

编译原理及实现



复 习

设 $M = (\{x, y\}, \{a, b\}, f, x, \{y\})$ 为一非确定的有限自动机，其中 f 定义如下：

$$f(x, a) = \{x, y\} \qquad f(x, b) = \{y\}$$

$$f(y, a) = \varnothing \qquad f(y, b) = \{x, y\}$$

试①构造相应确定有限自动机 M' ；

②对①构造的DFA M' 进行化简，求化简后的DFA M'' 。

解答：对照自动机的定义 $M=(S,\Sigma,f,S_0,Z)$ ，由 f 的定义可知 $f(x,a)$ 、 $f(y,b)$ 均为多值函数，所以是一非确定有限自动机，先画出NFA M 相应的状态图：

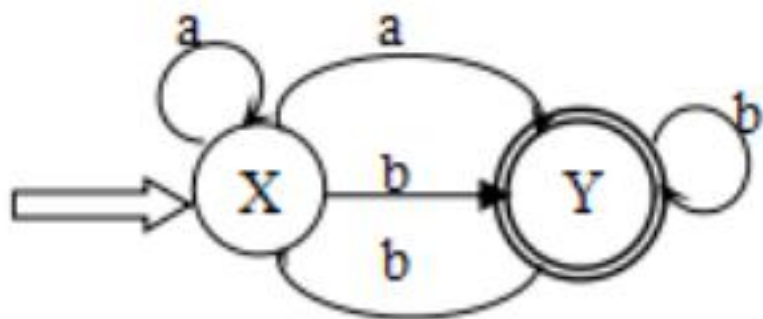


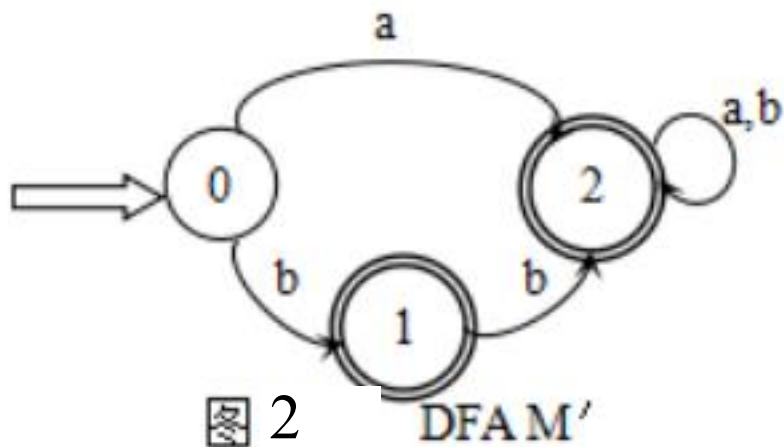
图 1 NFA M

用子集法构造状态转换矩阵表所示：

I	I_a	I_b
$\{x\}$	$\{x,y\}$	$\{y\}$
$\{y\}$	—	$\{x,y\}$
$\{x,y\}$	$\{x,y\}$	$\{x,y\}$

将转换矩阵中的所有子集重新命名而形成下表所示的状态转换矩阵。

	a	b
0	2	1
1	—	2
2	2	2



将图2的DFA M' 最小化:

首先, 将 M' 的状态分成终态组 $\{1, 2\}$ 与非终态组 $\{0\}$;

其次, 考察 $\{1, 2\}$ 。由于 $\{1, 2\}a = \{1, 2\}b = \{2\} \subset \{1, 2\}$, 所以不再将其划分了, 也即整个划分只有两组 $\{0\}$, $\{1, 2\}$: 令状态1代表 $\{1, 2\}$, 即把原来到达2的弧都导向1, 并删除状态2。

最后, 得到如图3所示化简DFA M'' 。

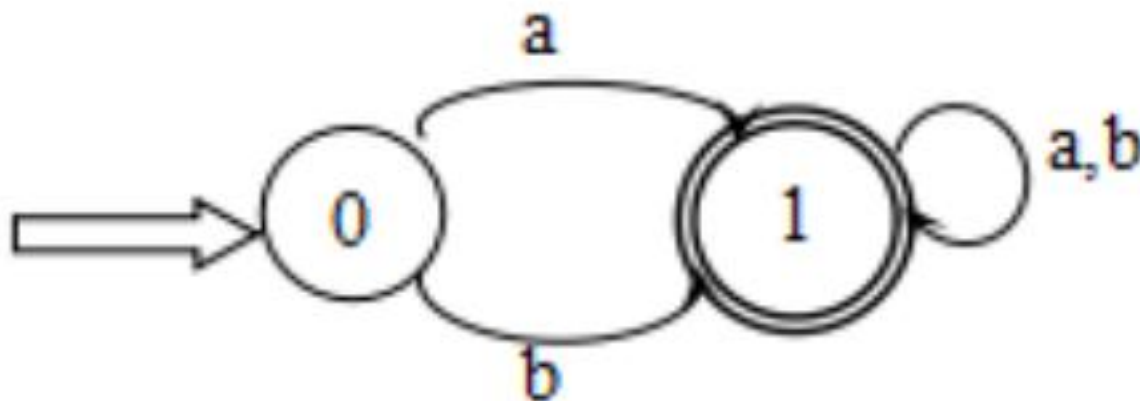


图 3

化简后的 DFAM ”

第3章 词法分析

- 3.1 词法分析基本思想
- 3.2 单词的描述工具
- 3.3 单词的识别
- 3.4 词法分析程序的设计及实现

3.1 词法分析程序的任务功能及实现方案

词法：描述语言的单词符号的文法。

例如描述某一语言标识符的文法也称为词法。

语言的**单词符号**种类很多，因此需要用**不同的词法**来描述他们。

思考：词法是哪类文法？

正规文法

∞ 定义：编译程序中完成词法分析任务的程序段，
又称词法分析器

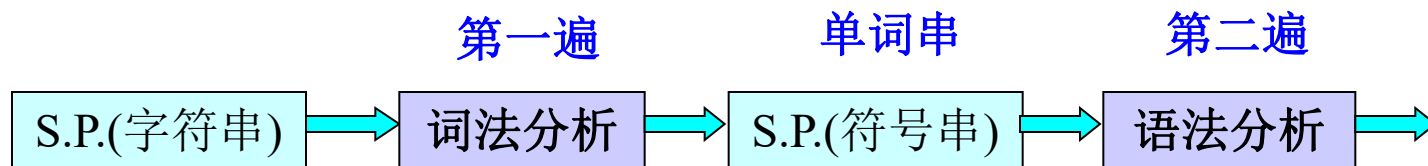
∞词法分析器任务：对源程序进行从左到右的顺序扫描，按照词法规则从中识别出一个个具有独立意义的单词符号并把源程序转换为等价的内部表示形式（属性字流）。

词法分析程序的功能

- 识别出单词（内部表示）；
- 过滤掉源程序中的注释和空白；
- 记录读入字符的行号，以便发现错误后能报告出
错位置；
- 进行预编译工作(对宏进行展开等工作)；
- 符号表操作和错误处理等。

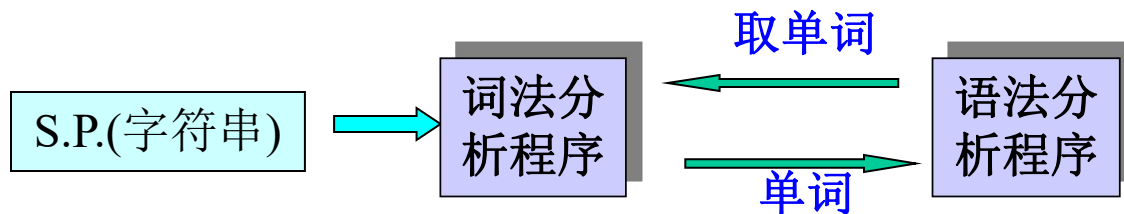
实现方案（编译程序中实现方式）：基本上有两种

1.词法分析单独作为一遍



优点：结构清晰、各遍功能单一
缺点：生成中间文件，效率低

2.词法分析程序作为单独的子程序



3.2 单词的种类及词法分析程序的输出形式

单词是语言中具有独立意义的`最小语法单位`.

单词的种类

1. 保留字: while、class、for、do
2. 标识符: a、m_a
3. 常数: 无符号数、布尔常数、字符串常数等
4. 运算符: +、-、*
5. 界限符: 逗号, 分号, 括号和引号

种类的划分是根据编译的目标和方便设定的

词法分析程序的输出形式-----单词的内部形式

单词类别	单词值
------	-----

常用的单词内部形式:

- 1、按单词种类分类
- 2、保留字和分界符采用一符一类
- 3、标识符和常数的单词属性值为符号表入口指针
(标识符、常数相关属性很多)

1、按单词种类分类

单词名称	类别编码	单词值
标识符	1	符号表入口指针
无符号常数(整)	2	整数值
无符号浮点数	3	数值
布尔常数	4	0 或 1
字符串常数	5	内部字符串
保留字	6	保留字或内部编码
分界符	7	分界符或内部编码

2、保留字和分界符采用一符一类

单词名称	类别编码	单词值
标识符	1	符号表入口指针
无符号常数(整)	2	整数值
无符号浮点数	3	数值
布尔常数	4	0 或 1
字符串常数	5	内部字符串
BEGIN	6	-
END	7	-
FOR	8	-
DO	9	-
.....
:	20	-
+	21	-
*	22	-
,	23	-
(.....	-

单词的描述方法--正规式（正规表达式）

正规式与正规集

正规式与有限自动机

正规式与正规文法

正规表达式与正规集

正规表达式是一种表示法，可以详细描述高级程序语言单词符号或者说每一个正规表达式 r 表示一个语言 $L(r)$.

例如可以用正规表达式表示C++语言的标识符（描述标识符的结构），而 C++语言的标识符（集合）可以看作一个语言，因此正规表达式 r 实质是用来表示一个语言 $L(r)$.

C++语言的标识符是字母或下划线开头的字母、下划线、数字串，如何表示？

正规表达式的递归定义

字母表 Σ 上的正规表达式递归定义如下:

1. ϵ 和 ϕ 都是 Σ 上的正规表达式, 它们所表示的语言分别为: $\{\epsilon\}$ 和 $\{\}$;
2. 任何 $a \in \Sigma$, a 是 Σ 上的正规表达式, 它所表示的语言为: $\{a\}$;
3. 假定 r 和 s 是 Σ 上的正规表达式, 它们所表示的语言分别记为 $L(r)$

和 $L(s)$, 那么 $r|s$, rs , r^* 也都是 Σ 上的正规表达式, 它们所表示的

语言分别为 $L(r) \cup L(s)$ 、 $L(r)L(s)$ 、 $L(r)^*$

$r|s$ $L(r) \cup L(s)$

rs $L(r)L(s)$

r^* $L(r)^*$

一个由正规表达式表示的语言称为一个正规集

规定 “*” , “连接” , “|” 运算左结合,
优先级由高到低

简化正规式 $(a) | ((b) * (c))$ 等价于 $a | b * c$

例: $\Sigma = \{A, B, \dots, Z, a, b, \dots, z, 0, 1, \dots, 9\}$

正规式1: $A | B \dots | Z | a | b \dots | z$

语言1: $L(A) \cup L(B) \dots \cup L(Z) \cup L(a) \cup L(b) \dots \cup L(z)$
 $= \{A, B, \dots, Z, a, b, \dots, z\}$

正规式2: $0 | 1 | \dots | 9$

语言2: $L(0) \cup L(1) \cup \dots \cup L(9) = \{0, 1, \dots, 9\}$

例 $\Sigma = \{a, b\}$

(a) $a \mid b$ $L(a) \cup L(b) = \{a, b\}$

(b) $(a \mid b)(a \mid b)$

$$(L(a) \cup L(b))(L(a) \cup L(b)) = \{aa, ab, ba, bb\}$$

(c) a^*

$$(L(a))^* = \{\epsilon, a, aa, aaa, aaaa, \dots\}$$

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

$$(L(a) \cup L(b))^* = \{\epsilon, a, b, aa, ab, ba, bb, aaa, \dots\}$$

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

符号串a和包括零个或多个a后跟一个b构成的所有符号串

思考：C++语言的标识符？

C++语言的标识符

正规式:

$(A | B... | Z | a | b... | z | _)((A | B... | Z | a | b... | z) | _ | (0 | 1 | .. | 9))^*$

语言:

$\{A, B, \dots, Z, a, b, \dots, z, _ \}(\{A, B, \dots, Z, a, b, \dots, z, _, 0, 1, \dots, 9\})^*$

描述单词符号的结构，是进行词法分析程序设计的第一步，在表示的基础上如何做进一步的工作？

正规式与有限自动机

定理: 设 r 是 Σ 上一个正规表达式, 则存在一个 FA M 接受 $L(r)$ 。反之亦然。

正规文法, 正规表达式和有限自动机三者之间在表示语言的意义下是等价的。

字母表 Σ 上的一个正规语言, 既可以由某一个正规文法产生, 也可以由某一个正规表达式所表示, 还可以由某一个有限自动机识别。

关系图:

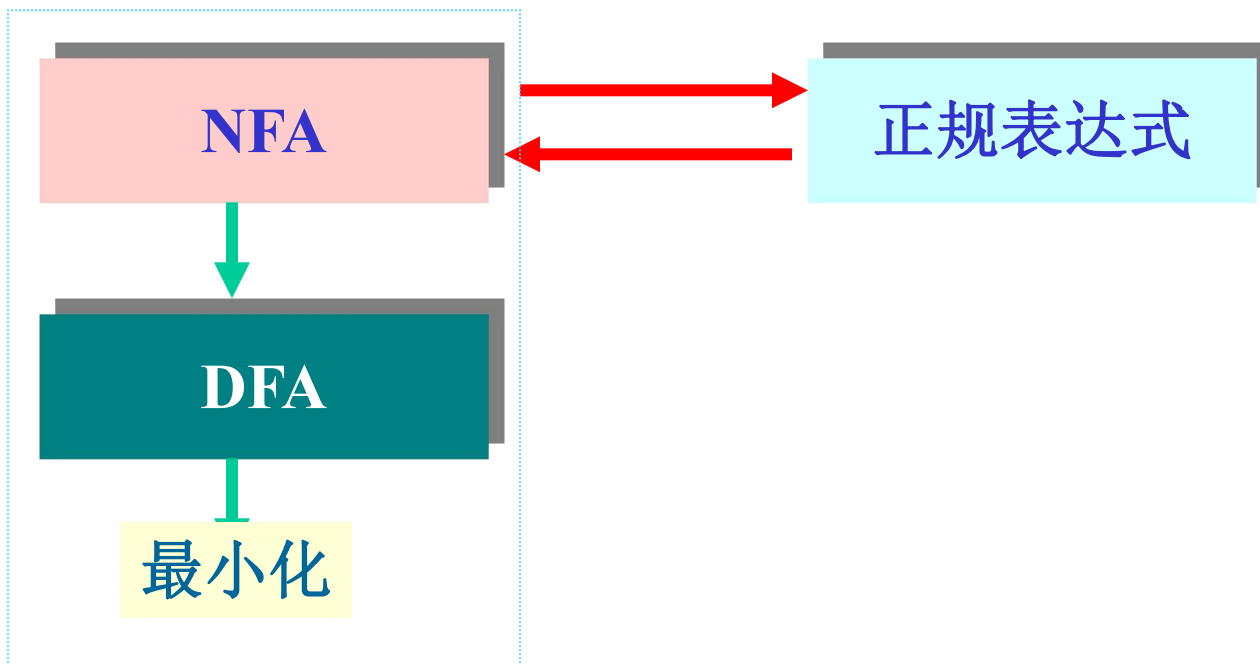
正规文法

NFA

正规表达式

DFA

最小化



转换法实现正规式与有限自动机的转换

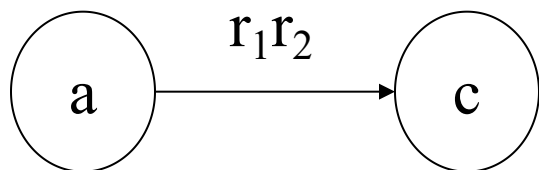
对于 Σ 上任一正规表达式 r ，一定能构造一个NFA m ，使得 $L(r)=L(m)$ 。

