



# 第三章 词法分析

**授课人：高珍**

**[gaozhen@tongji.edu.cn](mailto:gaozhen@tongji.edu.cn)**

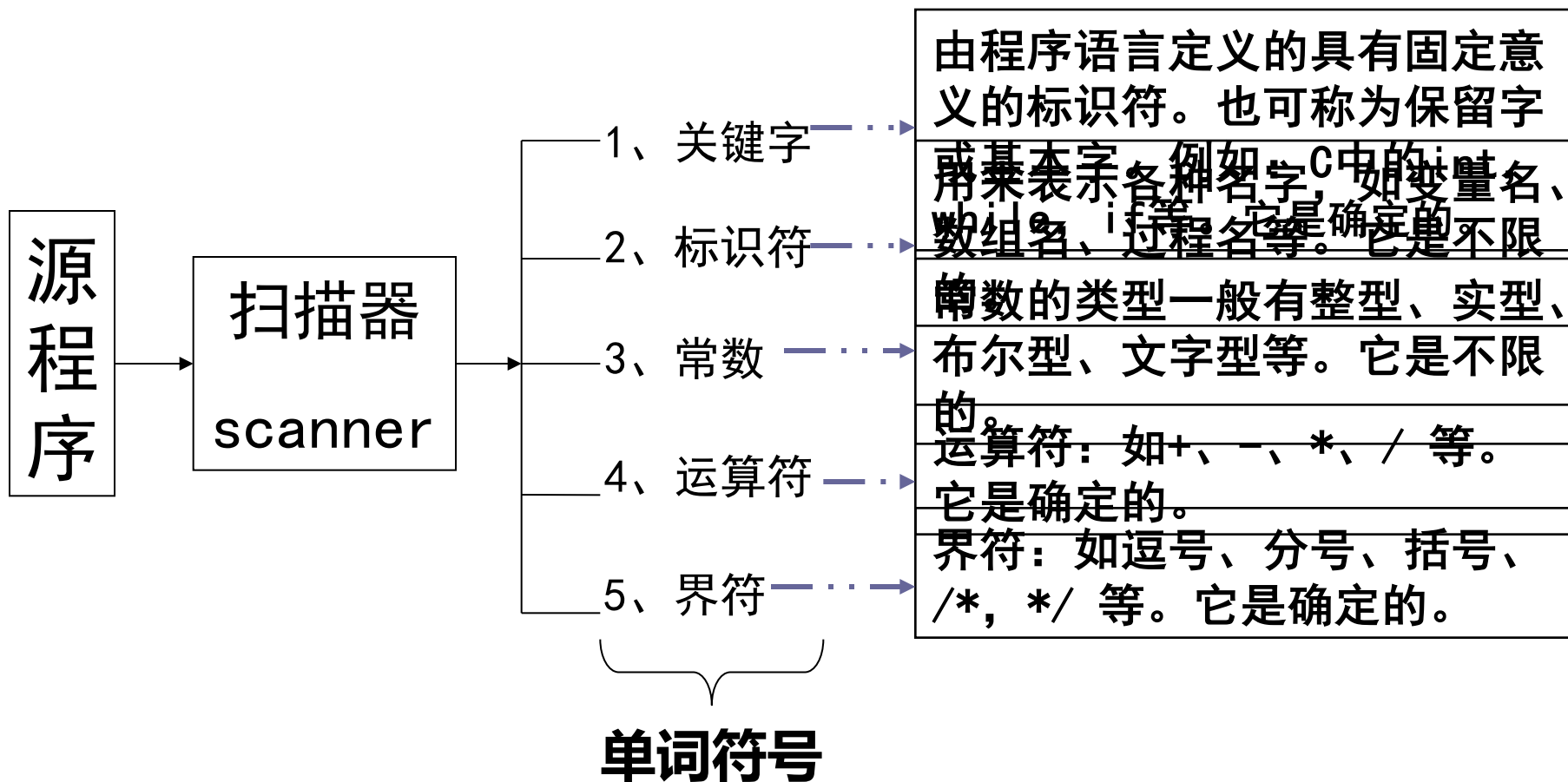
# 概述

- 编译程序首先是在单词级别上来分析和翻译源程序的。
- 词法分析的任务是：从左至右逐个字符地对源程序进行扫描，产生一个个单词符号，把作为**字符串**的源程序改造成成为**单词符号串**的中间程序。
- 词法分析是编译的基础。执行词法分析的程序称为词法分析器。

# 内容线索

- **对于词法分析器的要求**
- **词法分析器的设计**
- **正规表达式与有限自动机**
- **词法分析器的自动生成**

# 词法分析器的功能



# 单词符号表示形式

## ■ 词法分析器输出的单词符号常表示成二元式：

(单词种别, 单词符号的属性值)

- 单词种别是语法分析需要的信息
- 单词符号属性值则是编译其它阶段需要的信息, 简称单词值。

**例.** 语句 `int i,j;` 其中, `i`和`j`都是标识符(变量), 他们的属性值是标识符号表中的入口地址;

# 分类方法

- 单词种别:通常用整数编码。
- 一个语言的单词符号如何分类，分成几类，怎样编码取决于处理上的方便。
  - 标识符一般统归为一种。
  - 常数则宜按类型（整、实、布尔等）分种。
  - 关键字可视其全体为一种，也可以一字一种。采用一字一种的分法实际处理起来较为方便。
  - 运算符可采用一符一种的分法，但也可以把具有一定共性的运算符视为一类。
  - 至于界符一般用一符一种的分法。

# 单词符号的属性

- 单词符号的属性是指单词符号的特征或特性。属性值则是反映特性或特征的值。
  - 标识符的属性值是存放它符号表项的指针或内部字符串；
  - 常数的属性值是存放它的常数表项的指针或二进制形式；
  - 关键字、运算符和界符是一符一种，不需给出其自身的值。

## 例. 代码段 while (i>=j) i--; 词法分析结果

<while , - >

< ( , - >

< id , ptr-i>

< >= , - >

< id , ptr-j>

< ) , - >

< id , ptr-i>

< ..... >

< ; , - >

符号表

No	ID	Addr	type	... ..
				.
224	j	AF80	INT	
227	i	DF88	INT	



# FORTRAN编译实例

- FORTRAN编译程序的词法分析器在扫描输入串  
IF (5·EQ·M) GOTO 100 后，它输出的单词符号串是：

逻辑IF	(34, _)
左括号	(2, _)
整常数	(20, '5'的二进制表示)
等号	(6, _)
标识符	(26, 'M')
右括号	(16, _)
GOTO	(30, _)
标号	(19, '100'的二进制表示)

IF为关键字，种别编码34，

‘(’为界符，种别编码2，采用一符一种的编码方式。

常数类型，种别编码20，单词自  
等号为运算符，种别编码6，

M为标识符，种别编码26，单词自身值为‘M’

‘)’为界符，种别编码16，

GOTO为关键字，种别编码30，采用一符一种的编码方式

100为标号，种别编码19，单词内部的值用100的二进制表示。

# 词法分析程序的实现方式

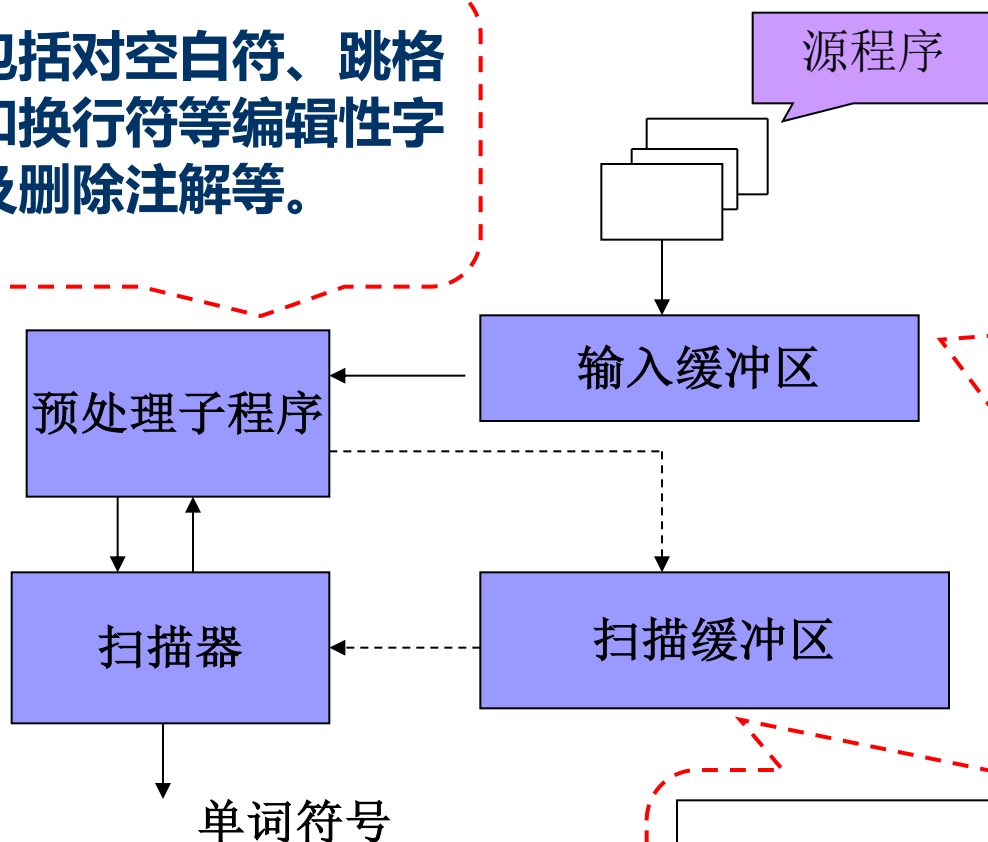
- **完全独立方式**：词法分析程序作为单独一遍来实现。词法分析程序读入整个源程序，它的输出作为语法分析程序的输入。
  - 编译程序结构简洁、清晰和条理化
- **相对独立方式**：把词法分析程序作为语法分析程序的一个独立子程序。语法分析程序需要新符号时调用这个子程序。
  - 优点：避免了中间文件生成，可以提高效率。

# 内容线索

- ✓ 对于词法分析器的要求
- 词法分析器的设计
- 正规表达式与有限自动机
- 词法分析器的自动生成

# 词法分析器的结构

预处理工作包括对空白符、跳格符、回车符和换行符等编辑性字符的处理，及删除注解等。



输入源程序文本。输入串一般放在一个缓冲区中，这个缓冲区称输入缓冲区。

设定两个指示器  
缓冲区一分为二

起点指示器

搜索指示器\*

# 单词符号的识别：超前搜索

## ■ 关键字识别

**例.** 在标准FORTRAN中2个合法句子：

1、 DO99K = 1,10 — 其中的DO为关键字

2、 DO99K = 1.10 — 其中的DO为标识符的一部分

# 单词符号的识别：超前搜索

## ■ 标识符的识别

- 多数语言的标识符是字母开头的“字母/数字”串，而且在程序中标识符的出现后都跟着算符或界符。因此，不难识别。

## ■ 常数的识别

- 对于某些语言的常数的识别也需要使用超前搜索。

## ■ 算符和界符的识别

- 对于诸如C++语言中的“+ +”、“- -”，这种复合成的算符，需要超前搜索。

# 状态转换图

- 大多数程序设计语言中单词符号的**词法规则**可以用**正规文法**描述。如：  
     $\langle \text{标识符} \rangle \rightarrow \text{字母} | \langle \text{标识符} \rangle \text{字母} | \langle \text{标识符} \rangle \text{数字}$   
     $\langle \text{整数} \rangle \rightarrow \text{数字} | \langle \text{整数} \rangle \text{数字}$   
     $\langle \text{运算符} \rangle \rightarrow + | - | \times | \div \dots$   
     $\langle \text{界符} \rangle \rightarrow ; | , | ( | ) | \dots$
- 利用这些规则识别单词符号的过程可用一张称为**状态转换图**的有限方向图来表示，而状态转换图识别单词符号的过程又可以方便地用程序实现。

# 状态转换图定义

## ■ 转换图：是一个有限方向图。

- 结点代表状态，用圆圈表示。

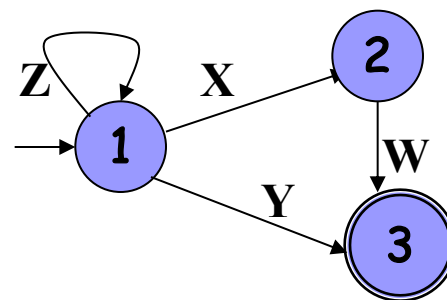
- 初态：一张转换图的启动条件，通常有一个，用圆圈表示。

- 终态：一张转换图的结束条件，至少有一个，用双圈表示。

- 状态之间用方向弧连接。弧上的标记（字符）代表在出射结点状态下可能出现的输入字符或字符类。

## ■ 状态转换图中只包含有限个状态（结点）

在状态1下，若输入字符为X，则读进X并转换到状态2；若输入字符为Y则读进Y并转换到状态3，输入字符Z，状态仍为1。



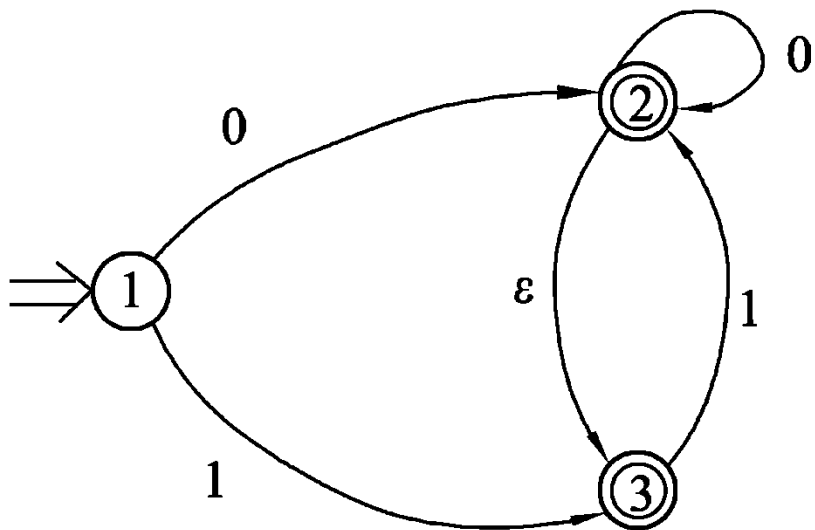


# 状态转换图的作用

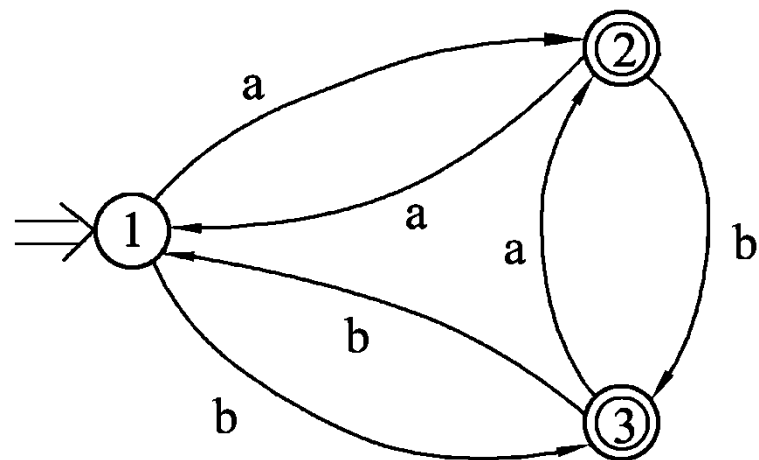
- 一个状态转换图可用于**接受**（或**识别**）一定的符号串。
- **路**:在状态转换图中从**初始状态**到**某一终止状态**的弧上的**标记序列**。
- 对于某一符号串 $\beta$ ，在状态转换图中，若存在一条路产生 $\beta$ ，则称状态转换图接受（或识别）该符号串 $\beta$ ，否则称符号串 $\beta$ 不能被接受。

