



# 第3章 词法分析 (3)

---

## Lexical Analysis

【对应教材 3.6, 3.7】



# Review

---

- 构造一个正则表达式，表示所有能被16整除的二进制数。
- 构造一个NFA，表示所有能被16整除的二进制数。



# 内容提要

---

- 词法分析器的作用
- 词法单元的规约
  - 串和语言；正则表达式、正则定义
- 词法单元的识别
- 词法分析器生成工具—LEX
- 有限自动机 (Finite Automata)
- 正则表达式到有限自动机
- 词法分析器生成工具的设计



# 回顾

## □ 正则表达式

- 正则表达式可以简洁、精确地描述词法单元的模式
- 人可以比较轻松地写出正则表达式

## □ 有限自动机

- 模拟DFA的执行可以高效地进行模式匹配
- 状态较多时，DFA不适合手动书写

## □ 目标：把人写的正则表达式转换为机器可以自动匹配的DFA

# 从正则表达式到自动机的转换

## □ 将正则表达式转换为DFA的步骤





# NFA与DFA的等价性

**定理：**对任何一个NFA  $M$ ，都存在DFA  $M'$ 使

$$L(M') = L(M)$$

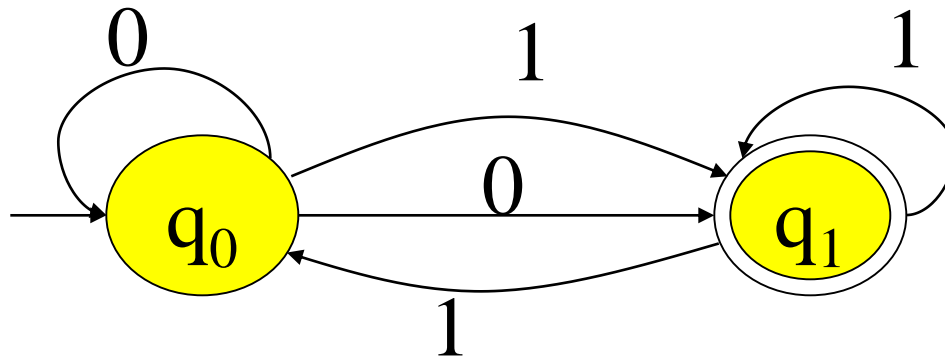
**证明思想：**用 $M'$ 的一个状态对应 $M$ 的一个状态集合，用这种方法，能从一个NFA  $M$ 构造一个DFA  $M'$ ，称作**子集构造法**。

**例3.2** NFA  $M = (\{0, 1\}, \{q_0, q_1\}, q_0, \delta)$ ，其中

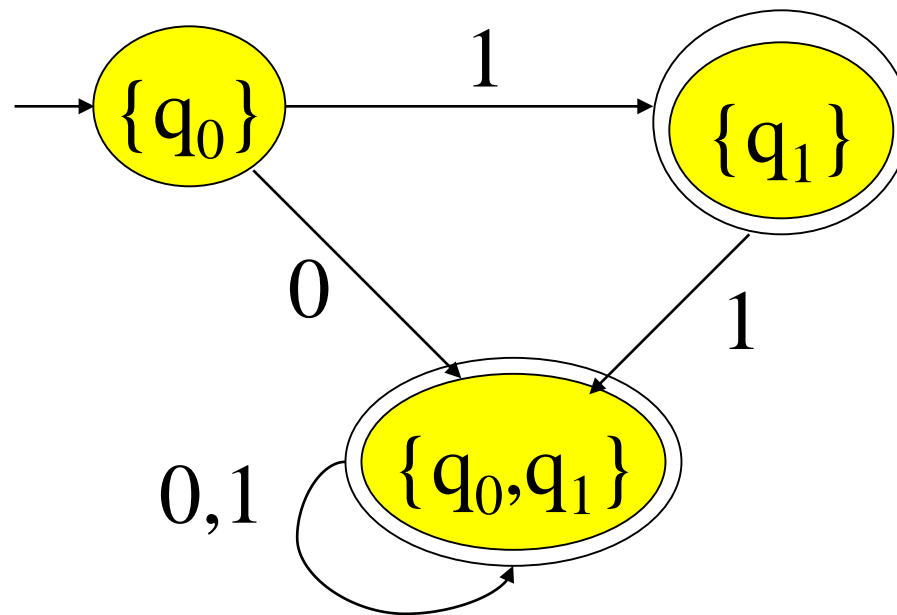
$$\delta(q_0, 0) = \{q_0, q_1\} \quad \delta(q_0, 1) = \{q_1\}$$