首先把转换图的概念拓广，每条弧上可以用一个正规式标记，对NFA m 构造一个广义的转换图。其中只有**开始状态S和终止状态Z**，连接S和Z的有向弧上是正规式 R ，然后**按照下面的替换规则对正规式不断进行分解**，不断加入状态结点和箭弧，**直到转换图上的所有弧标记都是 Σ 上的元素或 ϵ 为止**。

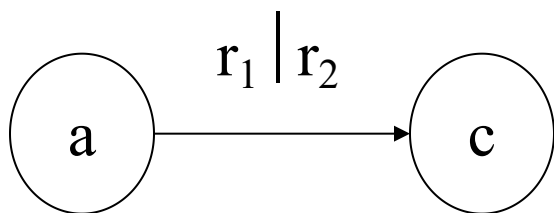
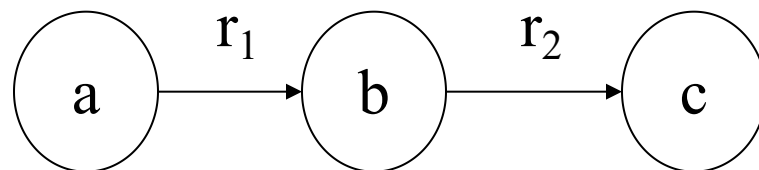


替换规则

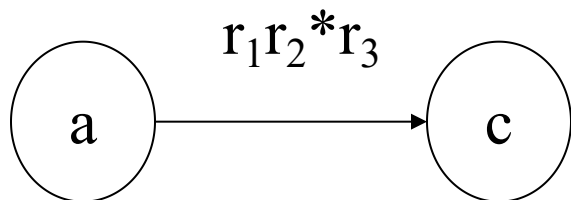
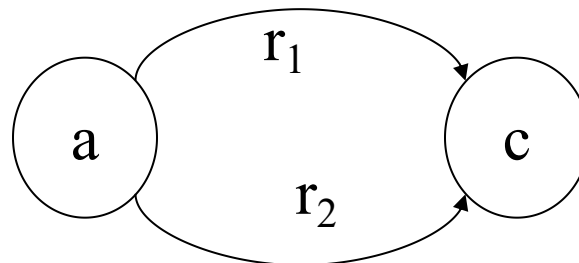
从正规式到有限自动机



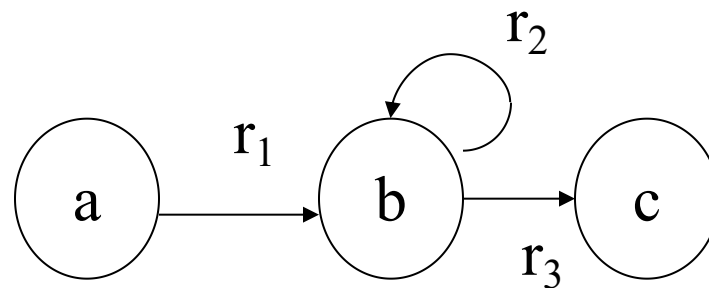
代之以



代之以

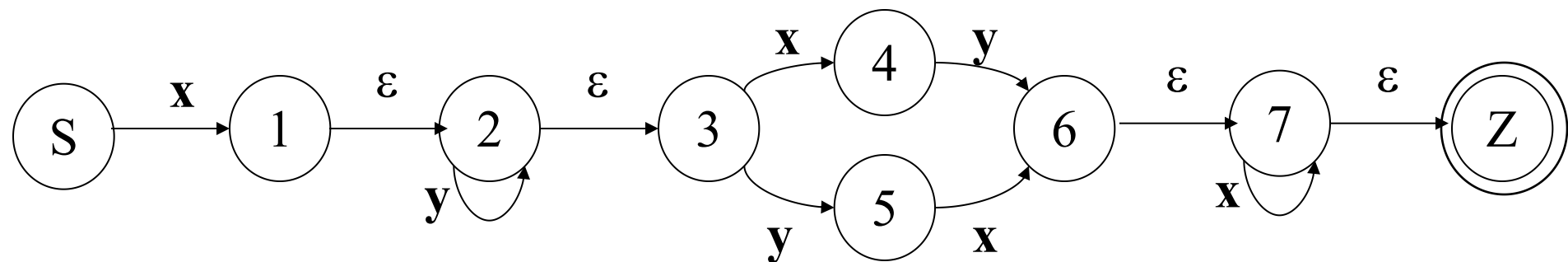
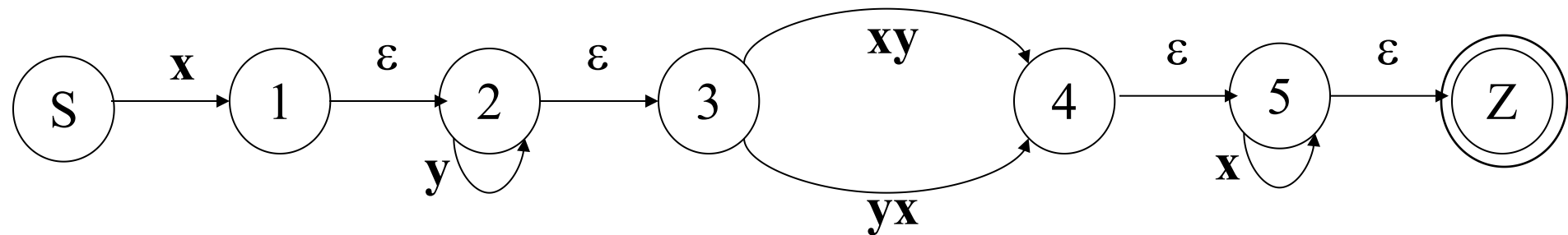
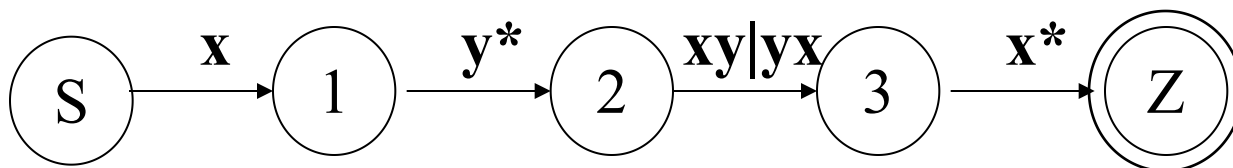
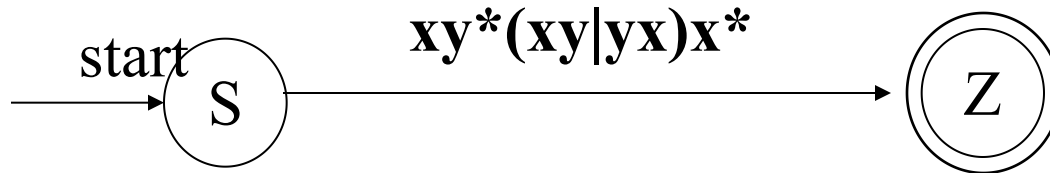


代之以





设 $\Sigma=\{x,y\}$, Σ 上的正规式 $R=xy^*(xy|yx)x^*$, 构造一个NFA M , 使 $L(M)=L(R)$.



转换法



从正规式到有限自动机

从有限自动机到正规式

对于 Σ 上任一NFA m ，能构造 Σ 上一个正规表达式 r ，使得 $L(r)=L(m)$ 。

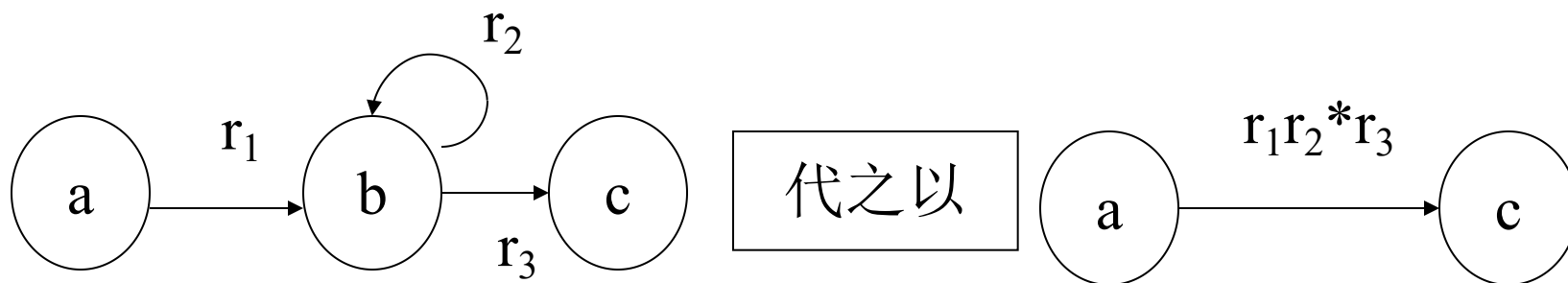
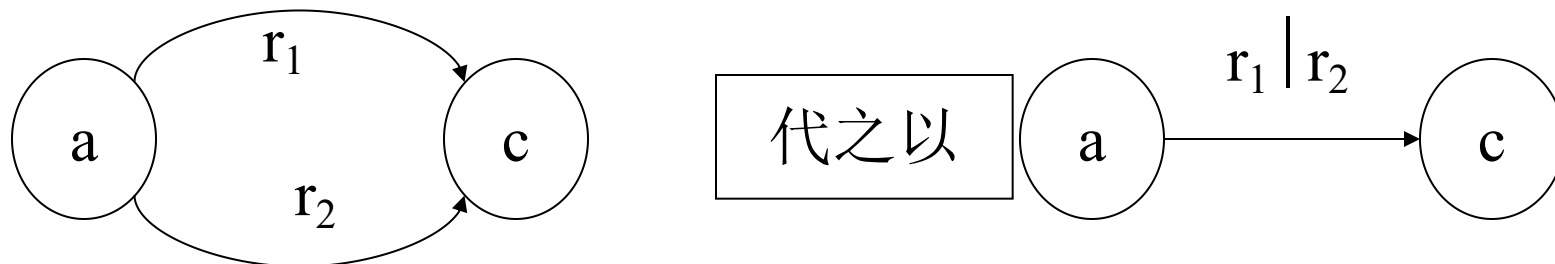
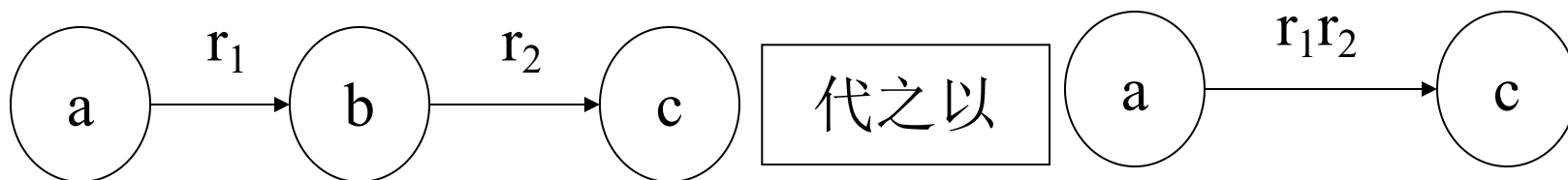
首先，在 m 的转换图上加进 S, Z 两个结点。从 S 用 ϵ 弧连接到 m 的所有初态结点，从 m 的所有终结（接受）结点用 ϵ 弧连接到 Z ，从而构成一个新的NFA m' ， $L(m)=L(m')$ 。

反复使用下面的替换规则逐步消去NFA m' 中的状态结点和弧，直至剩下 S, Z 为止。在消去结点的过程中，逐步用正规式标记弧。

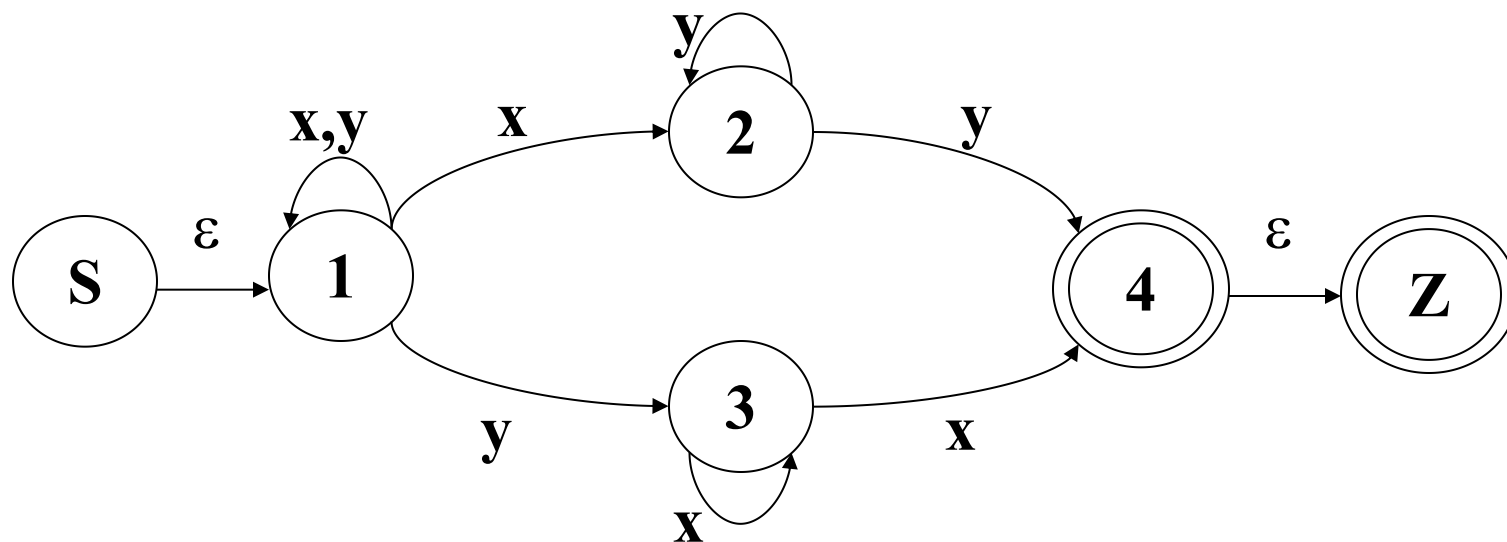
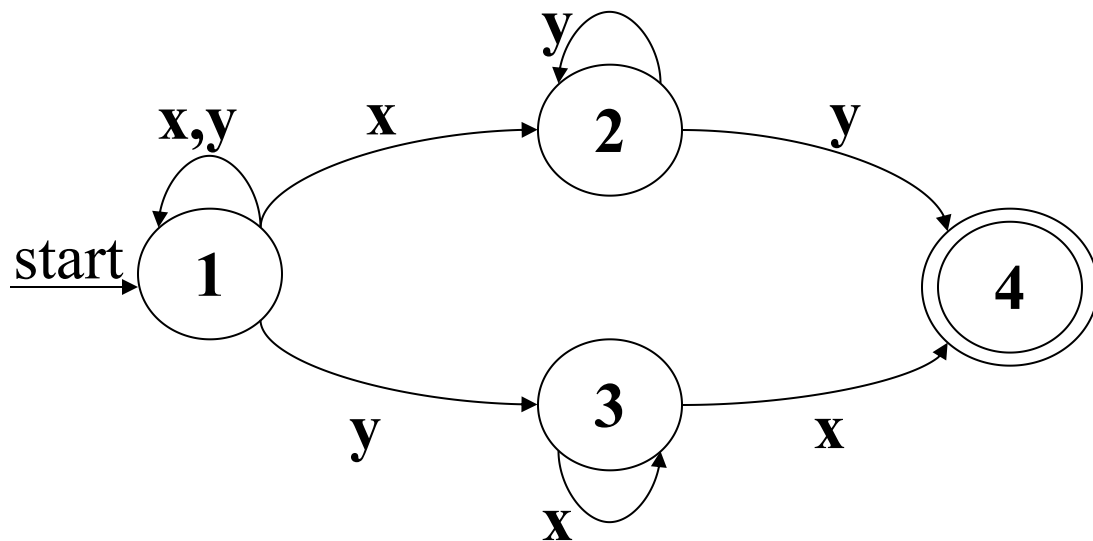