# 状态转换图所能识别的语言

- 能被状态转换图TG接受的符号串的集合记为  $L(TG)$ ，称它为**状态转换图所能识别的语言**。



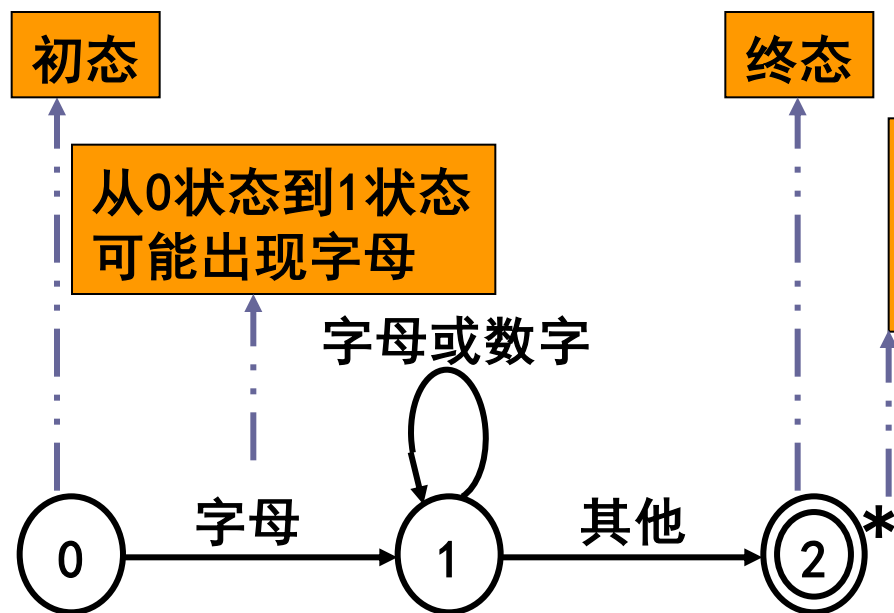
$L(TG) = \{ 0, 1, 00, 01, 11, 001, 010, \dots \}$



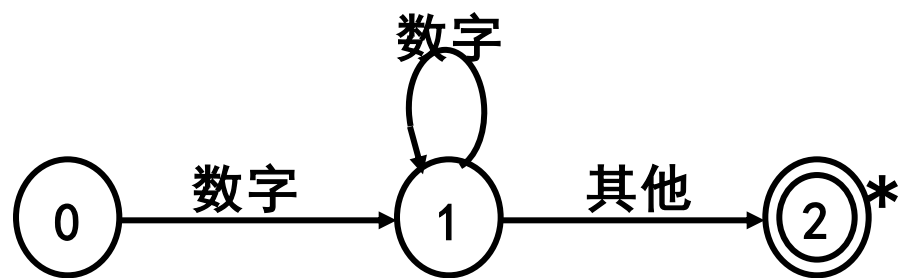
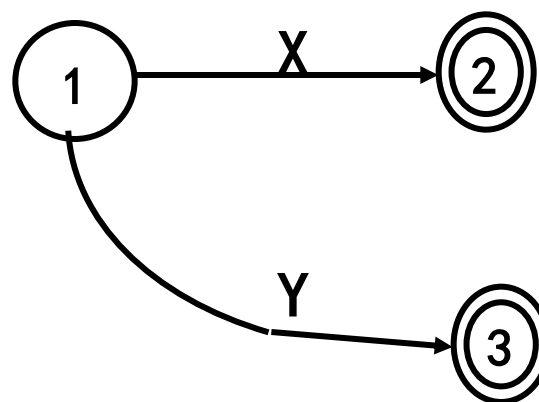
$L(TG) = \{ a, b, ab, ba, aaa, bbb, aab, bba, \dots \}$

# 状态转换图示例

- 大多数程序语言的单词符号都可以用状态转换图予以识别。



(b) 识别标识符的转换图



(c) 识别整数的转换图

# 状态转换图识别单词符号的过程

**Step1. 从初态开始;**

**Step2. 从输入串中读一个字符;**

**Step3. 判明读入字符与从当前状态出发的哪条弧上的标记相匹配, 便转到相应匹配的那条弧所指向的状态;**

**Step4. 重复Step3, 均不匹配时便告失败; 到达终态时便识别出一个单词符号。**

## 例. 设一小语言所有单词符号及其内部表示形式

单词符号	种别编码	助忆符	内码值
DIM	1	\$DIM	-
IF	2	\$IF	-
DO	3	\$DO	-
STOP	4	\$STOP	-
END	5	\$END	-
标识符	6	\$ID	内部字符串
整常数	7	\$INT	标准二进制形式
=	8	\$ASSIGN	-
+	9	\$PLUS	-
*	10	\$STAR	-
**	11	\$POWER	-
.	12	\$COMMA	-
(	13	\$LPAR	-
)	14	\$RPAR	-

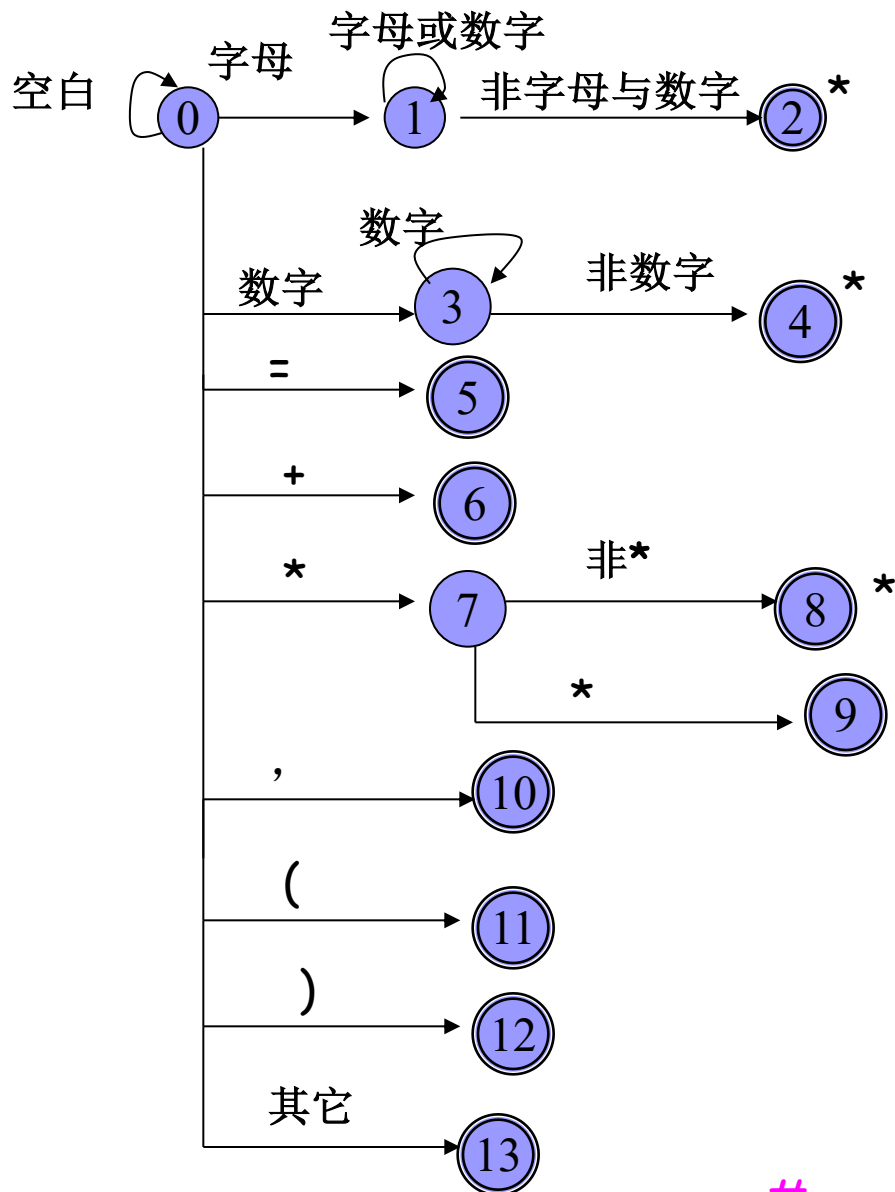
# 能识别小语言所有单词的状态转换图

约定（限制）：

✓ 关键字为保留字；

✓ 保留字作为标识符处理,并使用保留字表识别；

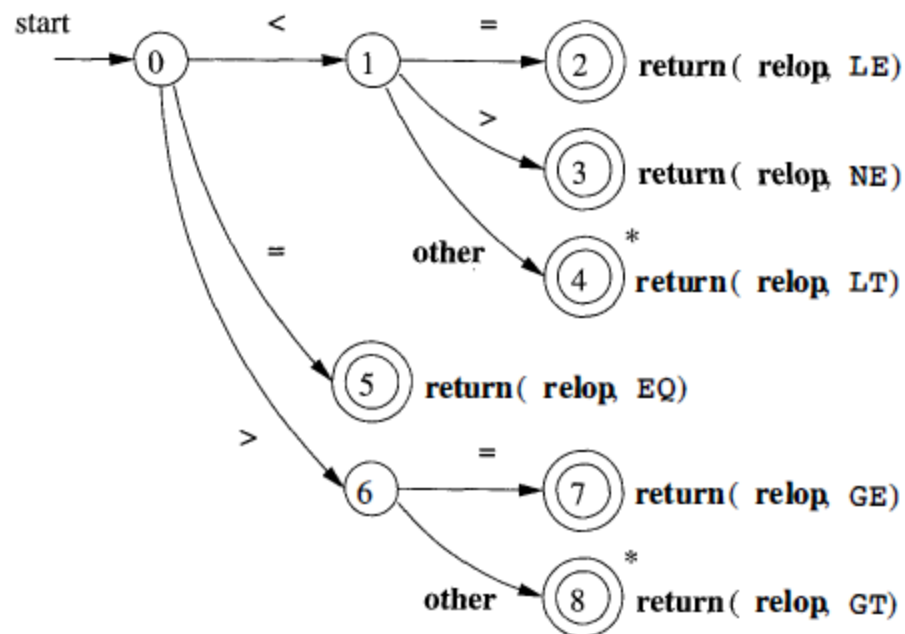
✓ 关键字、标识符、常数间若无运算符或界限符则加一空格



# 状态转换图示例

```

TOKEN getRelop()
{
    TOKEN retToken = new(RELOP);
    while(1) { /* repeat character proce
                or failure occurs */
        switch(state) {
            case 0: c = nextChar();
                    if ( c == '<' ) state = 1;
                    else if ( c == '=' ) state = 5;
                    else if ( c == '>' ) state = 6;
                    else fail(); /* lexeme is not a relop */
                    break;
            case 1: ...
            ...
            case 8: retract();
                    retToken.attribute = GT;
                    return(retToken);
        }
    }
}
    
```



# Simulating a DFA

```
s = s0;
c = nextChar() ;
while ( c != eof ) {
    s = move(s, c);
    c = nextChar() ;
}
if ( s is in F ) return " yes " ;
else return "no " ;
```



# 内容线索

- ✓ 对于词法分析器的要求
- ✓ 词法分析器的设计
- 正规表达式与有限自动机
- 词法分析器的自动生成

# 学习思路

**词法规则 → 正规式**  
**→ NFA → DFA**  
**→ 词法分析器 (Scanner)**

# 正规表达式与有限自动机

- 为了更好地使用状态转换图构造词法分析器，和讨论词法分析器的自动生成，还需要将转换图的概念形式化。
  - 正规式与正规集
  - 确定有限自动机 (DFA)
  - 非确定有限自动机(NFA)
  - 正规式与有限自动机的等价性
  - 确定有限自动机的化简

# 正规式与正规集

- 字母表 $\Sigma$ 上的**正规式**和**正规集**递归定义如下:

- (1)  $\varepsilon$ 和 $\varphi$ 都是 $\Sigma$ 上的正规式, 它们所表示的正规集分别为 $\{\varepsilon\}$ 和 $\varphi$ 。其中:  $\varepsilon$ 为空字符串,  $\varphi$ 为空集;
- (2) 任意元素 $a \in \Sigma$ ,  $a$ 是 $\Sigma$ 上的一个正规式, 它所表示的正规集是 $\{a\}$ ;
- (3) 假定 $U$ 和 $V$ 都是 $\Sigma$ 上的正规式, 它们所表示的正规集记为 $L(U)$ 和 $L(V)$ , 那么  $(U|V)$ ,  $(U \cdot V)$  和 $(U)^*$ 都是正规式, 他们所表示的正规集分别记为 $L(U) \cup L(V)$ ,  $L(U)L(V)$ 和 $(L(U))^*$ 。
- (4) 仅由有限次使用上述三步而得到的表达式才是 $\Sigma$ 上的正规式, 它们所表示的字集才是 $\Sigma$ 上的正规集。

# 三种运算示例

**例1.** 设  $L = \{ 001, 10, 111 \}$ ,  $M = \{ \varepsilon, 001 \}$ ,

$$L \cup M = \{ \varepsilon, 10, 001, 111 \}$$

**例2.** 设  $L = \{ 001, 10, 111 \}$ ,  $M = \{ \varepsilon, 001 \}$ , 则

$$LM = \{ 001, 10, 111, 001001, 10001, 111001 \}$$

**例3.** 设  $L = \{ 0, 11 \}$ , 则

$$L^* = \{ \varepsilon, 0, 11, 00, 011, 110, 1111, 000, 0011, 0110, 01111, 1100, 11011, 11110, 111111, \dots \}$$

# 说明

- 运算符 “|” 读为“或”；

“.” 读为“连接”

“\*” 读为“闭包”。

一般地，连接符“.”可省略不写，在不引起混淆的情况下，括号可省去

- 正规式运算符的**优先顺序**为：“\*” 最高，“.”次之，“|”最低。
- 若两个正规式所表示的正规集相同，则认为二者等价，记为  $U=V$ 。

# 例1. 令 $\Sigma = \{a,b\}$ , $\Sigma$ 上的正规式和相应的正规集有

正规式

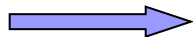
正规集

$(a|b)(a|b)$



$$\begin{aligned} L(a|b)(a|b) &= L(a|b) \cdot L(a|b) \\ &= (L(a) \cup L(b)) \cdot (L(a) \cup L(b)) \\ &= \{a,b\} \cdot \{a,b\} = \{aa,ab,ba,bb\} \end{aligned}$$

$ba^*$



$$\begin{aligned} L(ba^*) &= L(b)L(a^*) = L(b)(L(a))^* \\ &= \{b\}\{a\}^* = \{b\}\{\epsilon, a, aa, aaa, \dots\} \\ &= \{b, ba, baa, baaa, \dots\} \end{aligned}$$

$\Sigma$ 上所有以b为首后跟任意多个a的字的集合。

正规式

正规集

$a(a|b)^*$

→  $\Sigma$ 上所有以a为首的字集

$(a|b)^*(aa|bb)(a|b)^*$

→  $\Sigma$ 上所有含有两个相继a  
或两个相继b的字集

**例2.** C语言中“标识符”的正规式为:

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

**例3.** “整数”的正规式:

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



# 正规式的运算律

■ 令U、V和W均为正规式，则：

(1)  $U|V=V|U$  交换律

(2)  $U|(V|W)=(U|V)|W$  结合律

(3)  $U(VW)=(UV)W$  结合律

(4)  $U(V|W)=UV|UW$  分配律

(5)  $(V|W)U=VU|WU$  分配律

(6)  $\varepsilon U=U\varepsilon=U$

# 运算律证明-1

(1) 交换律:  $U \mid V = V \mid U$

$$\begin{aligned}\text{证明: } L(U \mid V) &= L(U) \cup L(V) \\ &= L(V) \cup L(U) \\ &= L(V \mid U)\end{aligned}$$

