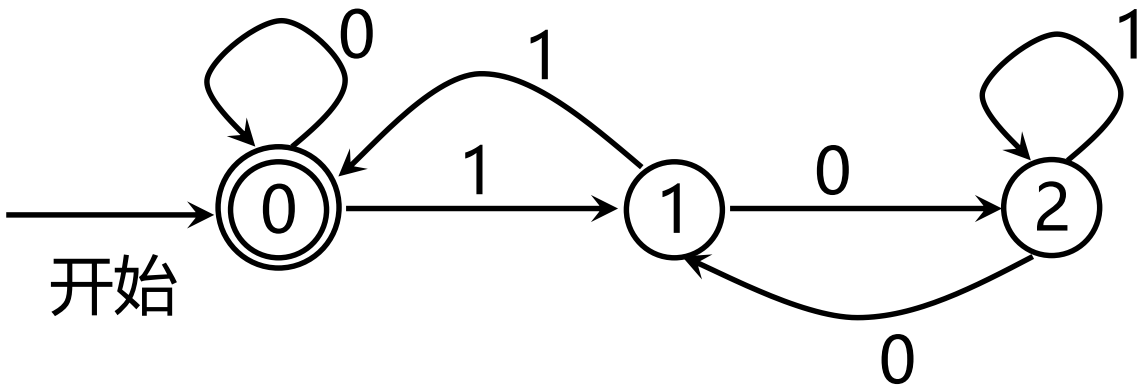
画出DFA，识别{0,1}上能被5整除的二进制数

	已读过	尚未读	已读部分的值
某时刻	101	0111000	5
读进0	1010	111000	5 * 2 = 10
读进1	10101	11000	10 * 2 + 1= 21