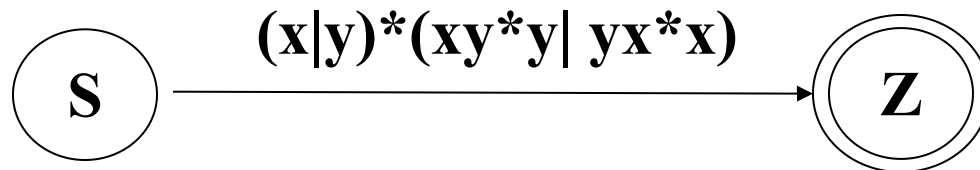
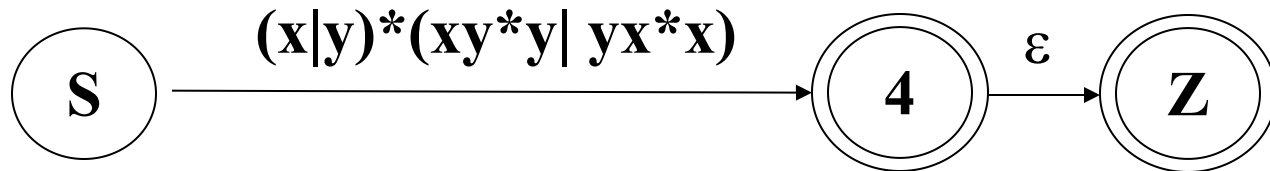
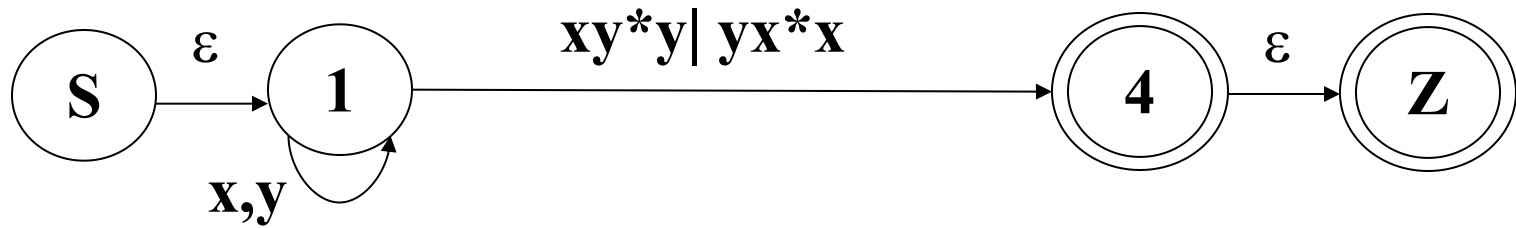
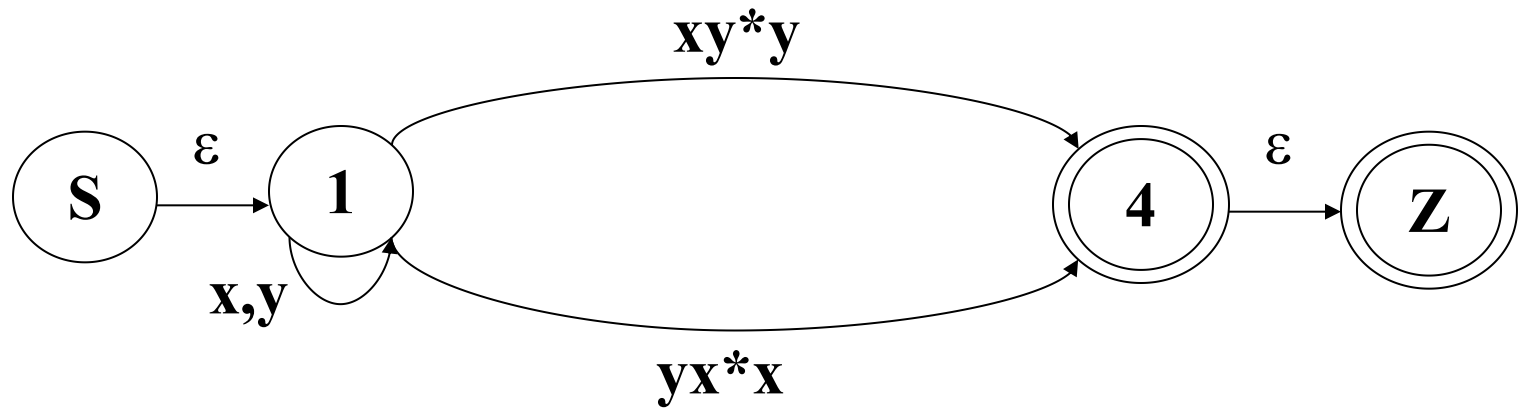
替换规则

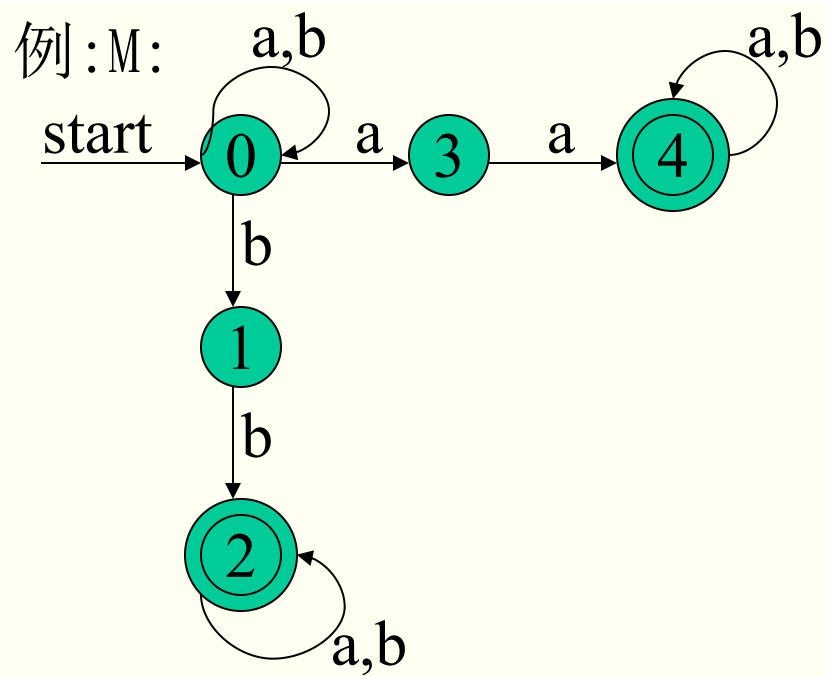
从有限自动机到正规式



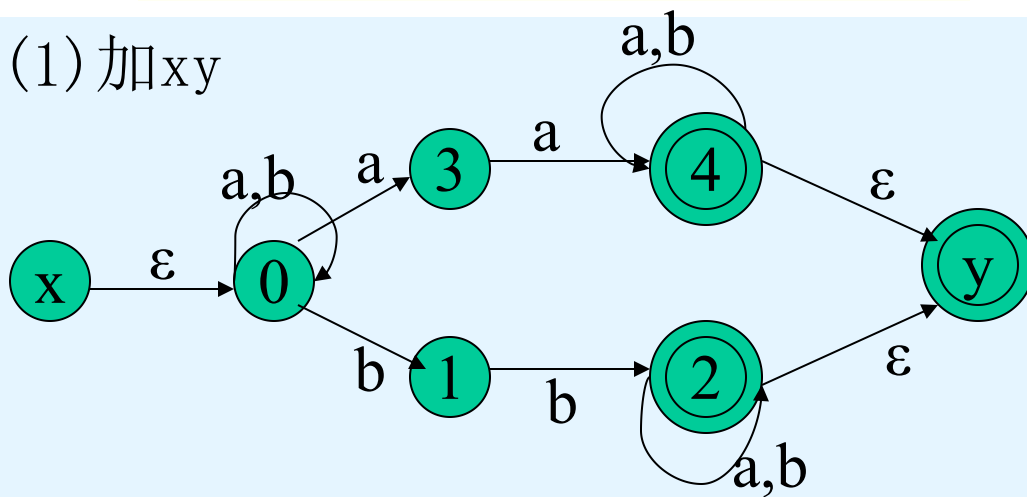
例： NFA M 如图所示，在 $\Sigma=\{x,y\}$ 上构造一个正规式 R ，使 $L(M)=L(R)$ 。



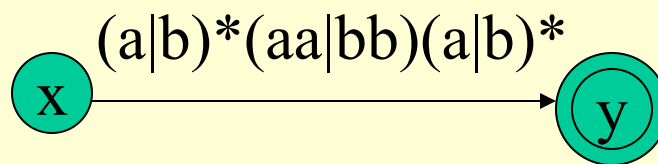
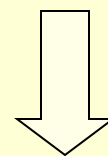
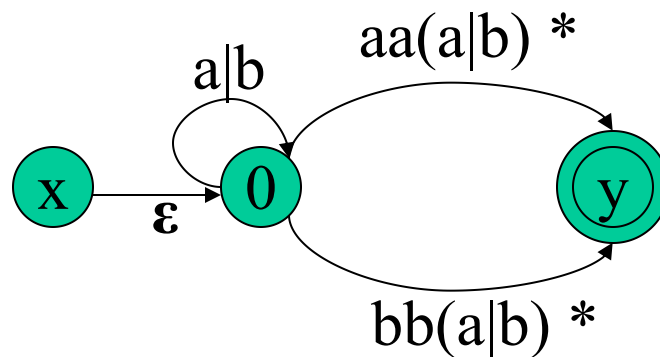
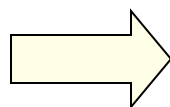
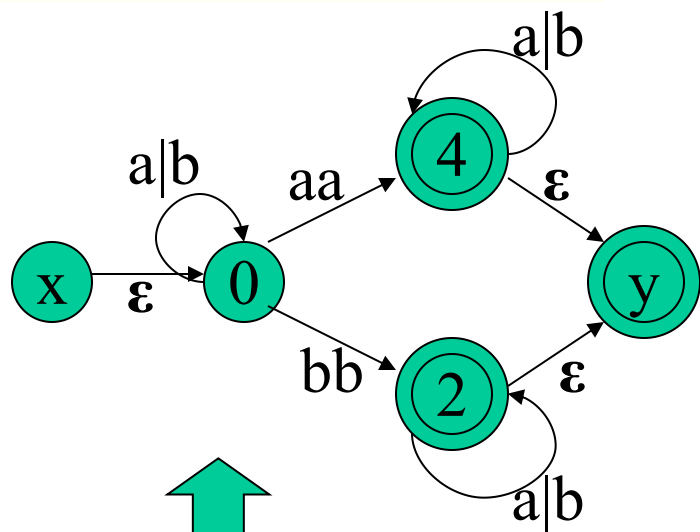




