

---

# 编译原理

- 第一章 编译程序概述
  - 第二章 PL/0编译程序的实现
  - 第三章 文法和语言
  - 第四章 词法分析
  - 第五章 自顶向下语法分析方法
  - 第六章 自底向上优先分析方法
  - 第七章 LR分析方法
  - 第八章 语法制导翻译和中间代码生成
  - 第九章 符号表
  - 第一〇章 代码优化
  - 第一一章 代码生成
-

---

# 词法分析 内容回顾

- 词法分析的主要任务
    - 从左到右逐个字符地对源程序进行扫描，产生一个个单词序列，用以语法分析。
  - 计算机实现词法自动分析的基本思路
    - 将单词符号的语法用有效的工具描述；
    - 基于该描述建立单词的识别机制；
    - 基于词法分析程序的自动构造原理，设计和实现词法分析程序
-

---

- 单词的描述工具

- 正规文法（判断一个文法是否正规文法）
- 正规式（给定字母表 $\Sigma$ ，写出 $\Sigma$ 上合法的正规式）、正规集（给定正规式，列出它对应的正规集；给定正规集，列出它对应的正规式）
- 正规文法与正规式的相互转换（二者的相互转换）

- 单词的识别机制

- 确定有穷自动机
- 不确定有穷自动机

---

## 自动识别单词的方法：

- (1) 把单词的结构用**正规式**描述；
  - (2) **把正规式转换为一个NFA**；
  - (3) **把NFA转换为相应的DFA**；
  - (4) 基于**DFA**构造词法分析程序。
-

## (一) 确定的有穷自动机**DFA** 定义

一个确定的有穷自动机**M**是一个五元组：

**M = (K, Σ, f, S, Z)**，其中：

- ① **K**是一个有穷集，每个元素表示一个状态；
- ② **Σ**是一个有穷字母表，每个元素是一个输入字符；
- ③ **f**是转换函数，是在 **$K \times \Sigma \rightarrow K$** 上的映象，如：  
$$f(K_i, a) = K_j \quad (K_i, K_j \in K);$$
- ④ **S**是初态， **$S \in K$** ；
- ⑤ **Z**  $\subset$  **K**，是终态集。

### 3、接受（识别）的概念： 自动识别单词

对于 $\Sigma^*$ 中的任何字符串 $t$ ，若存在一条初态到某一终态的路，且这条路上所有弧的标记符连接成的字符串等于 $t$ ，则称 $t$ 可为**DFA M**所接受。

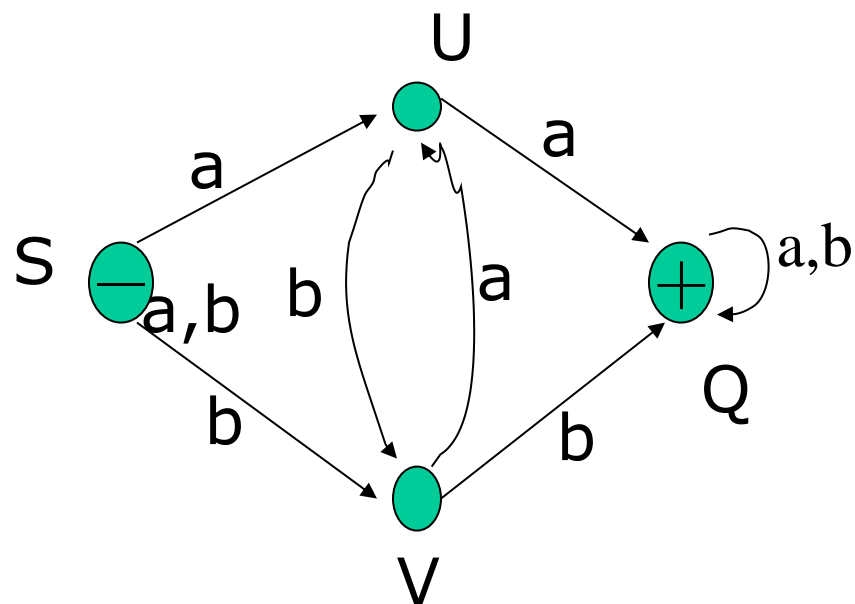
若**M**的初态同时又是终态，则空字可为**M**所接受。

### 4、DFA M 运行的理解：

① 设 $Q \in K$ ，函数 $f(Q, \varepsilon) = Q$ ，则输入字符串是空串，并停留在原状态上。

② 输入字符串 $t$ （ $t$ 表示成 $Tt_1$ 形式， $T \in \Sigma$ ， $t_1 \in \Sigma^*$ ），在**DFA M**上运行的定义为： $f(Q, Tt_1) = f(f(Q, T), t_1)$ ，其中 $Q \in K$ 。

状态图如下：



例如：**baab**字符串被**DFA**所接受，**DFA**见上例。

$$\begin{aligned} f(S, baab) &= f(f(S, b), aab) = f(V, aab) \\ &= f(f(V, a), ab) = f(U, ab) = f(f(U, a), b) \\ &= f(Q, b) = Q \quad (Q \text{ 是终态}) \end{aligned}$$

③**DFA M**所能接受的字符串的全体记为**L (M)** — 称为**语言**（也即句子的集合）

## 5、DFA的确定性：

当**f:  $K \times \Sigma \rightarrow K$** 是一个**单值函数**，即对任何状态  **$K \in R$** ，输入符号  **$a \in K$** ， **$f(k, a)$** 唯一确定下一状态。



---

**DFA的行为很容易用程序来模拟.**

**DFA  $M = (K, \Sigma, f, S, Z)$  的行为的模拟程序**

- $K := S;$**
- $c := \text{getchar};$**
- while  $c \neq \text{eof}$  do**
  - $\{ K := f(K, c);$**
  - $c := \text{getchar};$**
  - $\};$**
- if  $K$  is in  $Z$  then return ('yes')**
- else return ('no')**

---

## 自动识别单词的方法：

- (1) 把单词的结构用**正规式**描述；
- (2) **把正规式转换为一个NFA**；
- (3) **把NFA转换为相应的DFA**；
- (4) 基于**DFA**构造词法分析程序。

正则表达式  $\Rightarrow$  NFA  $\Rightarrow$  DFA  $\Rightarrow$  min DFA

---

## (二) 不确定的有穷自动机NFA

一个不确定的有穷自动机**NFA** **M**是一个五元组：  
**M** = (**K**, **Σ**, **f**, **S**, **Z**)，其中：

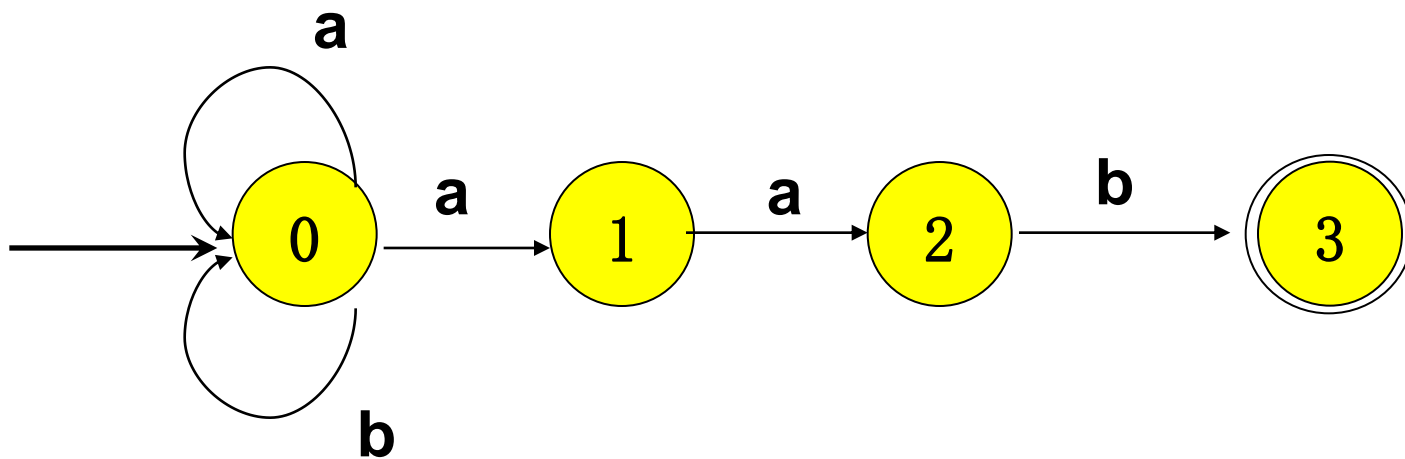
- ① **K**是一个有穷集，每个元素表示一个状态；
- ② **Σ**是一个有穷字母表，每个元素是一个输入字符；
- ③ **f**是一个从**K** × **Σ**<sup>\*</sup>到**K**上的子集的映象；

$$f: K \times \Sigma^* \rightarrow 2^K$$

- ④ **S** ⊂ **K**，是一个非空初态集；
- ⑤ **Z** ⊂ **K**，是一个终态集。

与DFA区别：多值函数  $f(K_i, a) = K_j$   $f(K_i, a) = K_t$ ；允许输入字符为 $\epsilon$

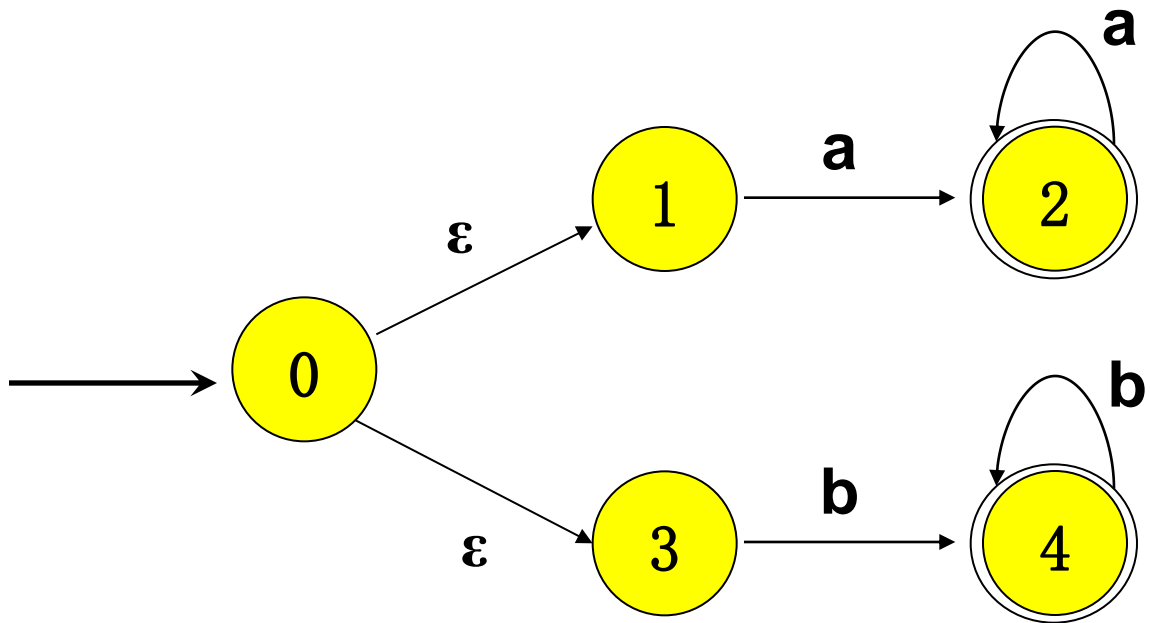
# NFA示例-1



NFA 表示的语言:

$(a|b)^*aab$

# NFA示例-2



NFA 表示的语言:

$aa^*|bb^*$

可不可以用DFA

来表示?