5个状态即可，分别代表已读部分的值除以5的余数



画出DFA，识别{0,1}上能被3整除的二进制数



写出正规式

(0 | ….)*

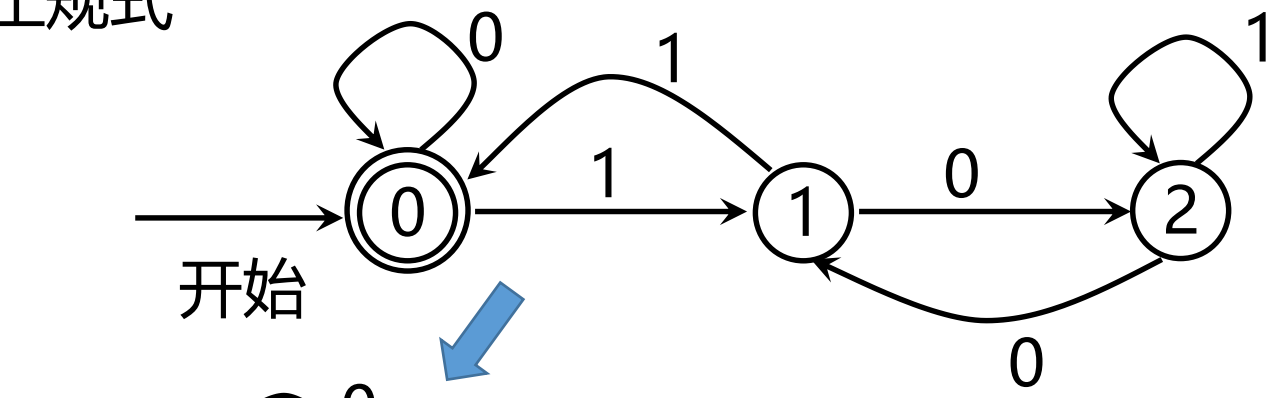
(0 | 1 (ε | ….) 1)*

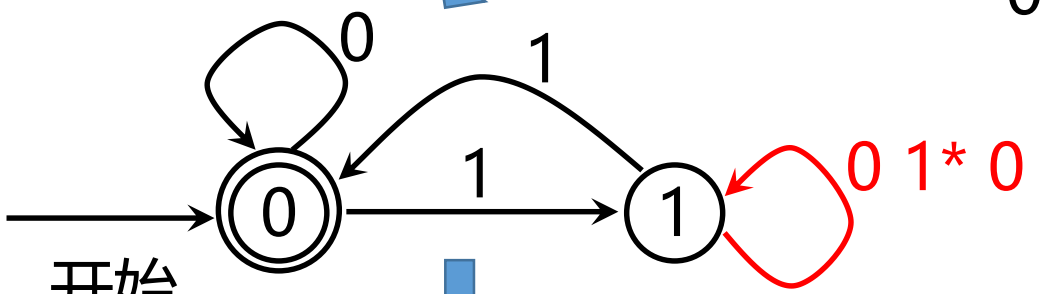
(0 | 1 (0 (ε | ….) 0)* 1)*

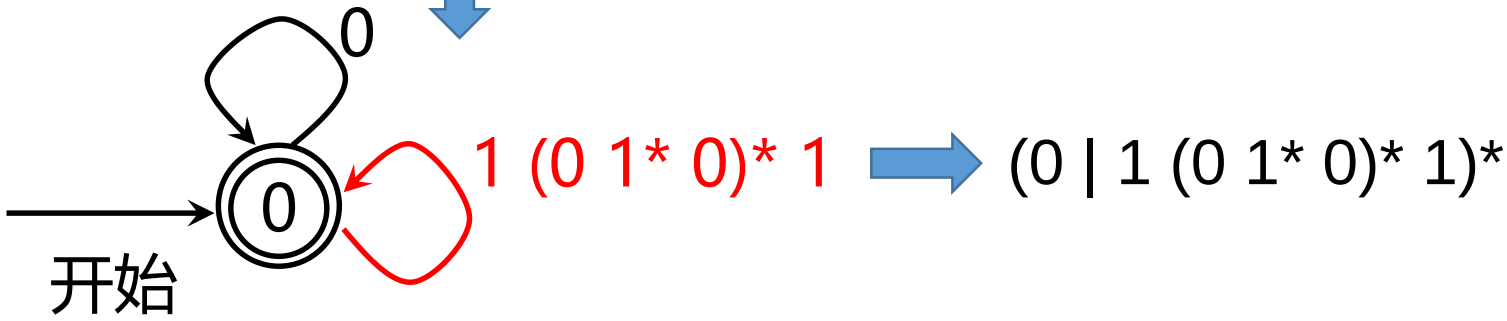
(0 | 1 (0 1* 0)* 1)*

画出DFA，识别{0,1}上能被3整除的二进制数

写出正规式

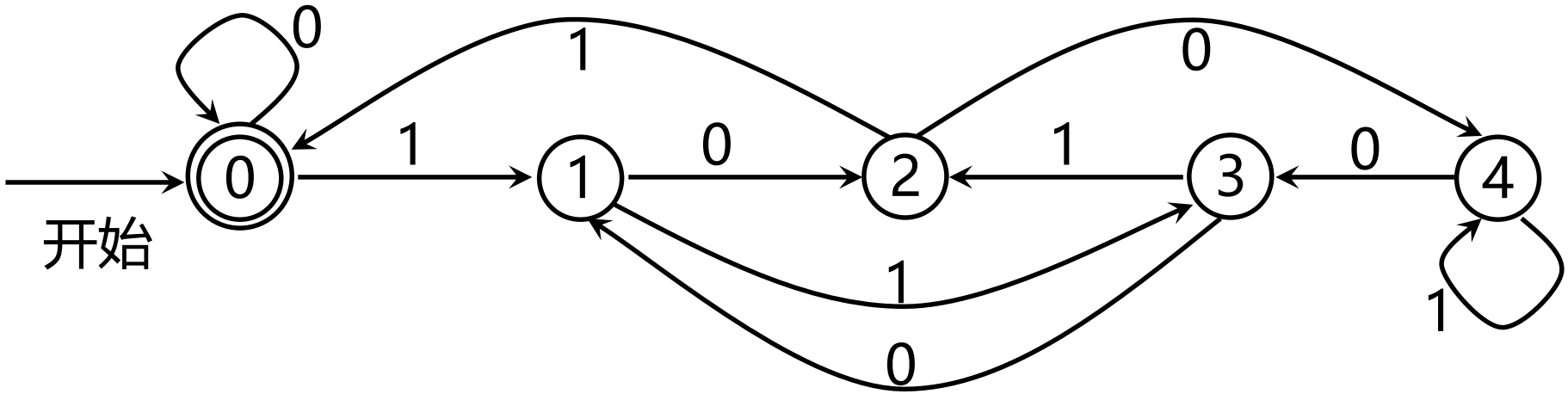


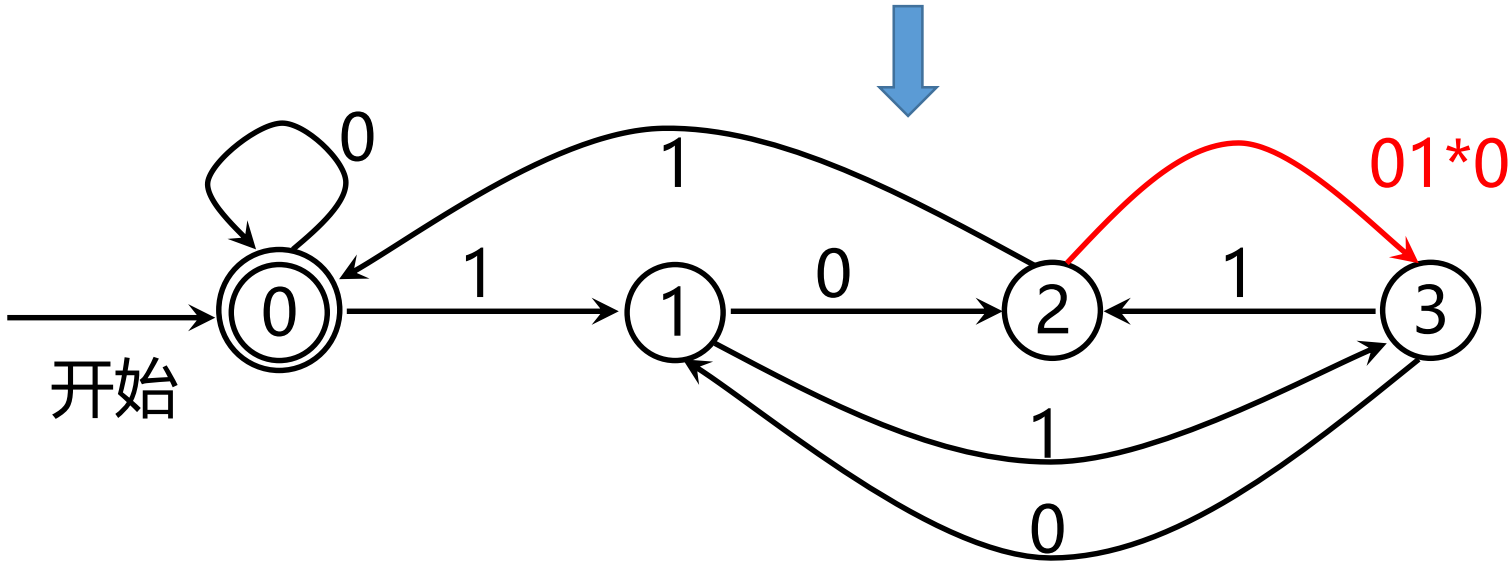




画出DFA，识别{0,1}上能被5整除的二进制数

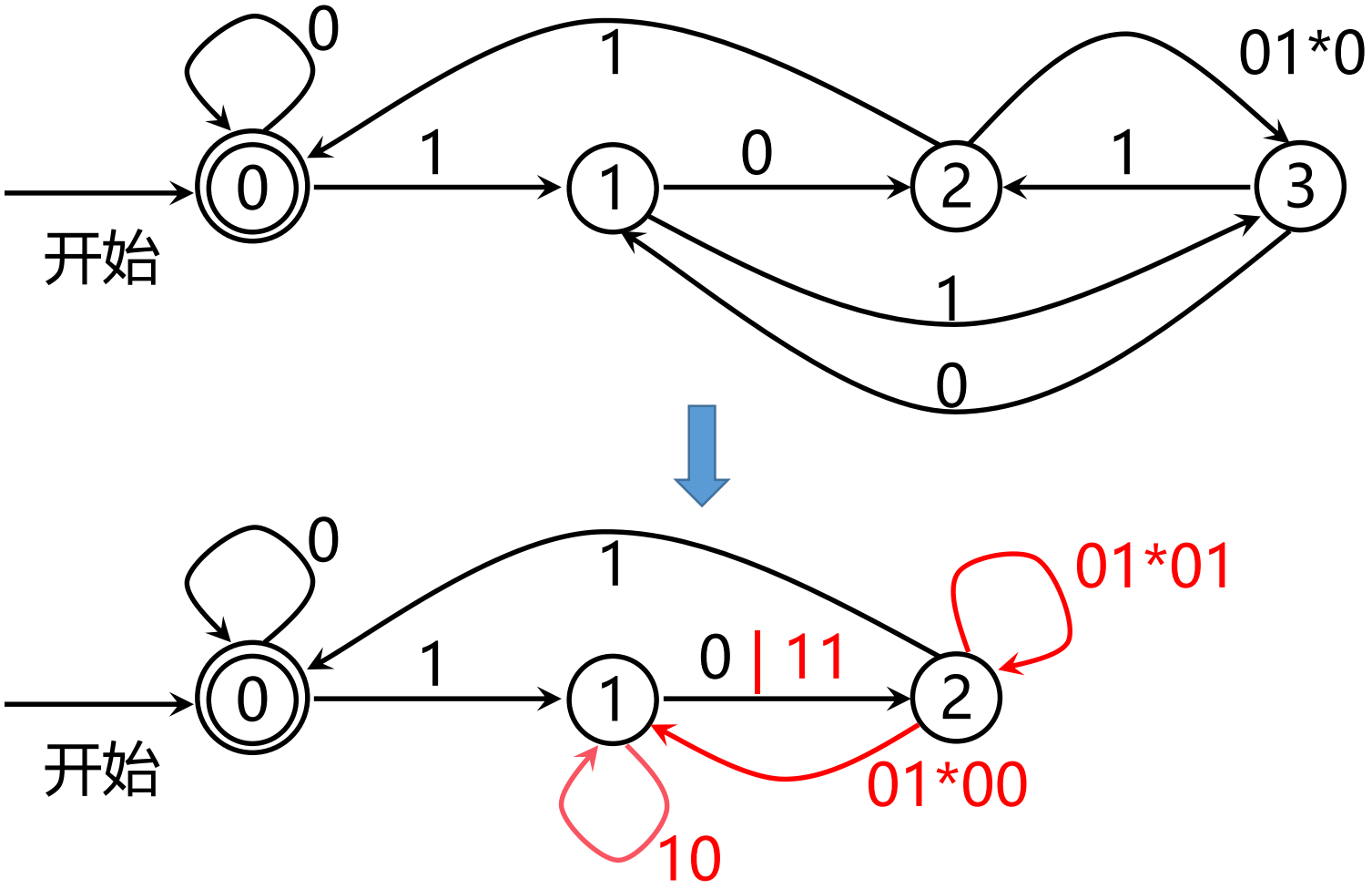
写出正规式





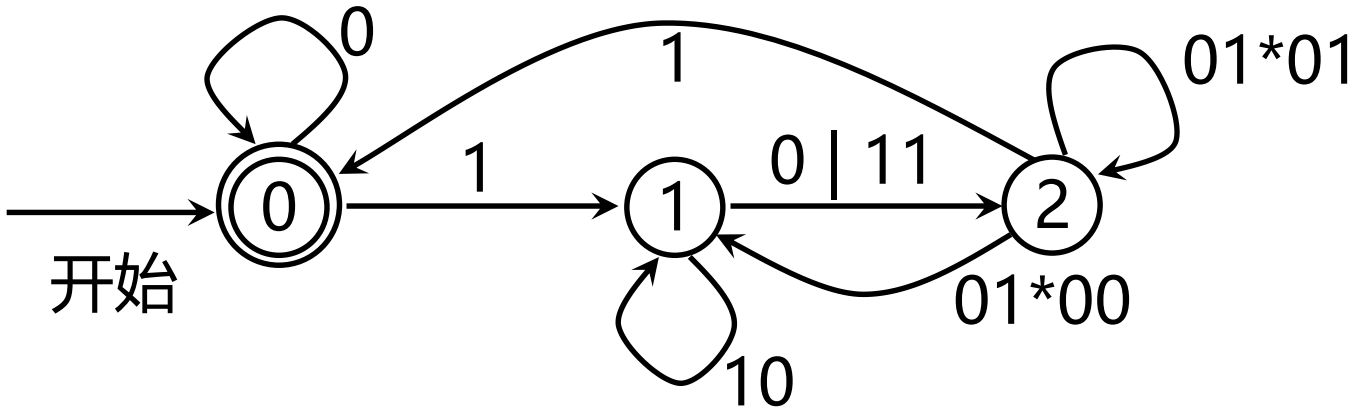
画出DFA，识别{0,1}上能被5整除的二进制数