$$\delta(q_1, 0) = \emptyset \quad \delta(q_1, 1) = \{q_0, q_1\}$$

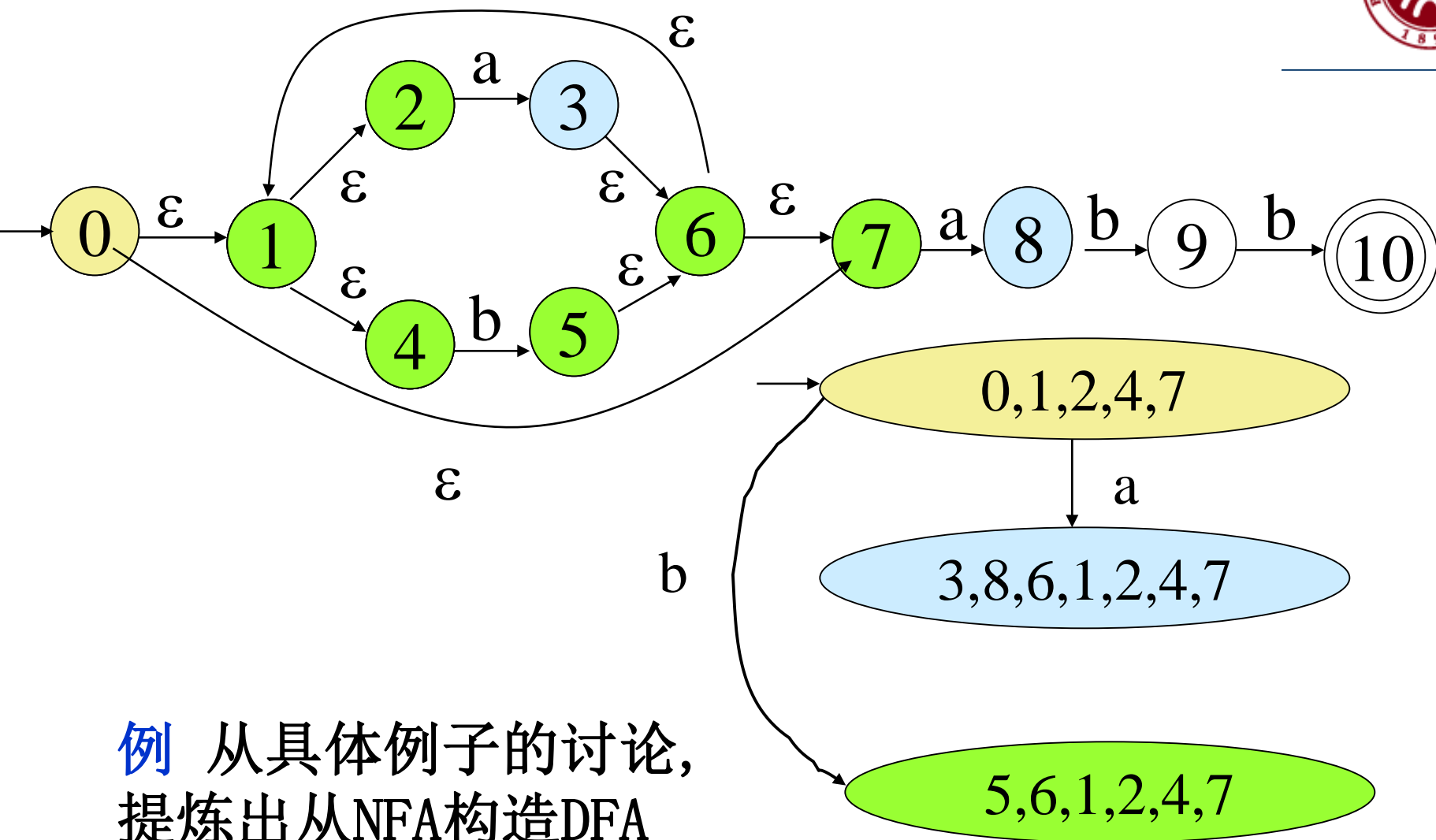
$$L(M) = L(M') = \{0,1\}^+ - 10\{0,1\}^*$$