(2) 结合律:  $U \mid (V \mid W) = (U \mid V) \mid W$

$$\begin{aligned}\text{证明: } L(U \mid (V \mid W)) &= L(U) \cup L(V \mid W) \\ &= L(U) \cup (L(V) \cup L(W)) \\ &= (L(U) \cup L(V)) \cup L(W) \\ &= L((U \mid V) \mid W)\end{aligned}$$

# 随堂练习

## ■ P64 8. 写出正规表达式

- (1) 以01结尾的二进制数串
- (2) 能被5整除的十进制整数
- (3) 包含奇数个0的二进制数串

# 自动机

## ■ 什么是自动机？

- 具有离散输入输出的数学模型。
- 自动机接受一定的输入，执行一定的动作，产生一定的结果。使用状态迁移描述整个工作过程。
  - 状态：一个标识，能区分自动机在不同时刻的状况。有限状态系统具有任意有限数目的内部“状态”

## ■ 为什么叫自动机？

- 可能的状态、运行的规则都是事先确定的。一旦开始运行，就按照事先确定的规则工作，因此叫“自动机”。

## ■ 自动机的本质？

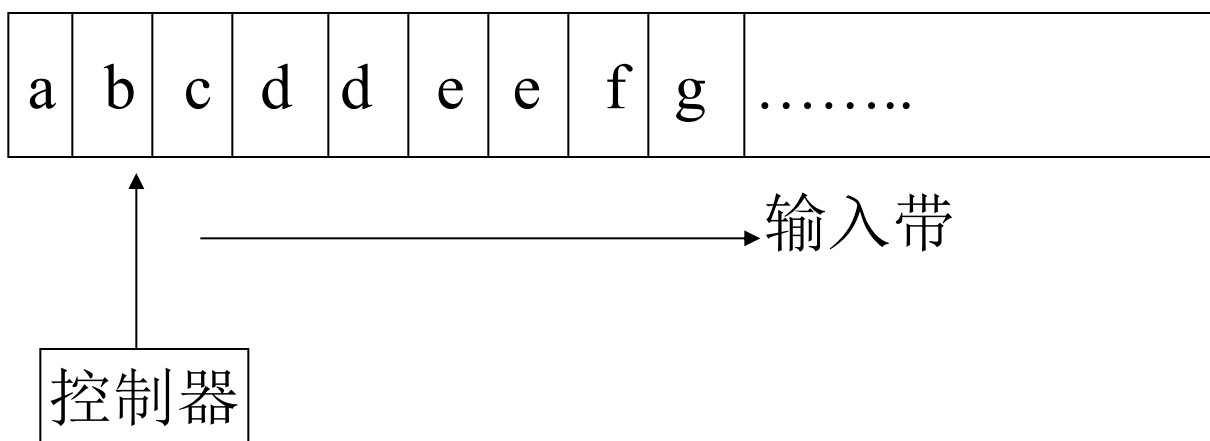
- 根据状态、输入和规则决定下一个状态
- 状态 + 输入 + 规则  $\rightarrow$  状态迁移

# 有限自动机的定义

- **有限自动机 (FA, Finite Automata)**
  - **有限状态机 (FSM, Finite State Machine)**
  - **一个机器或一种控制结构，设计它的目的是为了自动仿效一个事先确定的操作序列或响应一条已编码的指令。**

# FA的模型

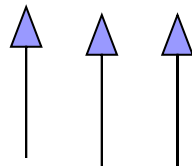
- FA可以理解成一个控制器，它读一条输入带上的字符。



**有限自动机=有限控制器+字符输入带**

# 示例

输入（字符串）： 0 0 1 0



控制器

# 确定有限自动机 (DFA)

## ■ DFA是一个五元组 $M=(S, \Sigma, \delta, s_0, F)$

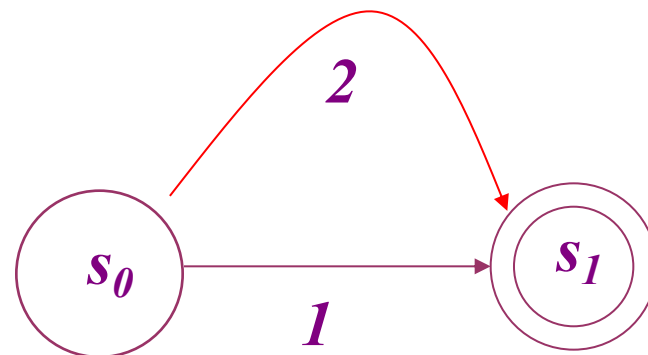
$S$ : 有限的状态集合, 每个元素称为一个**状态**;

$\Sigma$ : 有限的输入字母表, 每个元素称为一个**输入字符**;

$\delta$ : 转换函数(状态转移集合):  $S \times \Sigma \rightarrow S$  ;

$s_0$ : 初始状态,  $s_0 \in S$  ;

$F$ : 终止状态集,  $F \subseteq S$  ;





# 状态转换矩阵

- 一个DFA可用一个矩阵表示，该矩阵的行表示状态，列表表示输入字符，矩阵元素表示 $\delta(s,a)$ 的值。

例：DFA  $M = (\{0,1,2,3\}, \{a,b\}, \delta, 0, \{3\})$

其中

$\delta(0,a)=1$   $\delta(0,b)=2$

$\delta(1,a)=3$   $\delta(1,b)=2$

$\delta(2,a)=1$   $\delta(2,b)=3$

$\delta(3,a)=3$   $\delta(3,b)=3$

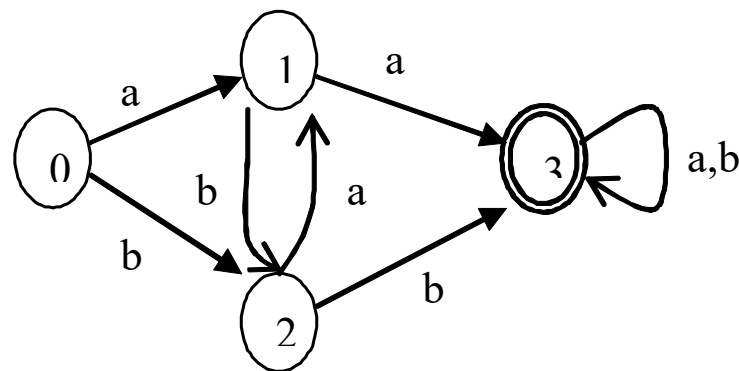
对应的状态转换矩阵

状态	a	b
0	1	2
1	3	2
2	1	3
3	3	3

# DFA与状态转换图

- 一个DFA可以表示成一张（确定的）状态转换图。
  - 假定DFA  $M$  含有  $m$  个状态和  $n$  个输入字符，那么，状态图必含有  $m$  个状态结点，每个结点最多有  $n$  条弧射出和别的结点相连。每条弧上用  $\Sigma$  中的一个不同输入字符做标记。整张图有唯一的一个初态结点和若干终态结点。

状态	a	b
0	1	2
1	3	2
2	1	3
3	3	3



# 扩展转移函数

## ■ $\delta'$ 函数

- 接收一个字符串的状态转移函数。

- $\delta': S \times \Sigma^* \rightarrow S$

## ■ 对任何 $s \in S$ , 定义:

1.  $\delta'(s, \varepsilon) = s$

2. 若  $\omega$  是一个字符串,  $a$  是一个字符

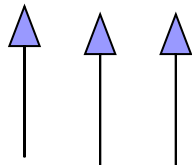
定义:  $\delta'(s, \omega a) = \delta(\delta'(s, \omega), a)$

## ■ 对于DFA: $\delta'(s, a) = \delta(\delta'(s, \varepsilon), a) = \delta(s, a)$

- 即对于单个字符时  $\delta$  和  $\delta'$  是相等的。

# 扩展转移函数

输入（字符串）： 0 0 1 0



控制器

$$\delta'(q_0, \varepsilon) = q_0$$

$$\delta'(q_0, 0) = \delta(q_0, 0) = q_2$$

$$\delta'(q_0, 00) = \delta(q_2, 0) = q_0$$

$$\delta'(q_0, 001) = \delta(q_0, 1) = q_1$$

$$\delta'(q_0, 0010) = \delta(q_1, 0) = q_3$$

# DFA接受的语言

- **被DFA接收的字符串**: 输入结束后使DFA的状态到达终止状态。否则该字符串不能被DFA接收.
- **DFA接收的语言**: 被DFA接收的字符串的集合.

$$L(M) = \{ \alpha \mid \delta'(s_0, \alpha) \in F \}$$

对任一输入字符串  $\alpha \in \Sigma^*$ , 若存在一条从初态结点  $s_0$  到某一终态结点的通路, 且这条通路上所有弧的标记连接成的字等于  $\alpha$ , 则称  $\alpha$  可以被 DFA  $M$  所识别(读出、接受)。

若  $s_0 \in F$ , 则  $\epsilon$  可被接受。

# Simulating a DFA

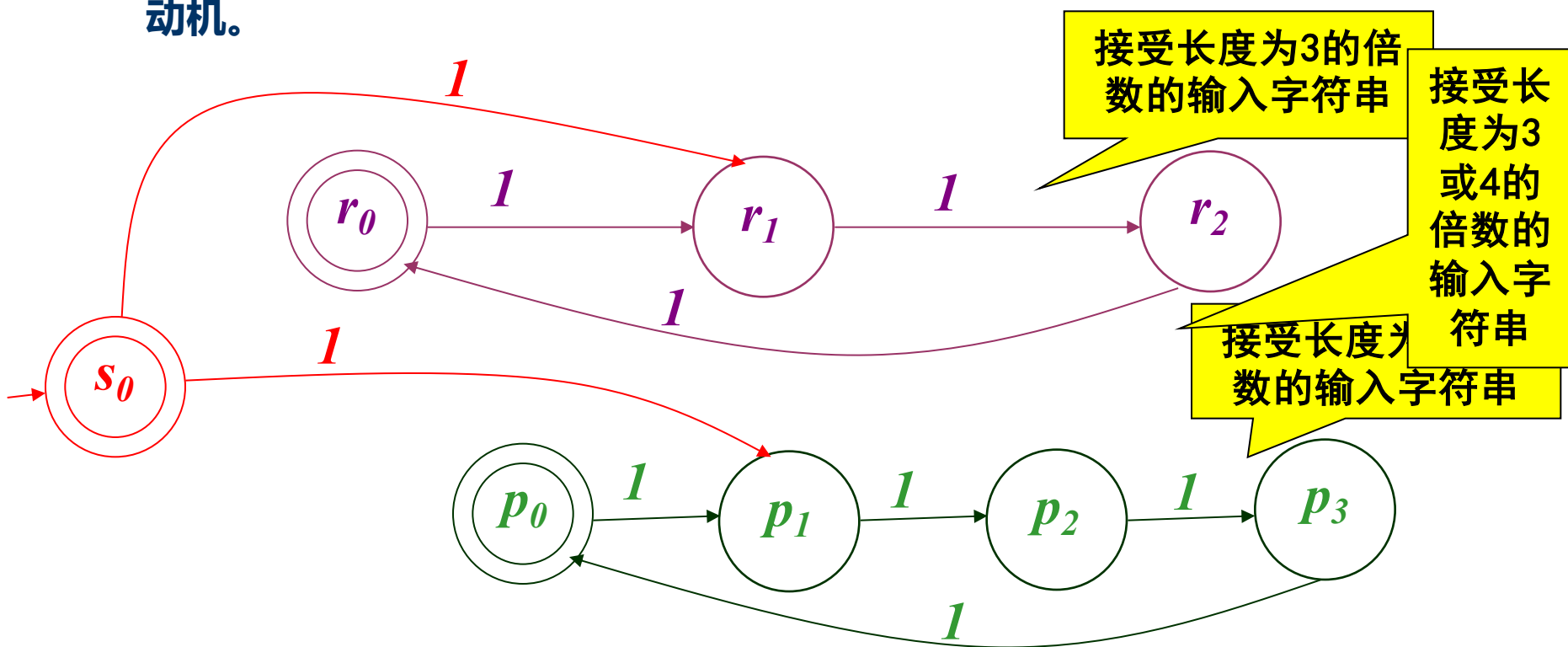
```
s = s0;  
c = nextChar() ;  
while ( c != eof ) {  
    s = move(s, c);  
    c = nextChar() ;  
}  
if ( s is in F ) return " yes " ;  
else return "no " ;
```

## 随堂练习

一个人带着狼、山羊、白菜在一条河的左岸。河边有一条船，只能容纳这个人和三样东西中的一样。也就是说，人每趟最多只能带一样东西过河。如果人将狼和羊留在同一岸边而无人照看的话，狼会把羊吃掉。类似地，如果人将羊和白菜留在同一岸边而无人照看的话，羊会把白菜吃掉。如果可能，请用有限自动机写出渡河的方法。

# 非确定的有限自动机

- 修改DFA的模型,使之在某个状态, 对应一个输入,可以有多个转移, 到达不同的状态, 即: **具有在同一情况下可有不同选择的能力**。则称为非确定的有限自动机。





# 非确定的有限自动机

- 一个非确定的有限自动机 (NFA)  $M$  是一个五元组:

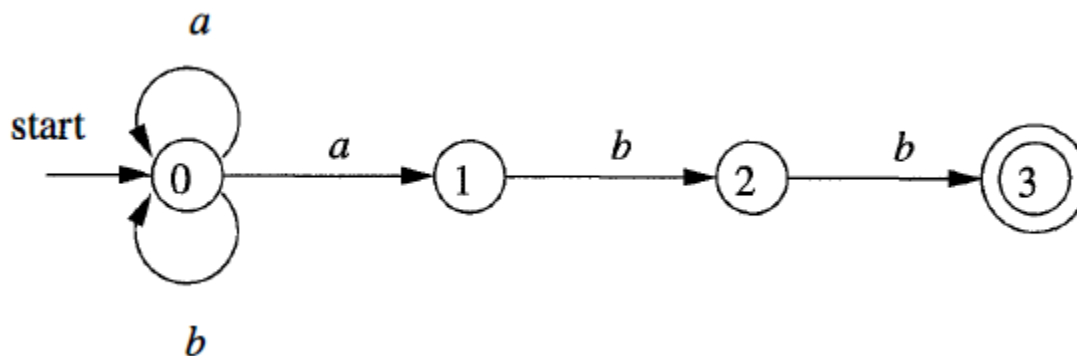
$M = (S, \Sigma, \delta, S_0, F)$ , 其中:

- (1)  $S$  和  $\Sigma$  的定义同前;
- (2)  $\delta: S \times \Sigma^* \rightarrow 2^S$  (状态子集);

对于某个状态  $s \in S$  和一个输入字母  $a$ :