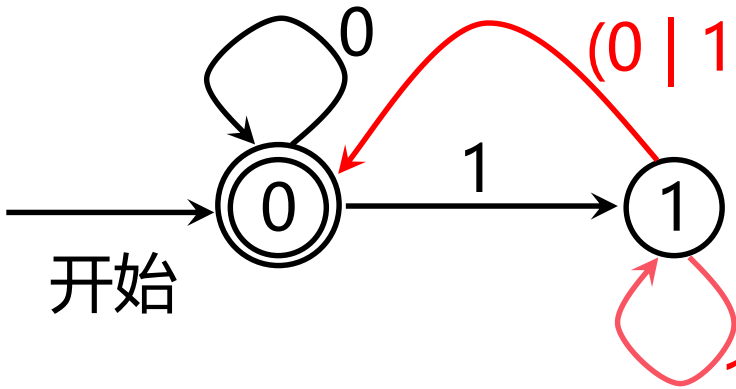
写出正规式



画出DFA，识别{0,1}上能被5整除的二进制数

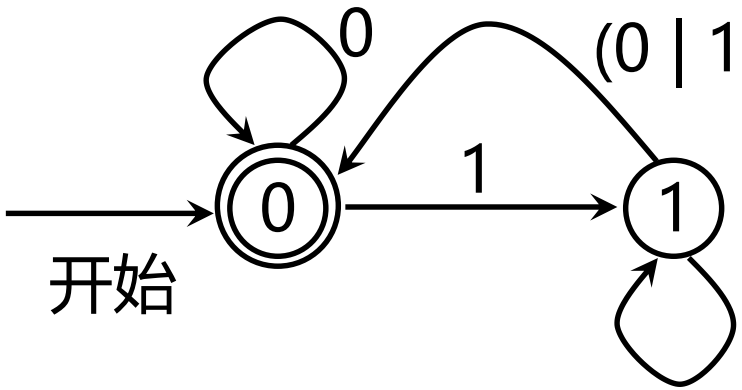
写出正规式





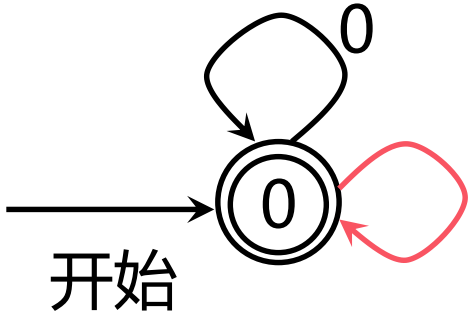
画出DFA，识别{0,1}上能被5整除的二进制数

写出正规式



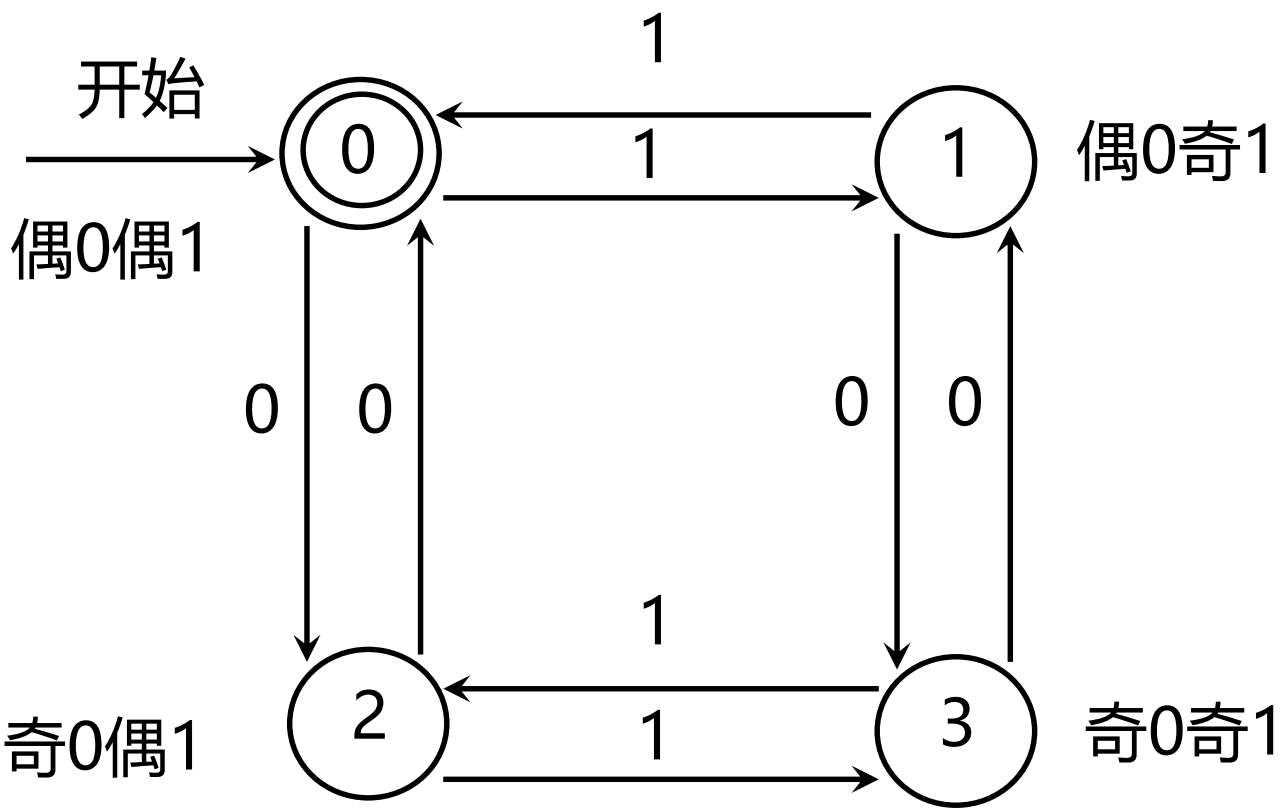
(0 | 11)(01*01)*1=str1

10 | (0 | 11)(01*01)*01*00=str2



1str2*str1 → (0 | 1str2*str1)*

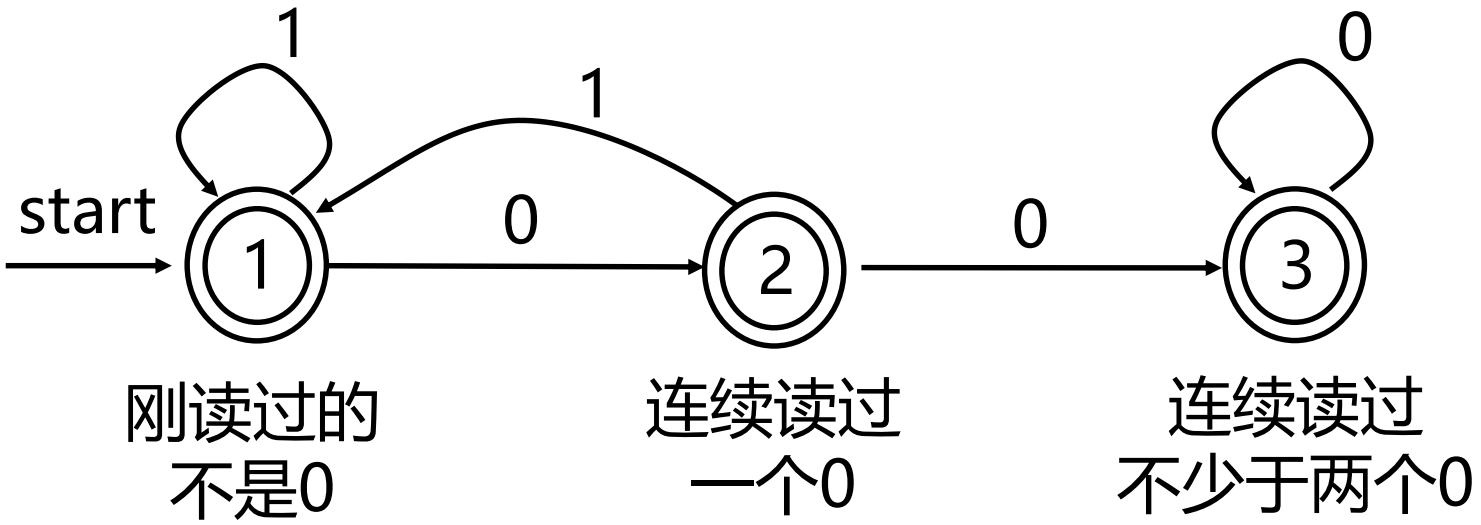
画出DFA，接受 0和1的个数都是偶数的字符串



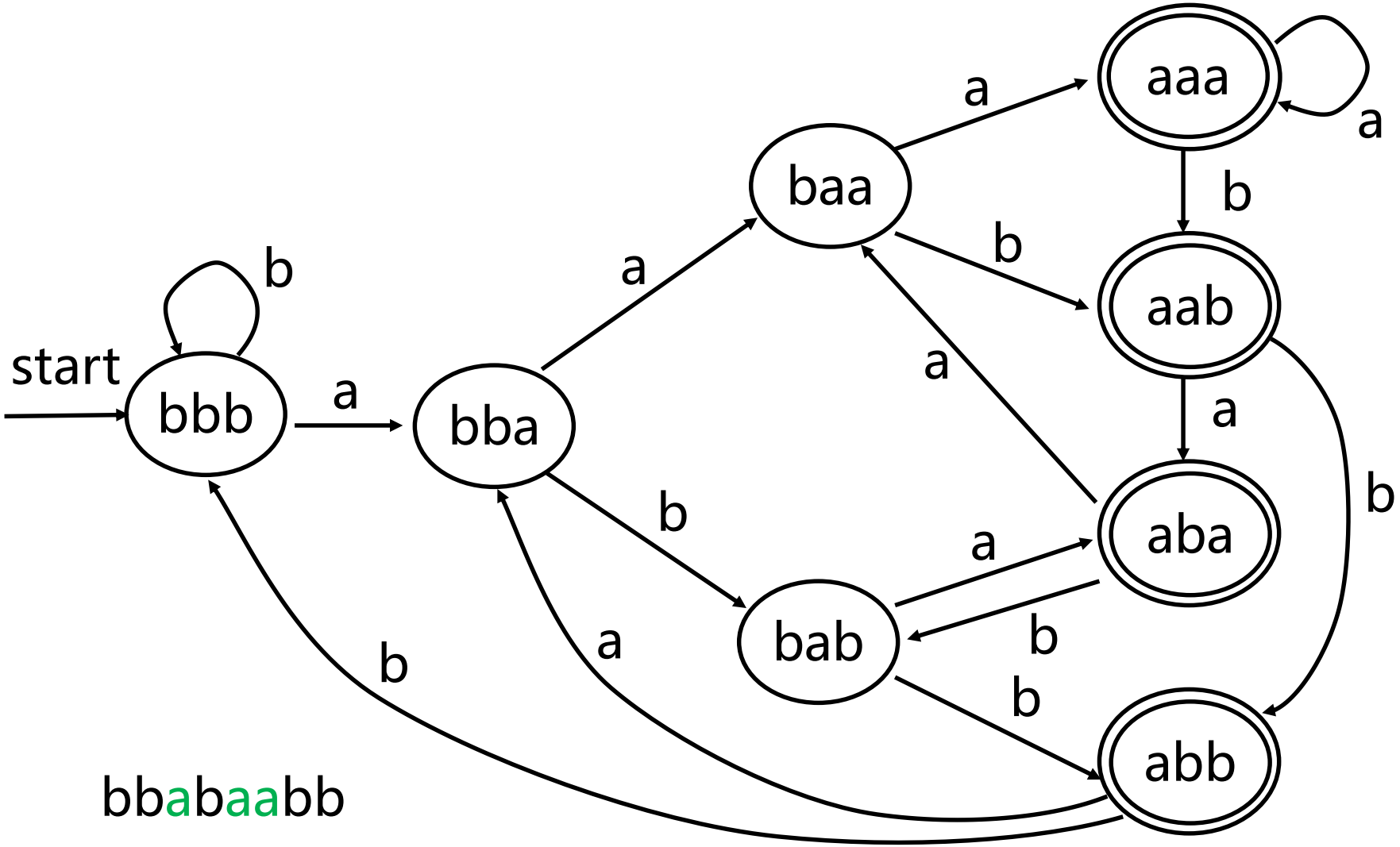
叙述下面的正规式描述的语言，并画出接受该语言的最简DFA的状态转换图

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