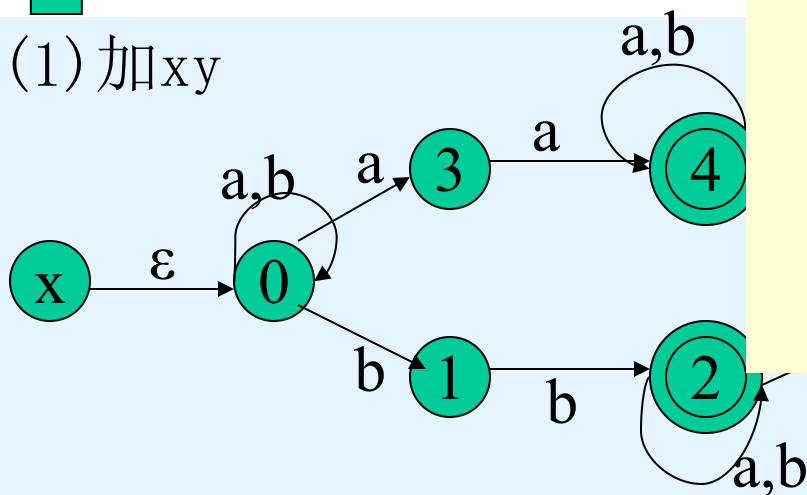
解: (1) 加xy



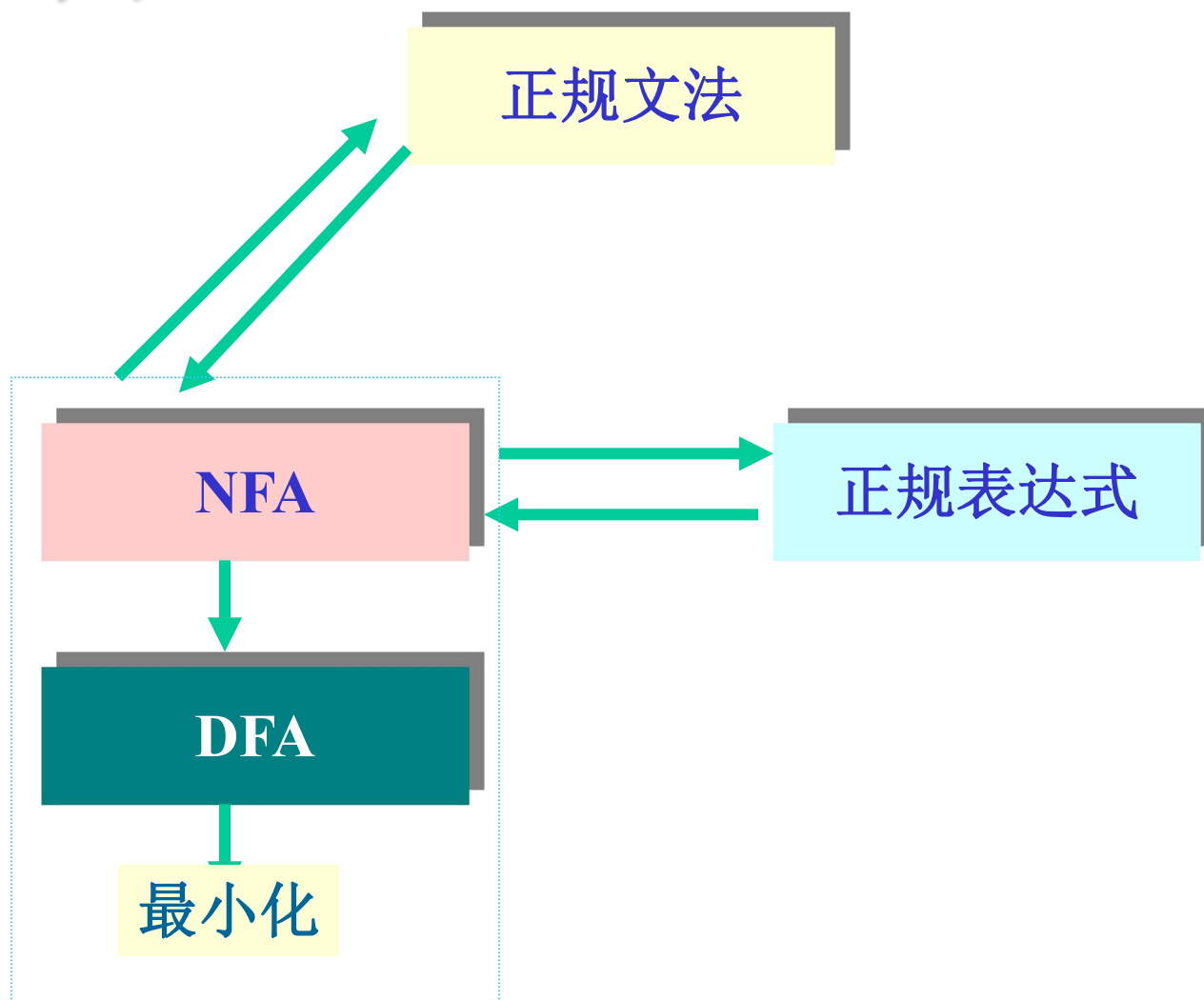
(2) 消除M'中的所有结点



解: (1) 加xy



关系图:



正规文法与有限自动机（FA）的等价性

如果对于某个正规文法G和某个有限自动机M, 有 $L(G)=L(M)$ （正规文法G所产生的语言和某个有限自动机所识别的语言相同）,则称G和M是等价的。

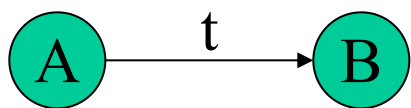
定理 对每一个右线性正规文法或左线性正规文法 G ，都存在一个FA M ，使 $L(M)=L(G)$

定理 对于每一个FA M ，都存在一个右线性正规文法 G 和一个左线性正规文法 G' ，使
$$L(M)=L(G)=L(G')。$$

有限自动机与正规文法

(1) 有限自动机 \Rightarrow 正规文法（右线性）

算法:

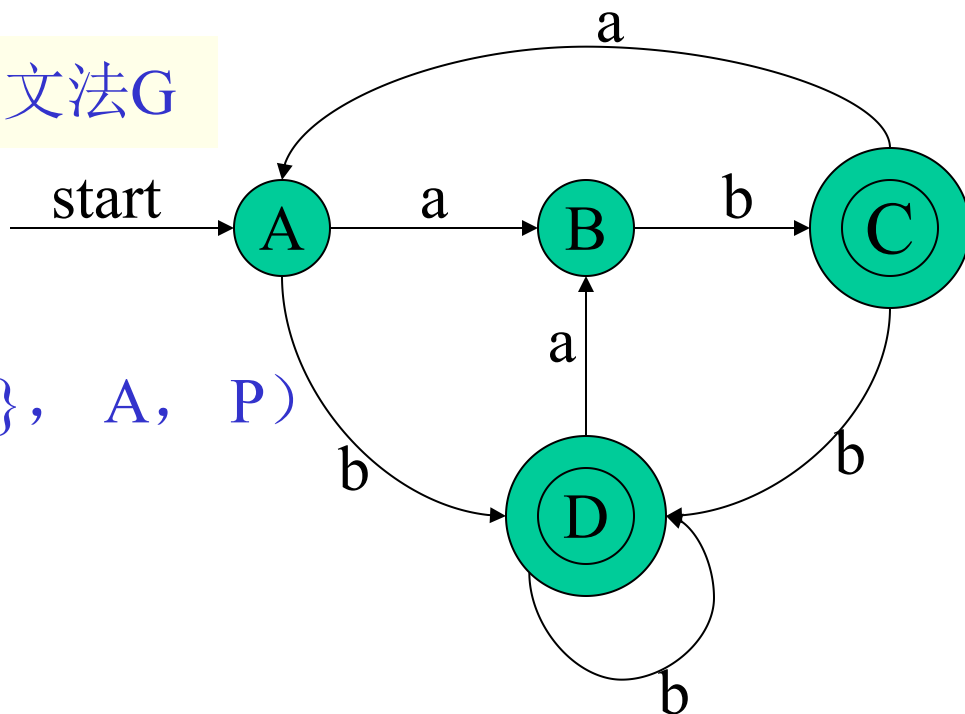


1.对转换函数 $f(A,t)=B$, 可写成一个产生式: $A \rightarrow tB$

2.对可接受状态Z,增加一个产生式: $Z \rightarrow \varepsilon$

3.有限自动机的初态对应于文法的开始符号,
有限自动机的字母表为文法的终结符号集.

例:给出如图FA等价的正规文法G



$G = (\{A, B, C, D\}, \{a, b\}, A, P)$

其中P: $A \rightarrow aB$

$A \rightarrow bD$

$B \rightarrow bC$

$C \rightarrow aA$

$C \rightarrow bD$

$C \rightarrow \varepsilon$

$D \rightarrow aB$

$D \rightarrow bD$

$D \rightarrow \varepsilon$



(2) 正规文法（右线性） \Rightarrow 有限自动机M

算法:

- 1.字母表与G的终结符号相同.
- 2.为G中的每个非终结符生成M的一个状态,G的开始符号S是开始状态S;
- 3.增加一个新状态Z,作为NFA的终态
- 4.对G中的形如 $A \rightarrow tB$,其中t为终结符或 ε ,A和B为非终结符的产生式,构造M的一个转换函数 $f(A,t)=B$
- 5.对G中的形如 $A \rightarrow t$ 的产生式,构造M的一个转换函数 $f(A,t)=Z$

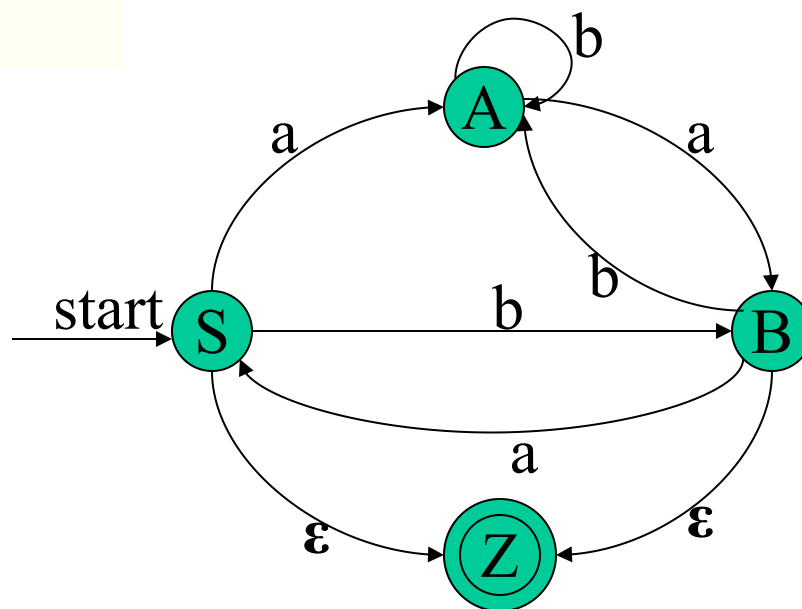
例:求与文法G[S]等价的NFA

G[S]: $S \rightarrow aA | bB | \epsilon$

$A \rightarrow aB | bA$

$B \rightarrow aS | bA | \epsilon$

求得:



课后思考

- 有限自动机 \Rightarrow 正规文法（左线性）？

补充

正规文法 \Rightarrow 正规式（自学）

利用以下转换规则,直至只剩下一个开始符号定义的产生式,并且产生式的右部不含非终结符.

规则	文法产生式	正则式
规则1	$A \rightarrow xB, B \rightarrow y$	$A = xy$
规则2	$A \rightarrow xA \mid y$	$A = x^*y$
规则3	$A \rightarrow x, A \rightarrow y$	$A = x \mid y$

例:有文法G[s]

$$S \rightarrow aA | a$$

$$A \rightarrow aA | dA | c$$

于是: $S = aA | a$

$$A = (aA | dA) | c \Rightarrow A = (a | d)A | c$$

由规则二: $A = (a | d)^* c$

$$\text{代入: } S = a(a | d)^* c | a$$

正规式 \Rightarrow 正规文法

算法:

1) 对任何正规式 r , 选择一个非终结符 S 作为识别符号.

并构造产生式 $S \rightarrow r$

2) 若 x, y 是正规式,

$A \rightarrow xy$ 重写为 $A \rightarrow xB \quad B \rightarrow y$

$A \rightarrow x*y$ $A \rightarrow xA \quad A \rightarrow y$

其中 B 为新的非终结符, $B \in V_N$

例:将 $a(a|d)^*$ 转换成等价的正规文法

解:1) $S \rightarrow a(a|d)^*$

2) $S \rightarrow aA$

$A \rightarrow (a|d)^*$

3) $S \rightarrow aA$

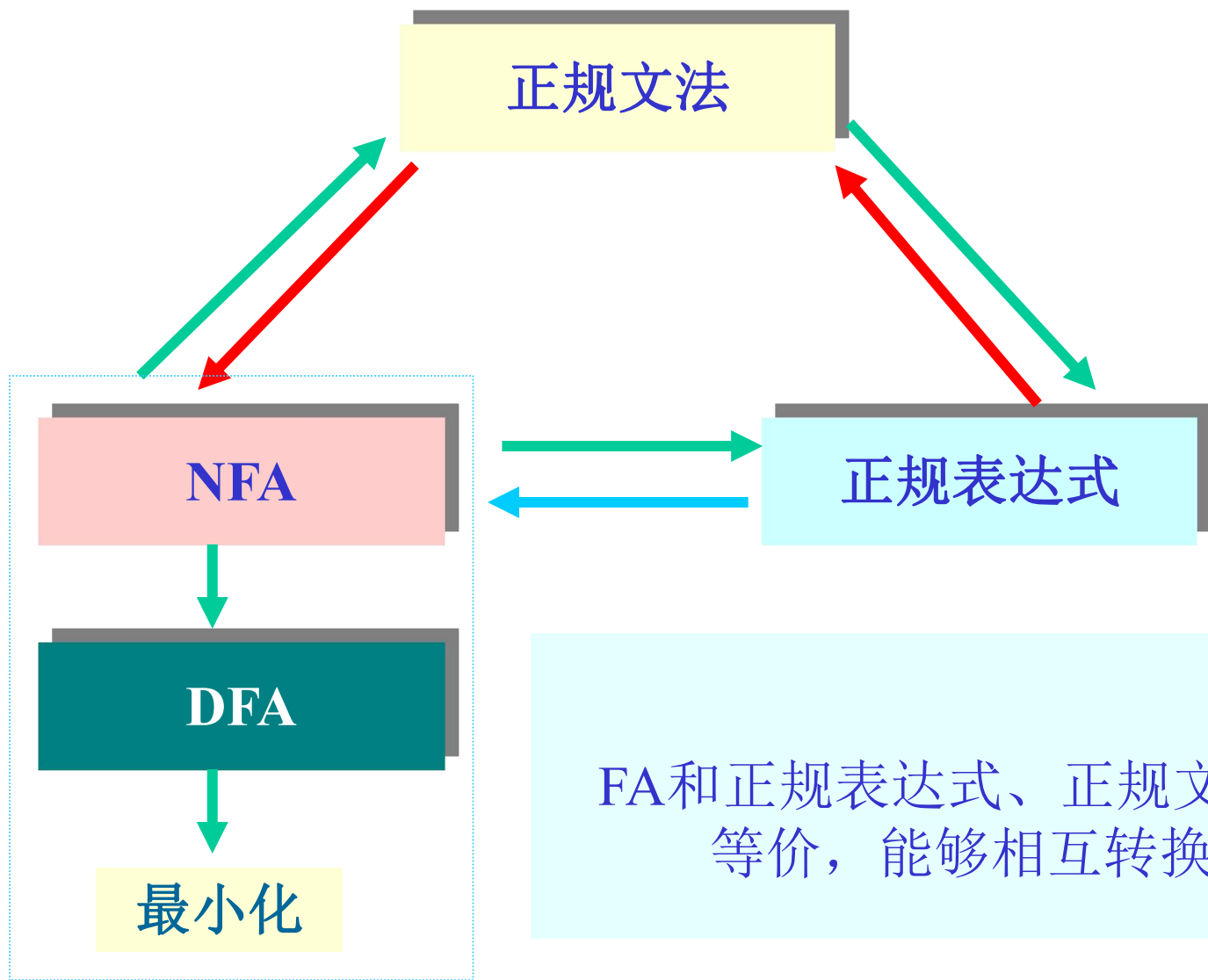
$A \rightarrow (a|d)A$

$A \rightarrow \epsilon$

4) $S \rightarrow aA$

$A \rightarrow aA|dA$

$A \rightarrow \epsilon$



FA和正则表达式、正规文法相互等价，能够相互转换.