---

例4.7：一个NFA，

$M = (\{0,1,2,3,4\}, \{a,b\}, f, \{0\}, \{2,4\})$

其中：  $f(0,a) = \{0,3\}$

$f(2,b) = \{2\}$

$f(0,b) = \{0,1\}$

$f(3,a) = \{4\}$

$f(1,b) = \{2\}$

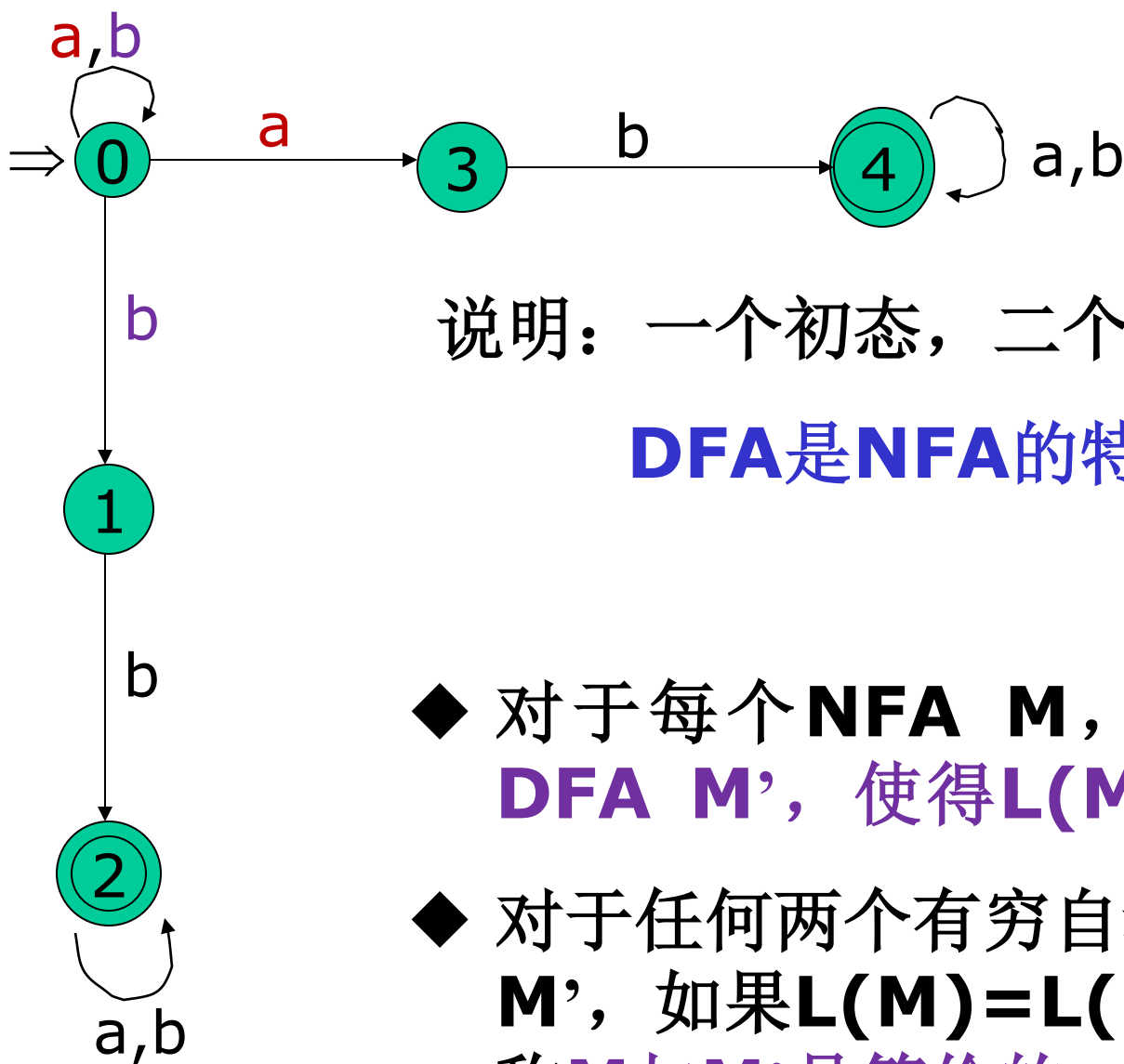
$f(4,a) = \{4\}$

$f(2,a) = \{2\}$

$f(4,b) = \{4\}$

状态图表示如下：

---



说明：一个初态，二个终态。

**DFA是NFA的特例。**

- ◆ 对于每个 **NFA M**，存在一个 **DFA M'**，使得  $L(M) = L(M')$ 。
- ◆ 对于任何两个有穷自动机 **M** 和 **M'**，如果  $L(M) = L(M')$ ，则称 **M** 与 **M'** 是等价的。

---

# NFA 转换为等价的DFA

- 从NFA的矩阵表示中可以看出，表项通常是一**状态的集合**，而在DFA的矩阵表示中，表项是一个**状态**
  - $NFA \Rightarrow DFA$  基本思路是：**DFA的每一个状态对应NFA的一组状态**，DFA使用它的状态去记录在NFA读入一个输入符号后可能达到的所有状态
-



---

定义对状态集合I的几个有关运算：

1. **状态集合I的  $\varepsilon$ -闭包**，表示为  $\varepsilon\text{-closure}(I)$ ，定义为一状态集，是状态集I中的任何状态S经任意条  $\varepsilon$  弧而能到达的状态的集合。

状态集合I的任何状态S都属于  $\varepsilon\text{-closure}(I)$

2. **状态集合I的a弧转换**，表示为  $\text{move}(I, a)$  定义为状态集合J，其中J是所有那些可从I中的某一状态经过一条a弧而到达的状态的全体。

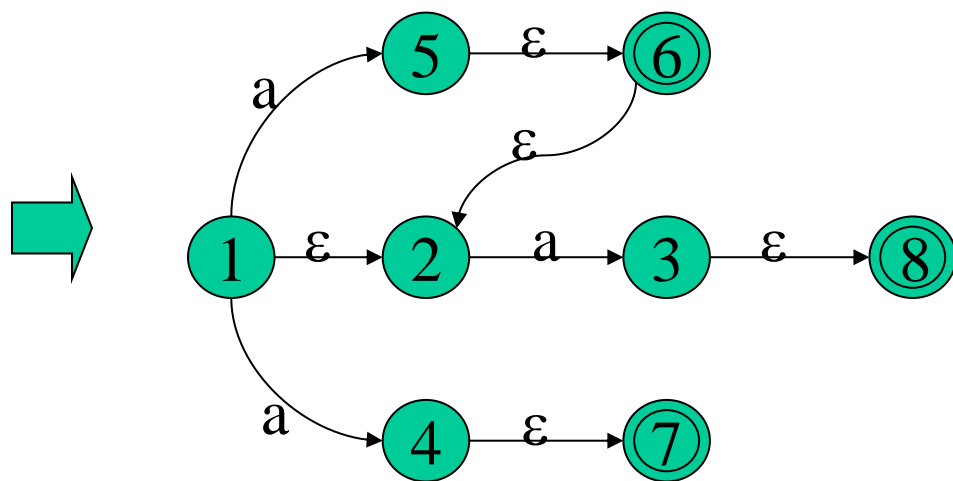
# 状态集合I的有关运算的例子

$I=\{1\}$ ,  $\epsilon$ -closure( $I$ )=?;

$I=\{5\}$ ,  $\epsilon$ -closure( $I$ )=?;

$\text{move}(\{1,2\},a)=?$

$\epsilon$ -closure( $\{5,3,4\}$ )=?;



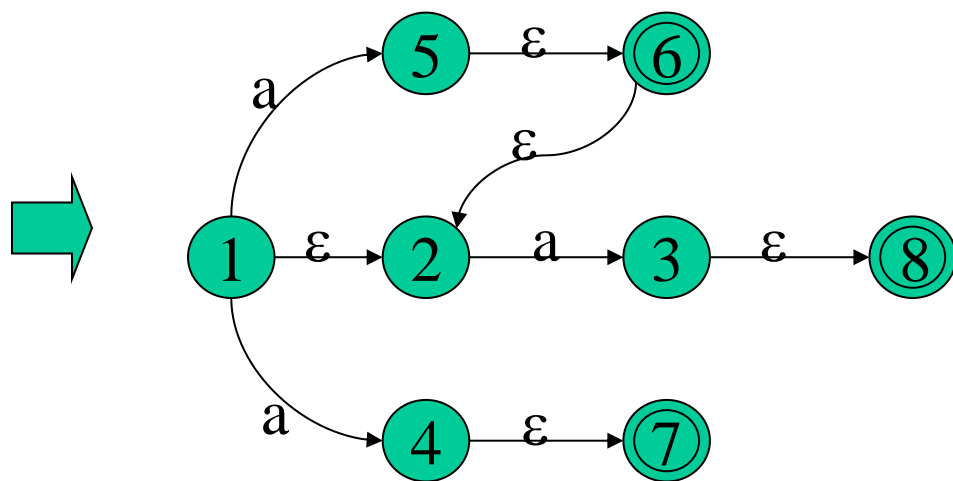
## 状态集合I的有关运算的例子

$I=\{1\}$ ,  $\epsilon$ -closure( $I$ )= $\{1,2\}$ ;

$I=\{5\}$ ,  $\epsilon$ -closure( $I$ )= $\{5,6,2\}$ ;

$\text{move}(\{1,2\},a)=\{5,3,4\}$

$\epsilon$ -closure( $\{5,3,4\}$ )= $\{2,3,4,5,6,7,8\}$ ;



# NFA确定化算法（子集法）

NFA  $N=(K, \Sigma, f, K_0, K_t)$  按如下办法构造一个DFA  $M=(S, \Sigma, d, S_0, S_t)$ , 使得  $L(M)=L(N)$ :

1. 构造DFA  $M$  的状态, 选择NFA  $N$  的状态的一些子集构成:

$M$  的状态集  $S$  由  $K$  的一些子集组成。用  $[S_1 S_2 \dots S_j]$  表示  $S$  的元素, 其中  $S_1, S_2, \dots, S_j$  是  $K$  的状态。并且约定, 状态  $S_1, S_2, \dots, S_j$  是按某种规则排列的, 即对于子集  $\{S_1, S_2\} = \{S_2, S_1\}$  来说,  $S$  的状态就是  $[S_1 S_2]$ ;

---

2. M和N的输入字母表是相同的，即是 $\Sigma$ ；

3. 构造DFA M的转换函数，根据新构造的状态和字母表中的字符构造：

转换函数是这样定义的：

$$d([S_1 S_2 \dots S_j], a) = [R_1 R_2 \dots R_t]$$

其中  $\{R_1, R_2, \dots, R_t\} = \varepsilon\text{-closure}(\text{move}(\{S_1, S_2, \dots, S_j\}, a))$

4  $S_0 = \varepsilon\text{-closure}(K_0)$  为M的开始状态；

5  $S_t = \{[S_i S_k \dots S_e], \text{ 其中 } [S_i S_k \dots S_e] \in S \text{ 且 } \{S_i, S_k, \dots, S_e\} \cap K_t \neq \Phi\}$

---

## 构造NFA $N$ 的状态 $K$ 的子集的算法:

把多值转换函数所转换到的多值（多状态）的集合作为一个子集，映射到DFA的一个新的状态

假定所构造的子集族为 $C$ ，即 $C = (T_1, T_2, \dots, T_I)$ ，其中 $T_1, T_2, \dots, T_I$ 为状态 $K$ 的子集。

- 1 开始，令 $\varepsilon\text{-closure}(K_0)$ 为 $C$ 中唯一成员，并且它是未被标记的。

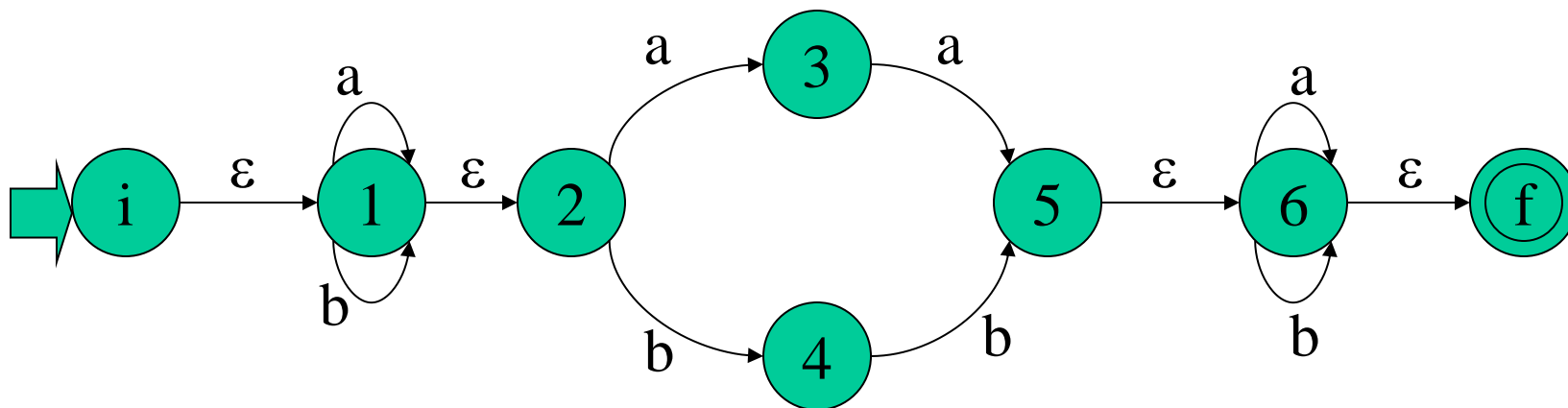
---

```
2  while (C中存在尚未被标记的子集T) do
    {
        标记T;
        for 每个输入字母a do
            {
                U:=  $\epsilon$ -closure(move(T,a));
                if U不在C中 then
                    将U作为未标记的子集加在C中
            }
    }
```