- 描述的语言是，所有不含子串001的0和1的串



- 用状态转换图表示接受 (a|b)*a(a|b)(a|b) 的DFA



2.3.3 NFA到DFA的变换

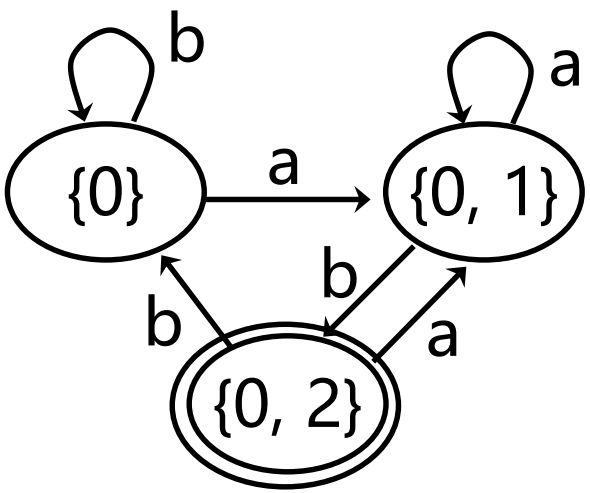
子集构造法

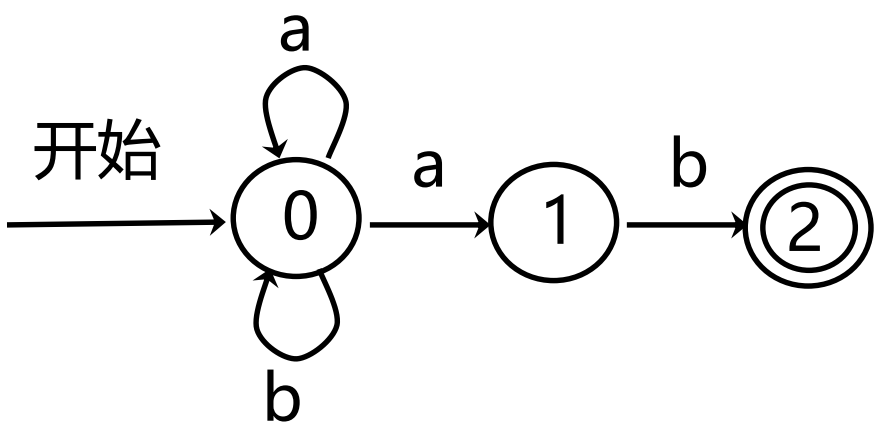
1、DFA的一个状态是NFA的一个状态集合

2、读了输入a_1，a_2，…，a_n后，

NFA能到达的所有状态：s_1, s_2, …，s_k，则

DFA到达状态{s_1, s_2, …，s_k}

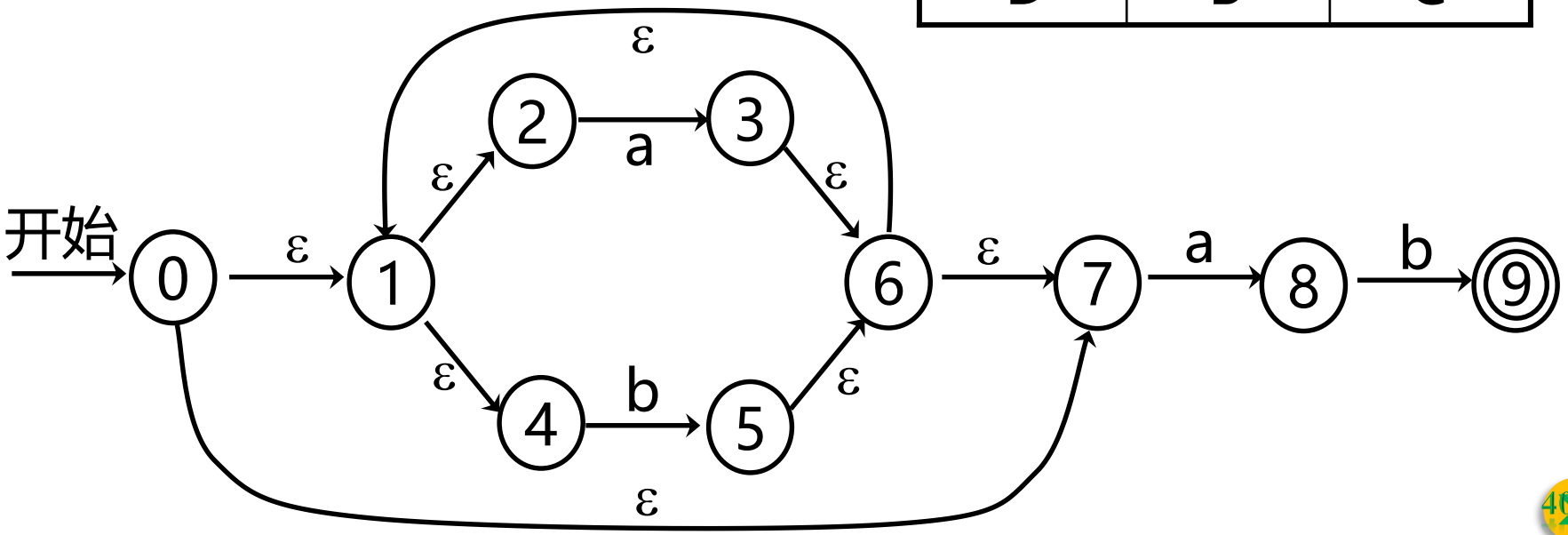




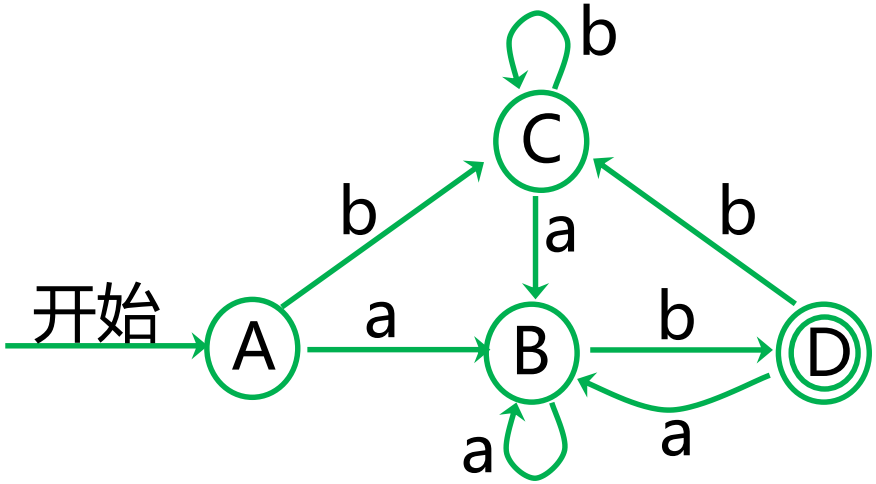
- 正规式(a|b)*ab，NFA如下，把它变换为DFA

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}

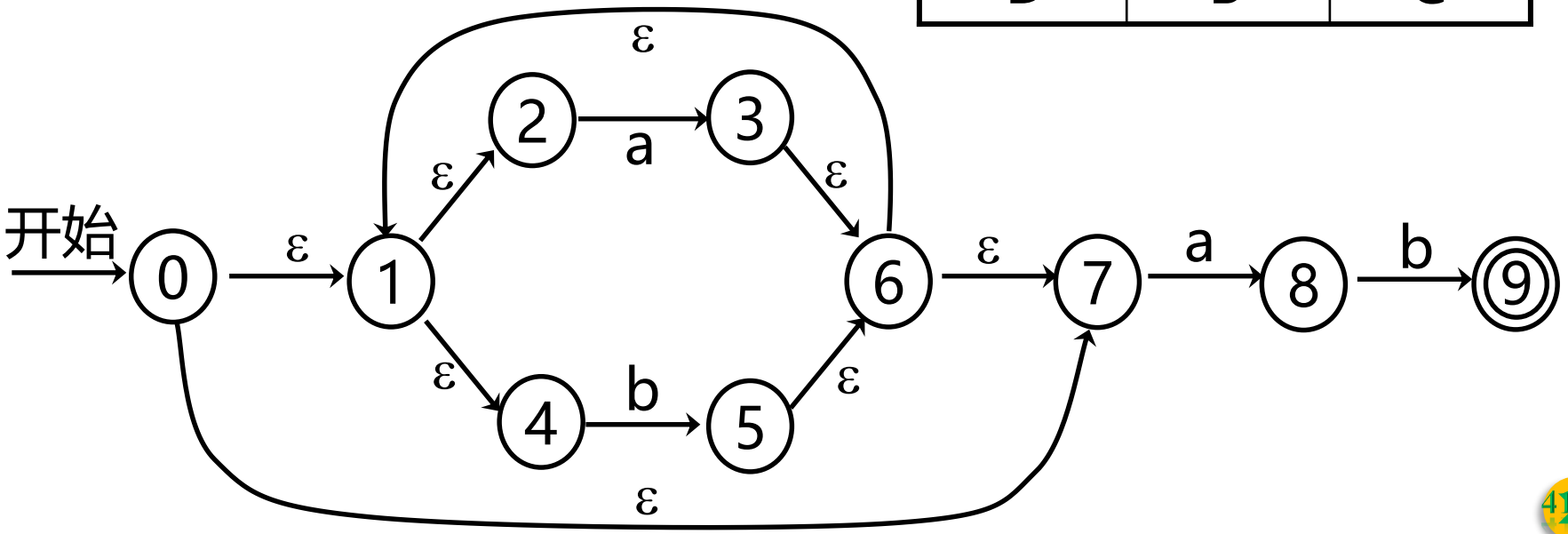
	输入符号	
	a	b
A	B	C
B	B	D
C	B	C