NFA



DFA



**例** 从具体例子的讨论,  
提炼出从NFA构造DFA  
的算法。





# 从NFA M 构造DFA M' 的算法

## 1. $\epsilon$ \_closure(S) 的定义和算法

**定义：**从状态集合S中任一状态出发, 仅沿 $\epsilon$ 弧到达的状态集合（包括S自身）称为**S的 $\epsilon$ \_闭包**, 记为 **$\epsilon$ \_closure(S)**：

$$T = S \cup (\cup \text{edge}(t, \epsilon)), t \in T$$

其中:  $\text{edge}(t, a)$ 是M中从状态t出发, 仅沿a弧到达的状态集合。

计算 T （即 $\epsilon$ \_closure(S)）的方法：

T:=S;

REPEAT

T':=T;

T:=T'  $\cup$  ( $\cup \text{edge}(t, \epsilon)$ ) ( $t \in T'$ )

UNTIL T=T'



# 从NFA M构造DFA M'的算法

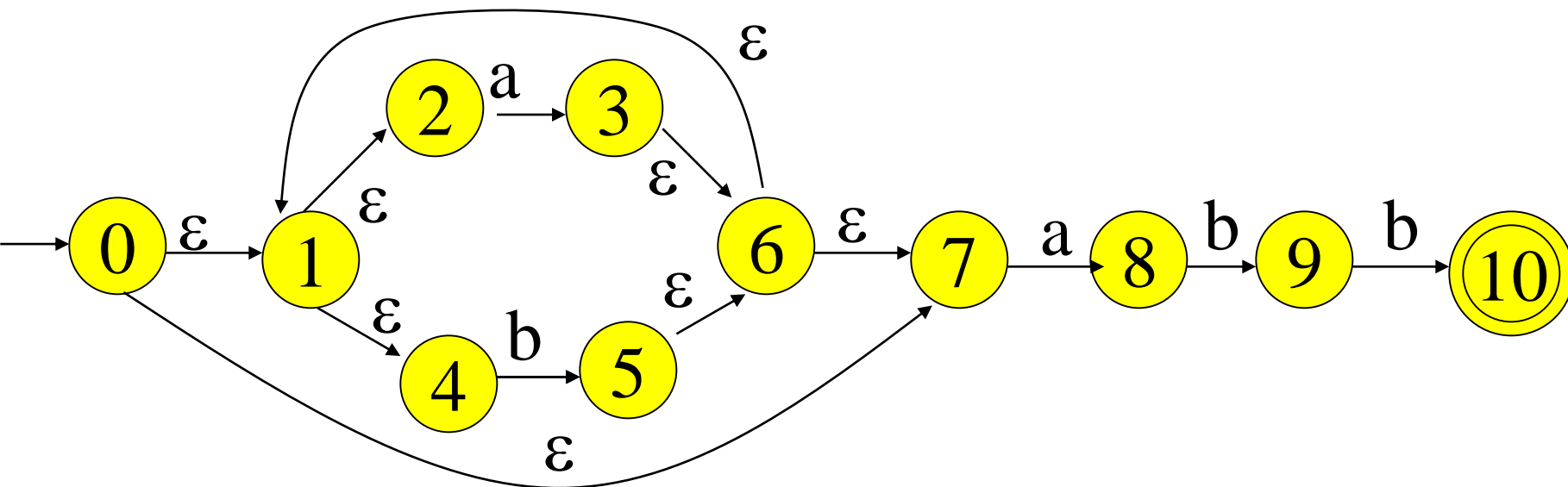
## 2. DFA M'中的状态

- M'中的每个状态是M的状态集合。
- 令 $t_0$ 是M的初始状态, M'的初始状态  $d_0 = \varepsilon\_closure(\{t_0\})$
- 包含M的任意终态的状态集合都是M'中的终止状态。