课堂练习1： 给出下面正规表达式

(1) 以**01**结尾的二进制数串。

(2) 能被**5**整除的十进制数。

(3) **{0,1}**上的含有子串**010**的所有串。

(4) 英文字母组成的所有符号串，要求符号串中的字母按字典序排列。

课堂练习1： 给出下面正规表达式

(1) 以01结尾的二进制数串。

$(0 \mid 1)^* 01$

(2) 能被5整除的十进制数。

$0 \mid 5 \mid (1 \mid 2 \mid 3 \mid 4 \mid 5 \mid 6 \mid 7 \mid 8 \mid 9) (0 \mid 1 \mid 2 \mid 3 \mid 4 \mid 5 \mid 6 \mid 7 \mid 8 \mid 9)^* (0 \mid 5)$

(3) $\{0,1\}$ 上的含有子串010的所有串。

$(0 \mid 1)^* 010 (0 \mid 1)^*$

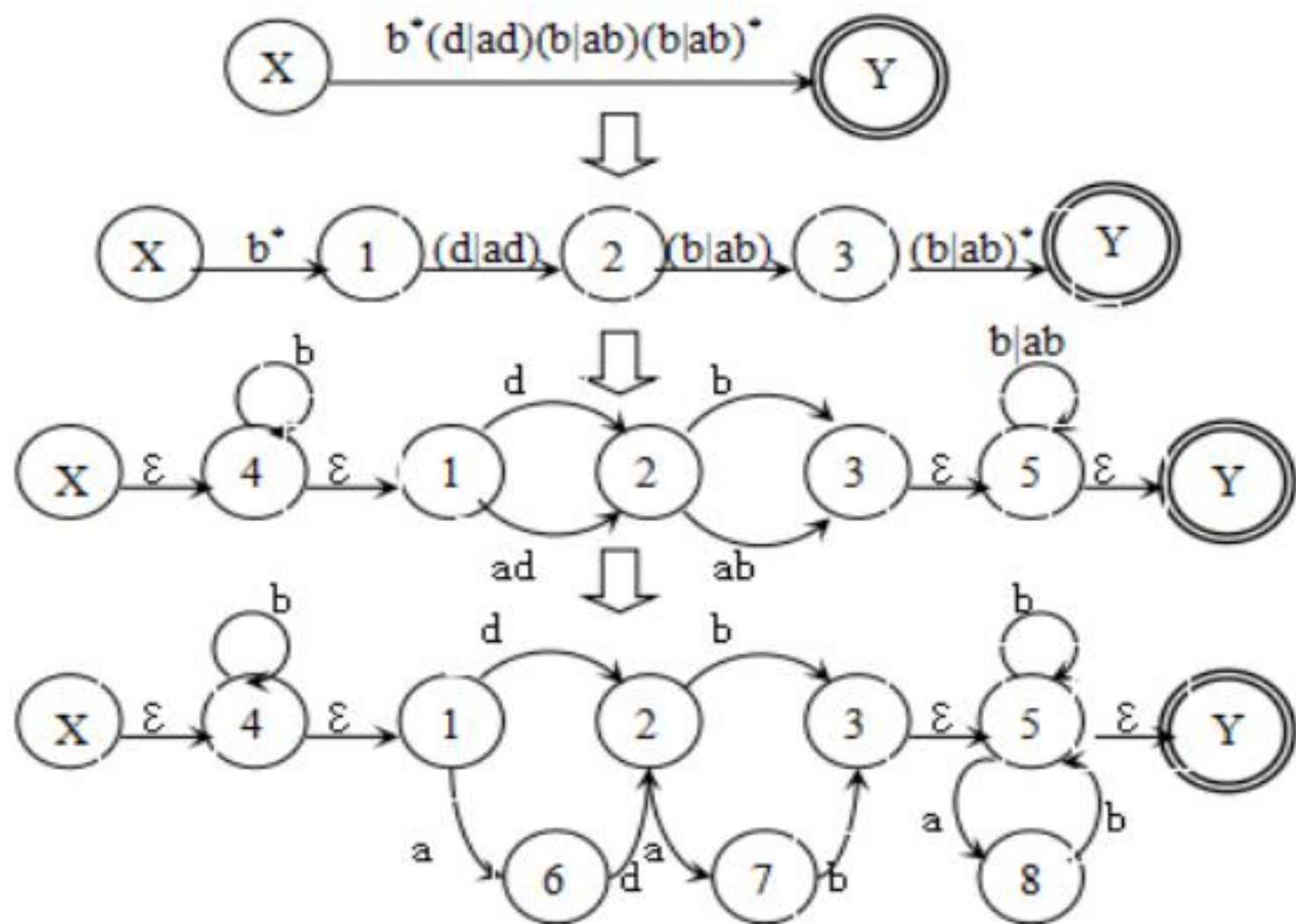
(4) 英文字母组成的所有符号串，要求符号串中的字母按字典序排列。

$(A \mid a)^* (B \mid b)^* (C \mid c)^* \dots (Z \mid z)^*$

课堂练习2:

对给定正规式 $b^*(d|ad)(b|ab)^+$,

构造其NFA M ;




课堂练习3：识别各进制无符号整数的 DFA

词法规则 \longrightarrow 正规表达式 \longrightarrow NFA \longrightarrow DFA(确定化、最简化)

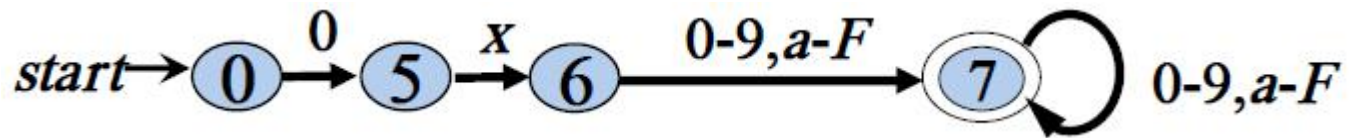
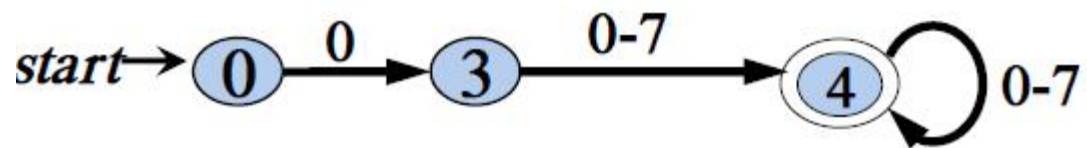
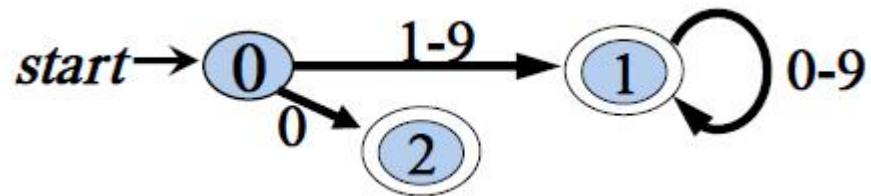
思考：

- 十进制整数 (DEC) : 0-9数字串, 高位不为0
- 八进制整数 (OCT) : 以0打头的0-7数字串
- 十六进制整数 (HEX) : 以0x打头的0-9, a-f的数字字母串

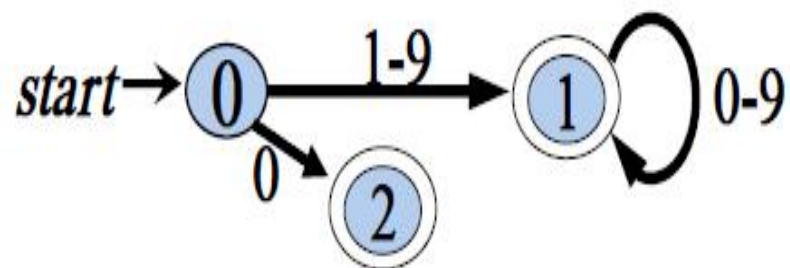
课堂练习3：识别各进制无符号整数的 DFA

词法规则  正规表达式

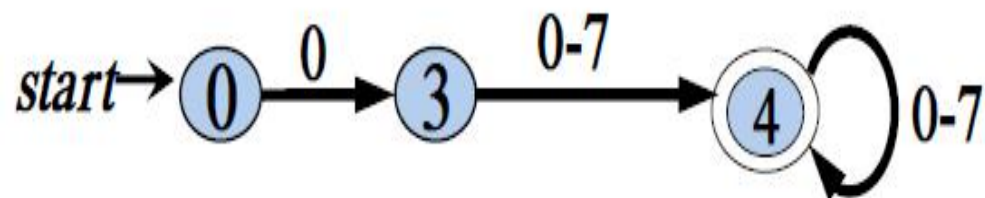
- $DEC \rightarrow (1|...|9)(0|...|9)^* | 0$
- $OCT \rightarrow 0(0|1|2|3|4|5|6|7)(0|1|2|3|4|5|6|7)^*$
- $HEX \rightarrow$
 $0x(0|1|...|9|a|...|f|A|...|F)(0|...|9|a|...|f|A|...|F)^*$



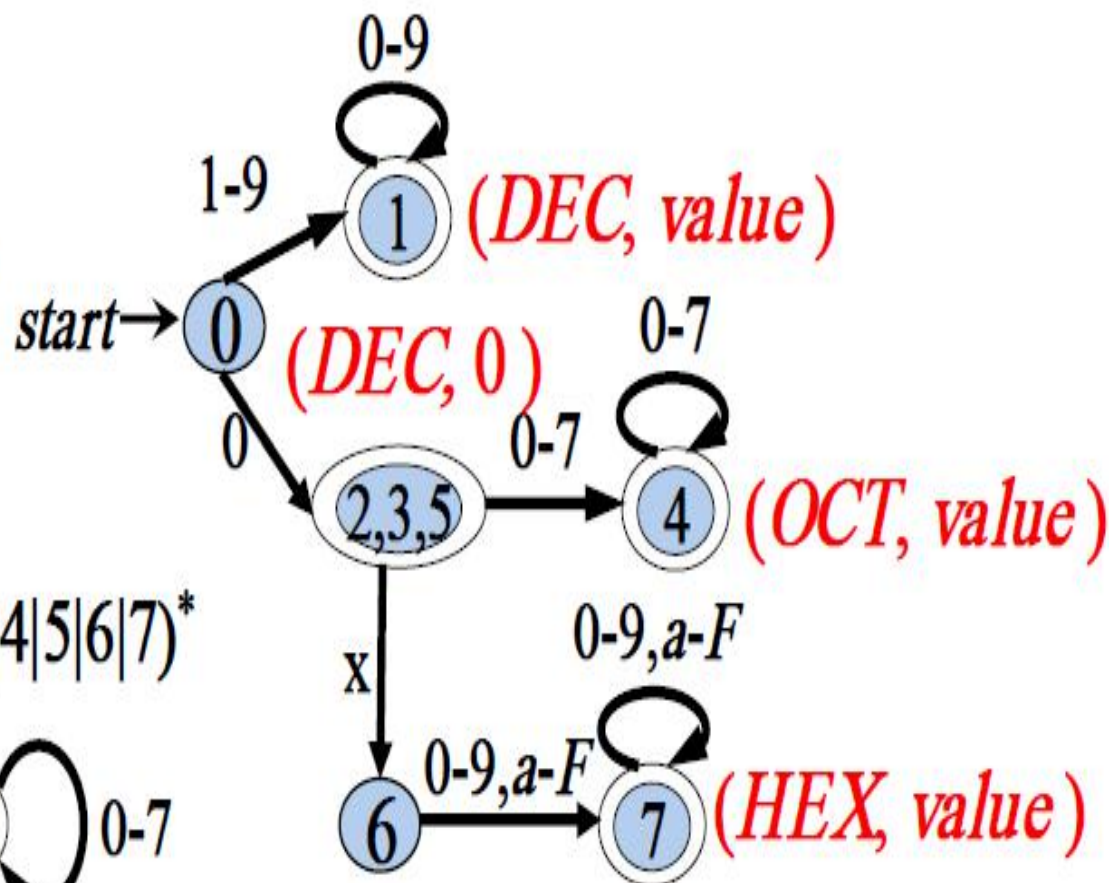
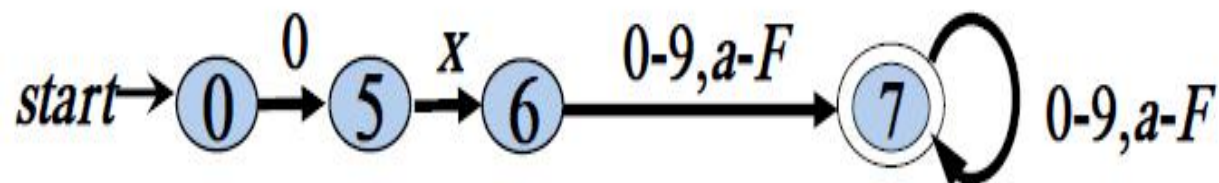
$DEC \rightarrow (1|...|9)(0|...|9)^*|0$



$OCT \rightarrow 0(0|1|2|3|4|5|6|7)(0|1|2|3|4|5|6|7)^*$



$HEX \rightarrow 0x(0|1|...|9|a|...|f|A|...|F)(0|...|9|a|...|f|A|...|F)^*$



课堂练习4

（整型或浮点型）无符号数的正规表示。

例如：

3

3.15

3.15E+4

3.15E-4

3.15E4

3E-4

课堂练习4

（整型或浮点型）无符号数的正规表示。

例如：

3	3. 15	3. 15E+4
3. 15E-4	3. 15E4	3E-4

$(0|1|2|\dots|9)(0|1|2|\dots|9)^*$ $(.(0|1|2|\dots|9)(0|1|2|\dots|9)^*|\epsilon)$ $(E(+|-|\epsilon)(0|1|2|\dots|9)(0|1|2|\dots|9)^*|\epsilon)$

数字串

是否有小数点

是否科学计数法

再复杂一点

$(1|2|\dots|9)(0|1|2|\dots|9)^*|0$

利用正则定义改写复杂的正规式

- 正则定义是具有如下形式的定义序列：

$d1 \rightarrow r1$

$d2 \rightarrow r2$

...

$dn \rightarrow rn$

给一些正规式命名，并在之后的正规式中像使用字母表中的符号一样使用这些名字

其中：

1. 每个 d_i 都是一个新符号，它们都不在字母表 Σ 中，而且各不相同
2. 每个 r_i 是字母表 $\Sigma \cup \{d1, d2, \dots, d_{i-1}\}$ 上的正则表达式

课堂练习4： 识别无符号数的 DFA

$\text{digit} \rightarrow 0|1|2|\dots|9$

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

$\text{optionalFraction} \rightarrow .\text{digits}|\epsilon$

$\text{optionalExponent} \rightarrow (E(+|-|\epsilon)\text{digits})|\epsilon$

$\text{number} \rightarrow \text{digits}\textcolor{red}{\text{optionalFraction}}\text{optionalExponent}$

例如：用正则定义重写C语言标识符

- $\text{digit} \rightarrow 0|1|2|\dots|9$
- $\text{letter_} \rightarrow A|B|\dots|Z|a|b|\dots|z|_$
- $\text{id} \rightarrow \text{letter_}(\text{letter_}|\text{digit})^*$

- 进一步：识别无符号数的 *DFA*

$\text{digit} \rightarrow 0|1|2|\dots|9$

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

$\text{optionalFraction} \rightarrow .\text{digits}|\varepsilon$

$\text{optionalExponent} \rightarrow (E(+|-|\varepsilon)\text{digits})|\varepsilon$

$\text{number} \rightarrow \text{digits}\text{optionalFraction}\text{optionalExponent}$