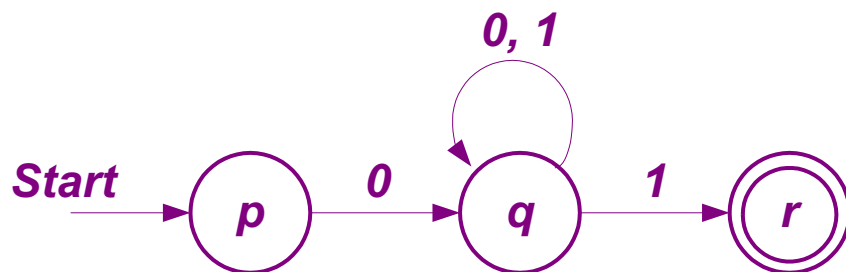
$$\delta(s, a) = S' \subseteq S$$

- (3)  $S_0 \subseteq S$  为非空初态集;
- (4)  $F \subseteq S$  为终态集



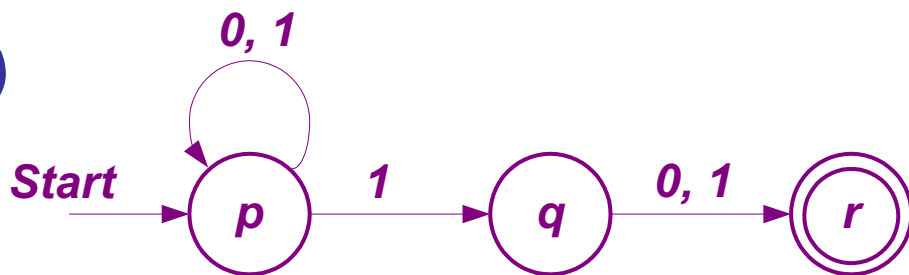
# 状态转换图和转换矩阵表示的NFA

(1)



	0	1
→ p	{q}	∅
q	{q}	{q, r}
* r	∅	∅

(2)



	0	1
→ p	{p}	{p, q}
q	{r}	{r}
* r	∅	∅

**注：状态转换矩阵中的每一项都是一个集合。含空集 $\emptyset$ ，即对于某些状态与输入字母的组合可能没有动作。**

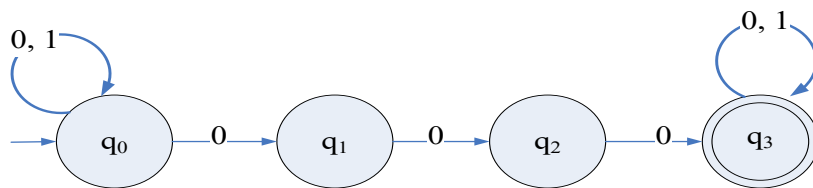
# Simulating a NFA

```
1)   $S = \epsilon\text{-closure}(s_0);$ 
2)   $c = \text{nextChar}();$ 
3)  while (  $c \neq \text{eof}$  ) {
4)       $S = \epsilon\text{-closure}(\text{move}(S, c));$ 
5)       $c = \text{nextChar}();$ 
6)  }
7)  if (  $S \cap F \neq \emptyset$  ) return "yes";
8)  else return "no";
```

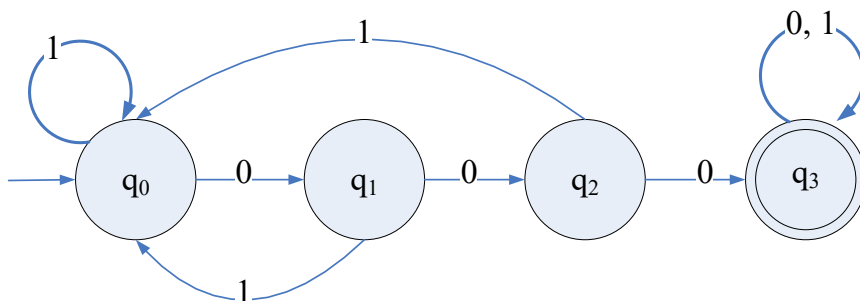
# NFA和DFA的比较

**例.** 构造NFA, 可识别 $\{0,1\}$ 上的语言

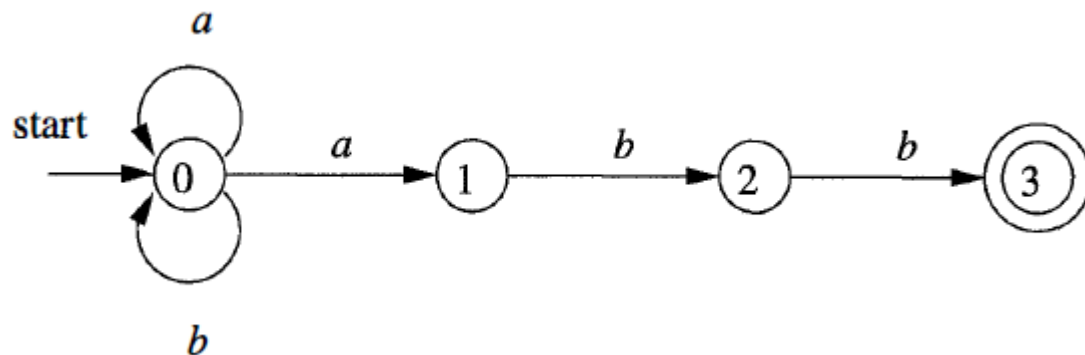
$$L = \{x000y \mid x, y \in \{0,1\}^*\}.$$



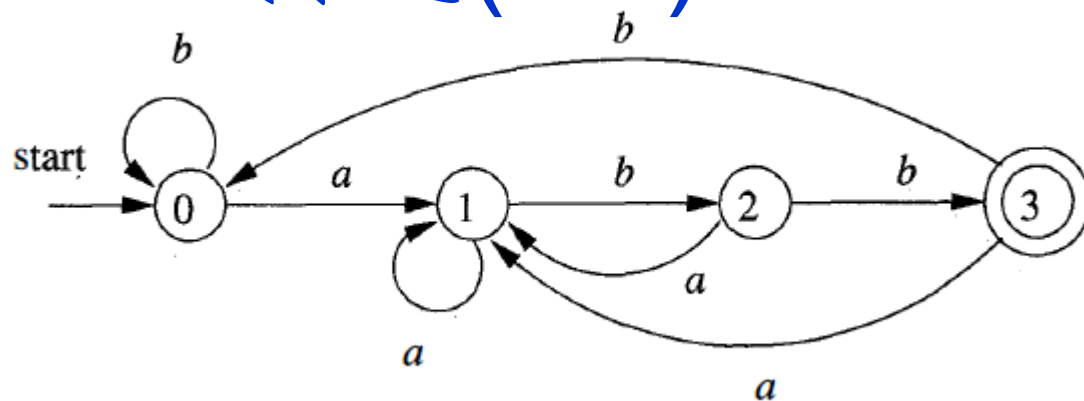
**比较对应的DFA**



# NFA接受 $(alb)^* abb$



# DFA接受 $(alb)^* abb$



# NFA和DFA的比较

## ■ 输入字母

- DFA的每一个状态对于字母表中的每一个符号都有一个转移函数。
- 在NFA中，一个状态对于字母表中的每一个符号可能不存在转移函数或者存在空转换。

## ■ 转移状态

- DFA中的下一状态是确定的，即唯一的
- NFA中的下一状态是不确定的，可能存在多个转移状态。

# NFA的状态转移函数

- 与 DFA 不同之处  $\delta: S \times \Sigma \rightarrow 2^S$
- 同样,  $\delta$  可扩展为  $\delta'$  ( $\delta': S \times \Sigma^* \rightarrow 2^S$ )
  1.  $\delta'(s, \varepsilon) = \{s\}$
  2.  $\delta'(s, \omega a) = \{p \mid \text{存在 } r \in \delta'(s, \omega) \wedge p \in \delta(r, a)\}$ 
    - 含义:  $\delta'(s, \omega a)$  对应的状态集合是  $\delta'(s, \omega)$  对应的每个状态下, 再接收字符  $a$  以后可能到达的状态集合的并集. 即  
若  $\delta'(s, \omega) = \{r_1, r_2, \dots, r_k\}$ , 则
$$\delta'(s, \omega a) = \bigcup \delta(r_i, a)$$
其中  $\omega \in \Sigma^*$ ,  $a \in \Sigma$ ,  $r_i \in S$

# 扩展转移函数适合于输入字符串

	0	1
$\rightarrow p$	$\{q\}$	$\phi$
$q$	$\{q\}$	$\{q, r\}$
$* r$	$\phi$	$\phi$

$$\delta'(p, \varepsilon) = \{p\}$$

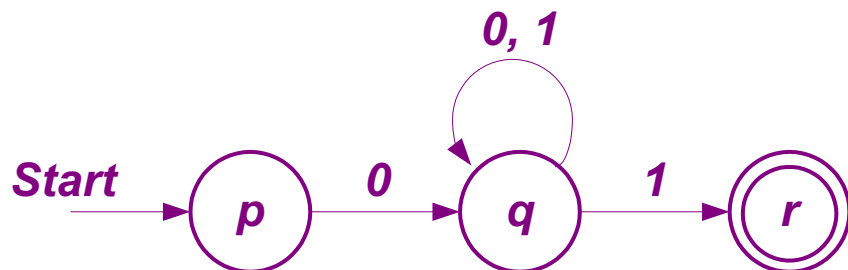
$$\delta'(p, 0) = \{q\}$$

$$\delta'(p, 01) = \{q, r\}$$

$$\delta'(p, 010) = \{q\}$$

$$\delta'(p, 0100) = \{q\}$$

$$\delta'(p, 01001) = \{q, r\}$$





# NFA 接受的语言

- 如果接收一个字符串后NFA进入一个状态集，而此集合包含一个以上F中的状态，则称NFA接收该字符串。

- 设一个  $M = (S, \Sigma, \delta, S_0, F)$ ，定义 M 的语言：

$$L(M) = \{ \alpha \mid \delta'(S_0, \alpha) \cap F \neq \phi \}$$

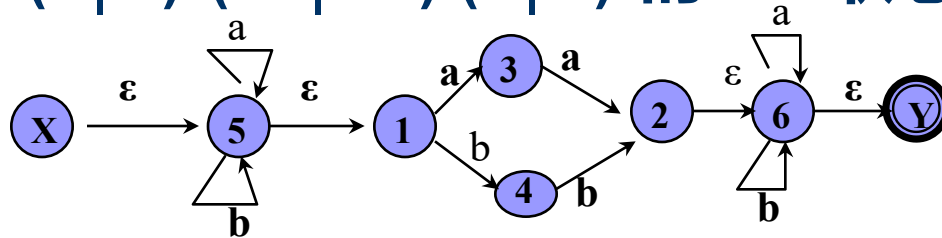
对任输入字符串  $\alpha \in \Sigma^*$ ，若存在一条从某初态结点  $S_0$  到某一终态结点的通路，且这条通路上所有弧的标记连接成的字等于  $\alpha$ （弧上的  $\epsilon$  忽略不计），则称  $\alpha$  可以被 NFA M 所识别（读出、接受）。

# NFA和DFA的等价性

- DFA是NFA的特例，所以NFA必然能接收DFA能接收的语言。
- 一个NFA所能接收的语言能被另一个DFA所接收？
- 设一个NFA接受语言 $L$ ，那么存在一个DFA接受 $L$ 。
  - 证明策略:对于任意一个NFA，构造一个接收它所能接收语言的DFA，这个DFA的状态对应了NFA的状态集合。

# 举例NFA转化为DFA

正规式  $V = (a \mid b)^*(aa \mid bb)(a \mid b)^*$  的NFA状态转换图为



## 1) 利用 子集法构造状态转换矩阵

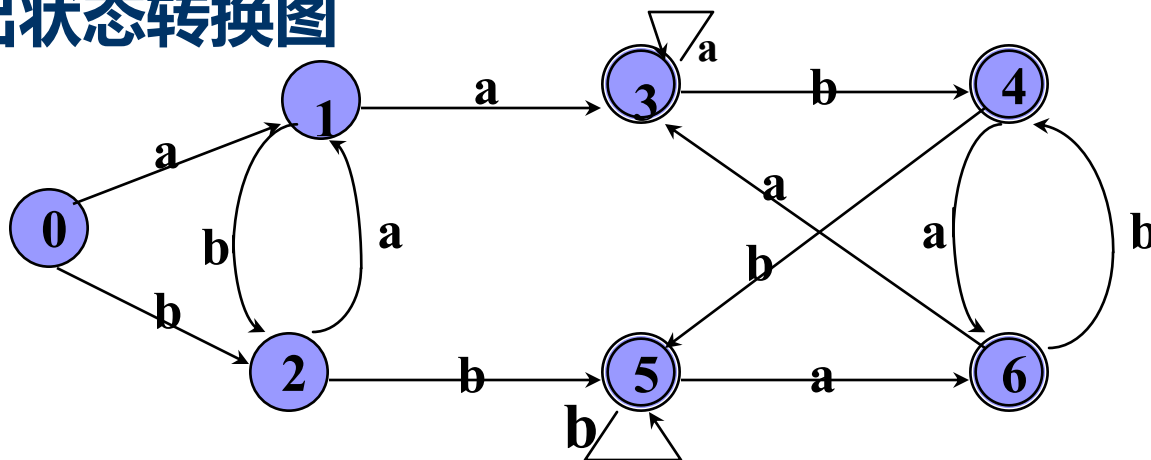
I	$I_a$	$I_b$
{X, 5, 1 }	{5, 3, 1 }	{5, 4, 1 }
{5, 3, 1 }	{5, 3, 1, 2, 6, Y }	{5, 4, 1 }
{5, 4, 1 }	{5, 3, 1 }	{5, 4, 1, 2, 6, Y }
{5, 3, 1, 2, 6, Y }	{5, 3, 1, 2, 6, Y }	{5, 4, 1, 6, Y }
{5, 4, 1, 2, 6, Y }	{5, 3, 1, 6, Y }	{5, 4, 1, 2, 6, Y }
{5, 4, 1, 6, Y }	{5, 3, 1, 6, Y }	{5, 4, 1, 2, 6, Y }
{5, 3, 1, 6, Y }	{5, 3, 1, 2, 6, Y }	{5, 4, 1, 6, Y }

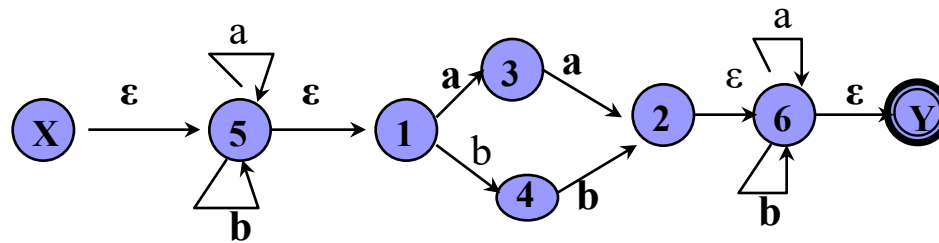
### 3) 对状态子集重新命名得新状态转换矩阵

I		I <sub>a</sub>	I <sub>b</sub>
{X, 5, 1}	0	{5, 3, 1}	{5, 4, 1}
{5, 3, 1}	1	{5, 3, 1, 2, 6, Y}	{5, 4, 1}
{5, 4, 1}	2	{5, 3, 1}	{5, 4, 1, 2, 6, Y}
{5, 3, 1, 2, 6, Y}	3	{5, 3, 1, 2, 6, Y}	{5, 4, 1, 6, Y}
{5, 4, 1, 6, Y}	4	{5, 3, 1, 6, Y}	{5, 4, 1, 2, 6, Y}
{5, 4, 1, 2, 6, Y}	5	{5, 3, 1, 6, Y}	{5, 4, 1, 2, 6, Y}
{5, 3, 1, 6, Y}	6	{5, 3, 1, 2, 6, Y}	{5, 4, 1, 6, Y}