## 3. DFA M'的转移函数

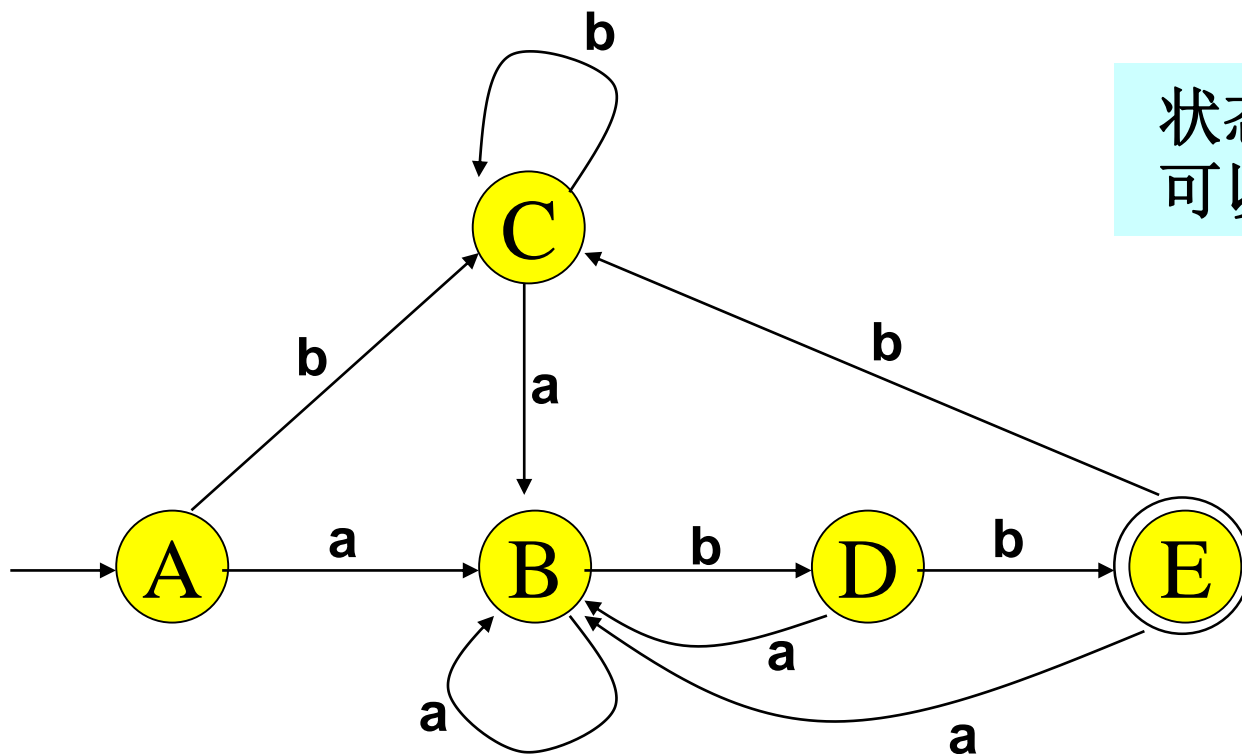
$$\text{DFAedge}(d, a) = \varepsilon\_closure(\bigcup_{t \in d} \text{edge}(t, a))$$

- $d$ 是M的状态集合
- $a \in \Sigma$
- $\text{edge}(t, a)$ 是M中从状态 $t$ 出发, 仅沿 $a$ 弧到达的状态集合。



states	a	b
(A) 0,1,2,4,7	(B)	(C)
(B) 3,8,6,1,2,4,7	(B)	(D)
(C) 5,6,1,2,4,7	(B)	(C)
(D) 5,9,6,1,2,4,7	(B)	(E)
(E) 5,10,6,1,2,4,7	(B)	(C)