---

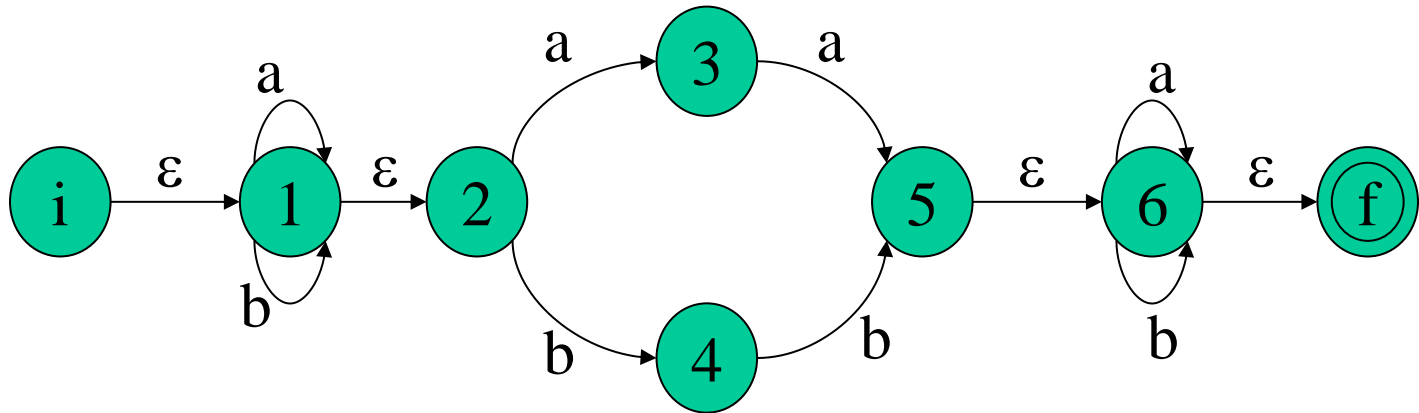
# NFA的确定化

## 例子



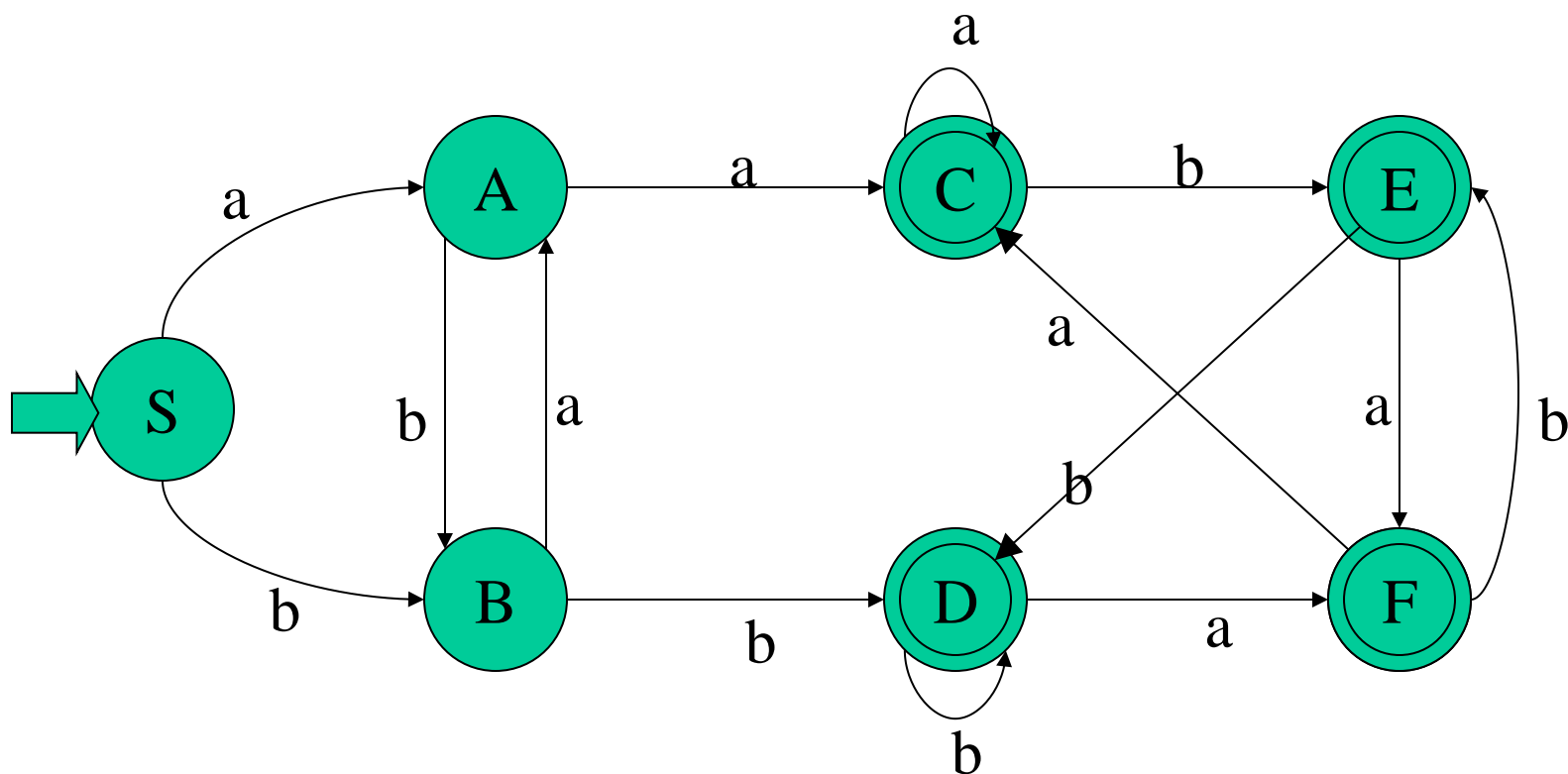
DFA还是NFA？





T		$\epsilon$ -clo (move(T,a))		$\epsilon$ -clo (move(T,b))
{i,1,2}	<b>S</b>	{1,2,3}	<b>A</b>	{1,2,4} <b>B</b>
{1,2,3}	<b>A</b>	{1,2,3,5,6,f}	<b>C</b>	{1,2,4} <b>B</b>
{1,2,4}	<b>B</b>	{1,2,3}	<b>A</b>	{1,2,4,5,6,f} <b>D</b>
{1,2,3,5,6,f}	<b>C</b>	{1,2,3,5,6,f}	<b>C</b>	{1,2,4,6,f} <b>E</b>
{1,2,4,5,6,f}	<b>D</b>	{1,2,3,6,f}	<b>F</b>	{1,2,4,5,6,f} <b>D</b>
{1,2,4,6,f}	<b>E</b>	{1,2,3,6,f}	<b>F</b>	{1,2,4,5,6,f} <b>D</b>
{1,2,3,6,f}	<b>F</b>	{1,2,3,5,6,f}	<b>C</b>	{1,2,4,6,f} <b>E</b>

# 等价的DFA



# DFA的最小化

- DFA的化简

- 设有DFA  $M$ , 寻找一个状态数更少的DFA  $M'$ , 使  $L(M') = L(M)$
- 可以证明, 存在一个最少状态的DFA  $M'$ , 使  $L(M) = L(M')$ 。

- 等价条件

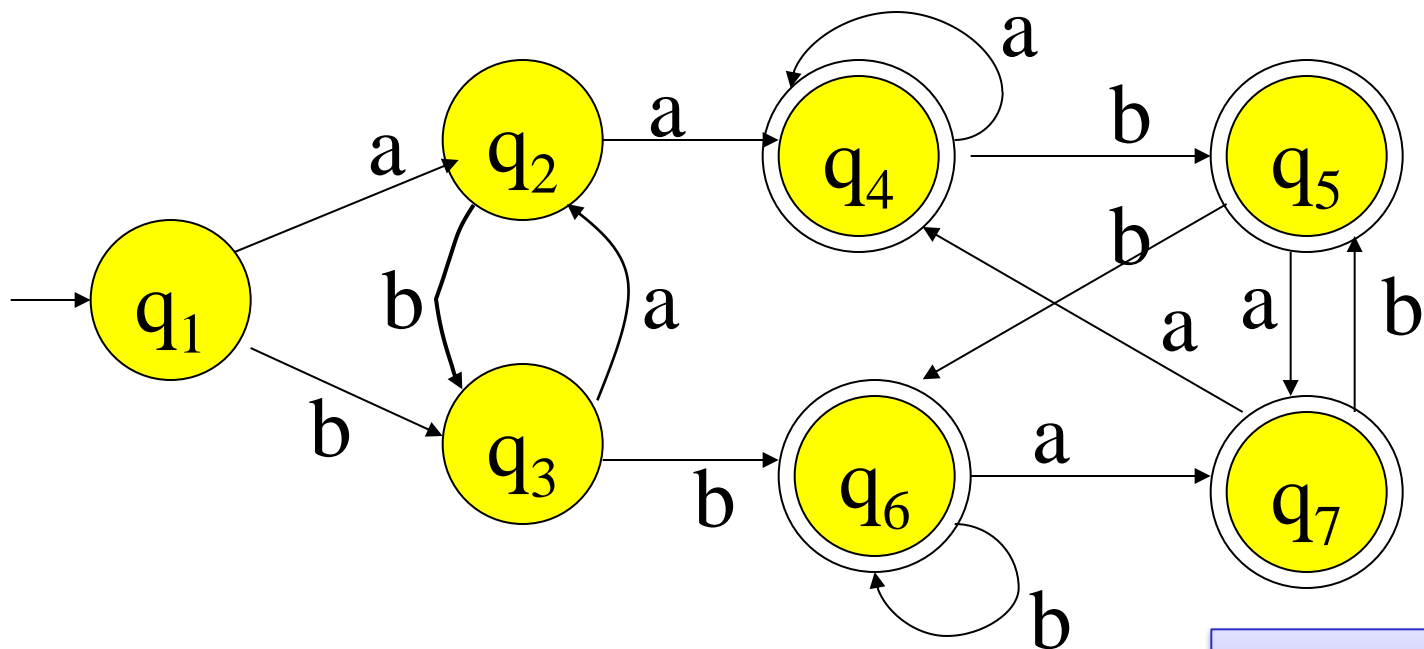
- 一致性条件: 状态 $s, t$ 必须同时为可接受状态或不可接受状态
- 蔓延性条件: 对任意输入符号, 状态 $s$ 和 $t$ 必须转换到等价的状态里, 否则是**可区分的**。

---

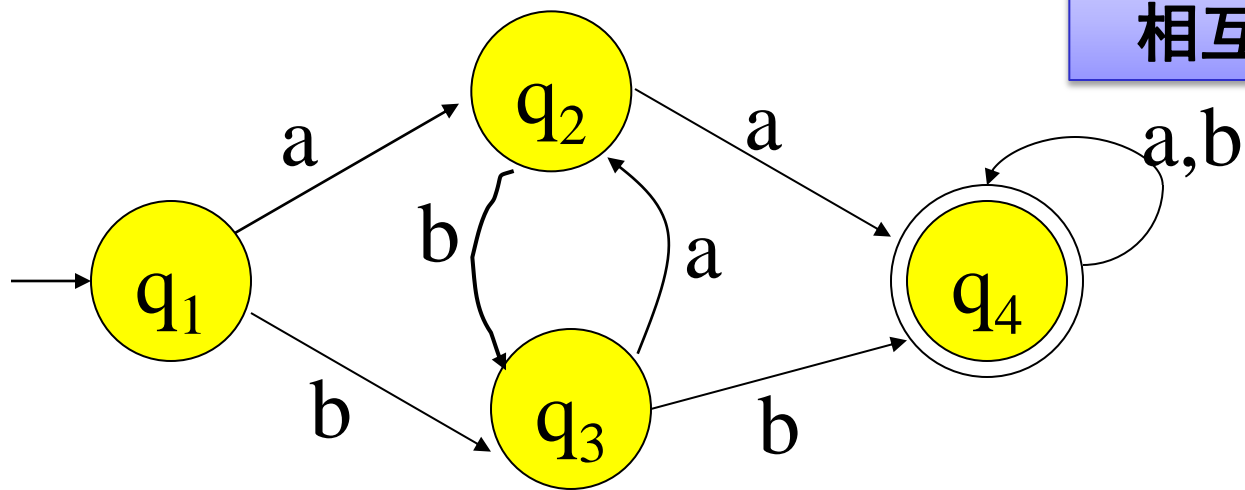
# 等价类划分方法

1. 把所有状态划分为两个组：接受状态组和非接受状态组。
  2. 任意选定一个输入符号 $a$ ，判断每个组中的各个状态对于 $a$ 的转换，如果落入不同的组中，就把该组中的状态按照转换之后的组进行分割，使分割之后的每个组对于 $a$ 的转换都会落入同一个组。
  3. 重复第2步，直至每个组中的所有状态都等价。
-