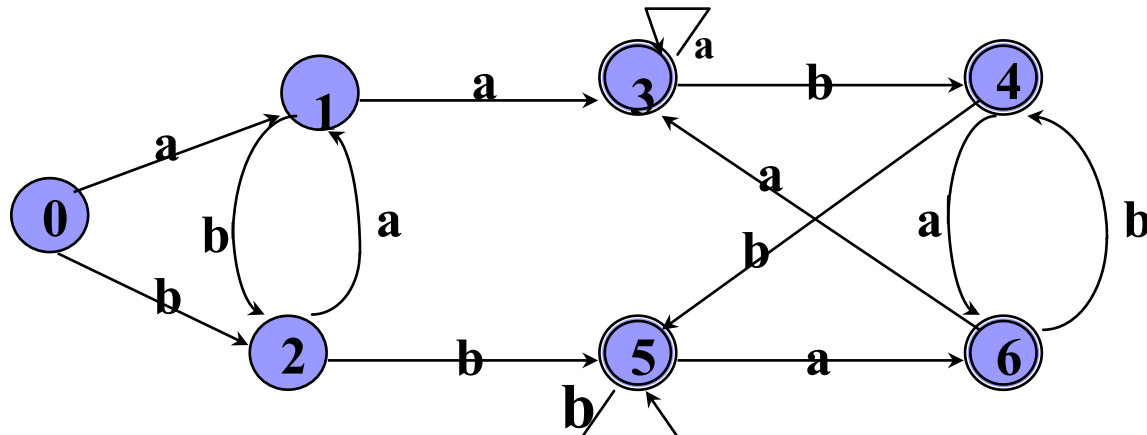
s	a	b
0	1	2
1	3	2
2	1	5
3	3	4
4	6	5
5	6	5
6	3	4

### 4) 画出状态转换图





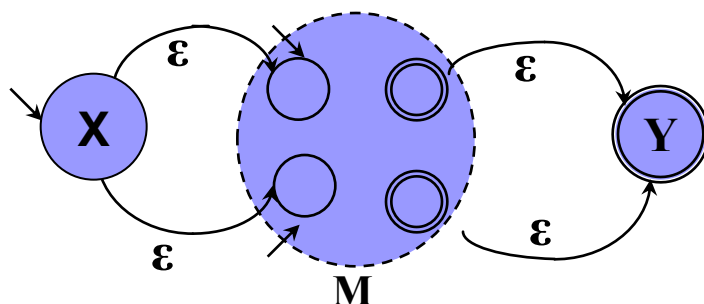
I		$I_a$		$I_b$	
{X, 5, 1}	0	{5, 3, 1}	1	{5, 4, 1}	2
{5, 3, 1}	1	{5, 3, 1, 2, 6, Y}	3	{5, 4, 1}	2
{5, 4, 1}	2	{5, 3, 1}	1	{5, 4, 1, 2, 6, Y}	5
{5, 3, 1, 2, 6, Y}	3	{5, 3, 1, 2, 6, Y}	3	{5, 4, 1, 6, Y}	4
{5, 4, 1, 6, Y}	4	{5, 3, 1, 6, Y}	6	{5, 4, 1, 2, 6, Y}	5
{5, 4, 1, 2, 6, Y}	5	{5, 3, 1, 6, Y}	6	{5, 4, 1, 2, 6, Y}	5
{5, 3, 1, 6, Y}	6	{5, 3, 1, 2, 6, Y}	3	{5, 4, 1, 6, Y}	4



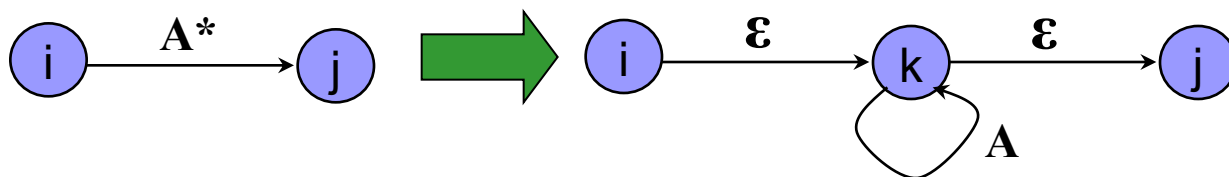
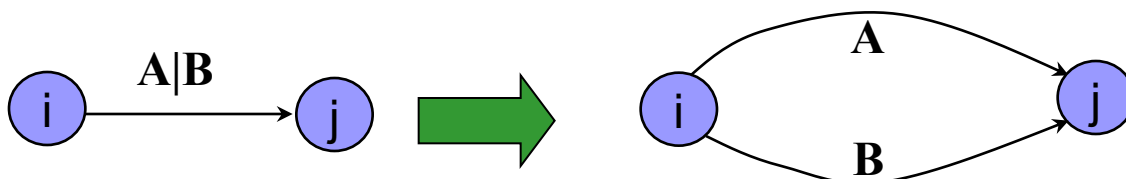
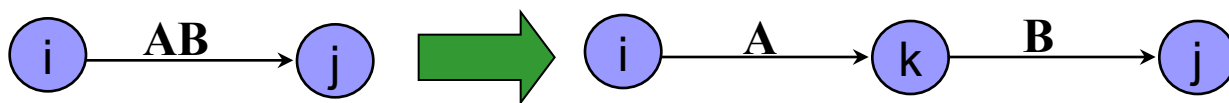
# NFA和DFA等价性证明

(1) 对NFA  $M$  的状态转换图进行改造, 得到 $M'$

1) 引进新的初始结点 $X$ 和终态结点 $Y$ ,  $X, Y \notin S$



2) 按以下规则扩展结点、加边



## (2) 将 $M'$ 进一步变换为DFA

- 设  $I$  是  $M'$  状态集的子集,  $I$  的  $\epsilon$ -闭包  $\epsilon\text{-CLOSURE}(I)$  为:
  - 1) 若  $q \in I$ , 则  $q \in \epsilon\text{-CLOSURE}(I)$ ;
  - 2) 若  $q \in I$ , 则从  $q$  出发经过任意条  $\epsilon$  弧可到达的任何状态  $q' \in \epsilon\text{-CLOSURE}(I)$ 。
- 设  $I$  是  $M'$  状态集的子集,  $a \in \Sigma$ , 定义
$$I_a = \epsilon\text{-CLOSURE}(J)$$
其中:  $J$  为从  $I$  中某一个状态结点出发, 经过一条  $a$  弧到达的状态结点的全体。

## (2) 将M'进一步变换为DFA (续)

### 1) 构造状态转换矩阵;

设 $\Sigma=\{a,b\}$ ,构造一张表, 表的形式为:

I	I <sub>a</sub>	I <sub>b</sub>
$\epsilon\_CLOSURE(\{X\})$		

2) 把表中第一列的每个子集看做一个新的状态, 重新命名, 其中, 第一行第一列的子集对应的状态是DFA的初态, 含有原终态Y的子集是DFA的终态

### 3) 画出新的 DFA



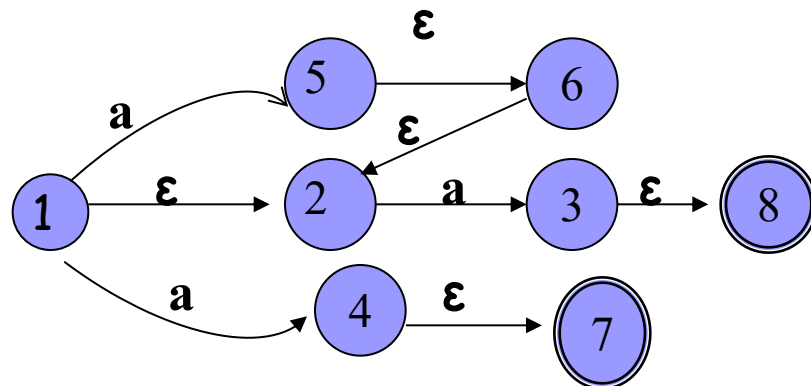
**例.** 如图所示的状态转换图

设  $I=\{1\}$ , 则

$\epsilon\_CLOSURE(I)=$

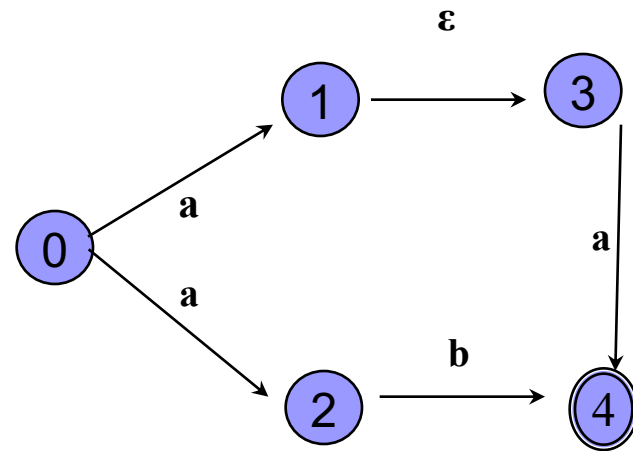
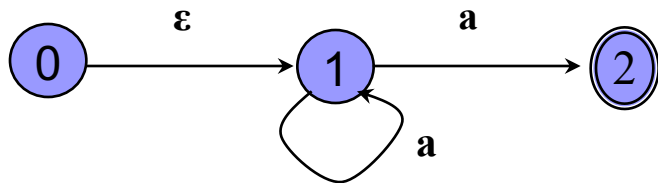
设  $I=\{1,2\}$ ,

$I_a=\epsilon\_CLOSURE\{5,4,3\}=$



注：实际上， $I_a$ 是从子集 $I$ 中任一状态出发，经 $a$ 弧（向后可跳过 $\epsilon$ 弧）而到达的状态集合。

# 随堂练习 (Canvas)



# 正规式与有限自动机的等价性

- 正规式与有限自动机是等价的。

(1) 对任何FA  $M$  , 都存在一个正规式 $V$  , 使得

$$L(V) = L(M);$$

(2) 对任何正规式 $V$  , 都存在一个FA  $M$  , 使得

$$L(M) = L(V)$$

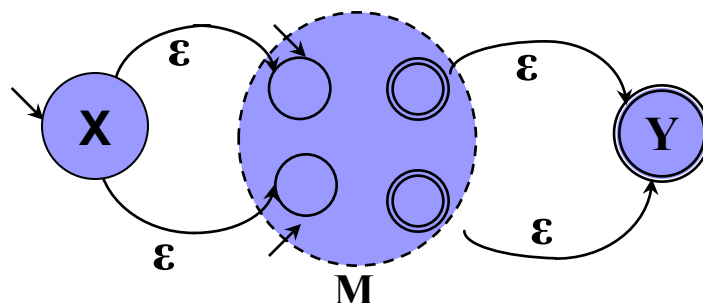
# 方向1: 从 NFA 构造等价的正规式 (状态消去法)

## ■ 思路:

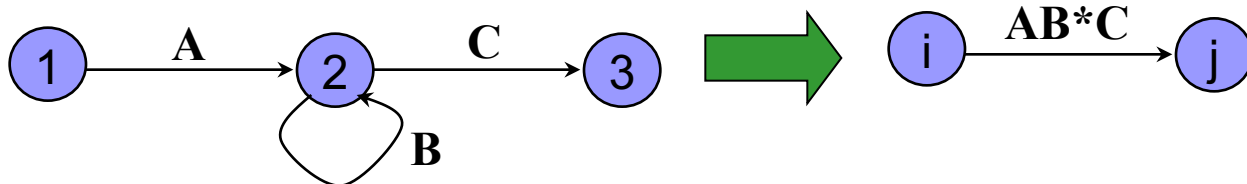
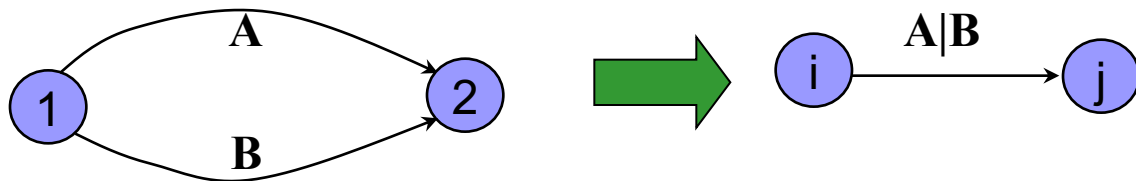
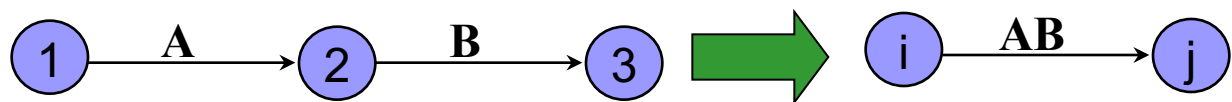
(1) 扩展自动机的概念, 允许正规式作为转移弧的标记.  
这样, 就有可能在消去某一中间状态时, 保证自动机能够接受的字符串集合保持不变.

(2) 在消去某一中间状态时, 与其相关的转移弧也将同时消去, 所造成的影响将通过修改从每一个前趋状态到每一个后继状态的转移弧标记来弥补.

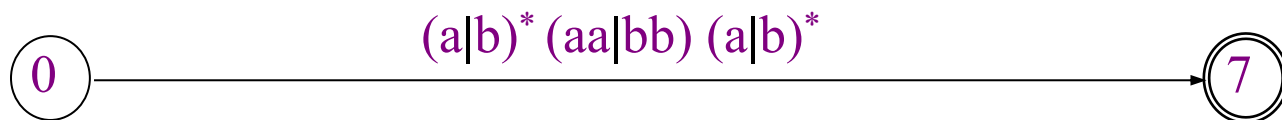
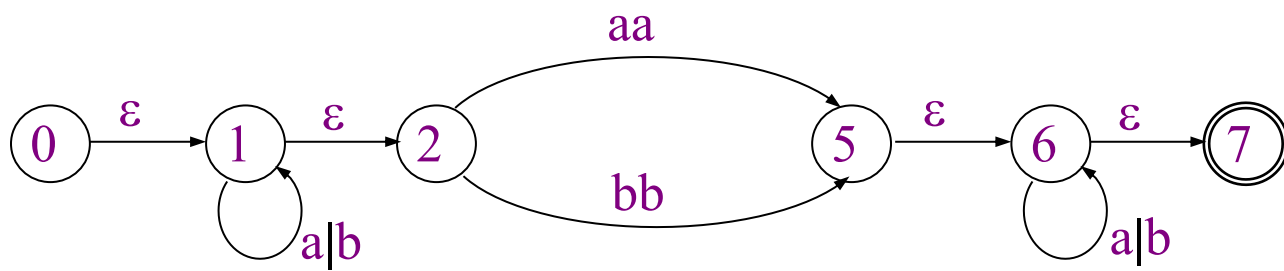
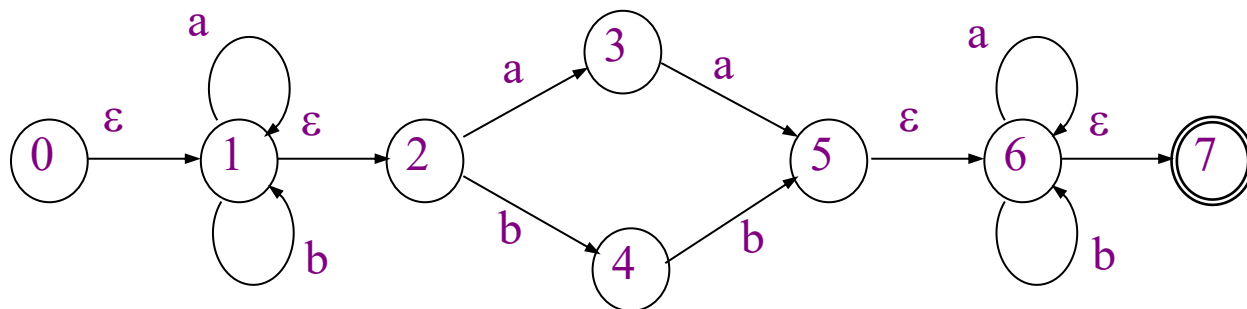
## 1) 增加X结、Y结，使初态、终态唯一