# 转换之后得到的DFA



状态是不是  
可以化简？



# 练习

---

- 把在讲义第2页构造的NFA转换为等价的DFA。
  - NFA: 表示所有能被16整除的二进制数。



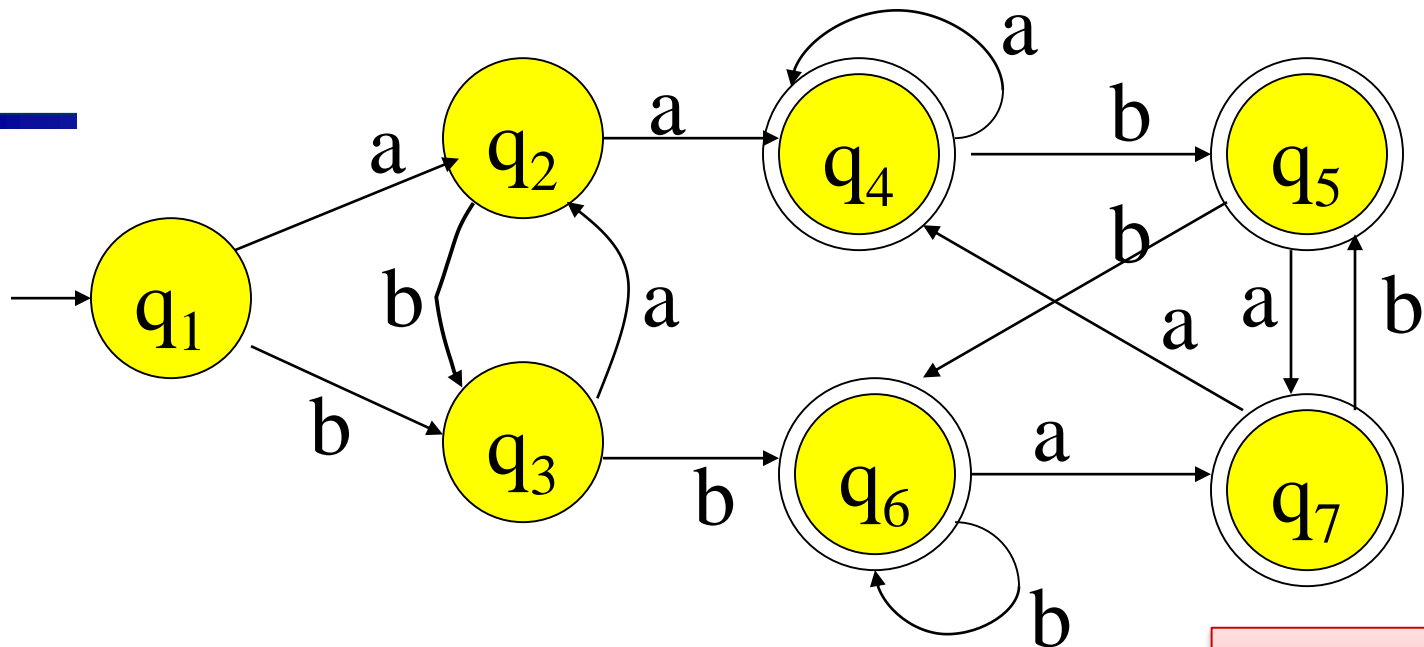
# DFA的最小化

## □ DFA的化简

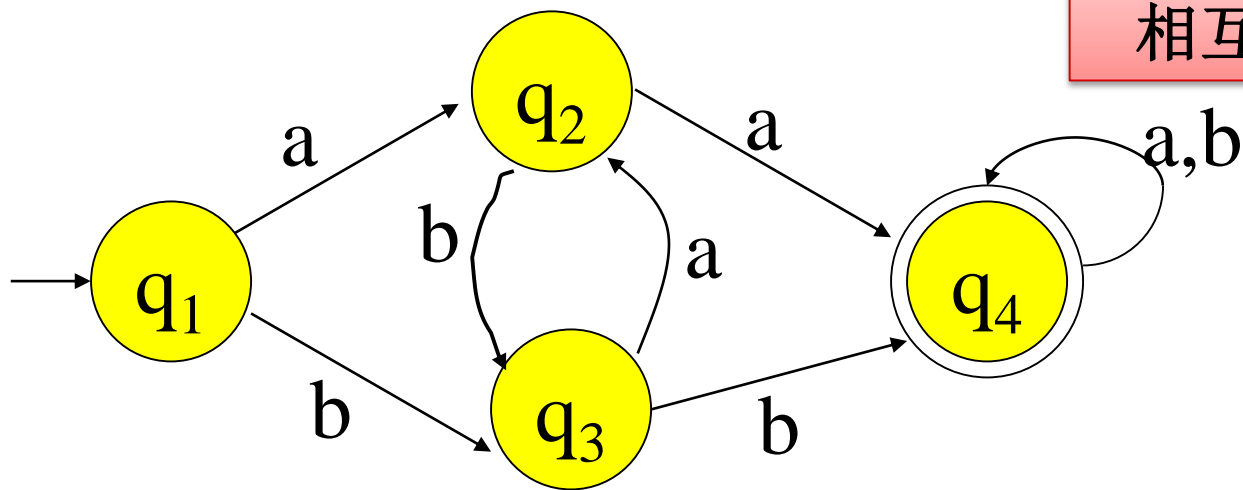
- 设有DFA  $M=(\Sigma, Q, q_0, F, \delta)$ , 寻找一个状态数更少的DFA  $M'$ , 使  $L(M') = L(M)$
- 可以证明, 存在一个最少状态的DFA  $M'$ , 使  $L(M)=L(M')$ 。