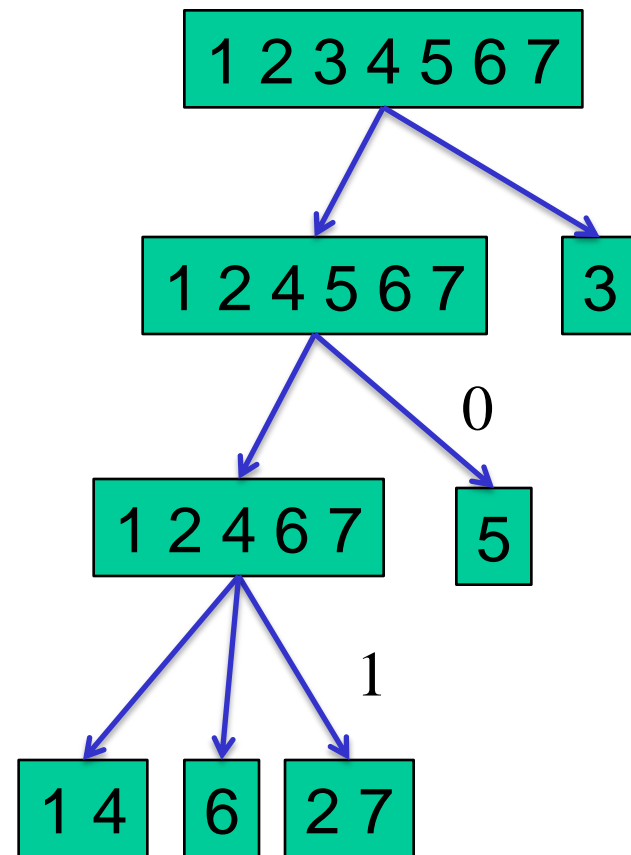
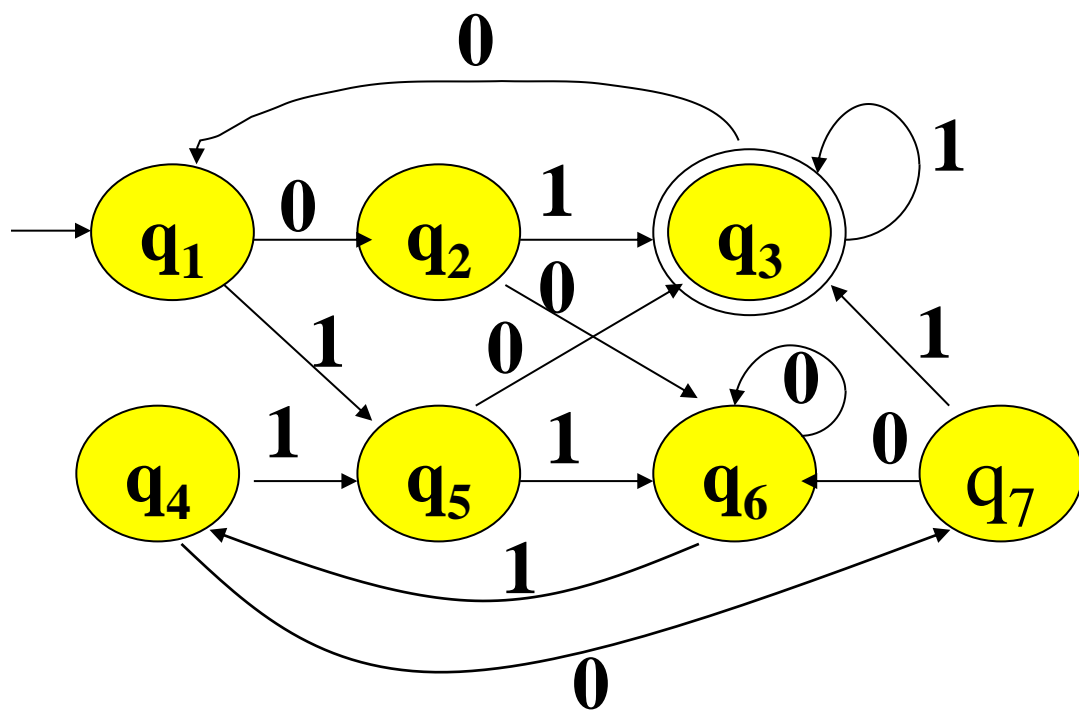
例



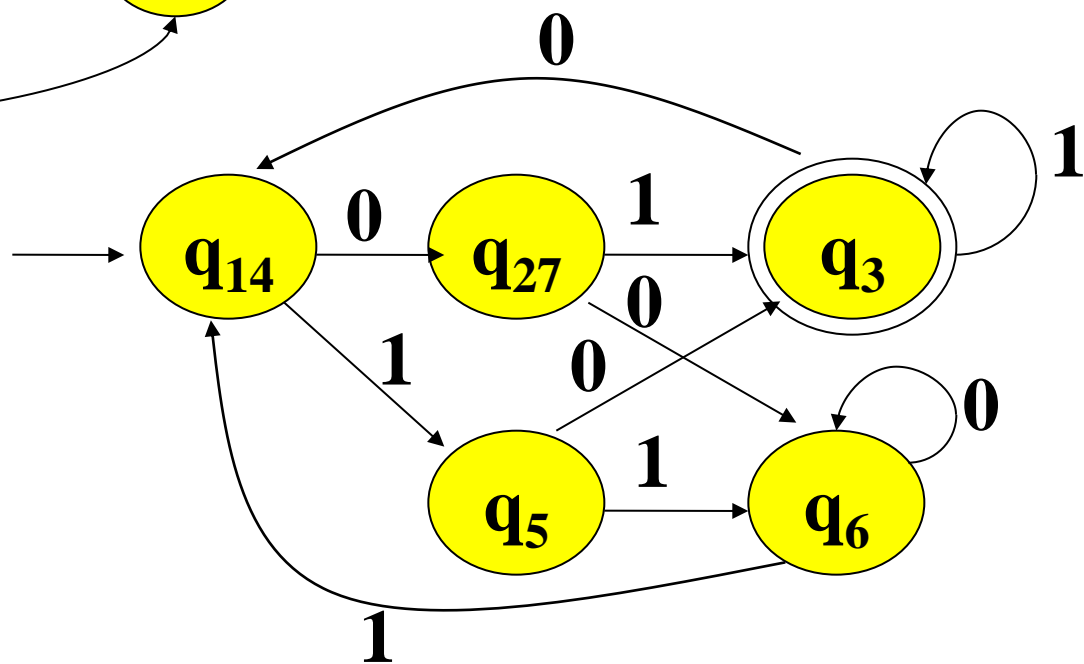
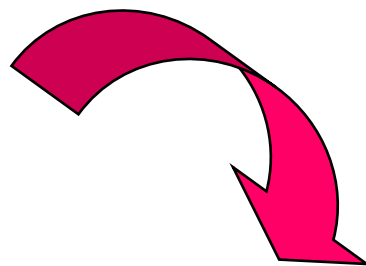
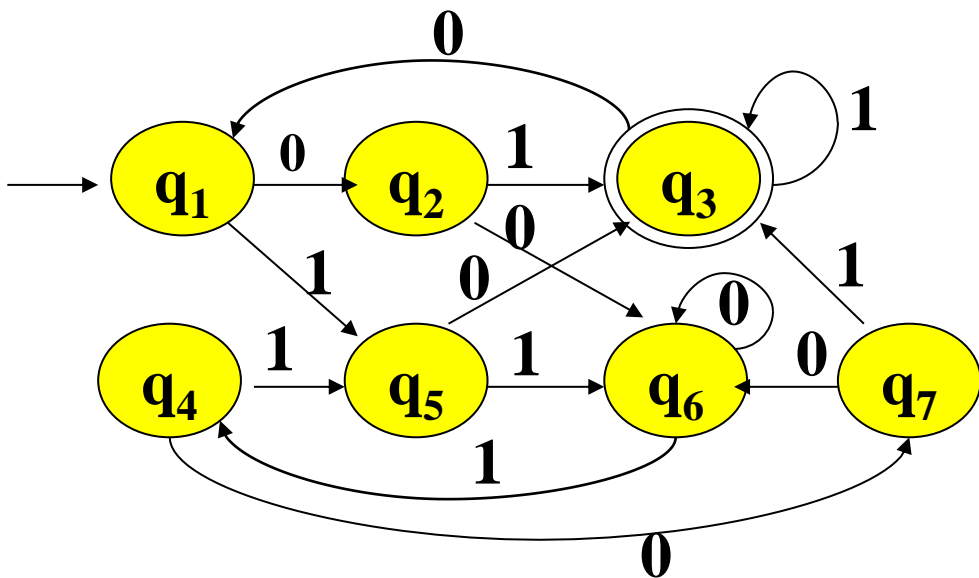
**q4 q5 q6 q7  
相互等价。**



# 例：DFA的最小化



# 化简之后



## 四、正规式和有穷自动机的等价性

正规式和有穷自动机的等价性由以下两点说明：

※对于 $\Sigma$ 上的**NFA M**，可以构造一个 $\Sigma$ 上的正规式**R**，使得 $L(R)=L(M)$ 。

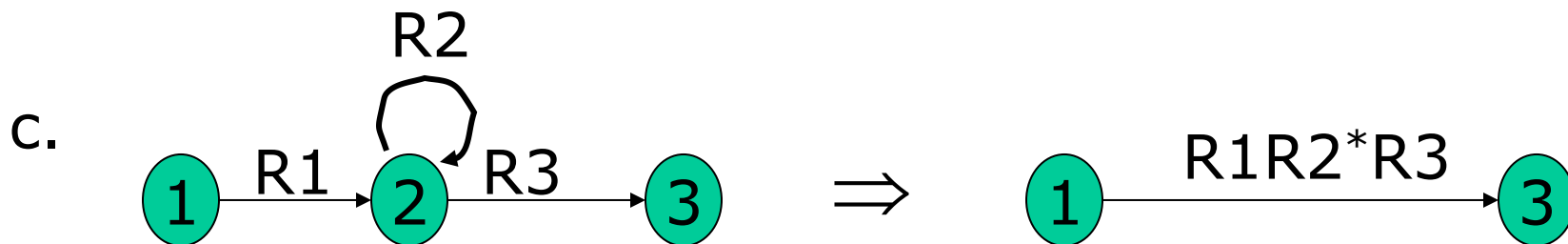
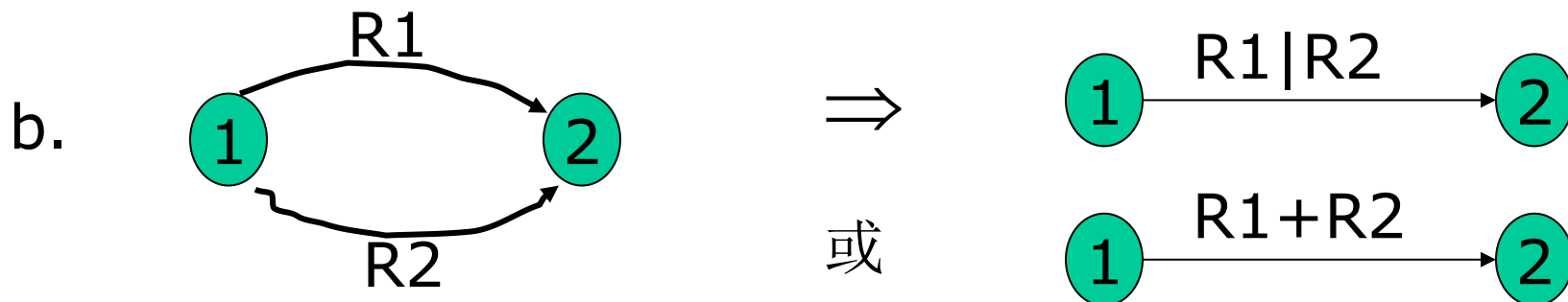
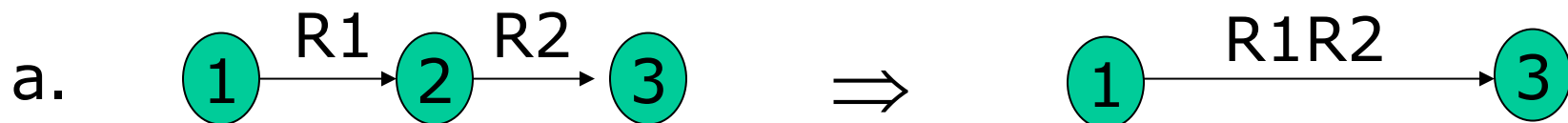
※对于 $\Sigma$ 上的每个正规式**R**，可以构造一个 $\Sigma$ 上的**NFA M**，使得 $L(M)=L(R)$ 。



**1、在NFA M上构造正规式R。即从 $L(M) \Rightarrow L(R)$**

方法：在每一条弧上用一个正规式作标记。

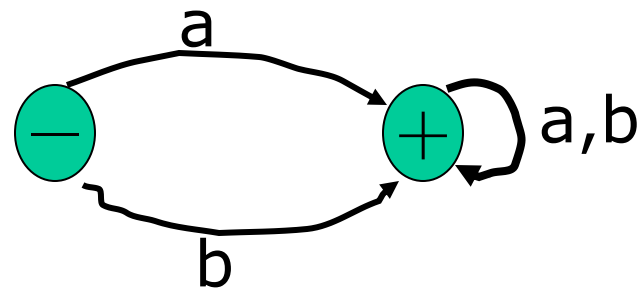
规则如下：



例1:  $L(M)$ 如下图:

求正规式 $R$ , 使 $L(R)=L(M)$ .