D	B	C



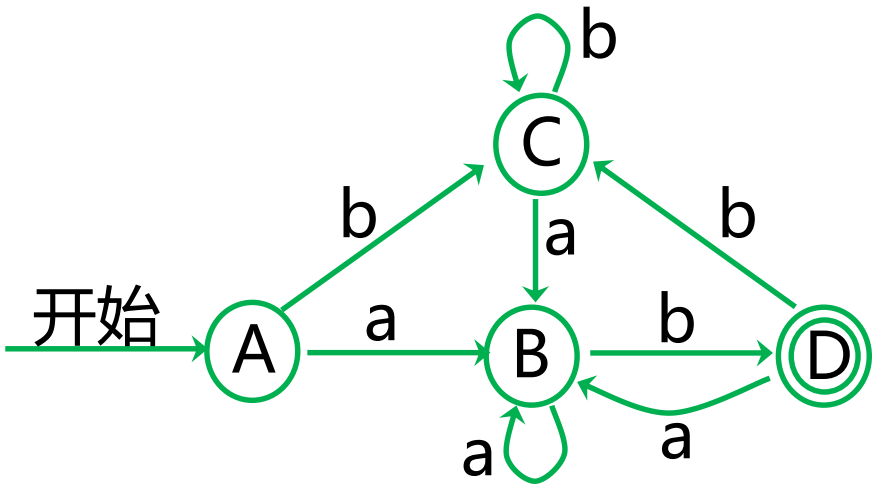
- 正规式(a|b)*ab，NFA如下，把它变换为DFA



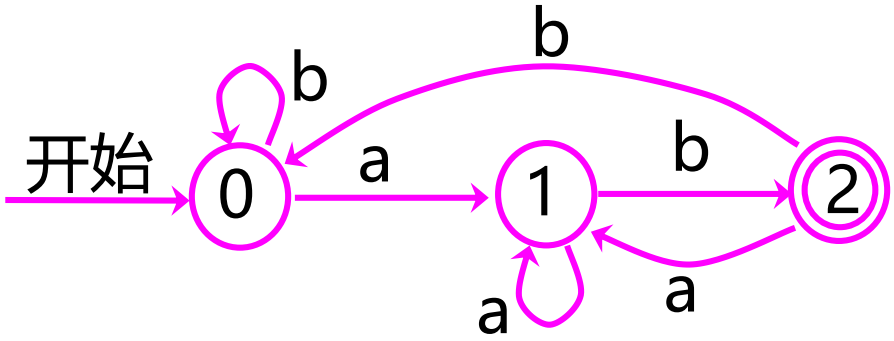
	输入符号	
	a	b
A	B	C
B	B	D
C	B	C
D	B	C

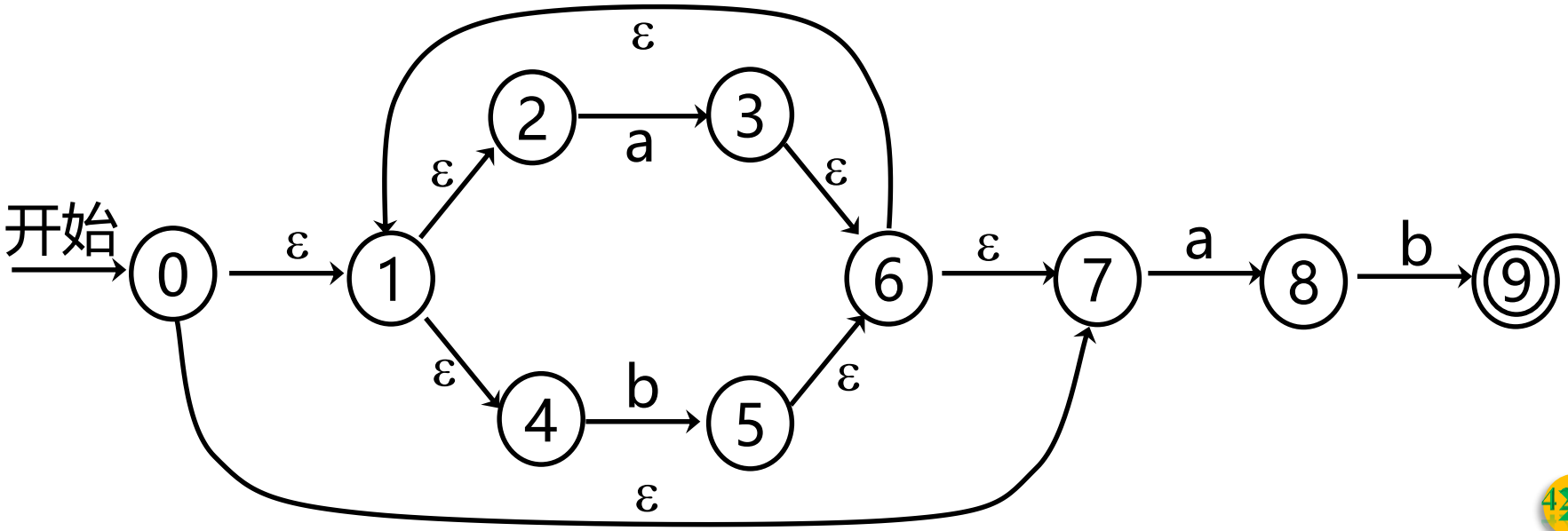


- 正规式(a|b)*ab，NFA如下，把它变换为DFA



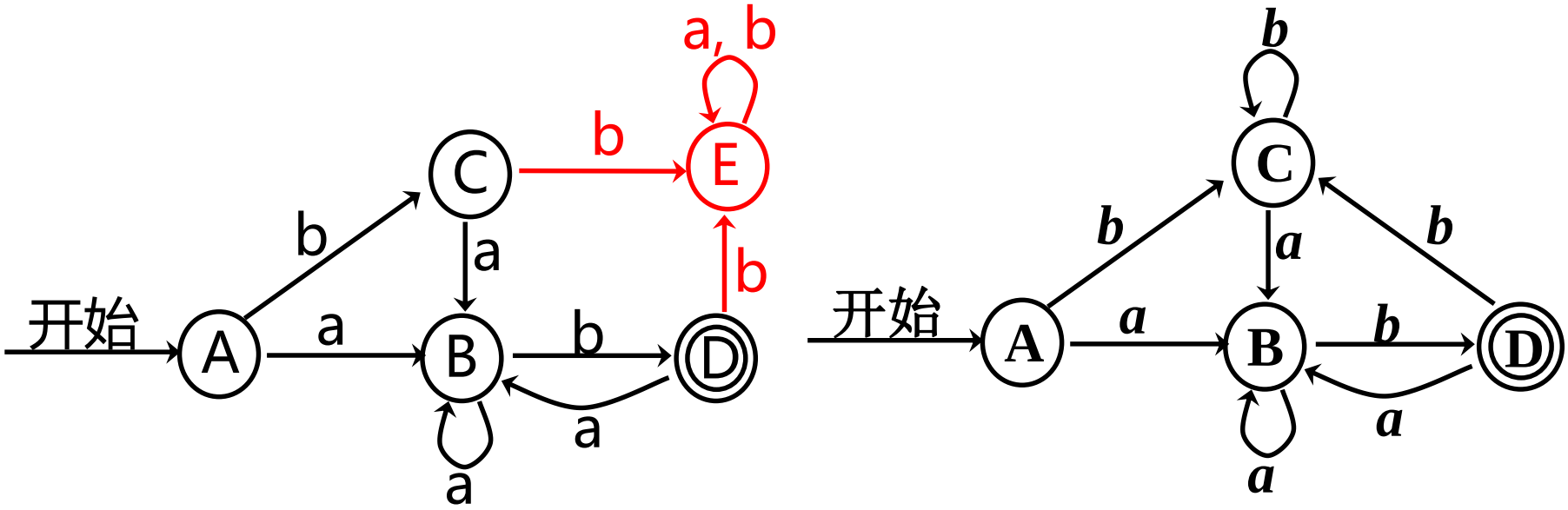
子集构造法不一定
得到最简DFA





2.3.4 DFA的化简

- 死状态
  - 在转换函数由部分函数改成全函数表示时引入
  - 左图需要引入死状态E； 右图无须引入死状态

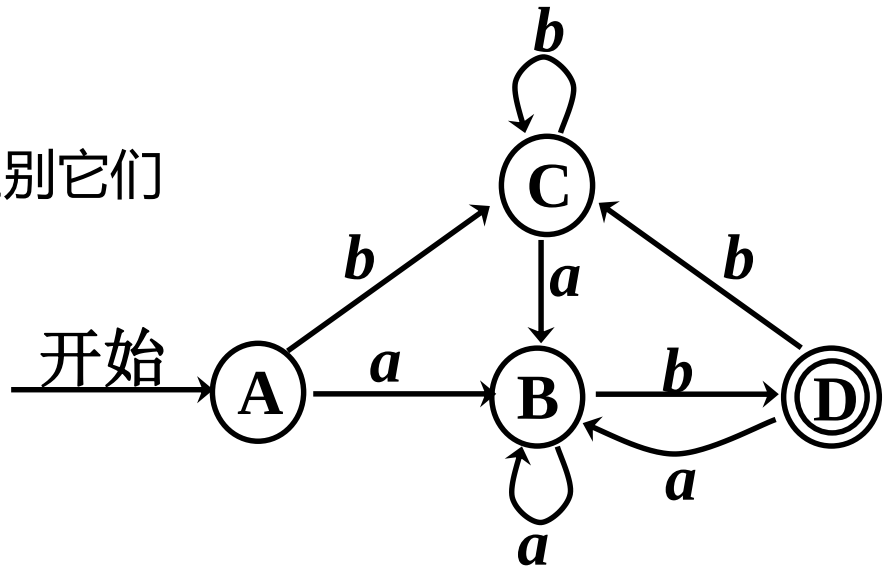


- 可区别的状态
  - A和B是可区别的状态

    从A出发，读过单字符b，到达非接受状态C，而从B出发，读过单字符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