完整正规式: $dd^*(.dd^*|\varepsilon)((E(+|-|\varepsilon)dd^*)|\varepsilon)$

- 进一步：识别无符号数的 *DFA*

$\text{digit} \rightarrow 0|1|2|\dots|9$

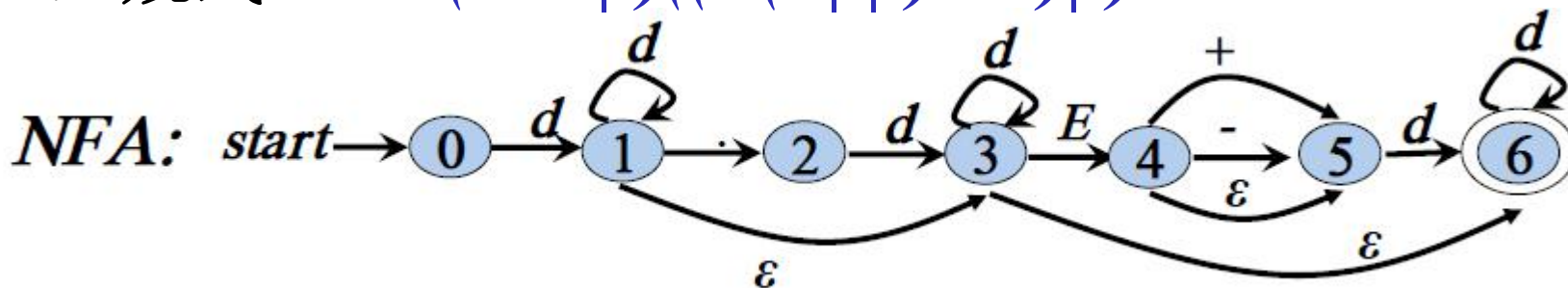
$\text{digits} \rightarrow \text{digit digit}^*$

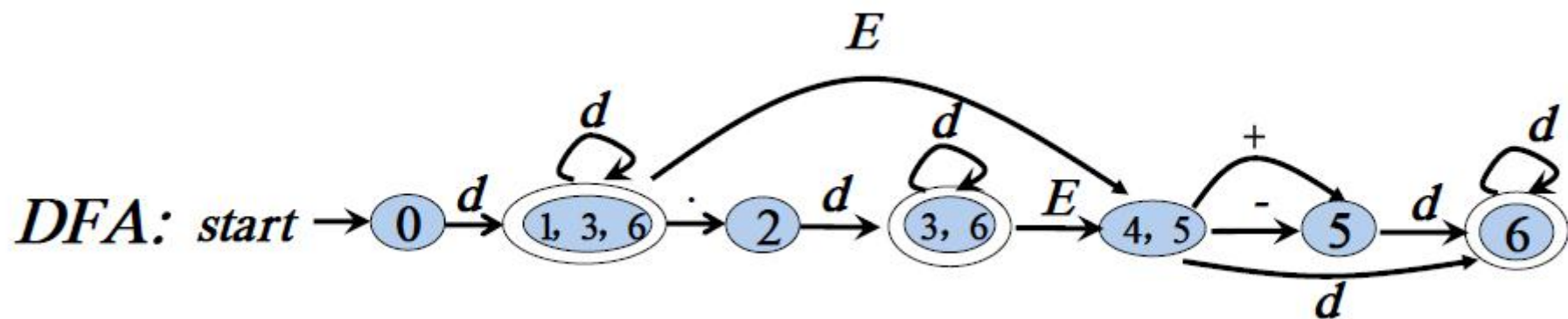
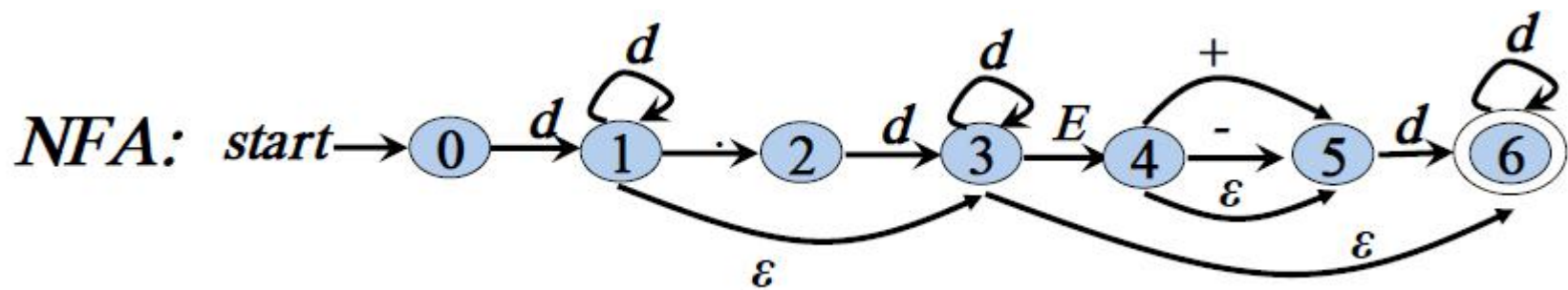
$\text{optionalFraction} \rightarrow .\text{digits}|\epsilon$

$\text{optionalExponent} \rightarrow (E(+|-|\epsilon)\text{digits})|\epsilon$

$\text{number} \rightarrow \text{digits}\text{optionalFraction}\text{optionalExponent}$

正规式: $dd^*(.dd^*|\epsilon)((E(+|-|\epsilon)dd^*)|\epsilon)$





3.3 词法分析程序的设计与实现

词法规则 \longrightarrow 正规表达式 \longrightarrow 状态转换图（最小DFA）

1.构造识别单词的状态转换图

- (1)对程序语言的单词按种类分别构造对应的状态转换图.
- (2)对各类转换图合并，构成一个能识别语言所有单词的状态转换图.

2.编程实现状态转换图 ✨



2.编程实现状态转换图



根据有限自动机实现词法分析器一般有两种方式：

（1）基于表驱动的方式

（2）硬编码方式

1.基于表驱动的词法分析

以标识符为例的状态转移矩阵

状态	-	a-z	A-Z	0-9	其他
0	id	id	id	数字	其他
id	id	id	id	id	accept

主要动作：

- ①查询状态转移矩阵
- ②状态比较
- ③状态处理

1.基于表驱动的词法分析伪代码

```
cur_state=0;
cur_char=GetNextChar()
while(1){
    next_state=table[cur_state,cur_char]; //查表获取下一个状态
    if(next_state==accept){
        return PROCESS_ACCEPT(cur_state); //处理接受状态
    }
    else if(next_state==error){
        return ERROR_PROCESS(); //错误处理
    }
    else{
        HANDLE_STATE(cur_state,cur_char); // 处理当前状态
        cur_state=next_state; // 进入下一个状态
        cur_char=GetNextChar(); // 获取下一个字符
    }
}
```

1.基于表驱动的词法分析

优缺点：

- ①实现简单；
- ②需要大量的存储空间；
- ③灵活性较差；

词法分析器的自动生成工具一般都采用表驱动的方法。

2.硬编码方式的词法分析

硬编码方式的词法分析不需要显式地确立有限自动机的状态，它使用程序的控制结构直接对词法分析进行解析，即根据词法记号本身的含义，使用代码解析词法记号的内容。

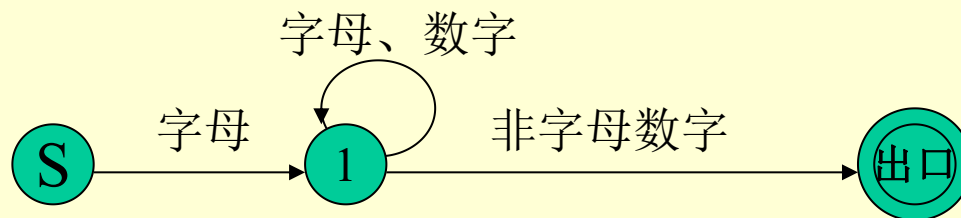
```
if(ch==下划线or字母){  
    ch==GetNextChar();  
    while(ch==下划线or字母or数字){  
        ch=GetNextChar();  
    }  
}
```

3.3.1 词法及其状态转换图

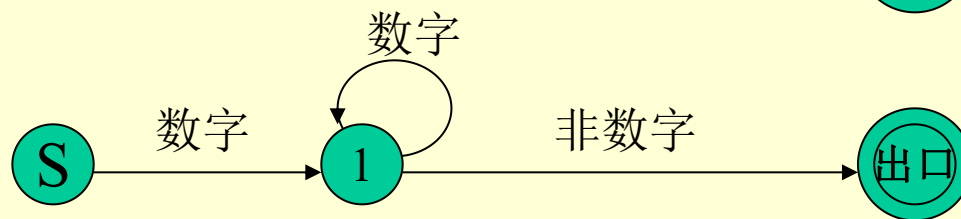
例1 假定语言X的字母表 $\Sigma = \{A-Z, a-z, 0-9, ;, =\}$
单词符号定义如下：