## 2) 反复用以下规则消去结点、合并边，直到剩下X结和Y结，X到Y上正规式为所求。

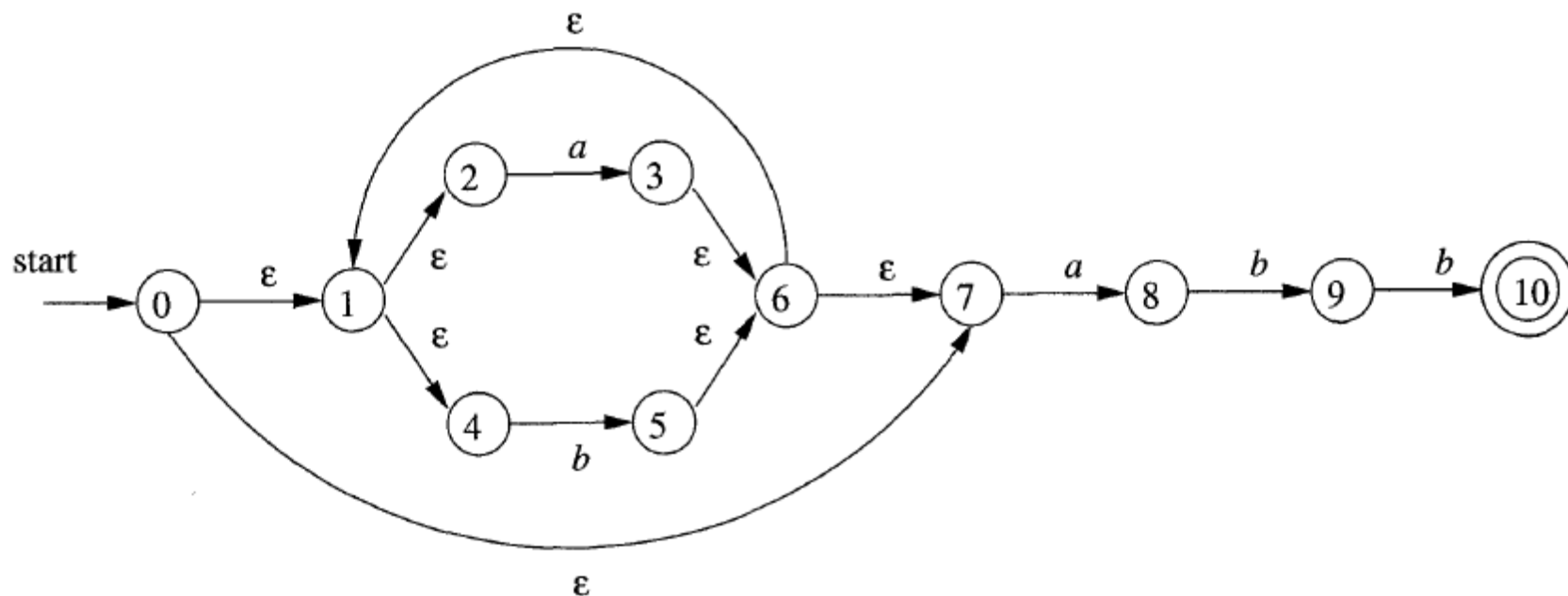


# 状态消去法举例



# 随堂练习 (Canvas)

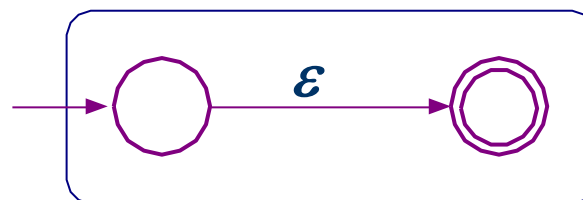
NFA  $\rightarrow$  正规式



# 方向2: 从正规式构造等价的NFA (Thompson算法)

基础:

1 对于  $\varepsilon$ , 构造为



2 对于  $\phi$ , 构造为



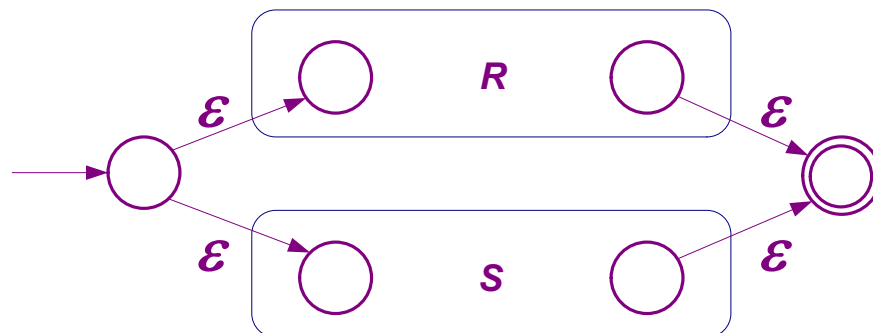
3 对于  $a$ , 构造为



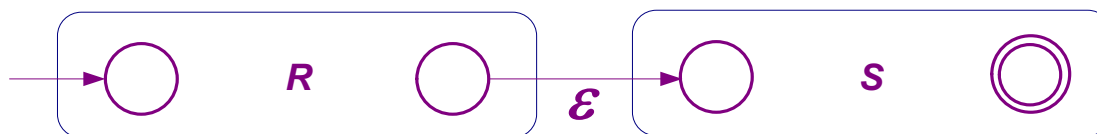


归纳:

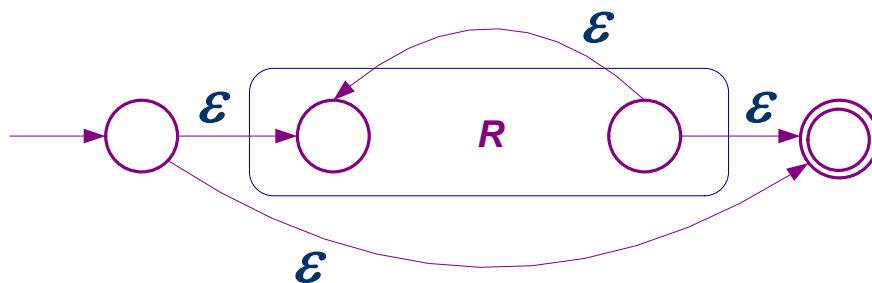
1 对于  $R|S$  , 构造为



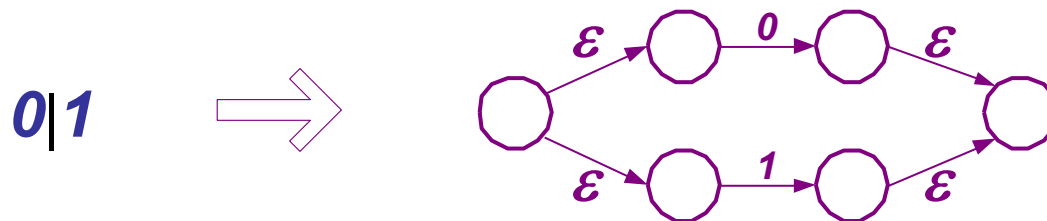
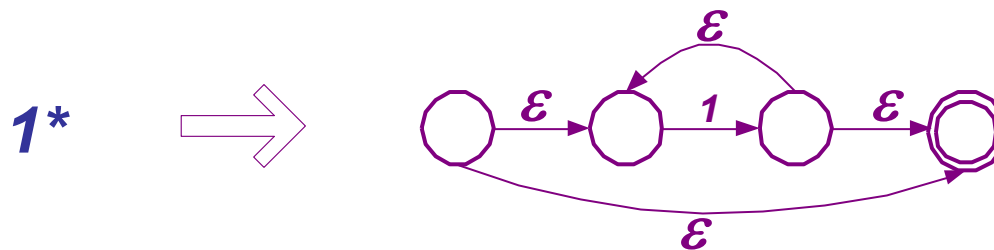
2 对于  $RS$  , 构造为



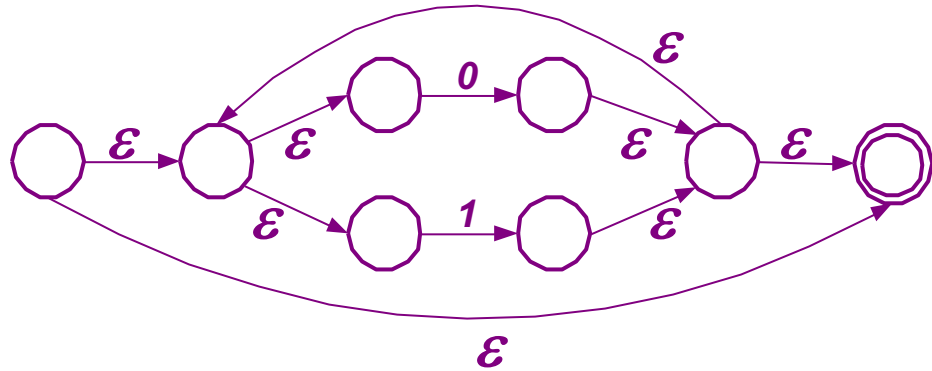
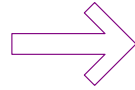
3 对于  $R^*$  , 构造为



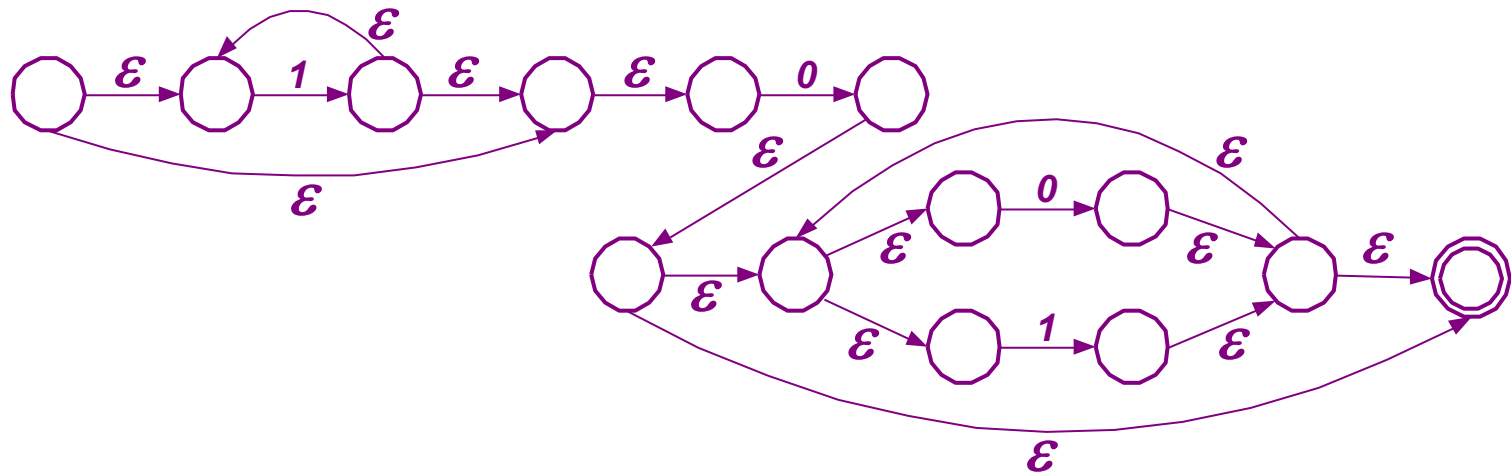
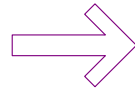
例. 设正则表达式  $1^*0(0|1)^*$ , 构造等价的NFA.



$(0|1)^*$



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



# 随堂练习

■  $(alb)^* abb(alb)^*$

# DFA的化简

- DFA  $M$  的化简是指寻找一个状态数比  $M$  少的 DFA  $M'$ , 使  $L(M) = L(M')$ 。
- 术语
  - **状态 $s$ 和 $t$ 等价**: 若从状态 $s$ 出发能读出字 $\alpha$ 停于终态, 则从 $t$ 出发也能读出 $\alpha$ 而停于终态; 反之, 若从状态 $t$ 出发能读出字 $\alpha$ 停于终态, 则从 $s$ 出发也能读出 $\alpha$ 而停于终态。
  - **状态 $s$ 和 $t$ 可区别**: 状态 $s$ 和 $t$ 不等价。 状态 $s$ 读入 $\alpha$ 停于终态, 但状态 $t$ 读入 $\alpha$ 不停于终态。
  - 例如: 终态与非终态是可区别的。

# DFA化简的思路

- 将 DFA  $M$  的状态集划分为不相交的子集, 使不同的两个子集的状态可区别, 同一个子集的状态都等价。

# DFA化简的步骤

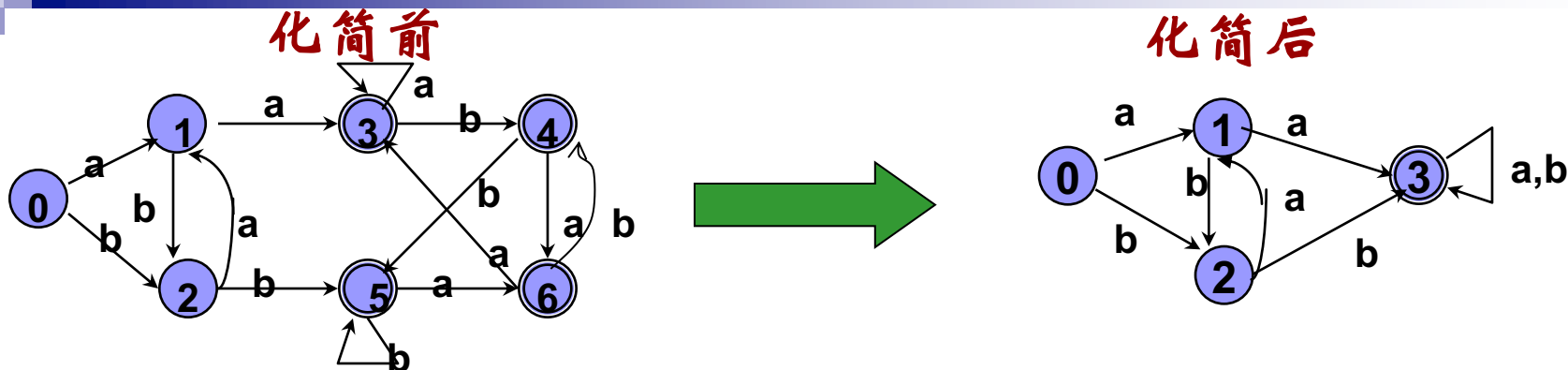
- 把状态集S划分为两个子集，得初始划分

$\Pi = \{ I^{(1)}, I^{(2)} \}$ , 其中  $I^{(1)}$  为终态集,  $I^{(2)}$  为非终态集;

- 设当前  $\Pi = \{ I^{(1)}, I^{(2)}, \dots, I^{(m)} \}$ , 检查 $\Pi$ 中每个 $I^{(k)}$ 是否可以再分

**依据为:**如果存在一个输入字符a, 使得 $I^{(k)}_a$ 不全包含在现行 $\Pi$ 的子集中, 就将 $I^{(k)}$ 进行划分。

- 一般地, 如果 $I^{(k)}_a$ 落入现行 $\Pi$ 的N个子集中, 则应将 $I^{(k)}$ 划分成N个不相交的组, 使得每个组 $I^{(ki)}$ 的 $I^{(ki)}_a$ 都落入 $\Pi$ 的同一子集。
- 重复第二步, 直至 $\Pi$ 中子集数不再增长为止。



初始划分  $\Pi_0 = \{ I^{(1)}, I^{(2)} \}$ ,  $I^{(1)} = \{ 3, 4, 5, 6 \}$ ,  $I^{(2)} = \{ 0, 1, 2 \}$

考察  $I^{(1)}_a = \{ 3, 6 \}$  包含于  $\{ 3, 4, 5, 6 \}$

$I^{(1)}_b = \{ 4, 5 \}$  包含于  $\{ 3, 4, 5, 6 \}$

$I^{(1)}$  不可再分,  $\Pi_0$  不变.