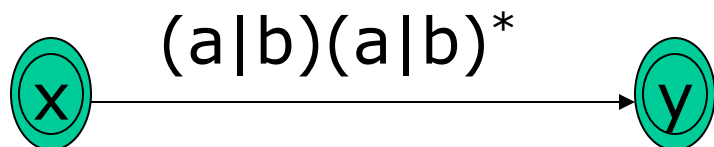
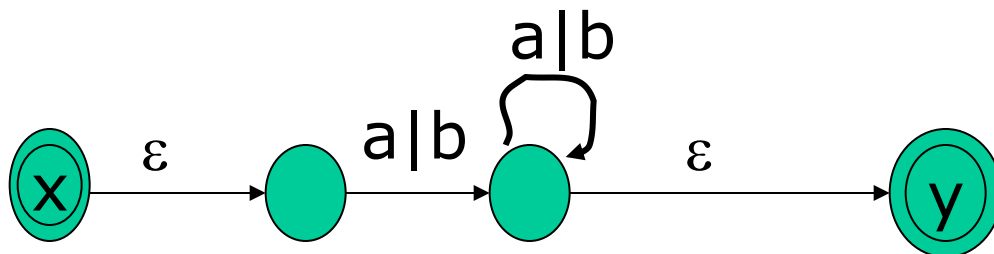
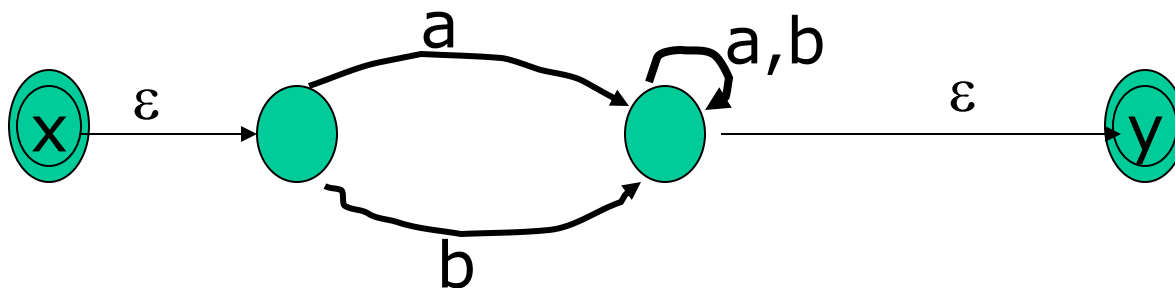
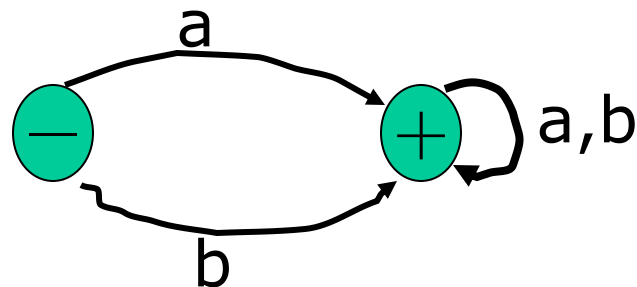
解:



例1:  $L(M)$ 如下图:

求正规式 $R$ , 使 $L(R)=L(M)$ .

解:

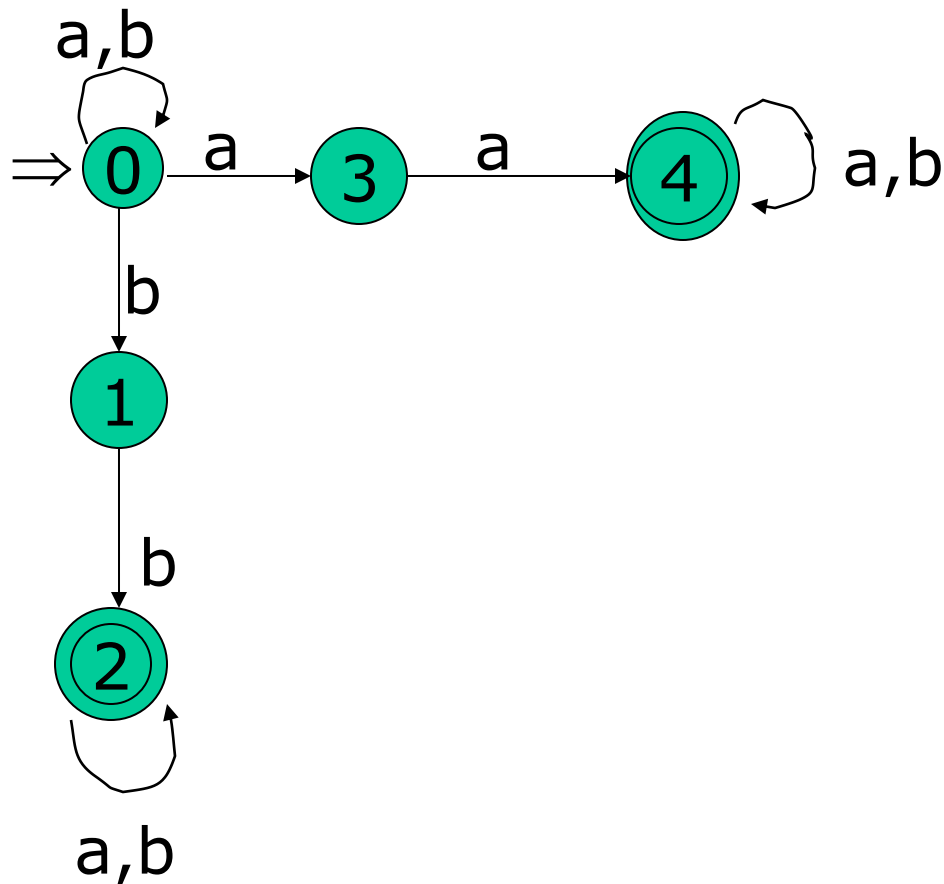


因此:

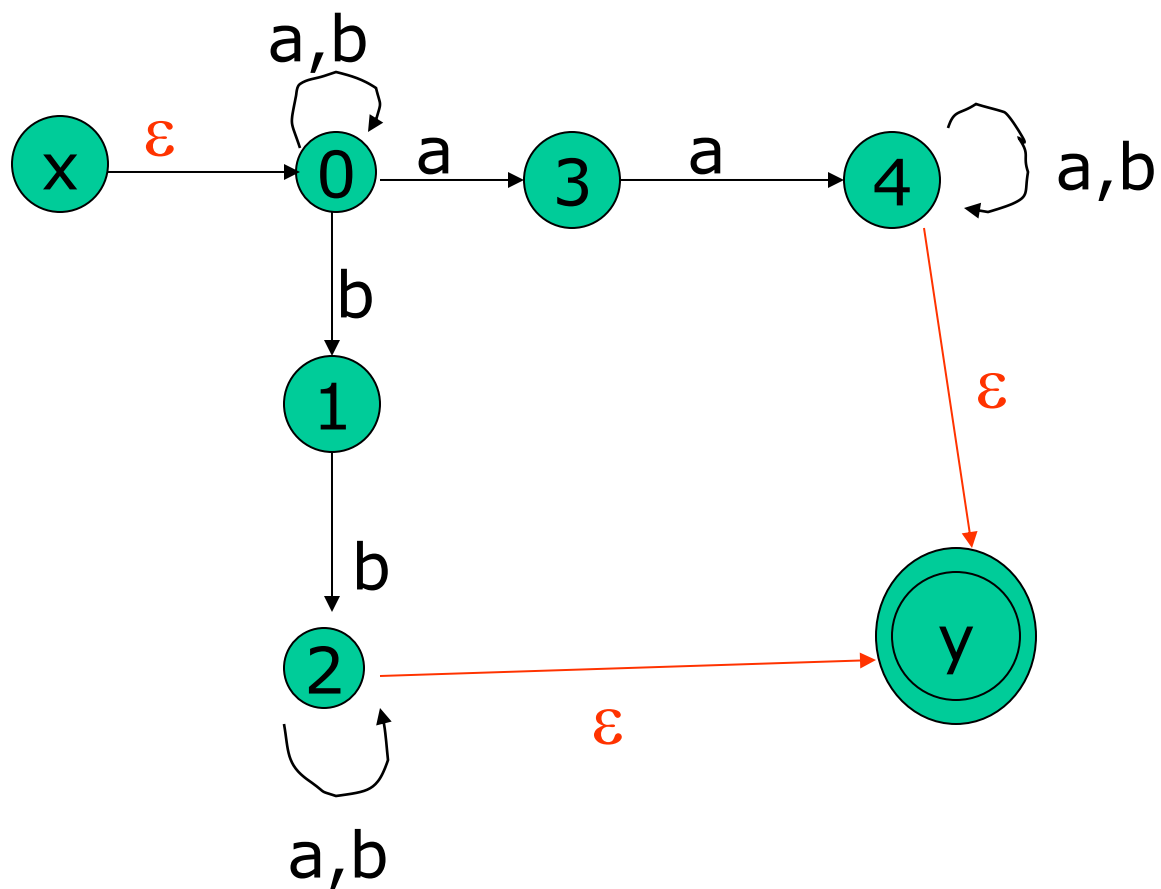
$$L(R) = (a+b)(a+b)^*$$

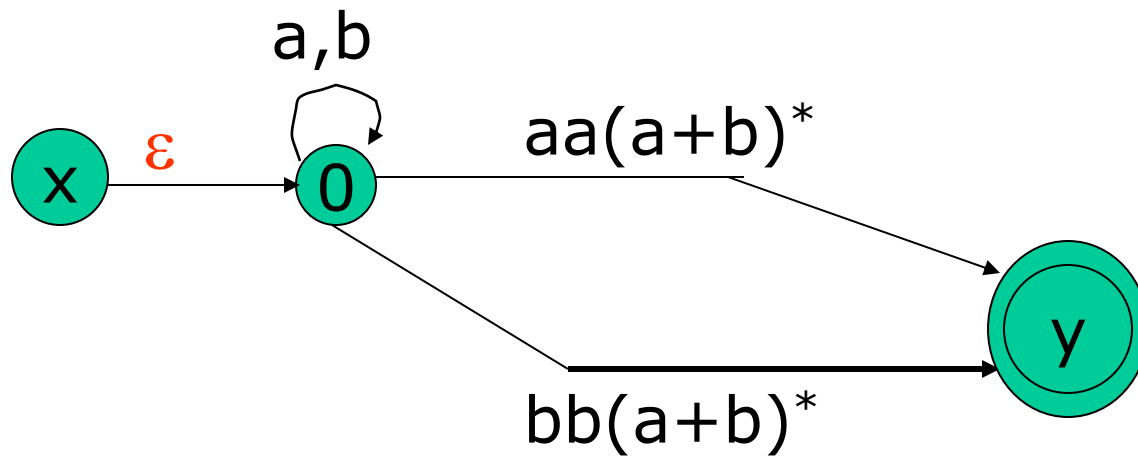
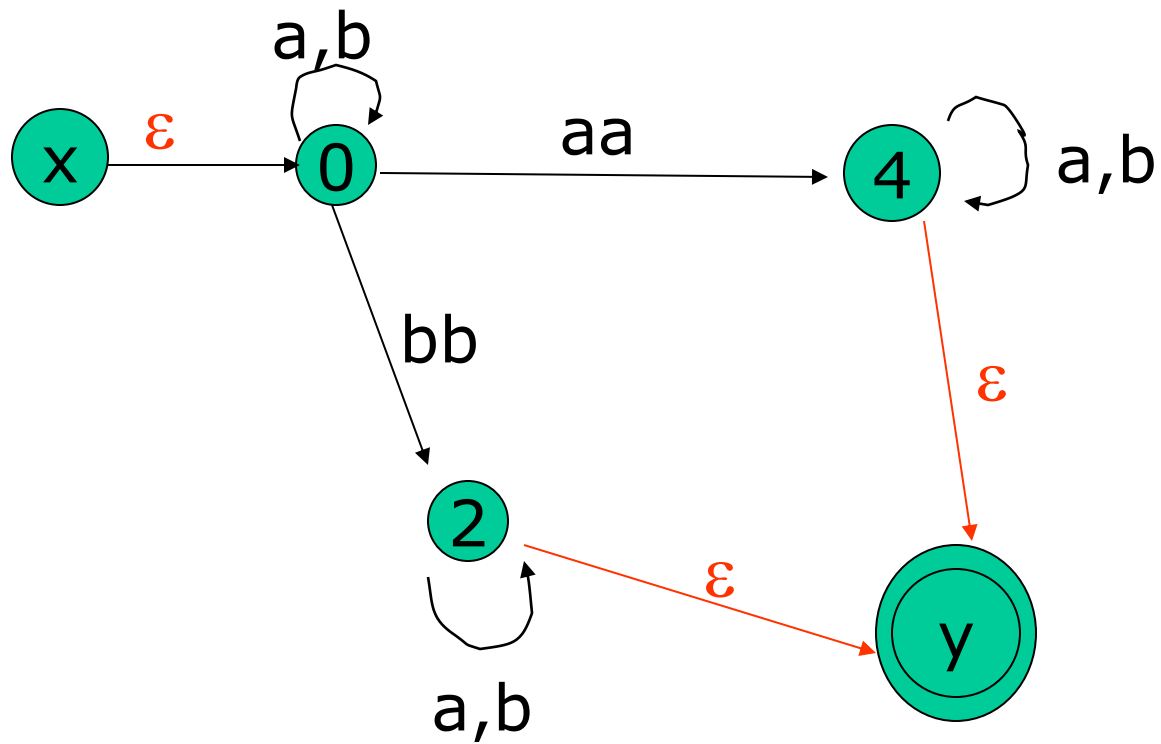
例2: M状态图如下:

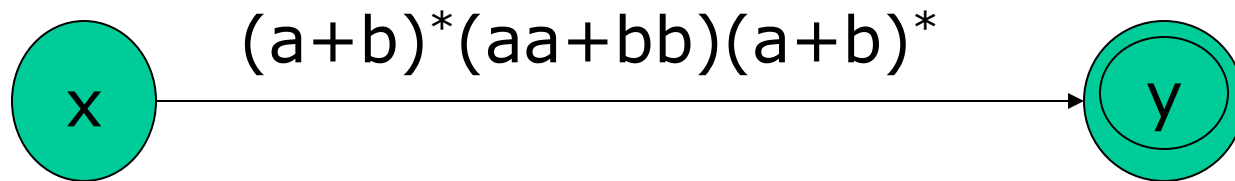
求正规式R, 使 $L(R)=L(M)$ .