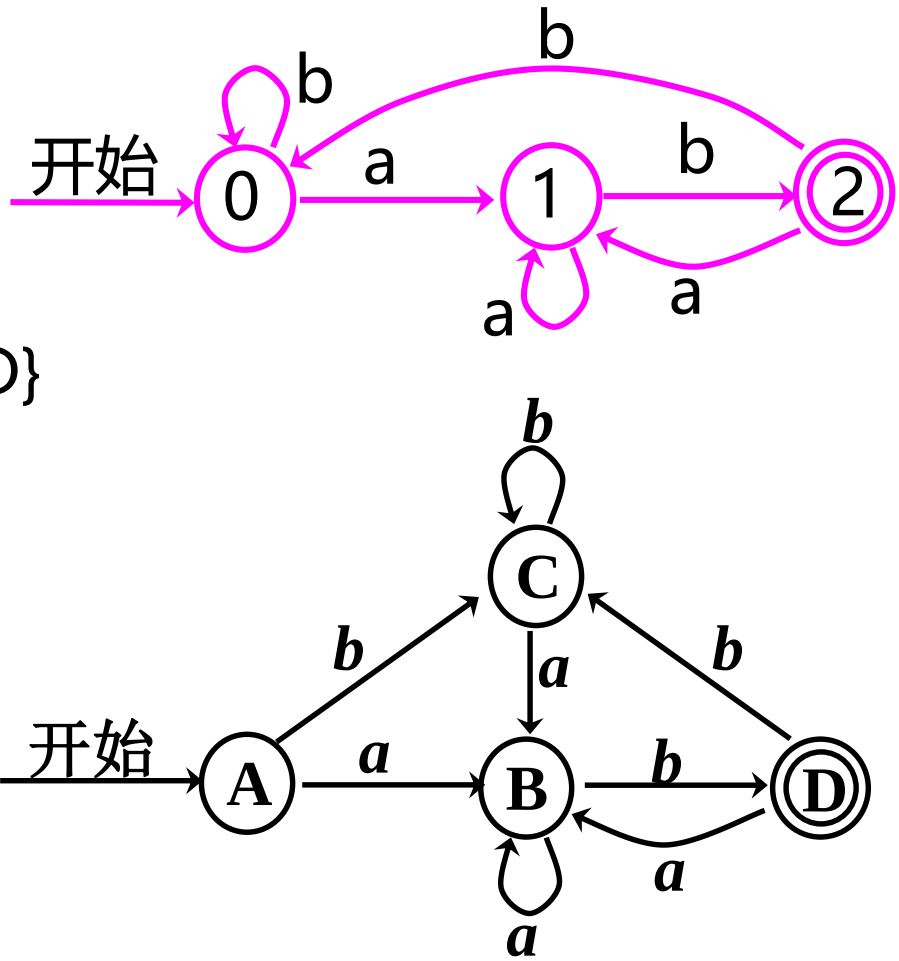
- 化简方法
  1. {A, B, C}, {D}

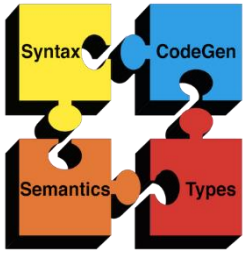
     move({A, B, C}, a) = {B}

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

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

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





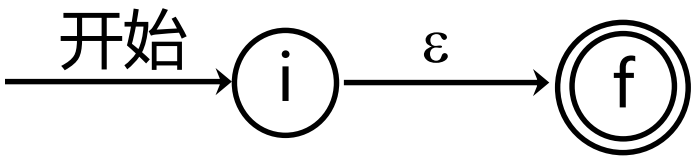
第2章 词法分析

2.4 从正规式到有限自动机

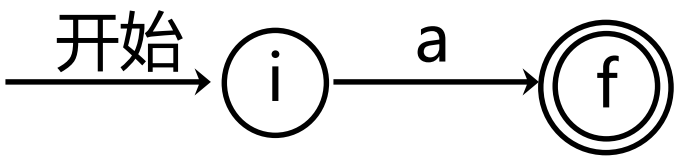
- 从正规式建立识别器的步骤
  - 从正规式构造NFA（本节介绍）
    用语法制导的算法，它用正规式语法结构来指导构造过程
  - 把NFA变成DFA （子集构造法，已介绍）
  - 将DFA化简 （合并不可区别状态，也已介绍）