- 1、标识符：字母打头的字母数字串
- 2、无符号整数：无符号数字串
- 3、分界符： $;$
- 4、运算符： $=$

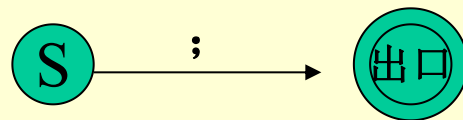
标识符



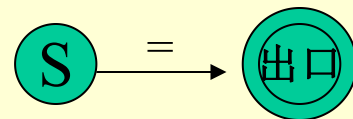
无符号整数

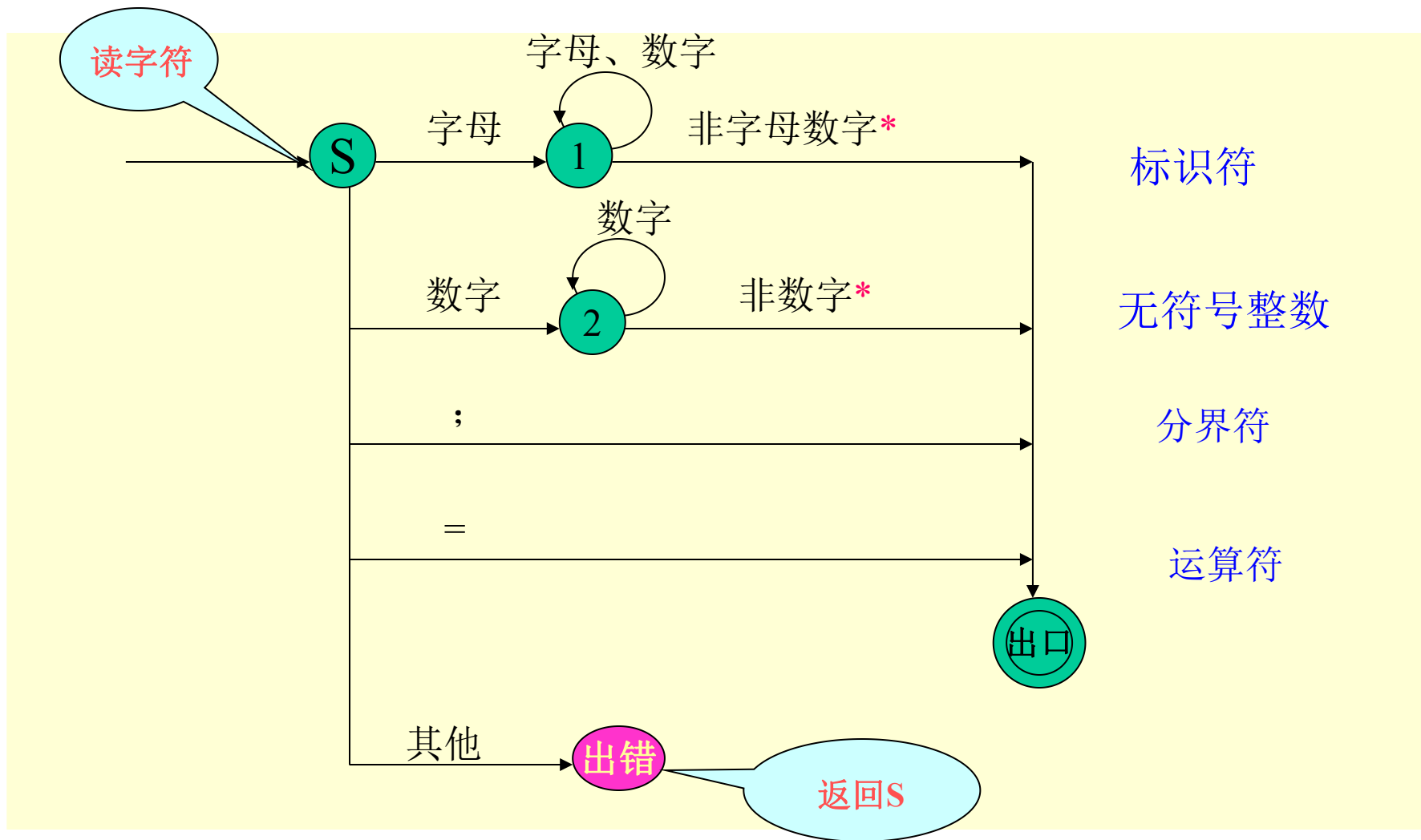


分界符



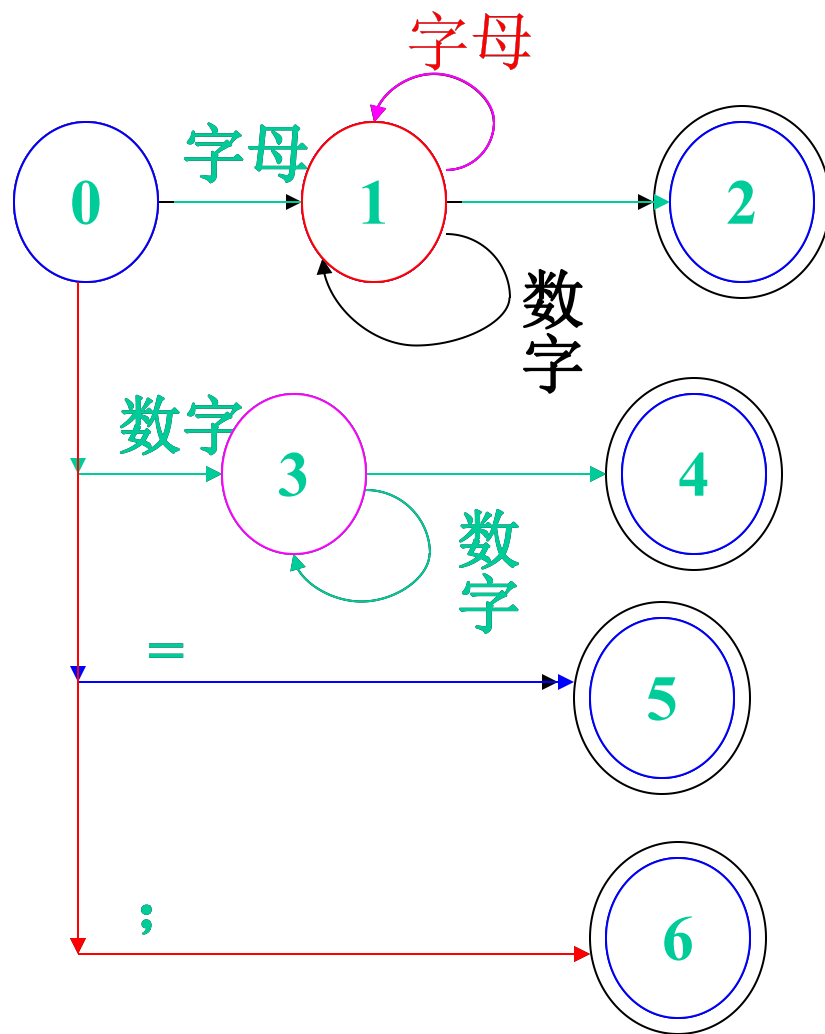
运算符





L	i	n	e	=	8	0	;
---	---	---	---	---	---	---	---

输入



输出

⟨ 1, 'Line' ⟩

⟨ 3, '=' ⟩

⟨ 2, '80' ⟩

⟨ 4, ';' ⟩

3.3.2 状态转换图的实现——构造词法分析程序

例1 假定语言X的字母表 $\Sigma = \{A-Z, a-z, 0-9, ;, =\}$
单词符号定义如下：

- 1、标识符：字母打头的字母数字串
- 2、无符号整数：无符号数字串
- 3、分界符： $;$
- 4、运算符： $=$

标识符

<1, 标识符名字>

分界符

<3, “.” >

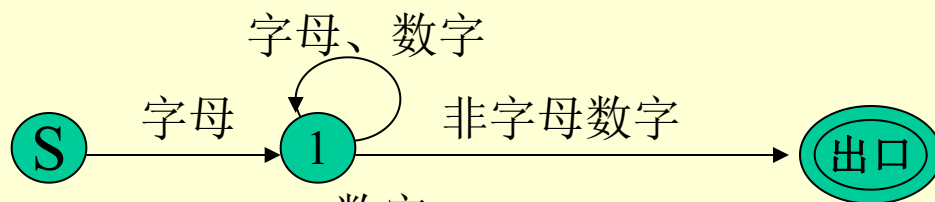
无符号整数

<2, 整数值>

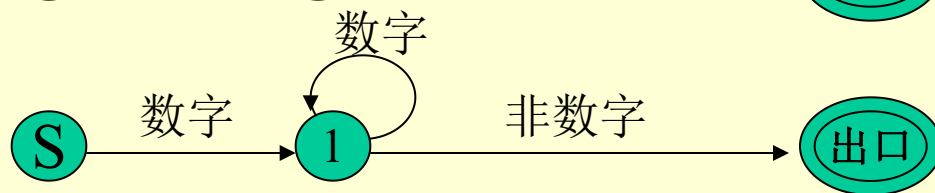
运算符

<4, “=” >

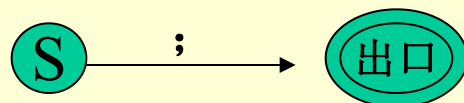
标识符



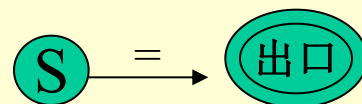
无符号整数



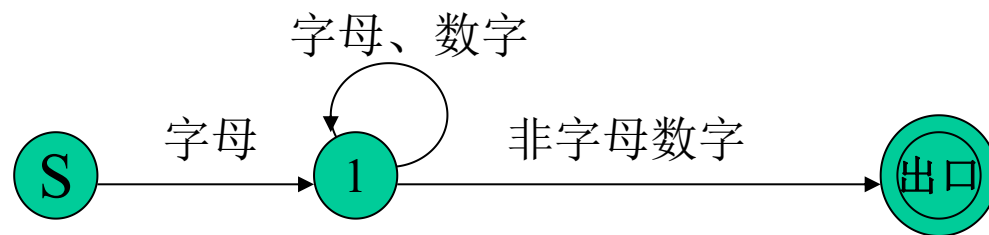
分界符



运算符



标识符



L	i	n	e	=	8	0	;
---	---	---	---	---	---	---	---

If (ISLETTER)

{

WHILE (ISLETTER OR ISDIGIT) DO

{

当前字符放入一临时字符数组;

GETNEXTCHAR ; //从缓冲区取下一字符

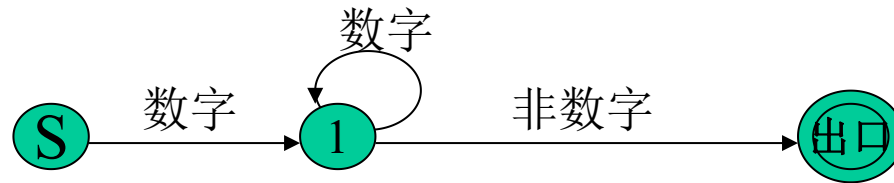
};

UNGETCH; //回退一字符

OUTPUT(1, 标识符名字);

};

无符号整数



L	i	n	e	=	8	0	;
---	---	---	---	---	---	---	---

If (ISDIGIT)

{

WHILE ISDIGIT DO

{

当前字符放入一临时字符数组;

GETNEXTCHAR ; //从缓冲区取下一字符

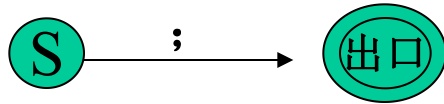
};

UNGETCH; //回退一字符

OUTPUT (2, 整数);

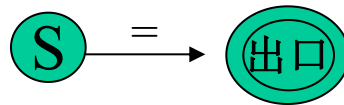
};

分界符

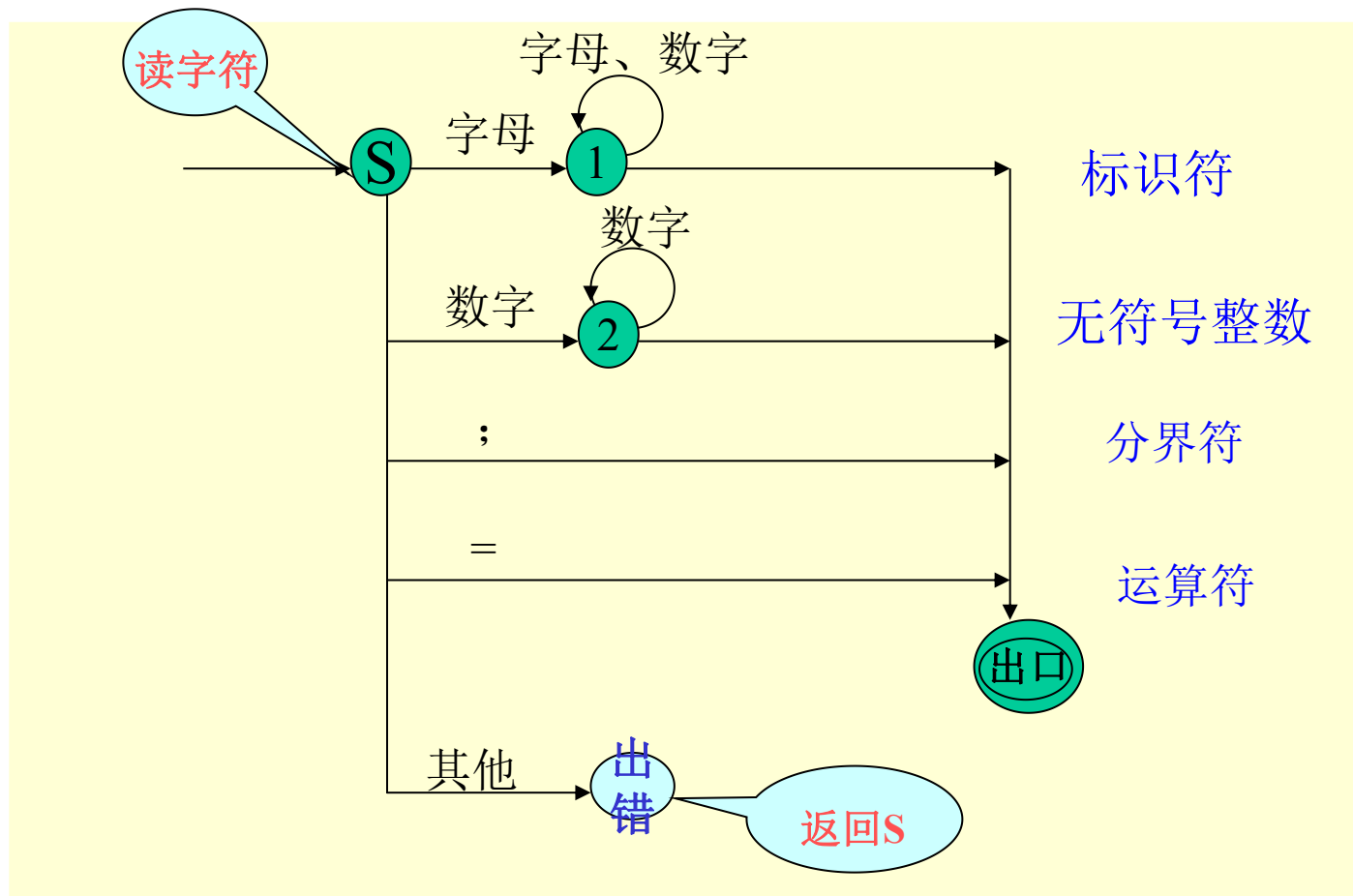


If (CH==';') OUTPUT (3 , " ; ");

运算符



If (CH=='=') OUTPUT (4 , " =");



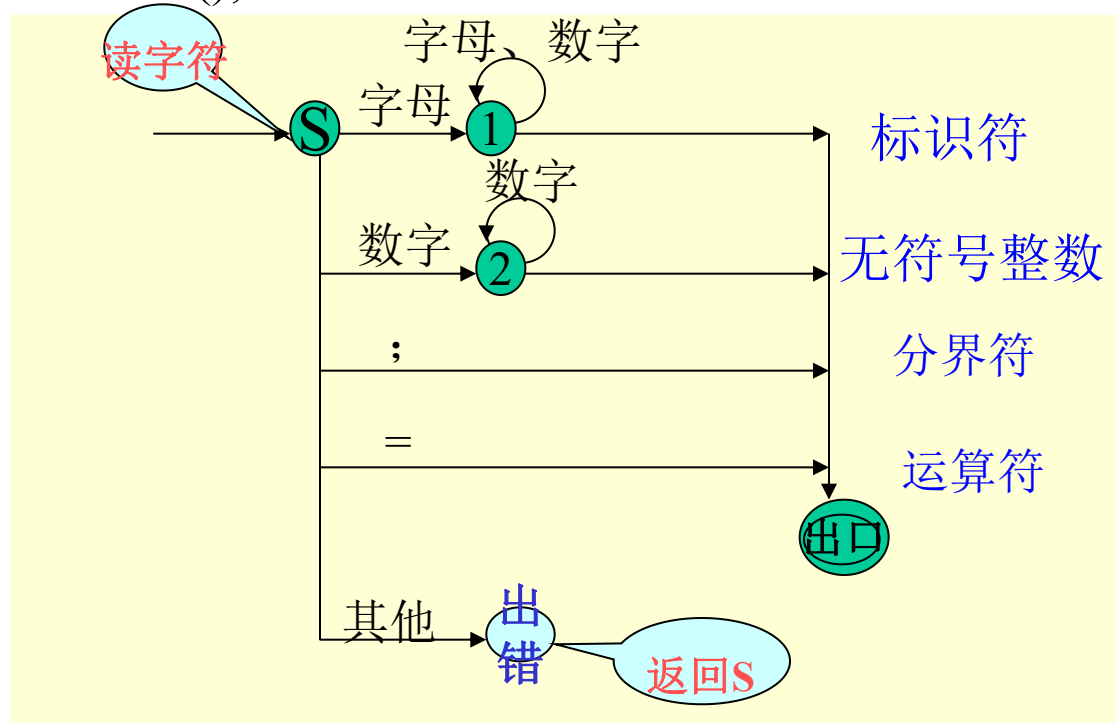
例1程序语言的词法分析程序

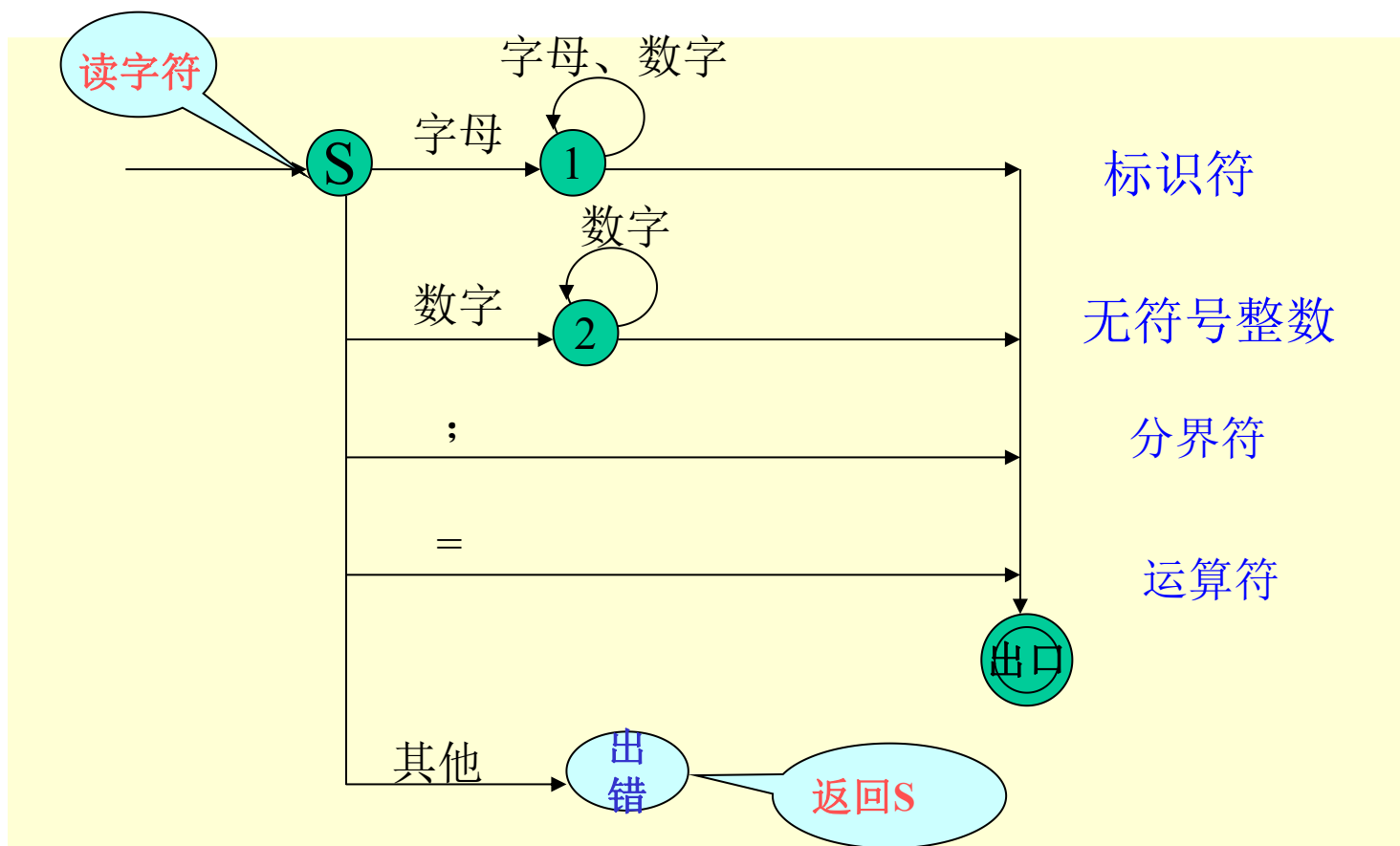
```
GETNEXTCHAR( ) ;
SWITCH(CHCODE);
{
CASE 1: { WHILE (ISLETTER OR ISDIGIT) DO
        {
            SAVE( ); // 当前字符放入一临时字符数组;
            GETNEXTCHAR( ) ; //从缓冲区取下一字符
        };
        UNGETCH; //回退一字符
        OUTPUT(1, 标识符名字);
    }; BREAK;
CASE 2: { WHILE ISDIGIT DO
        {
            SAVE( ); // 当前字符放入一临时字符数组;
            GETNEXTCHAR ; //从缓冲区取下一字符
        };
        UNGETCH; //回退一字符
        OUTPUT (2, 整数);
    }; BREAK;
```

CASE 3 : OUTPUT (3 , “;”); BREAK;



CASE 4 : OUTPUT (4 , “=”); BREAK;

DEFAULT: Error();






采用面向对象的方法设计词法分析器

词法规则  正规表达式  状态转换图（最小DFA）

 合并状态转换图

合并状态转换图  编程实现状态转换图

文法、正规表达式、有限自动机

有限自动机(最小化?)