解：加 $x, y$ 结点。







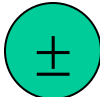
所以  $L(R) = (a+b)^*(aa+bb)(a+b)^*$

---

## 2、 $L(R) \Rightarrow \text{NFA}$ ，从正规式 $R$ 构造NFA

规则如下：

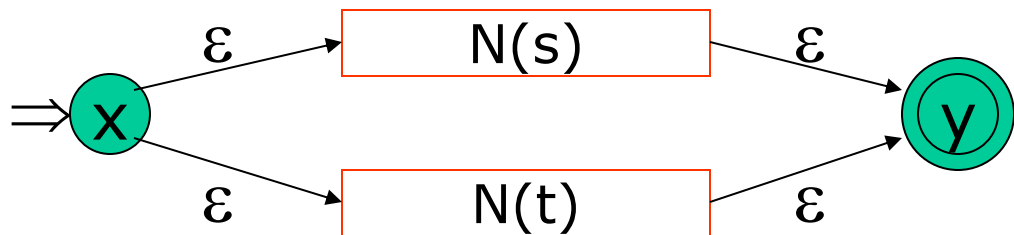
① 正规式 $\phi$ ，构造NFA为： $\Rightarrow$  

或：

② 对应正规式 $\varepsilon$ ，构造NFA为： $\Rightarrow$  

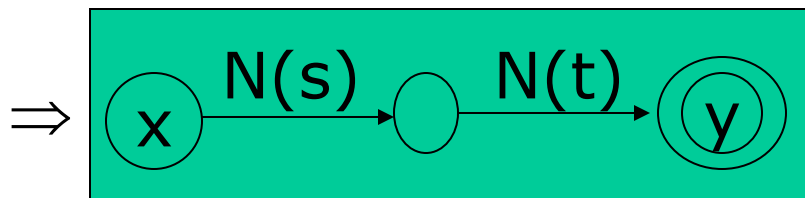
③ 对应正规式 $a$ ，构造NFA为： $\Rightarrow$  

④  $s, t$ 是正规式，相应NFA为 $N(s), N(t)$ ，  
则正规式 $R = s|t$ ，构造 $\text{NFA}(R)$  为：

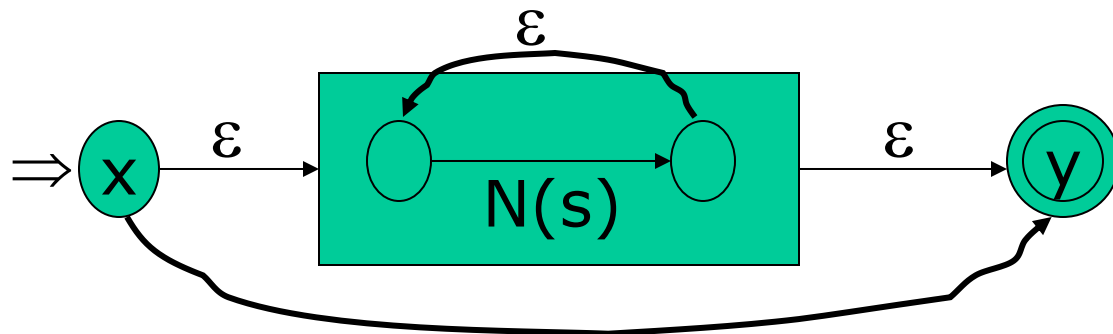




⑤  $s, t$  是正规式，相应NFA为  $N(s), N(t)$ ，则正规式  $R=st$ ，构造  $NFA(R)$  为：



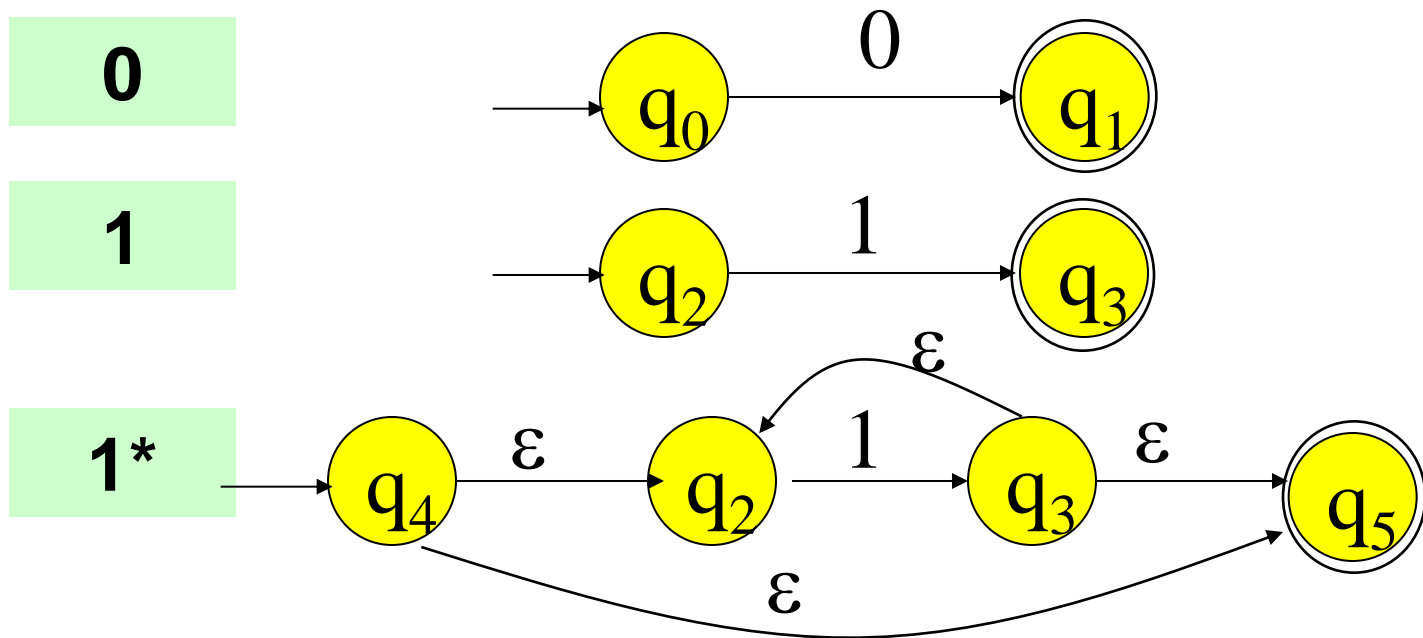
⑥  $s$  是正规式，相应NFA为  $N(s)$ ，则正规式  $R=s^*$ ，构造  $NFA(R)$  为：