- 首先构造识别ε和字母表中一个符号的NFA

  重要特点：仅一个接受状态，它没有向外的转换



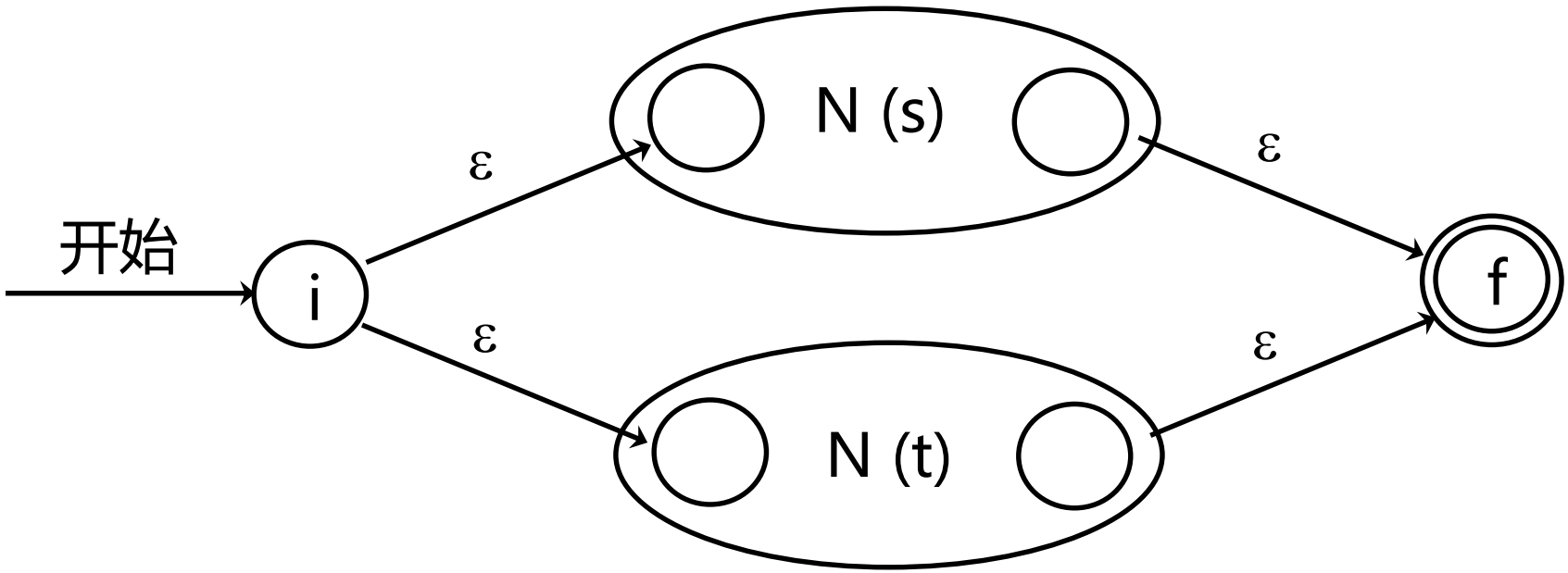
识别正规式 ε 的NFA



识别正规式 a 的NFA

- 构造识别主算符为选择的正规式的NFA

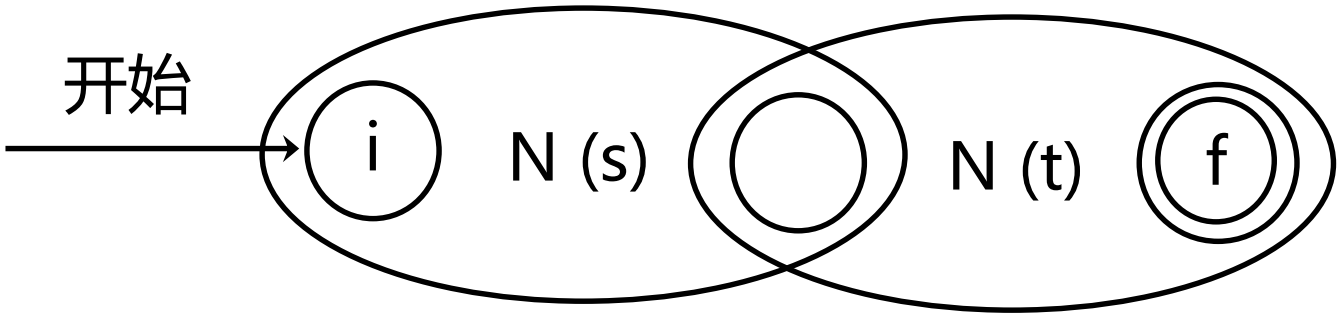
重要特点：仅一个接受状态，它没有向外的转换



识别正规式 s | t 的NFA

- 构造识别主算符为连接的正规式的NFA

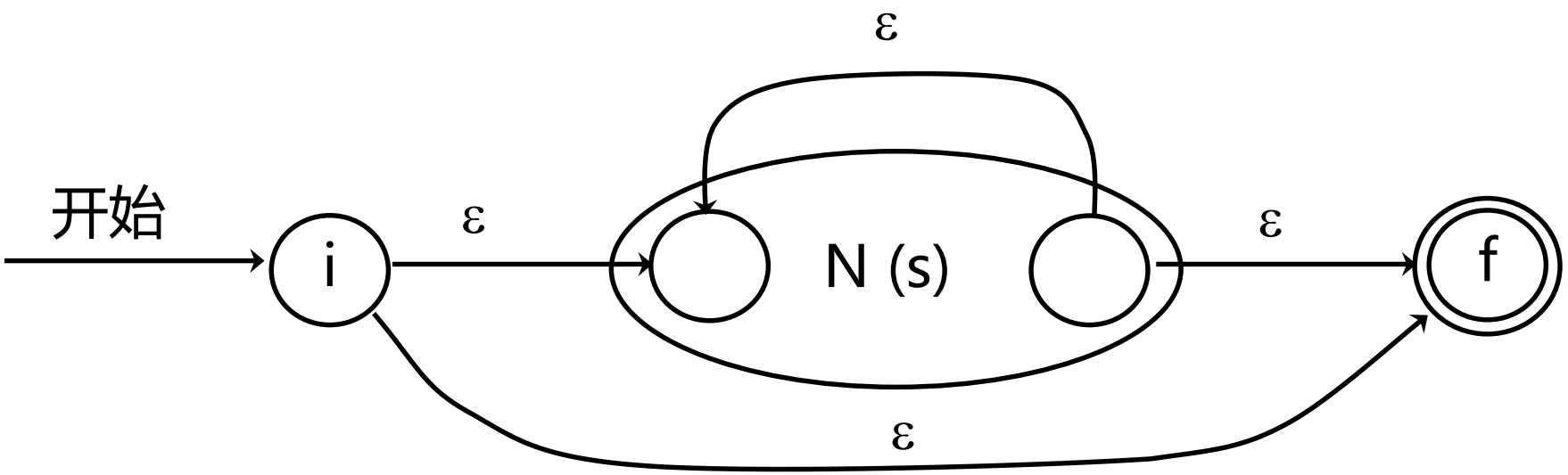
  重要特点：仅一个接受状态，它没有向外的转换



识别正规式 st 的NFA

- 构造识别主算符为闭包的正规式的NFA

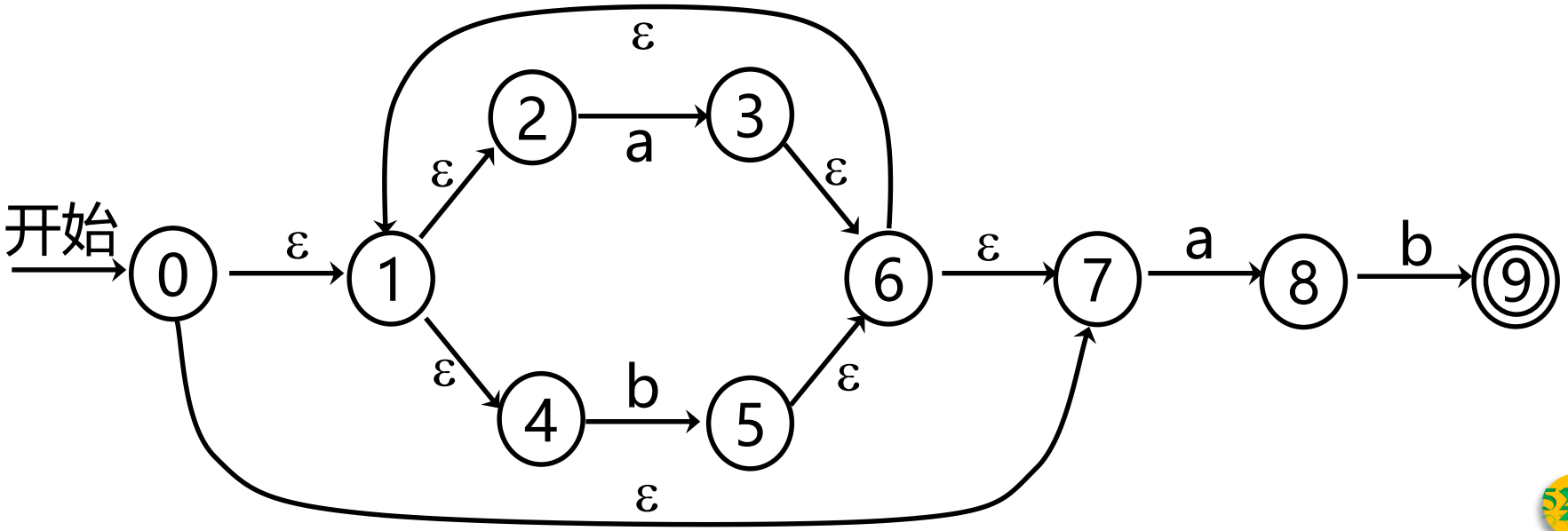
重要特点：仅一个接受状态，它没有向外的转换



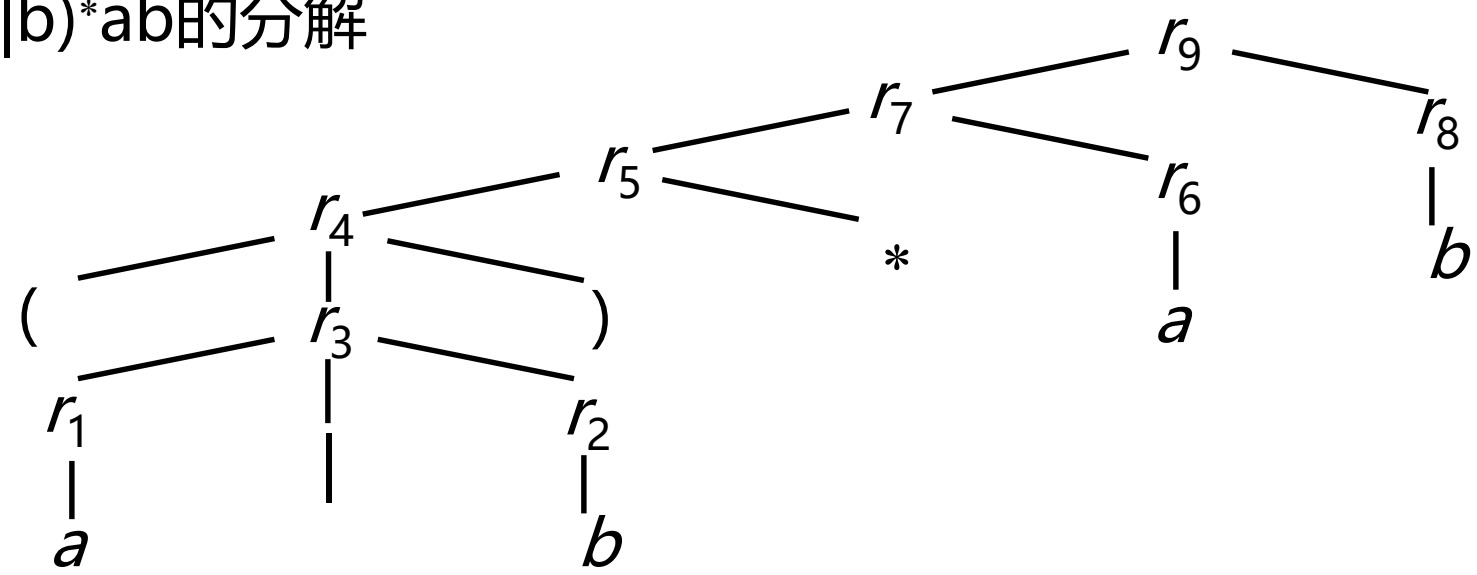
识别正规式 s^* 的NFA

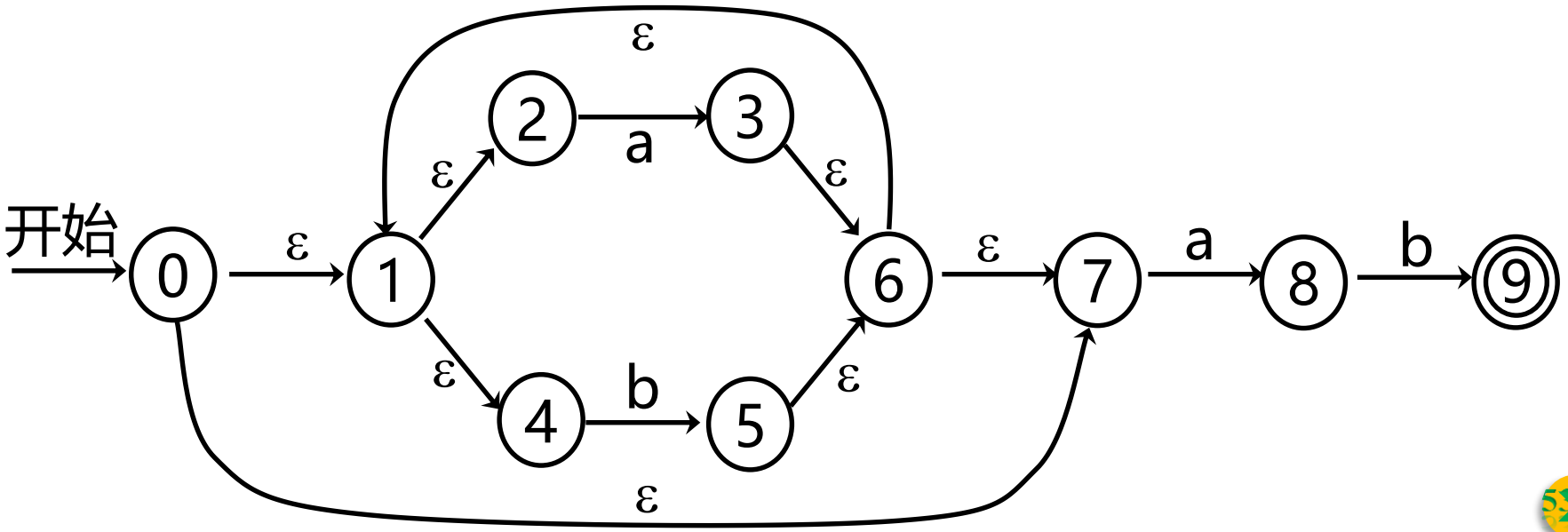
- 对于加括号的正规式(s)，使用N(s)本身作为它的NFA

- 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)的每个状态有一个用Σ的符号标记的指向其它结点的转换，或者最多两个指向其它结点的ε转换