词法分析阶段的错误处理

- 词法分析阶段可检测错误的类型
 - 单词拼写错误
 - 非法字符
 - 字符串丢失右引号
 - 字符丢失右单引号
 - 多行注释没有正常结束
 - 词法记号不存在
 -

词法分析阶段的错误处理

- 词法错误检测
 - 如果当前状态与当前输入符号在转换表对应项中的信息为空，而当前状态又不是终止状态，则调用错误处理程序

错误处理

- 查找已扫描字符串中最后一个对应于某终态的字符
 - 如果找到了，将该字符与其前面的字符识别成一个单词。然后将输入指针退回到该字符，扫描器重新回到初始状态，继续识别下一个单词
 - 如果没找到，则确定出错，采用错误恢复策略

错误恢复策略

- 最简单的错误恢复策略：

“恐慌模式(*panic mode*)”恢复

- 从剩余的输入中不断删除字符，直到词法分析器能够在剩余输入的开头发现一个正确的字符为止

课后思考题

•识别常量

- 识别数字常量（整数、浮点数）
- 识别字符常量
- 识别字符串常量

•识别无效词法记号

- 空白字符（空格、制表符、换行符）
- 注释（单行注释、多行注释（是否能嵌套））

•识别关键字

- 关键字是一类特殊的单词。本质上与标识符没有任何区别。只是词法分析器将其作为系统保留的标识符。不允许用户重新定义。我们在分析标识符后可以查询关键字表，来确定当前识别的标识符是用户定义的还是系统保留的。

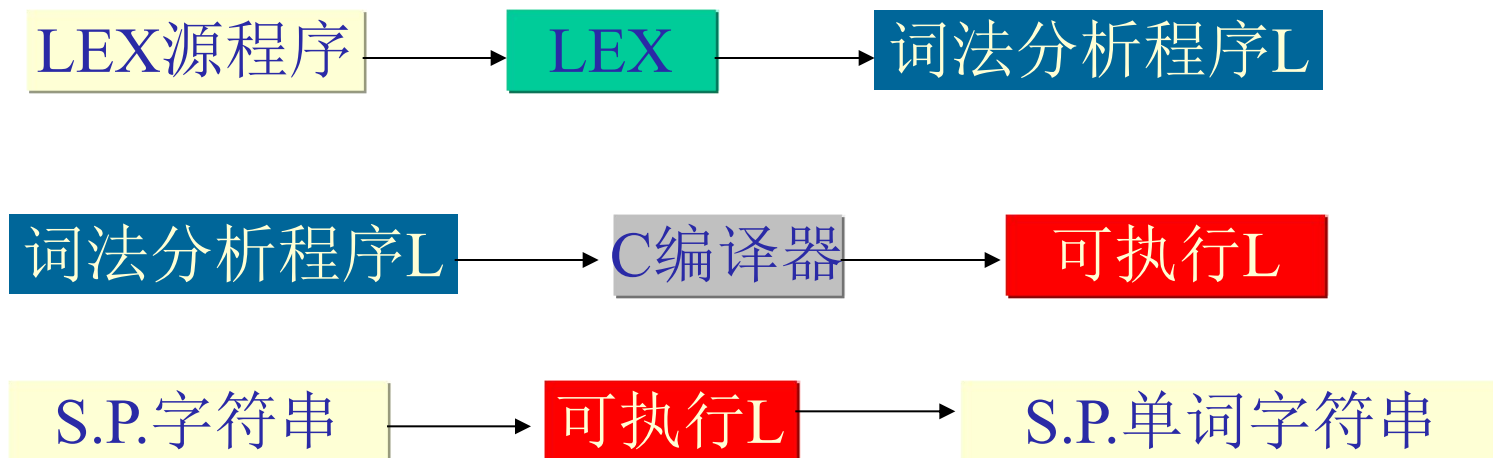
作业

- 画出识别课后思考题中三个问题对应的DFA
- 给出对应的伪代码
- 提交到QQ群的词法分析作业文件夹里

3.4 词法分析程序的自动生成器—LEX (LEXICAL Analyzer Generator)

LEX是1972年在贝尔实验室的UNIX上首先实现
FLEX是1984年GNU工程推出，是LEX的扩充，并兼容LEX

原理：



3.4.1 LEX源程序

一个LEX源程序主要由三个部分组成:

1. 辅助定义式(说明部分)
2. 识别规则。
3. 用户子程序

各部分之间用%%隔开，同时%%在最左边.

一、辅助定义式是如下形式的LEX语句：

$$\begin{array}{ll} D_1 & R_1 \\ D_2 & R_2 \\ & \vdots \\ & \vdots \\ D_n & R_n \end{array}$$

其中： R_1, R_2, \dots, R_n 为正则表达式。

D_1, D_2, \dots, D_n 为正则表达式名字，称简名字

例： C++标识符

letter $A|B|\dots|Z|_$

digit $0|1|\dots|9$

iden $\text{letter}(\text{letter}|\text{digit})^*$



二、识别规则：是一串如下形式的LEX语句：

$$\begin{array}{ll} P_1 & \{A_1\} \\ P_2 & \{A_2\} \\ & \vdots \\ & \vdots \\ P_m & \{A_m\} \end{array}$$

P_i ：定义在 $\Sigma \cup \{D_1, D_2, \dots, D_n\}$ 上的正规表达式，也称词形。

$\{A_i\}$ ： A_i 为语句序列，它指出，在识别出词形为 P_i 的单词以后，词法分析器所应作的动作。

其基本动作是返回单词的类别编码和其属性。

三、用户子程序

定义模式动作需要的函数或包括主函数

在这三部分中一和三为可选项，但是二是必须的

除辅助定义之外定义的部分必须用符号%{和%}括起来，其内容为：

1. 所对应语言的库文件
2. 外部变量
3. 外部函数和函数原型

下面是识别某语言单词符号的LEX源程序：

例：LEX 源程序

AUXILIARY DEF

letter [A—z]

digit [0—9]

%%

RECOGNITION RULES

“BEGIN”

“END”

“FOR”

RETURN是LEX过程，该过程将单词传给语法分析程序

RETURN (C, LEXVAL)

其中C为单词类别编码

LEXVAL:

标识符: TOKEN (字符数组)

整常数: DTB (数值转换函数, 将TOKEN
中的数字串转换二进制值)

其他单词: 无定义。

/ 添加规则 /

{RETURN(1,—) }

{RETURN(2,—) }

{RETURN(3,—) }

“DO”

{RETURN(4,—) }

“IF”

{RETURN(5,—) }

“THEN”

{RETURN(6,—) }

“ELSE ”

{RETURN(7,—) }

{letter}({letter} | {digit})*

{RETURN(8,TOKEN) }

{digit}({digit})*

{RETURN(9,DTB) }

“.”

{RETURN(10,—) }

“+”

{RETURN(11,—) }

“*”

{RETURN(12,—) }

“ ”
 ,

{RETURN(13,—) }

“ (”

{RETURN(14,—) }

“) ”

{RETURN(15,—) }

“ .:= ”

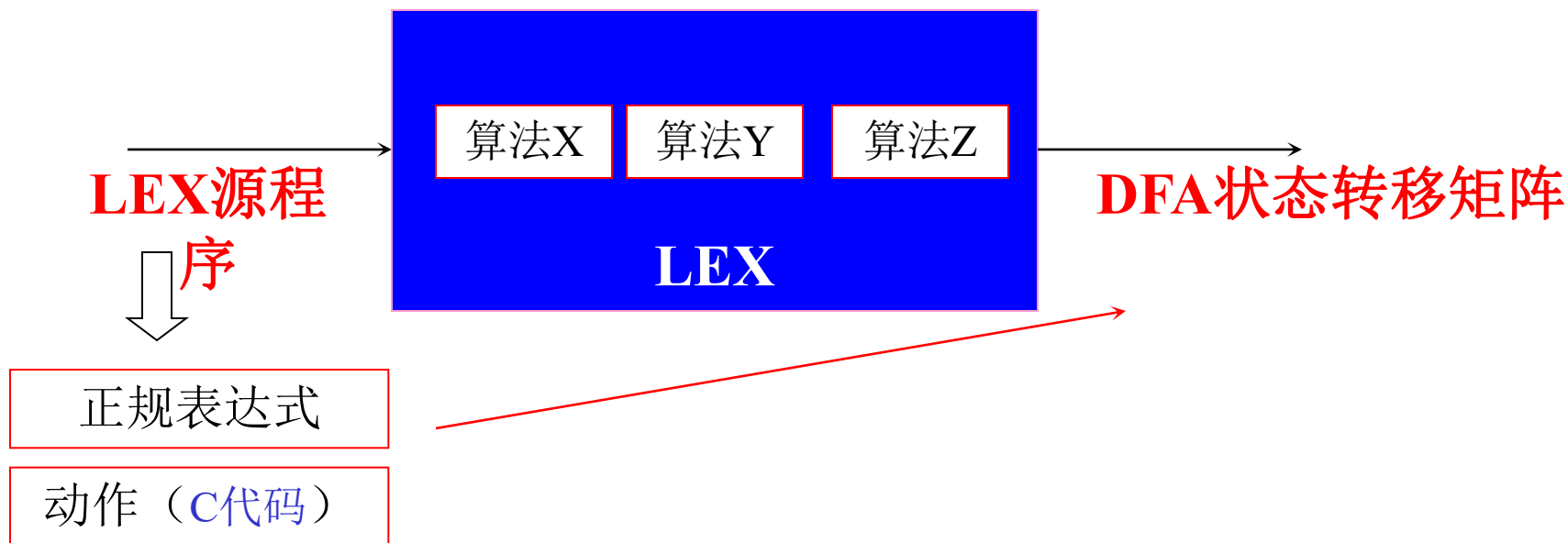
{RETURN(16,—) }

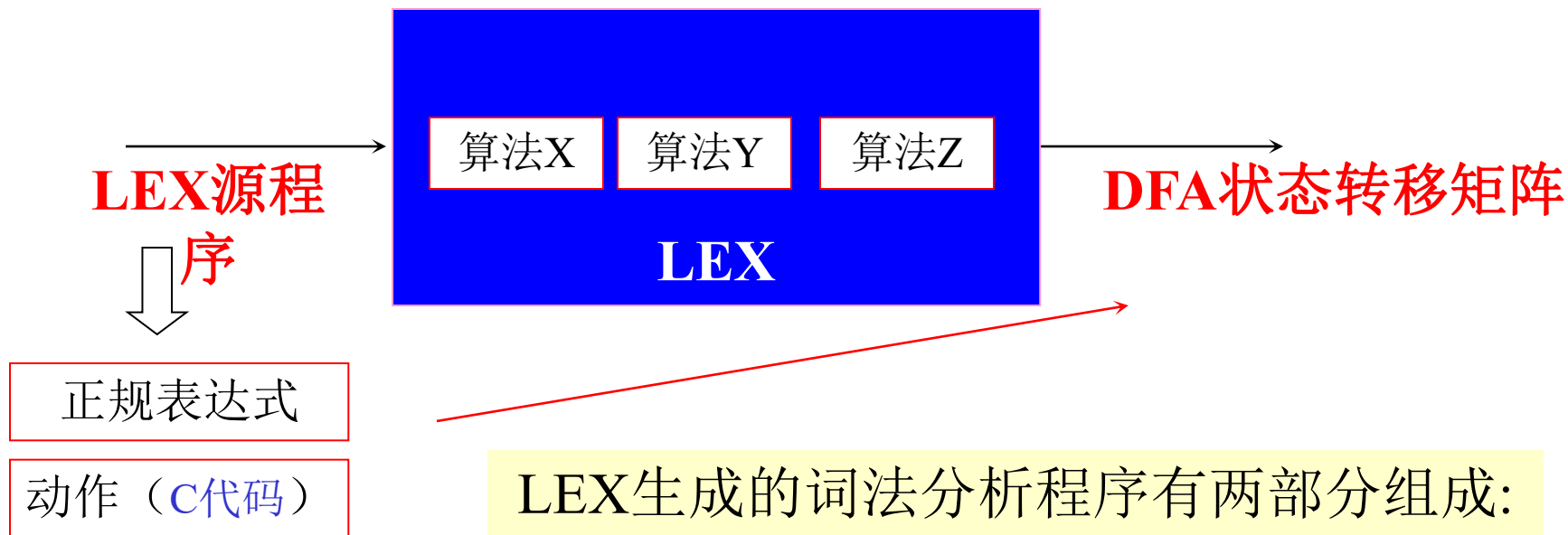
“ == ”

{RETURN(17,—) }

3.4.2 LEX的实现

LEX的功能是根据LEX源程序构造一个词法分析程序，该词法分析器实质上是一个有限自动机。





LEX生成的词法分析程序有两部分组成:

词法分析程序

状态转换矩阵(DFA)

控制执行程序

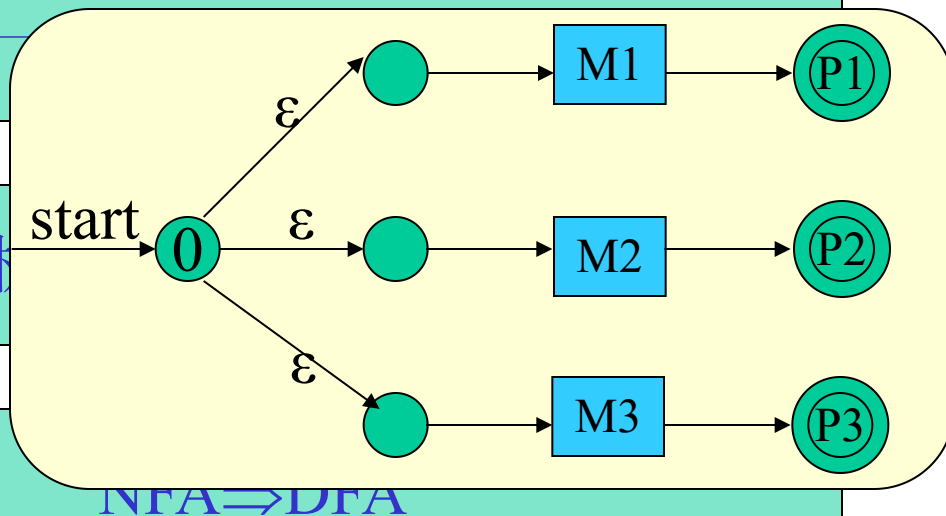
∴ LEX的功能是根据LEX源程序生成状态转换矩阵和控制程序

LEX的处理过程:

- 扫描每条识别规则 P_i 构造一个

- 将各条规则的有穷自动机

- 确定化



生成该DFA的状态转换矩阵和控制执行程序