## □ 等价状态

- 设  $p, q \in Q$ , 若对任意  $w \in \Sigma^*$ ,  $\delta(p, w) \in F$  当且仅当  $\delta(q, w) \in F$ , 则称  $p$  和  $q$  是等价状态。
- 否则, 称  $p$  和  $q$  是可区别的。



**q4 q5 q6 q7  
相互等价。**





# 等价状态的判别条件

- 等价状态定义了状态集合上的等价关系。因此状态集合能被划分成等价类
- 两个状态p和q等价应满足如下条件：
  - 一致性条件
    - p和q必须同时或为接受状态或为非接受状态
  - 蔓延性条件：
    - 对于 $\forall a \in \Sigma$ ,  $\delta(p,a)=r$ ,  $\delta(q,a)=s$ , r和s必须等价;
    - 反过来, r和s不等价, 则p和q不等价。
- 可以按照上述条件把所有状态划分为不同的等价类

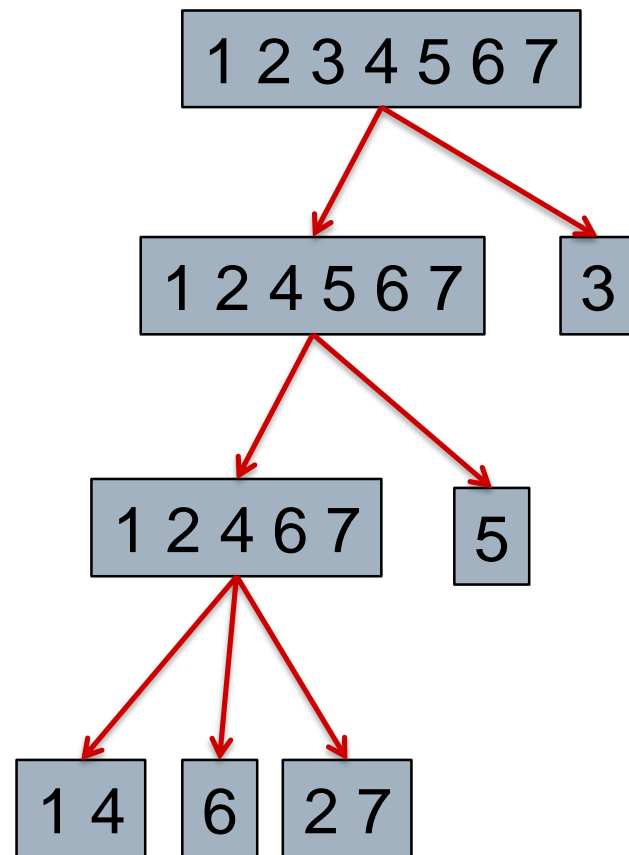
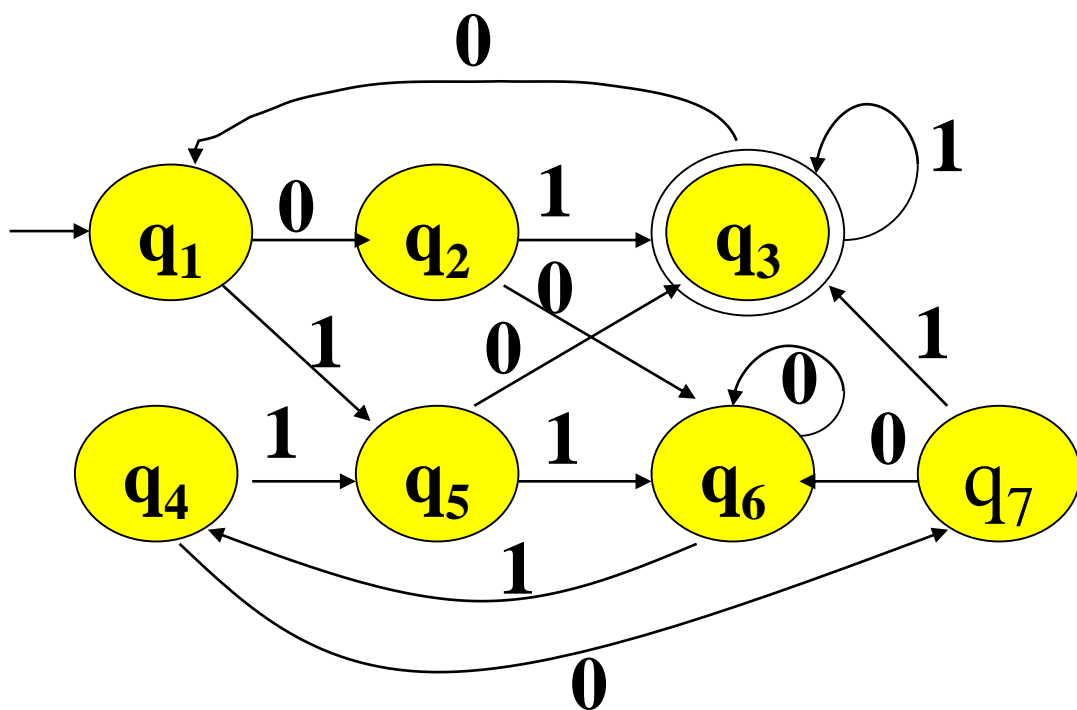