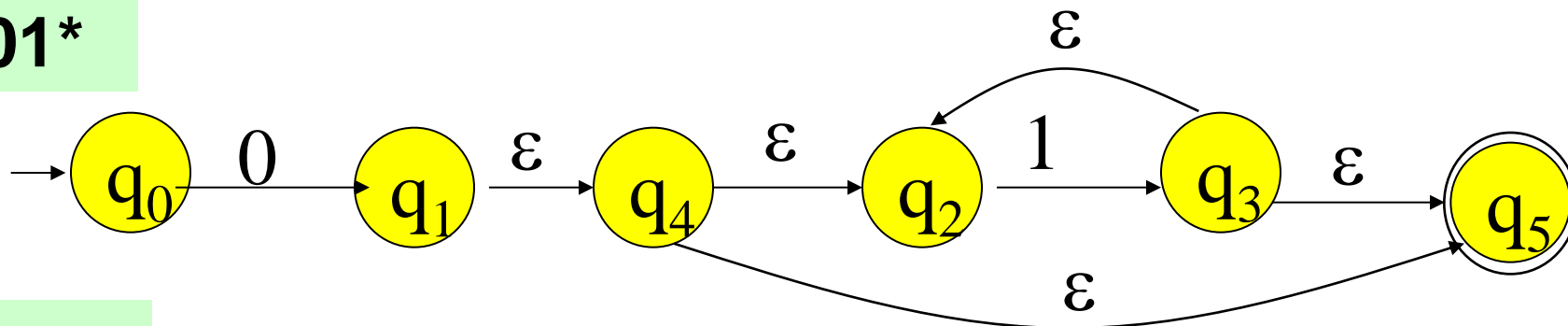
例：构造与下列正则式

$$r = 01^* | 1$$

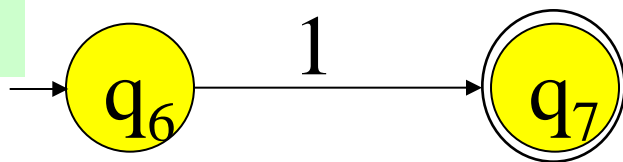
等价的有限自动机。



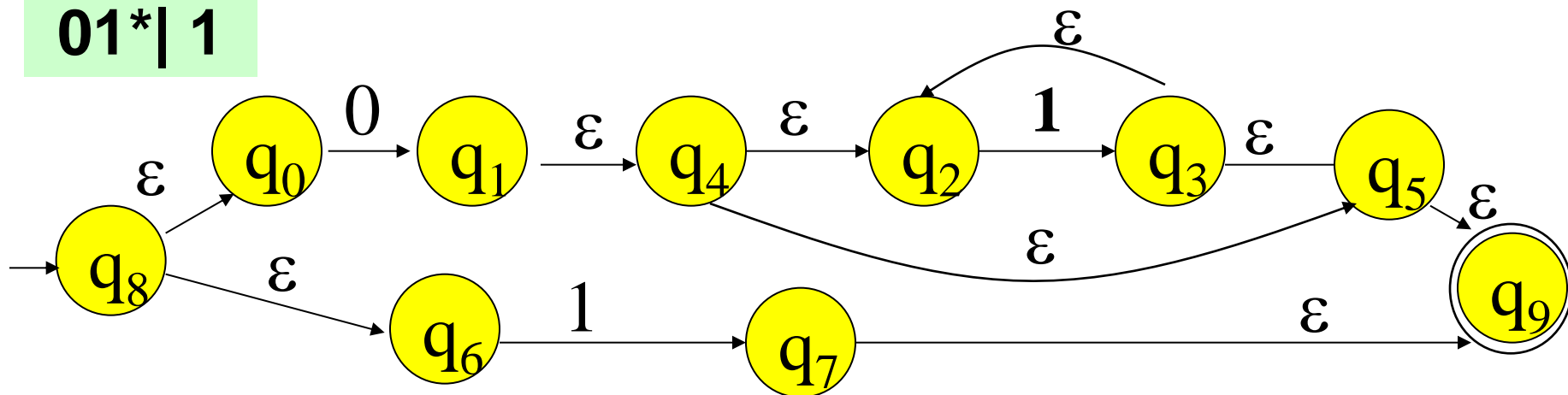
**$01^*$**



**$1$**



**$01^* | 1$**



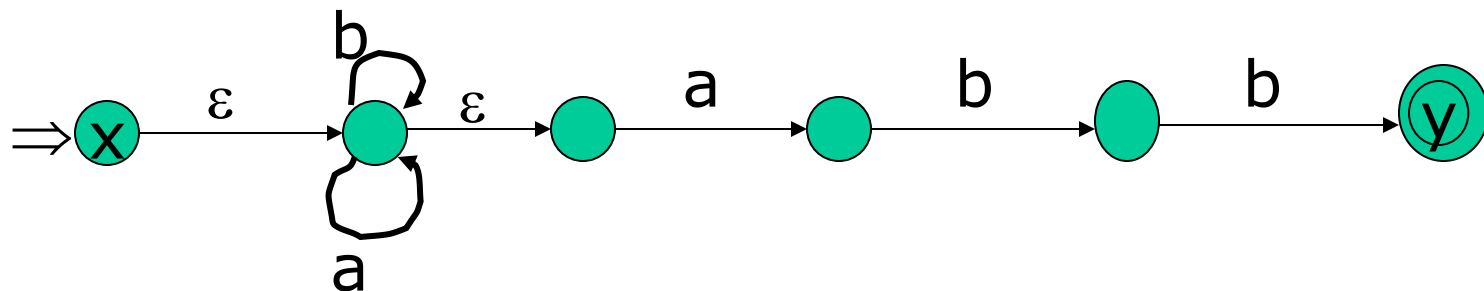
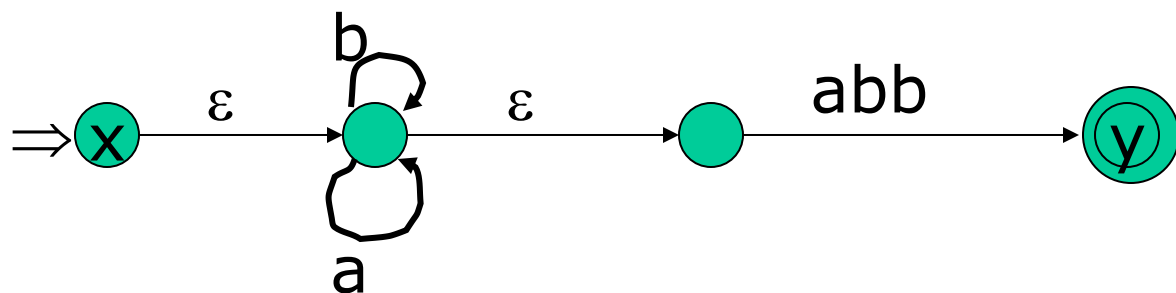
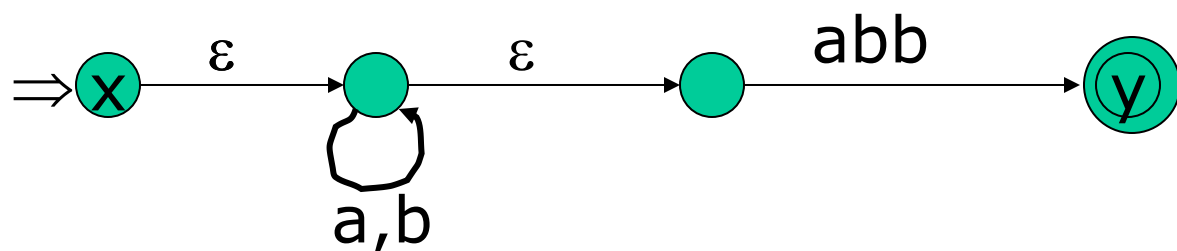
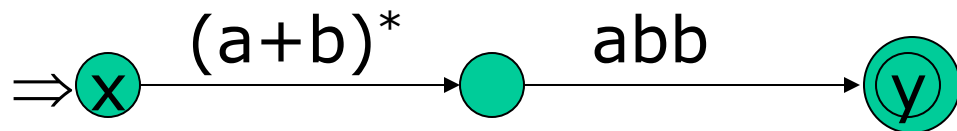
---

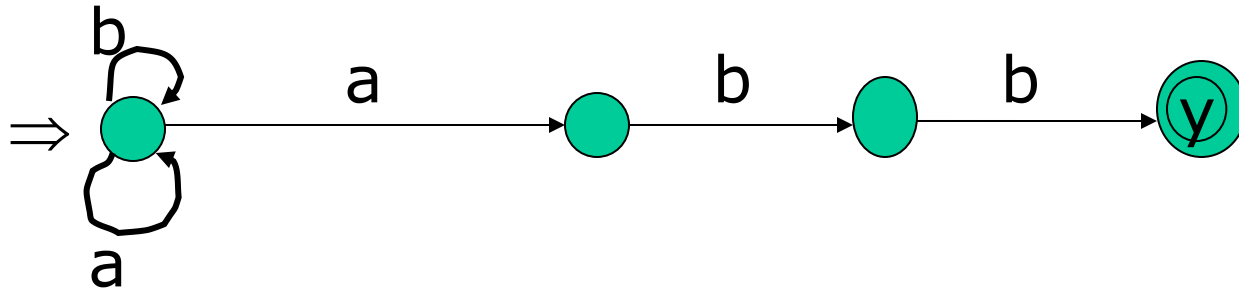
例1:  $L(R) = (a+b)^*abb$ , 构造NFA使 $L(N)=L(R)$

解:

例1:  $L(R) = (a+b)^*abb$ , 构造NFA使 $L(N)=L(R)$

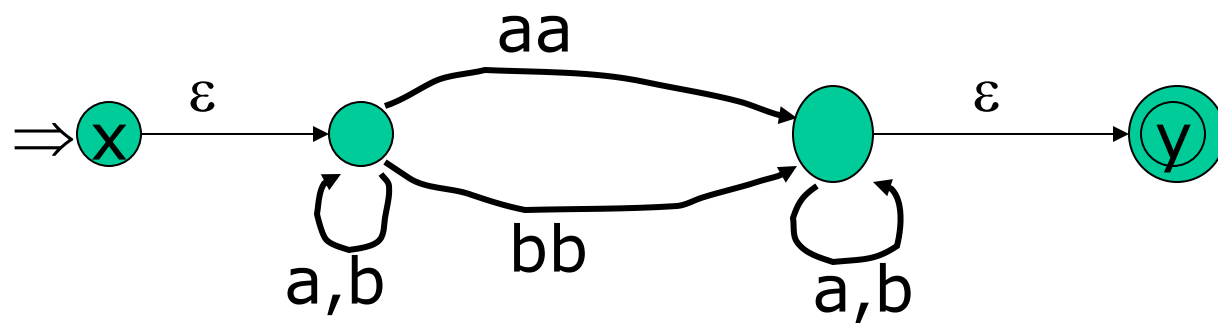
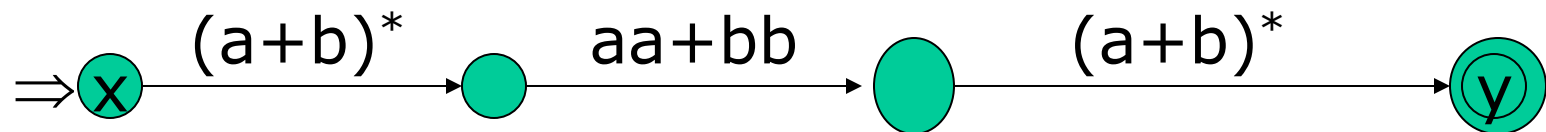
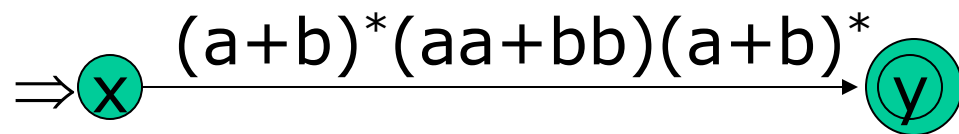
解:

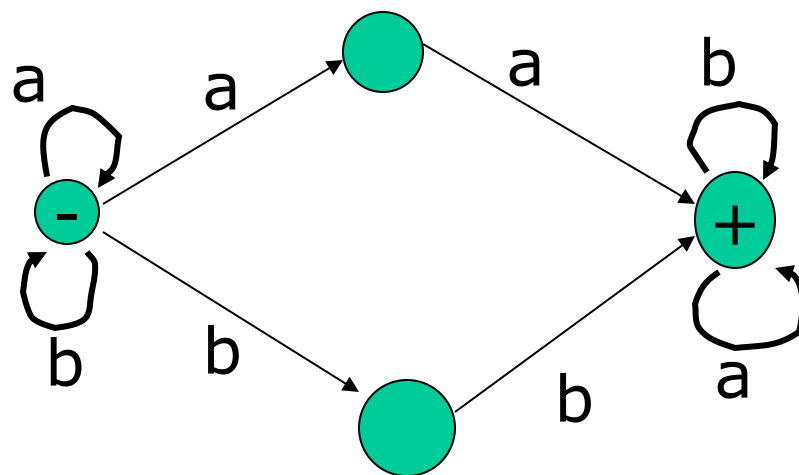
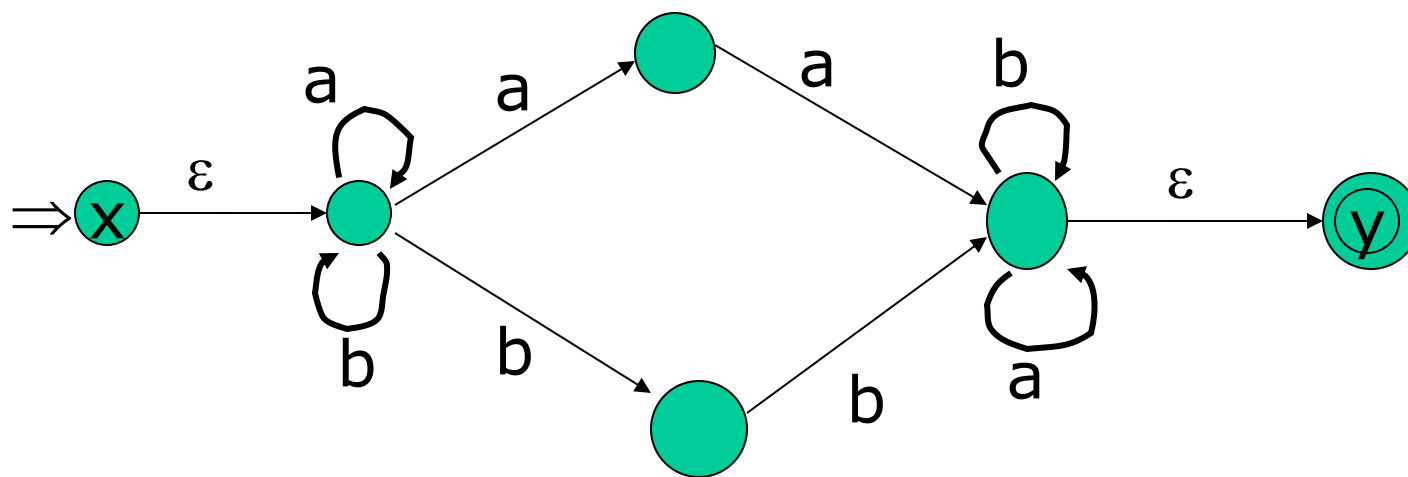




例4:  $L(R) = (a+b)^*(aa+bb)(a+b)^*$

构造 $L(N)$ 使与 $L(R)$  等价。







---

# 词法分析程序的自动构造

自动识别单词的方法：

- (1) 把单词的结构用**正规式**描述；
- (2) 把正规式转换为一个NFA；
- (3) 把NFA转换为相应的DFA；
- (4) 基于**DFA**构造词法分析程序。

上机内容：词法分析程序的自动构造工具  
LEX介绍及使用练习

---

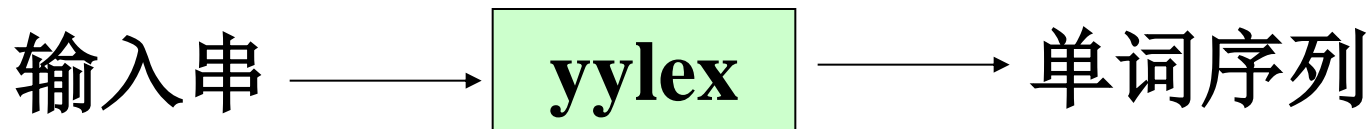
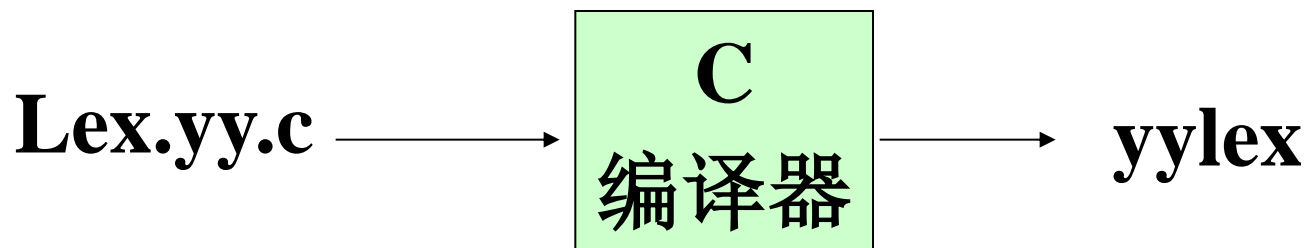
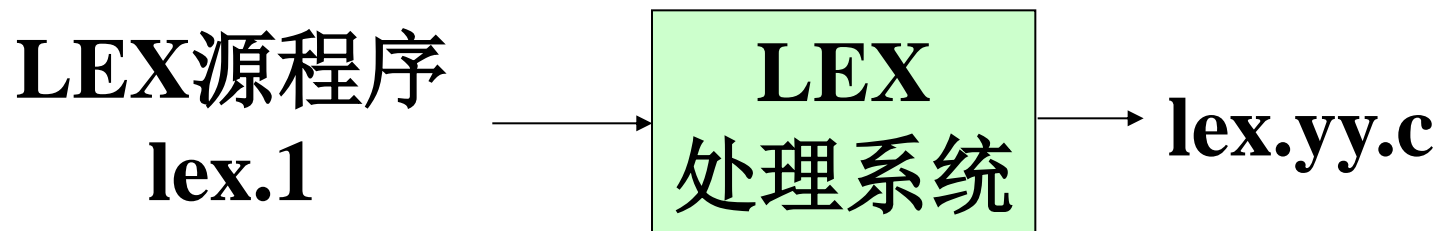
---