考察  $I^{(2)}_a = \{ 1, 3 \}$ , 其中  $\{ 1 \}_a = \{ 3 \}$ ,  $\{ 0, 2 \}_a = \{ 1 \}$ , 所以

$\{ 0, 1, 2 \}$  可分为  $\{ 1 \}$ ,  $\{ 0, 2 \}$  得  $\Pi_1 = \{ \{ 1 \}, \{ 0, 2 \}, \{ 3, 4, 5, 6 \} \}$

考察  $\{ 0, 2 \}_b = \{ 2, 5 \}$ ,  $\{ 0, 2 \}$  可分为  $\{ 0 \}$ ,  $\{ 2 \}$

得  $\Pi_2 = \{ \{ 0 \}, \{ 1 \}, \{ 2 \}, \{ 3, 4, 5, 6 \} \}$

令状态3代表  $\{ 3, 4, 5, 6 \}$ , 画出化简后的DFA

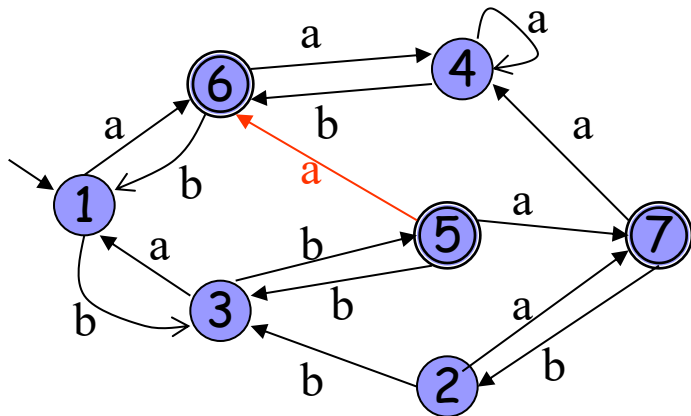
$I^{(1)}, I^{(2)}$  被 omi ga 划分, 分别接受 omi ga 前者可以到终态, 后者到不了终态

$\{ 1 \}, \{ 0, 2 \}$  被 a 划分, 分别接受 a 前者可以到  $\{ 3, 4, 5, 6 \}$ , 后者到  $\{ 0, 1, 2 \}$

$\{ 0 \}, \{ 2 \}$  被 b 划分, 分别接受 b 前者可以到  $\{ 0, 1, 2 \}$ , 后者到  $\{ 3, 4, 5, 6 \}$

化简后的DFA怎么画:  $\{ 0 \}$  接受 a 能到  $\{ 1 \}$ , 所以画一条状态0到状态1的线...  $\{ 3, 4, 5, 6 \}$  在原图中接受 a 或 b 都落到  $\{ 3, 4, 5, 6 \}$





$$\Pi_0 = \{\{1,2,3,4\}, \{5,6,7\}\}$$

因为  $\{1,2,3,4\}_a = \{6,7,1,4\}$  不全包含在  $\Pi_0$  的子集中, 需划分。又因为

$\{1,2\}_a = \{6,7\}$  落在  $\{5,6,7\}$  集合中,

$\{3,4\}_a = \{1,4\}$  落在  $\{1,2,3,4\}$  集合中,

$$\text{所以得 } \Pi_1 = \{\{1,2\}, \{3,4\}, \{5,6,7\}\}$$

因为  $\{3,4\}_a = \{1,4\}$ , 不全包含在  $\Pi_1$  的子集中, 需划分为  $\{3\}$ ,  $\{4\}$  得:

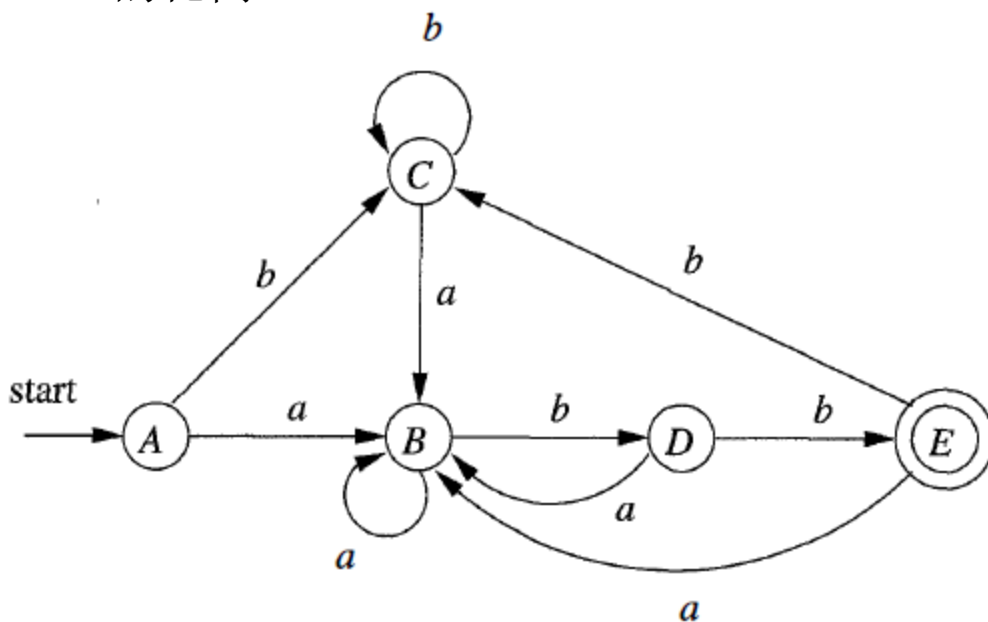
$$\Pi_2 = \{\{1,2\}, \{3\}, \{4\}, \{5,6,7\}\}$$

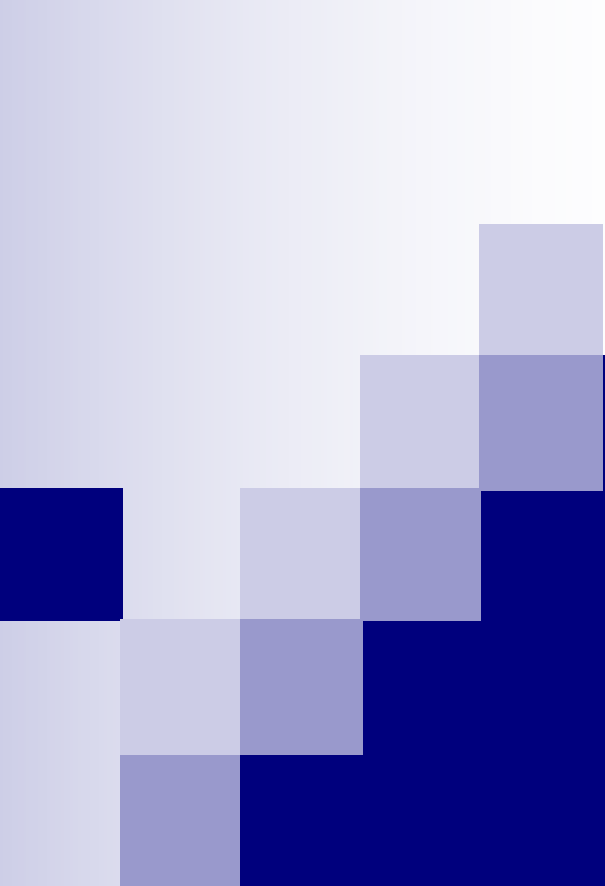
因为  $\{5,6,7\}_a = \{7,4\}$ , 所以

$$\Pi_3 = \{\{1,2\}, \{3\}, \{4\}, \{5\}, \{6,7\}\}$$

# 随堂练习 (Canvas)

DFA的化简





# 正规文法和有限自动机的等价性

# 正规文法与有限自动机的等价性

- 对于正规文法G和有限自动机M, 如果 $L(G)=L(M)$ , 则称**G和M是等价的**
- 正规文法与有限自动机的等价性
  - (1) 对每一个右 (左) 线性正规文法G, 都存在一个有限自动机M, 使得 $L(G)=L(M)$
  - (2) 对每一个有限自动机M, 都存在一个右 (左) 线性正规文法G, 使得 $L(G)=L(M)$

# 右线性正规文法生成NFA方法

设右线性正规文法  $G=(V_N, V_T, S, \epsilon)$ 。

将  $V_N$  中的每一非终结符号视为状态符号，并增加一个新的终态符号  $f \notin V_N$ 。

令  $M=(V_N \cup \{f\}, V_T, \delta, S, \{f\})$ ，其中  $\delta$  由以下规则定义：

(a) 若对某个  $A \in V_N$  及  $a \in V_T \cup \{\epsilon\}$ ，存在  $A \rightarrow a$ ，则令

$$\delta(A, a) = f;$$

(b) 对任意的  $A \in V_N$  及  $a \in V_T \cup \{\epsilon\}$ ，存在

$$A \rightarrow aA_1 \mid aA_2 \mid \dots \mid aA_k$$

则令

$$\delta(A, a) = \{A_1, \dots, A_k\};$$

## 右线性正规文法

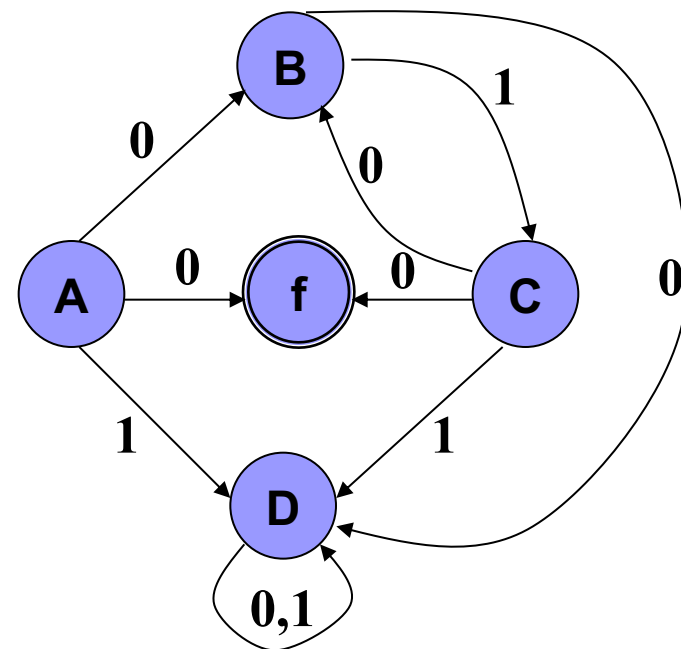
$G = (\{A, B, C, D\}, \{0, 1\}, A, \epsilon)$ , 其中

$A \rightarrow 0|0B|1D$

$B \rightarrow 0D|1C$

$C \rightarrow 0|0B|1D$

$D \rightarrow 0D|1D$



NFA  $M = (\{A, B, C, D, f\}, \{0, 1\}, \delta, A, \{f\})$ , 其中

$\delta(A, 0) \rightarrow \{B, f\}$     $\delta(A, 1) \rightarrow \{D\}$

$\delta(B, 0) \rightarrow \{D\}$     $\delta(B, 1) \rightarrow \{C\}$

$\delta(C, 0) \rightarrow \{B, f\}$     $\delta(C, 1) \rightarrow \{D\}$

$\delta(D, 0) \rightarrow \{D\}$     $\delta(D, 1) \rightarrow \{D\}$

# 左线性正规文法生成NFA方法

设左线性正规文法  $G=(V_N, V_T, S, \mathbf{f})$ 。

应该是起始态符号

将  $V_N$  中的每一非终结符号视为状态符号，并增加一个新的终态符号  $q_0 \notin V_N$ 。

令  $M=(V_N \cup \{q_0\}, V_T, \delta, q_0, \{S\})$ ，其中  $\delta$  由以下规则定义：

(a) 若对某个  $A \in V_N$  及  $a \in V_T \cup \{\epsilon\}$ ，存在  $A \rightarrow a$ ，则令

$$\delta(q_0, a) = A;$$

(b) 对任意的  $A \in V_N$  及  $a \in V_T \cup \{\epsilon\}$ ，存在

$$A_1 \rightarrow Aa, A_2 \rightarrow Aa, \dots, A_k \rightarrow Aa$$

则令

$$\delta(A, a) = \{A_1, \dots, A_k\};$$

# 课堂练习

右线性正规文法，求等价的FA

$G = (\{A, B\}, \{l, d\}, A, \epsilon)$ ，其中  $\epsilon$  为

$A \rightarrow l \mid IB$

$B \rightarrow l \mid d \mid IB \mid dB$

左线性正规文法，求等价的FA

$G = (\{A\}, \{l, d\}, A, \epsilon)$ ，其中  $\epsilon$  为

$A \rightarrow l \mid Al \mid Ad$



# 课后练习

- P65, 15 给定右线性文法, 画NFA

# DFA产生右线性正规文法方法

设DFA  $M = (S, \Sigma, \delta, s_0, F)$ .

(1) 若  $s_0 \notin F$ , 我们令  $G = (S, \Sigma, s_0, \epsilon)$ , 其中,  $\epsilon$  是由以下规则定义的产生式集合:

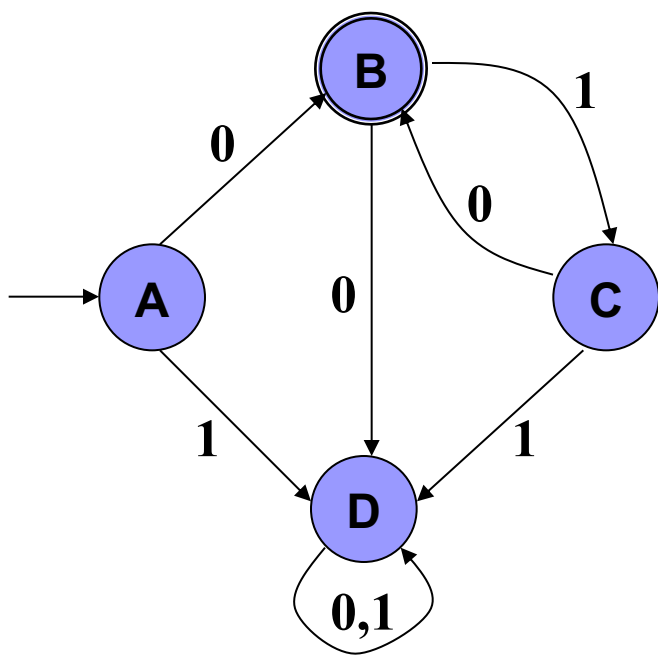
对任何  $a \in \Sigma$  及  $A, B \in S$ , 若有  $\delta(A, a) = B$ , 则

(a) 当  $B \notin F$ , 令  $A \rightarrow aB$

(b) 当  $B \in F$ , 令  $A \rightarrow a \mid aB$

(2) 若  $s_0 \in F$ , 添加新的非终结符号  $s_0'$  和产生式  $s_0' \rightarrow \epsilon \mid s_0$ , 并用  $s_0'$  代替  $s_0$  作为开始符号