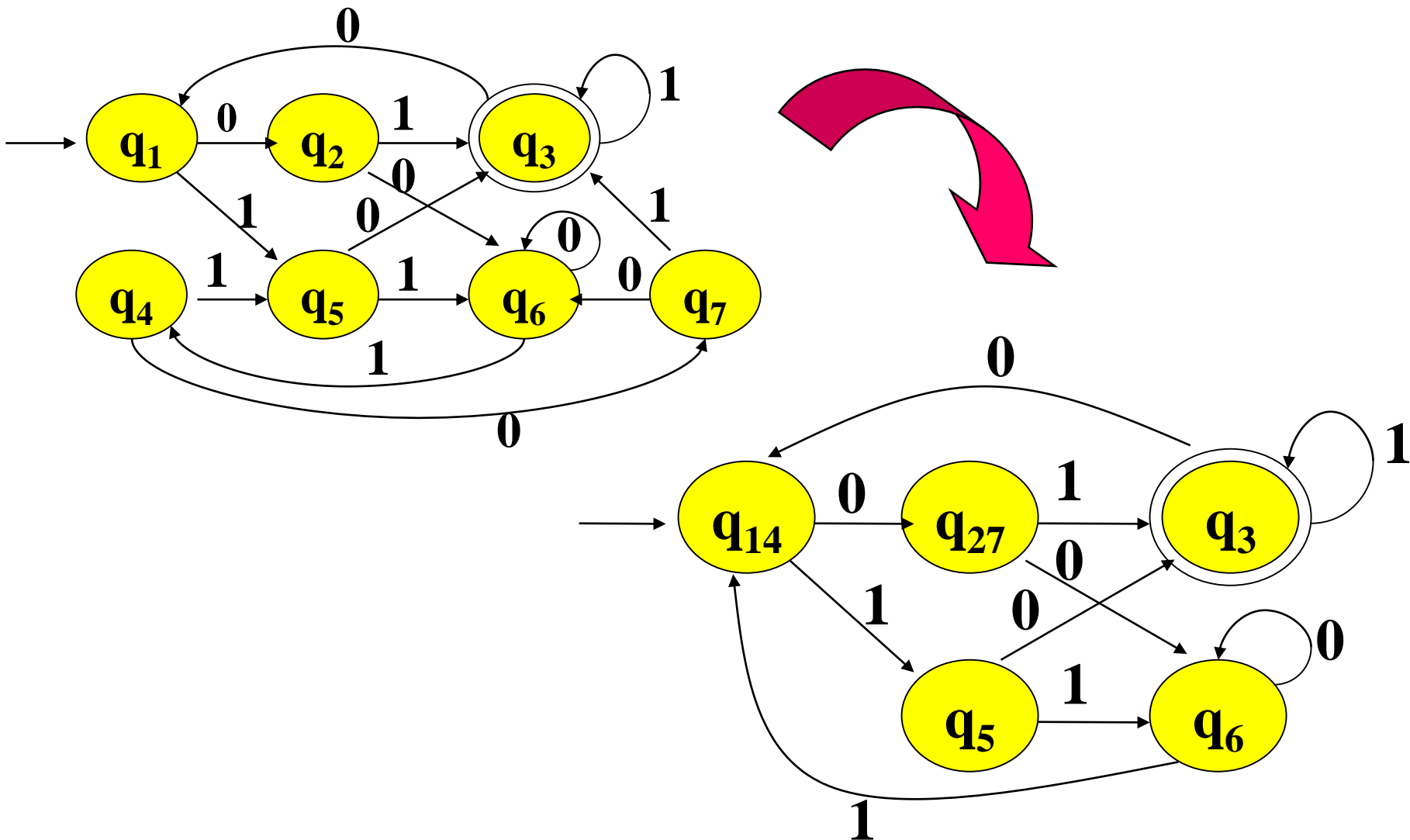
# 等价类划分方法

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：DFA的最小化



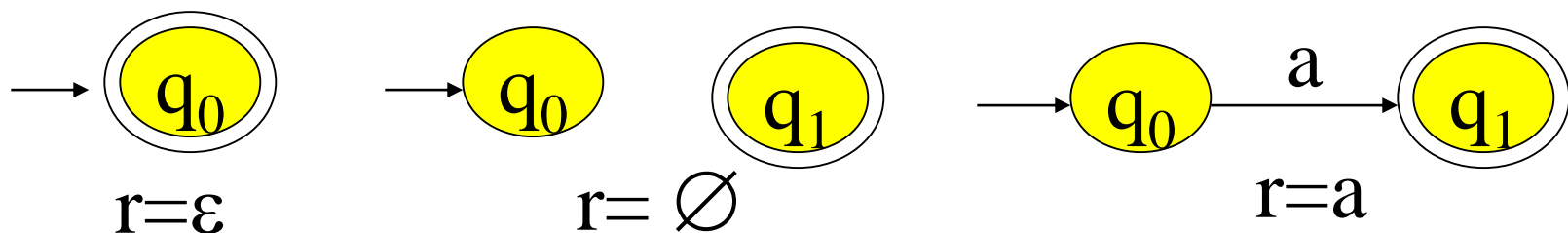
# 化简之后



# 从正则表达式构造FA

**定理：** 设 $r$ 是 $\Sigma$ 