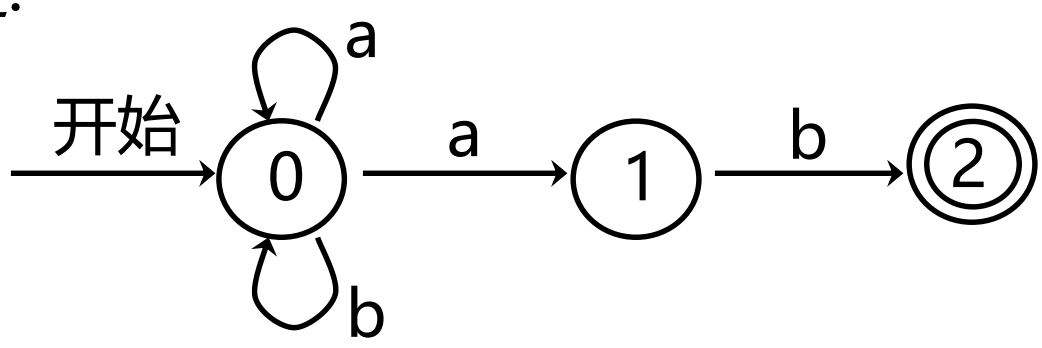
- (a|b)*ab的分解



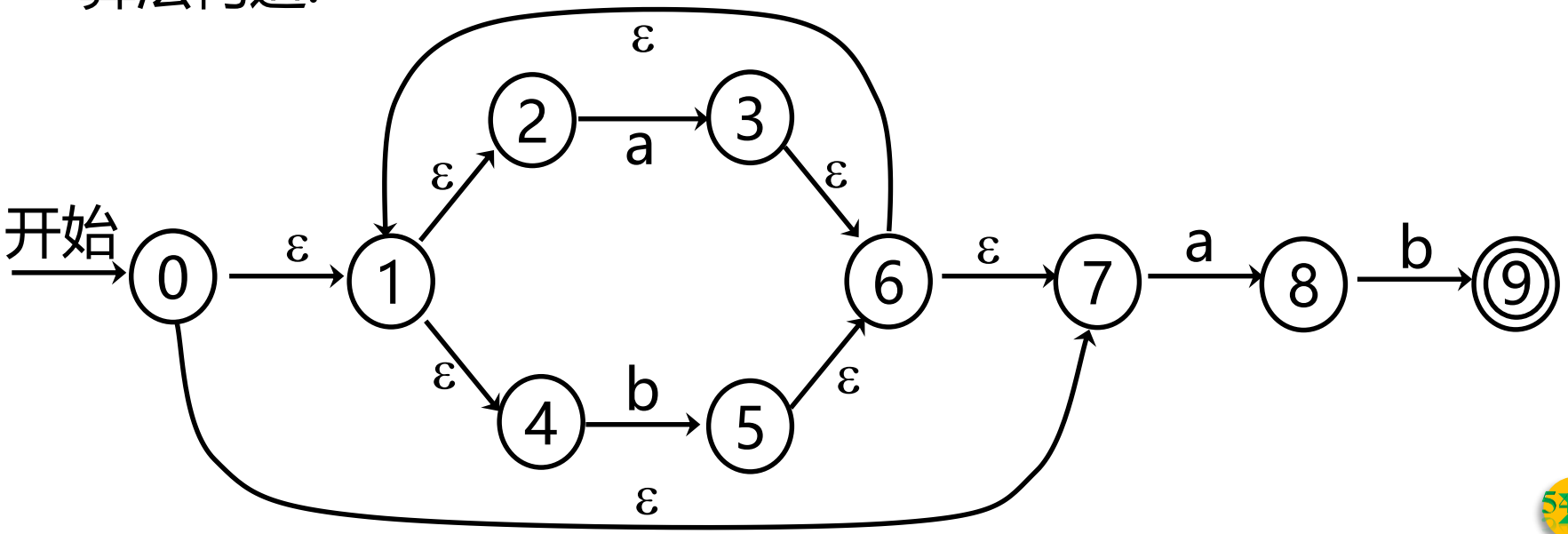


- (a|b)*ab的两个NFA的比较

手工构造:



算法构造:

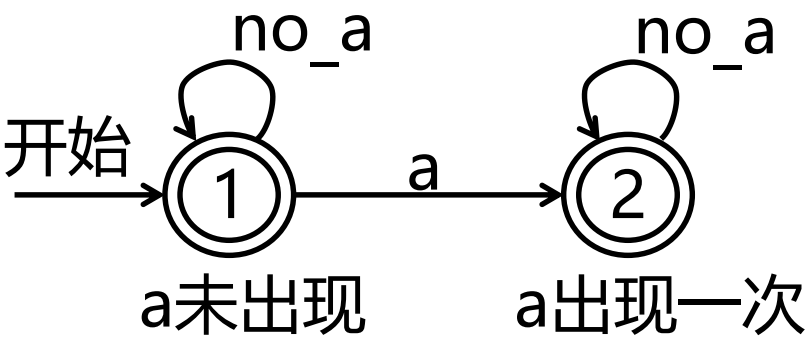


- 小结：从正规式建立识别器的步骤
  - 从正规式构造NFA
  - 把NFA变成DFA
  - 将DFA化简
- 存在其它办法

- 在字母表{a..z}上，用正规式表示字母a出现次数不超过1次的所有句子的集合，并画出接受该语言的最简DFA
- 考虑任意字母出现次数都不超过1次的所有句子的集合，分析接受该语言的最简DFA有多少个状态

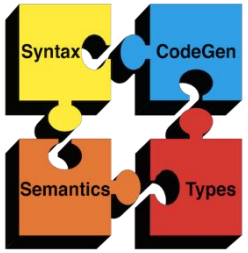
no_a=b|c|d|...|y|z

acc=(no_a)*(a|ε)(no_a)*



对每个字母，用0表示它不出现，1表示它出现；每个状态用26个比特表示，对应句子中各字母的出现情况

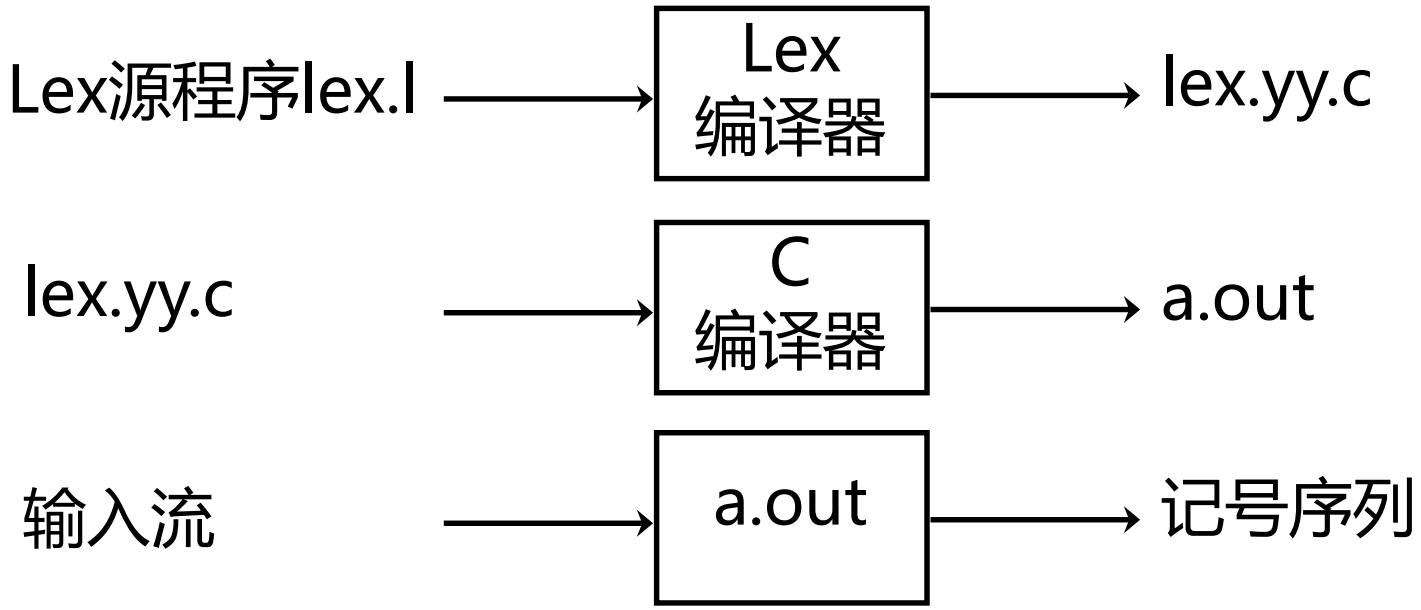
共2^{26}个状态



第2章 词法分析

2.5 词法分析器的生成器 *

- 用Lex建立词法分析器的步骤



- 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}

- 例——声明部分

```
%{
/* 常量LT, LE, EQ, NE, GT, GE,
        WHILE, DO, ID, NUMBER, RELOP的定义*/
%}
/* 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}+)?
```

- 例——翻译规则部分

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

- 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 */
}
```

- 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

- 字母表、串、句子、语言的概念

- 掌握下面涉及的一些概念，相互转换的技巧、方法或算法
  - 非形式描述的语言 ↔ 正规式
  - 非形式描述的语言 ↔ NFA
  - 非形式描述的语言 ↔ DFA（或最简DFA）
  - 正规式 ↔ NFA
  - NFA ↔ DFA
  - DFA ↔ 最简DFA

非形式描述的语言 ↔ 正规式

- 2.3
- 2.4 (c) (e) (g)

正规式 ↔ NFA ↔ DFA

- 2.7 (c) (d)
- 2.8 (仅为2.7 (c) 做)

非形式描述的语言 ↔ DFA

- 2.10
- 2.12
- 2.15

- 2.3

2.3 叙述由下列正规式描述的语言。

(a) $0\,(0 \mid 1)^{*}\,0$

(b) $((\varepsilon \mid 0)\,1^{*})^{*}$

(c) $(0 \mid 1)^{*}0\,(0 \mid 1)\,(0 \mid 1)$

(d) $0^{*}10^{*}10^{*}10^{*}$

(e) $(00 \mid 11)^{*}((01 \mid 10)\,(00 \mid 11)^{*}(01 \mid 10)\,(00 \mid 11)^{*})^{*}$

- 2.4 (c) (e) (g)

*2.4 为下列语言写出正规定义。

(c) 某语言的注释,它是以/*开始并以*/结束的任意字符串,但它的任何前缀(本身除外)不以*/结尾。

(e) 最多只有一处相邻数字相同的所有数字串。

(g) 由偶数个0和奇数个1构成的所有0和1的串。

- 2.7 (c) (d)

2.7　用算法 2.4 为下列正规式构造不确定有限自动机，给出它们处理输入串 *ababbab* 的状态转换序列。

(a) $(a \mid b)^*$

(b) $(a^* \mid b^*)^*$

(c) $((\varepsilon \mid a)b^*)^*$

(d) $(a \mid b)^* abb(a \mid b)^*$

- 2.8 (仅为2.7 (c) 做)

2.8　用算法 2.2 把习题 2.7 的 NFA 变换成 DFA。给出它们处理输入串 *ababbab* 的状态转换序列。

(c) $((\varepsilon \mid a)b^*)^*$

- 2.10

2.10 某语言的注释是以/*开始和以*/结束的任意字符串，但它的任何前缀（本身除外）不以*/结尾。画出接受这种注解的 DFA 的状态转换图。

- 2.12

2.12 为下列正规式构造最简的 DFA。

(a) $(a \mid b)^* a\,(a \mid b)$

(b) $(a \mid b)^* a\,(a \mid b)\,(a \mid b)$

(c) $(a \mid b)^* a\,(a \mid b)\,(a \mid b)\,(a \mid b)$

- 2.15

2.15 构造一个最简的 DFA，它接受所有大于 101 的二进制整数。