# 举例：DFA产生正规文法



## 右线性正规文法

$G = (\{A, B, C, D\}, \{0, 1\}, A, \epsilon)$ , 其中

$A \rightarrow 0|0B|1D$

$B \rightarrow 0D|1C$

$C \rightarrow 0|0B|1D$

$D \rightarrow 0D|1D$

## 左线性正规文法

$G = (\{B, C, D\}, \{0, 1\}, B, \epsilon)$ , 其中

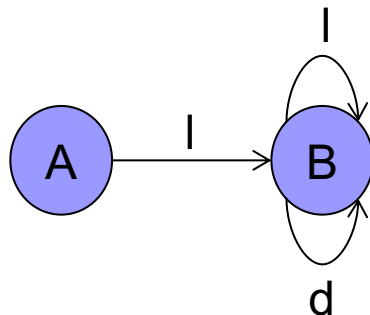
$B \rightarrow C0|0$

$C \rightarrow B1$

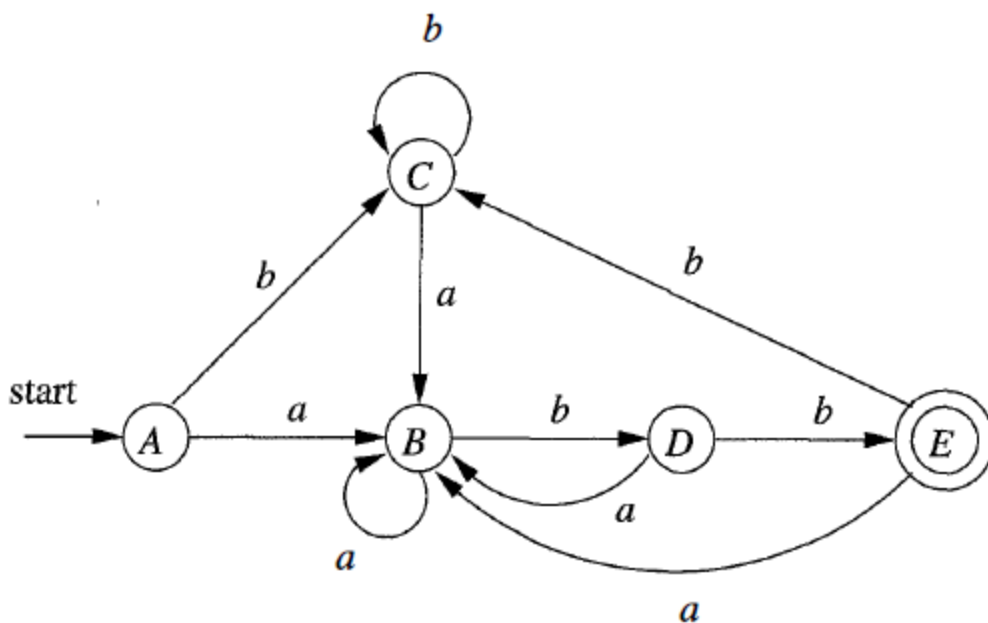
$D \rightarrow D0|D1|C1|B0|1$

# 课堂练习

求与下面DFA等价的左线性文法和右线性文法



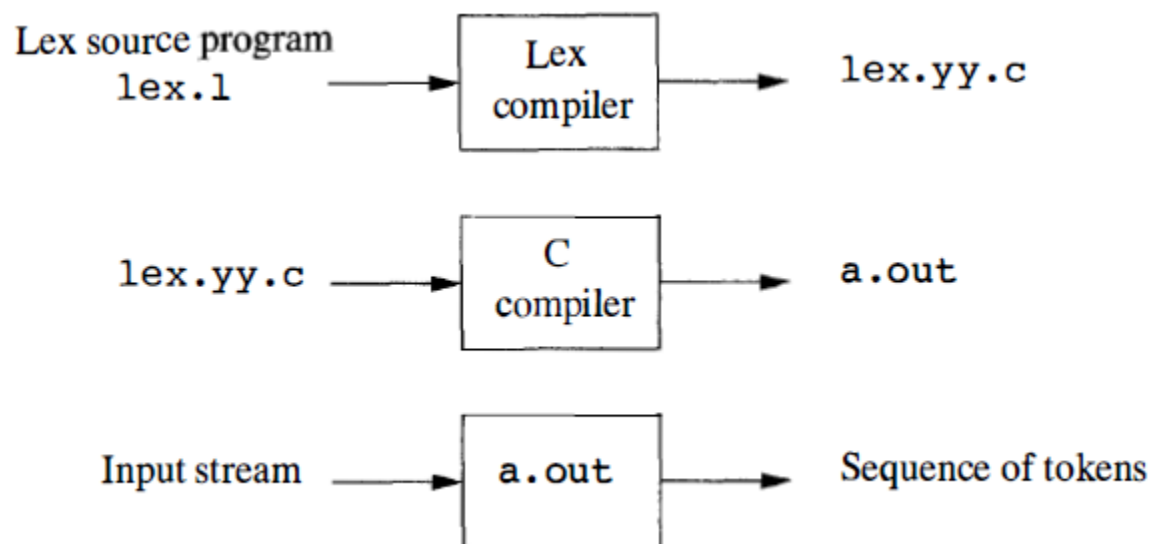
# 求DFA的正规文法 (Canvas)



# 内容线索

- ✓ 对于词法分析器的要求
- ✓ 词法分析器的设计
- ✓ 正规表达式与有限自动机
- 词法分析器的自动生成

# 词法分析器的自动产生——LEX

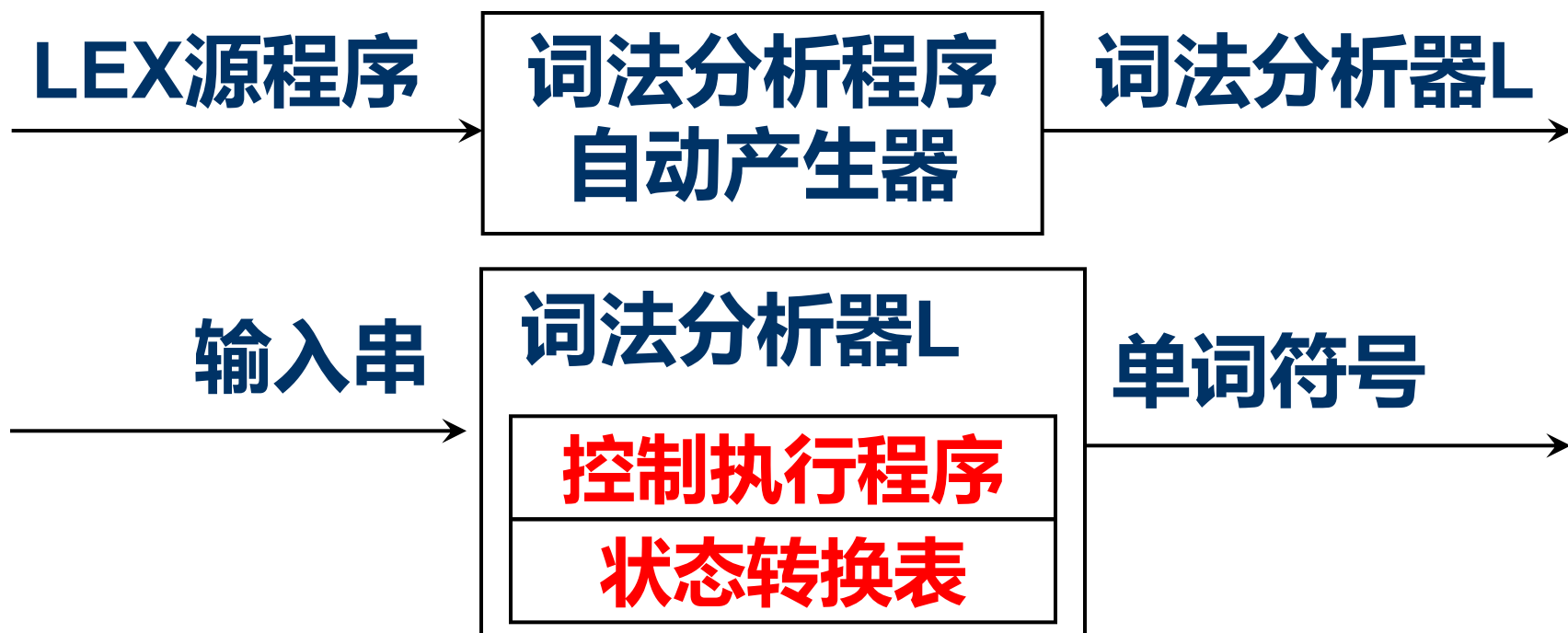


Lex 和 Yacc 从入门到精通

熊春雷

# 词法分析器的自动产生——LEX

- LEX程序由一组正规式以及与每个正规式相应的动作组成。
  - 动作本身是一小段程序代码，它指出了当按正规式识别出一个单词符号时应采取的行动。





# 语言LEX的一般描述

## (1) 正规式辅助定义式

$d_1 \rightarrow r_1$

$d_2 \rightarrow r_2$

...

$d_n \rightarrow r_n$

$r_i$  为正规式,  $d_i$  为该正规式的简名,

$r_i$  中只允许出现  $\Sigma$  中的字符和已定义的简名  $d_1, d_2, \dots, d_{i-1}$

## (2) 识别规则: 是一串下述形式的LEX语句

$P_1 \quad \{A_1\}$

$P_2 \quad \{A_2\}$

...

$P_m \quad \{A_m\}$

LEX源程序包括:

{辅助定义部分}

declarations

%%

识别规则部分

translation rules

%%

{用户子程序部分}

auxiliary functions

$P_i$  为  $\Sigma \cup \{d_1, d_2, \dots, d_n\}$  上的正规式;

$A_i$  为识别出词形  $P_i$  后应采取的动作, 是一小段程序代码。

## 示例. 正规式辅助定义式

标识符:  $\text{letter} \rightarrow A \mid B \mid \dots \mid Z$

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

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

整常数:  $\text{integer} \rightarrow \text{digit}(\text{digit})^*$

$\text{sign} \rightarrow + \mid - \mid \varepsilon$

$\text{signedinteger} \rightarrow \text{sign integer}$

不带指数部分的实常数:

$\text{decimal} \rightarrow \text{signedinteger} . \text{integer}$   
 $\quad \mid \text{signedinteger} . \mid \text{sign} . \text{Integer}$

带指数部分的实常数:

$\text{exponential} \rightarrow (\text{decimal}$   
 $\quad \mid \text{signedinteger}) E \text{ signedinteger}$

## 示例. 识别小语言单词符号的 LEX 程序

AUXILIARY DEFINITIONS /\* 辅助定义 \*/

letter  $\rightarrow$  A | B | ... | Z

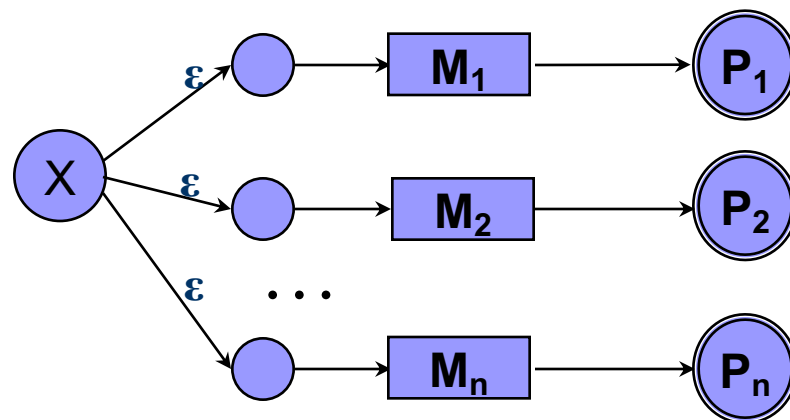
digit  $\rightarrow$  0 | 1 | ... | 9

正规式

RECOGNITION RULES /\* 识别规则 \*/

1	DIM	{RETURN (1, _ )}
2	IF	{RETURN (2, _ )}
3	DO	{RETURN (3, _ )}
4	STOP	{RETURN (4, _ )}
5	END	{RETURN (5, _ )}
6	letter(letter   digit)*	{RETURN (6, getSymbolTableEntryPoint() )}
7	digit (digit)*	{RETURN (7, getConstTableEntryPoint() )}
8	=	{RETURN (8, _ )}
9	+	{RETURN (9, _ )}
10	*	{RETURN (10, _ )}
11	**	{RETURN (11, _ )}
12	,	{RETURN (12, _ )}
13	(	{RETURN (13, _ )}
14	)	{RETURN (14, _ )}

# LEX 的实现



## ■ 方法

- 由LEX 编译程序将 LEX 源程序改造为一个词法分析器，即构造相应的 DFA

## ■ 步骤

- 对每条识别规则 $P_i$ 构造一个相应的 NFA  $M_i$
- 引入一个新的初态 $X$ , 连接成 NFA  $M$
- 用子集法将其确定化并化简
- 将 DFA 转换为词法分析程序

## ■ 注意

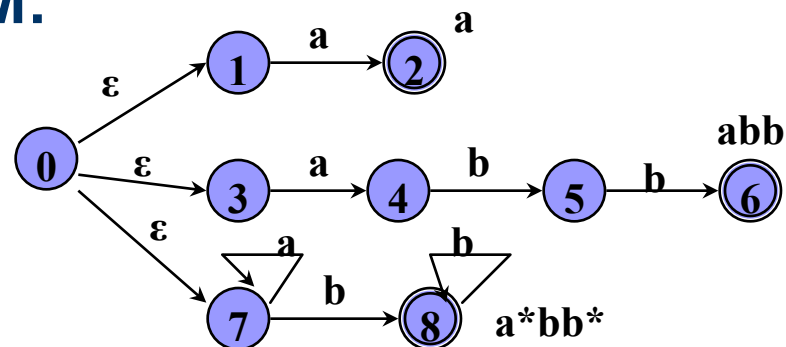
- 匹配最长子串(最长匹配原则)
- 多个最长子串匹配 $P_i$ , 以前面的 $P_i$ 为准(优先匹配原则)

## 例. LEX 程序:

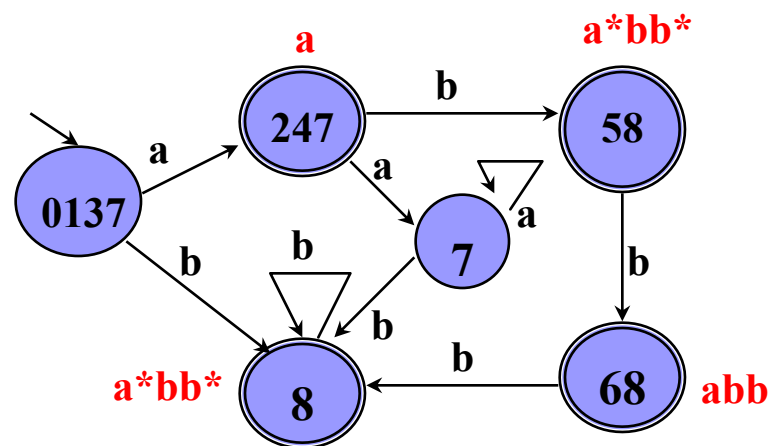
```

a      { }
abb    { }
a*bb*  { }
    
```

## NFA M:



状态	a	b	识别单词
0 1 3 7	2 4 7	8	
2 4 7	7	5 8	a
8		8	a*bb*
7	7	8	
5 8		6 8	a*bb*
6 8		8	abb



输入: abbbabb

输出: abbb abb \*

# 总结

- 算法1 (Thompson算法)
  - 正规式->NFA
- 算法2 (子集法)
  - NFA->DFA
- 算法3 (状态等价法)
  - DFA化简
- 其他 (DFA代码化)
  - DFA转为代码

**Dank u**

Dutch

**Merci**

French

**Спасибо**

Russian

**Gracias**

Spanish

شكراً

Arabic

감사합니다

Korean

**Tack så mycket**

Swedish

धन्यवाद

Hindi

תודה רבה

Hebrew

**Obrigado**

Brazilian  
Portuguese

谢谢

Chinese

**Dankon**

Esperanto

***Thank You !***

ありがとうございます

Japanese

**Trugarez**

Breton

**Danke**

German

**Tak**

Danish

**Grazie**

Italian

நன்றி

Tamil

**děkuji**

Czech

ขอบคุณ

Thai

**go raibh maith agat**

Gaelic

# 章节习题

- NFA转DFA: P64, 12 (a)
- DFA的化简: P64 12(b)
- DFA和正规式转化: P65, 15