# Lex 简介

- Lex 是一种词法分析程序的自动构造工具。
    - 通常和Yacc一起使用，生成编译器的前端
  - 实现原理：
    - 根据给定的正则表达式自动生成相应的词法分析程序。
    - 利用正则表达式与DFA的等价性
  - 转换方式：

正则表达式  $\Rightarrow$  NFA  $\Rightarrow$  DFA  $\Rightarrow$  min DFA
-

# 用Lex建立词法分析程序的过程



---

# Lex源程序

- 一个LEX源程序主要由三个部分组成：
    - 声明
    - 转换规则及动作
    - 辅助子程序
  - 各部分之间用%%隔开
  - 声明包括变量，C语言常量和正则定义式。
  - 词法分析器返回给语法分析器一个单词，把单词的属性值存放于全局变量yylval中。
-

# Lex转换规则及动作

- 转换规则及动作的形式为：

$p_1$	{动作1}
$p_2$	{动作2}
...	...
$p_n$	{动作n}

- 每个  $p_i$  是正则定义式的名字，每个动作i 是正则定义式  $p_i$  识别某类单词时，词法分析器应执行动作的程序段。
  - 动作用C语言书写。
- 辅助子程序是执行动作所必需的。
  - 这些子程序用C语言书写，可以分别进行编译。

---

# 词法分析器的工作方式

- Lex生成的词法分析器作为一个函数被调用
  - 在每次调用过程中，不断读入余下的输入符号
  - 发现**最长**的、与某个模式匹配的输入前缀时，调用相应的动作；
    - 该动作进行相关处理，并把控制返回；
    - 如果不返回，则词法分析器继续寻找其它词素
-

# Lex程序示例

```
%{  
    /* definitions of manifest constants  
    LT , LE , EQ , NE , GT , GE ,  
    IF , THEN , ELSE , ID , NUMBER , RELOP */  
}%  
  
/* 正则定义*/  
  
delim    [ \t\n]  
ws        {delim}+  
Letter    [A-Za-z]  
digit     [0-9]  
id         {letter} ({letter} | {digit}) *  
Number    {digit}+ ( \ . {digit } + )?(E [+ -] ?{digit}+ ) ?  
  
%%
```

# Lex程序示例（续）

```
{ws}          { /* no action and no return */ }
If      {return(IF) ;}
then    {return (THEN) ;}
else     {return (ELSE) ;}
{id}     {yylval = (int ) installID () ; return(ID) ;}
{number} {yylval = (int) installNum() ; 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) ;}
```

```
%%
```

```
int installID () /*向符号表添加指向yytext,长度为yyleng的词法单元*/
int installNum () /*把数字常量添加到另外一个单独的表格中*/
```



# 例题分析

- 单词的描述工具

- 正规文法（判断一个文法是否正规文法）
- 正规式（给定字母表 $\Sigma$ ，写出 $\Sigma$ 上合法的正规式）、正规集（给定正规式，列出它对应的正规集；给定正规集，列出它对应的正规式）
- 正规文法与正规式的相互转换（二者的相互转换）

- 单词的识别机制

- 确定有穷自动机（给定DFA，画出它的状态图和矩阵表示；给定DFA，判断它所识别的语言；给定一种语言，构造它对应的DFA；给定DFA，写出DFA识别语言的伪代码）
- 不确定有穷自动机（给定NFA，构造等价的DFA，写出构造DFA状态的伪代码）

---

# 例题分析

1. 构造正规式1 (0|1) \*101相应的DFA
  2. 1(2),(3)
  3. 叙述正规式  
 $(00|11)^*((01|10)(00|11)^*(01|10)(00|11)^*)^*$   
描述的语言
  4. 构造一个NFA，它接受 $\Sigma=\{0,1\}$ 上连续两个0和连续两个1组成的字符串。
-

---

# 课后作业

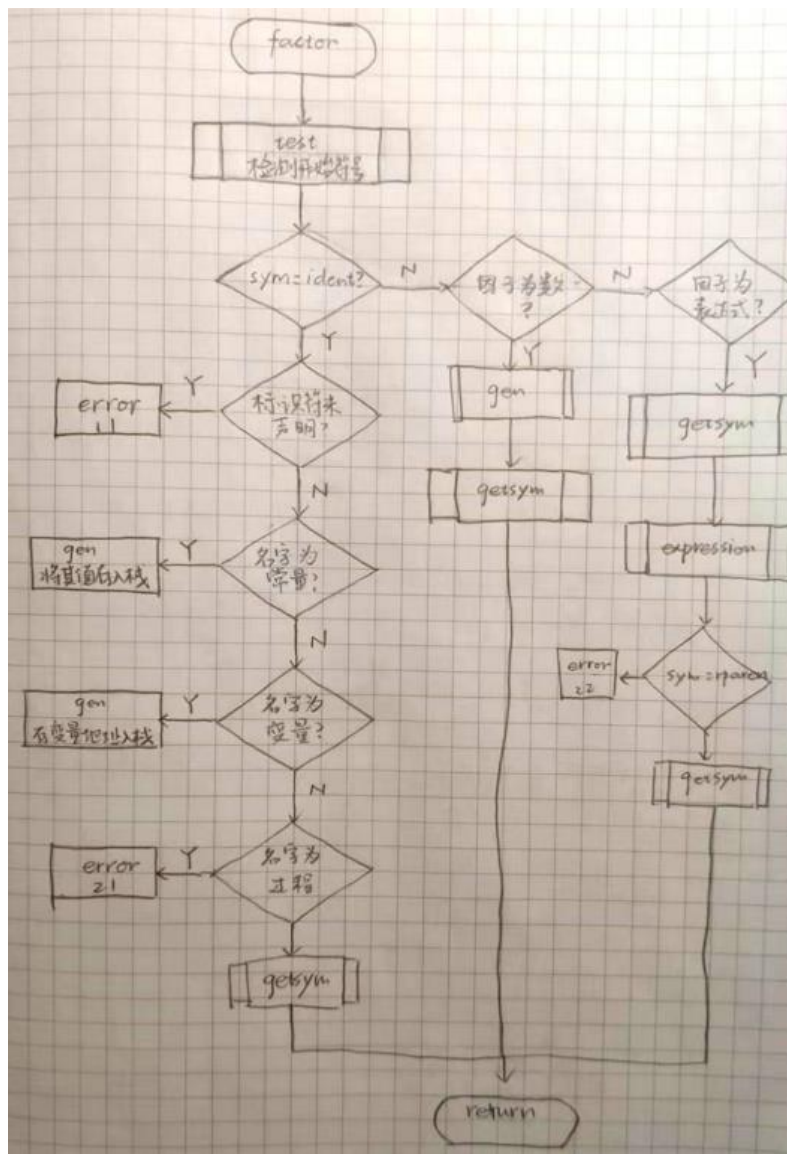
- **P72 : 1 (1) , 2, 5**
- **P73: 8, 9**

---

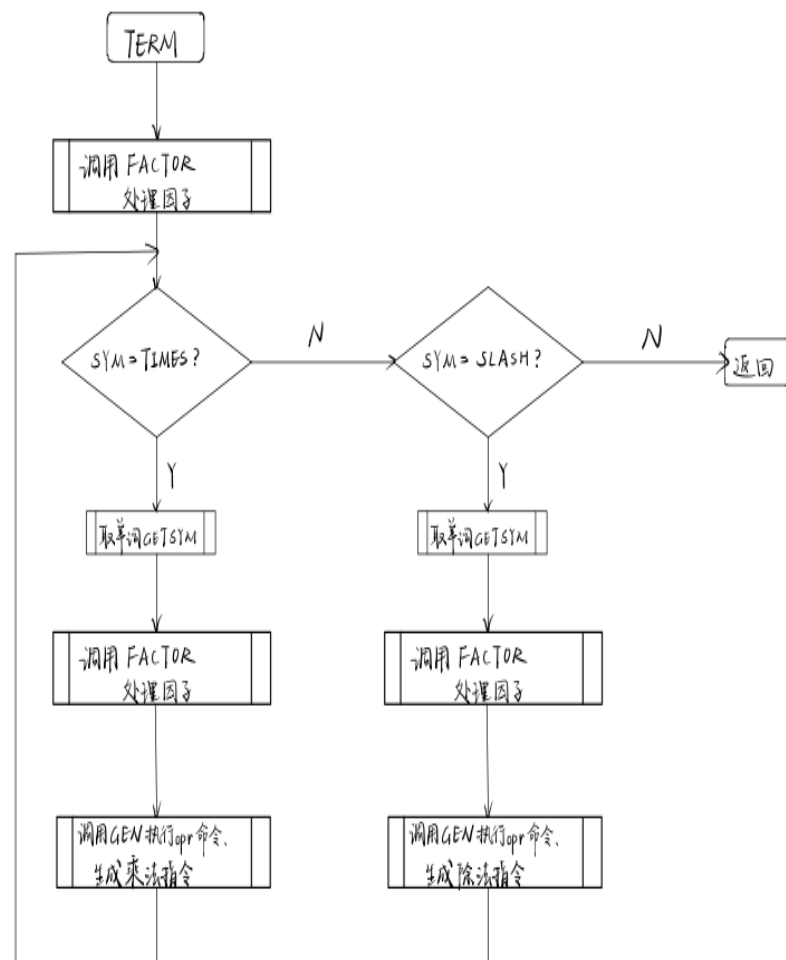
# 习题讲解

- **P48**
- **4**
- **14 (1) (2)**

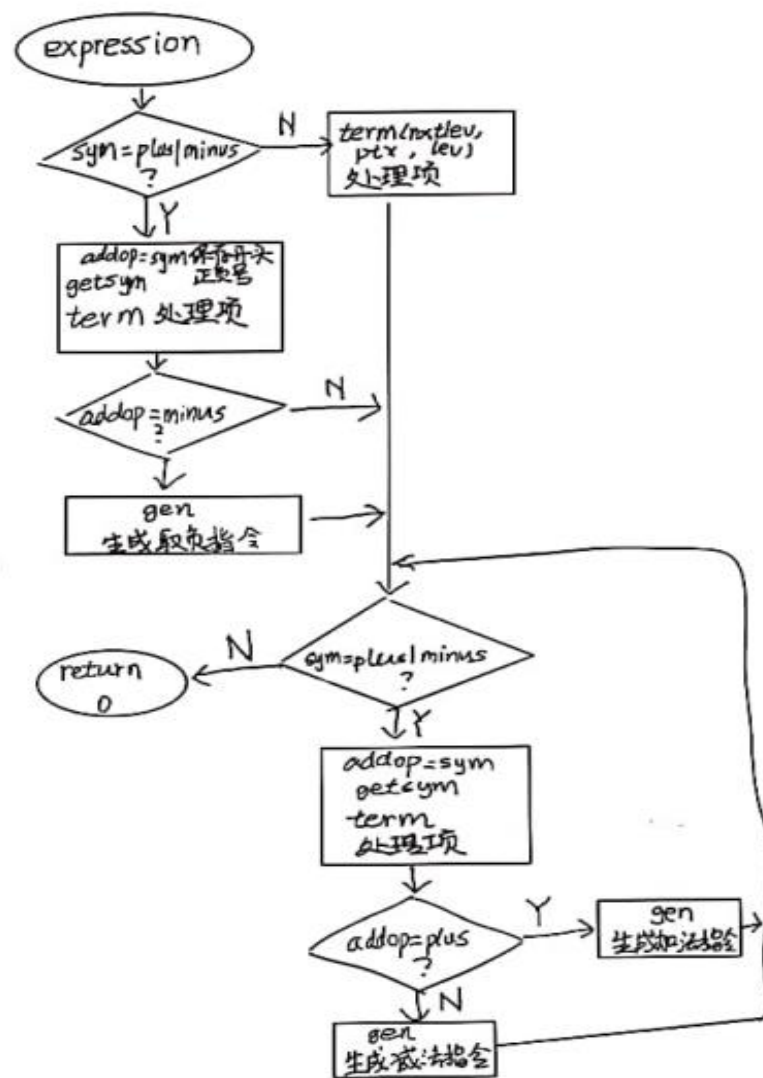
# 语句处理、表达式、项、因子



项处理



# 语句处理、表达式、项、因子



# 语句处理、表达式、项、因子