上一个正则表达式，则存在 FA  $M$  接受  $L(r)$ ，并且 $M$ 的终态是唯一的且无有向边射出。

证： $\Rightarrow$  对正则表达式 $r$ 的运算符数目作归纳。设 $r$ 具有零个运算，必有 $r=\varepsilon$ 或 $r=\emptyset$ 或 $r=a \in \Sigma$ ，则FA分别为：

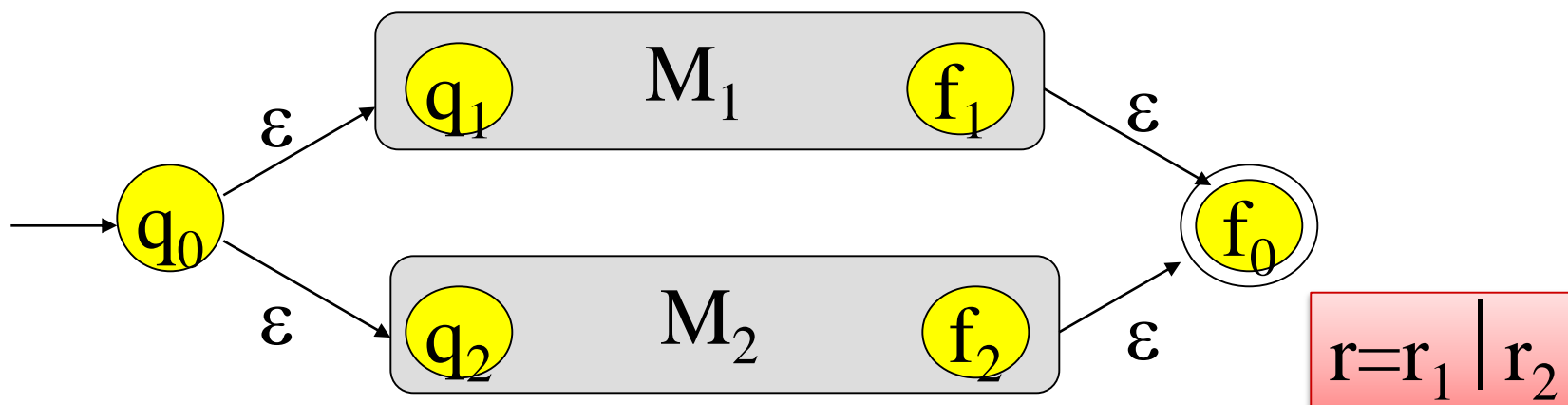


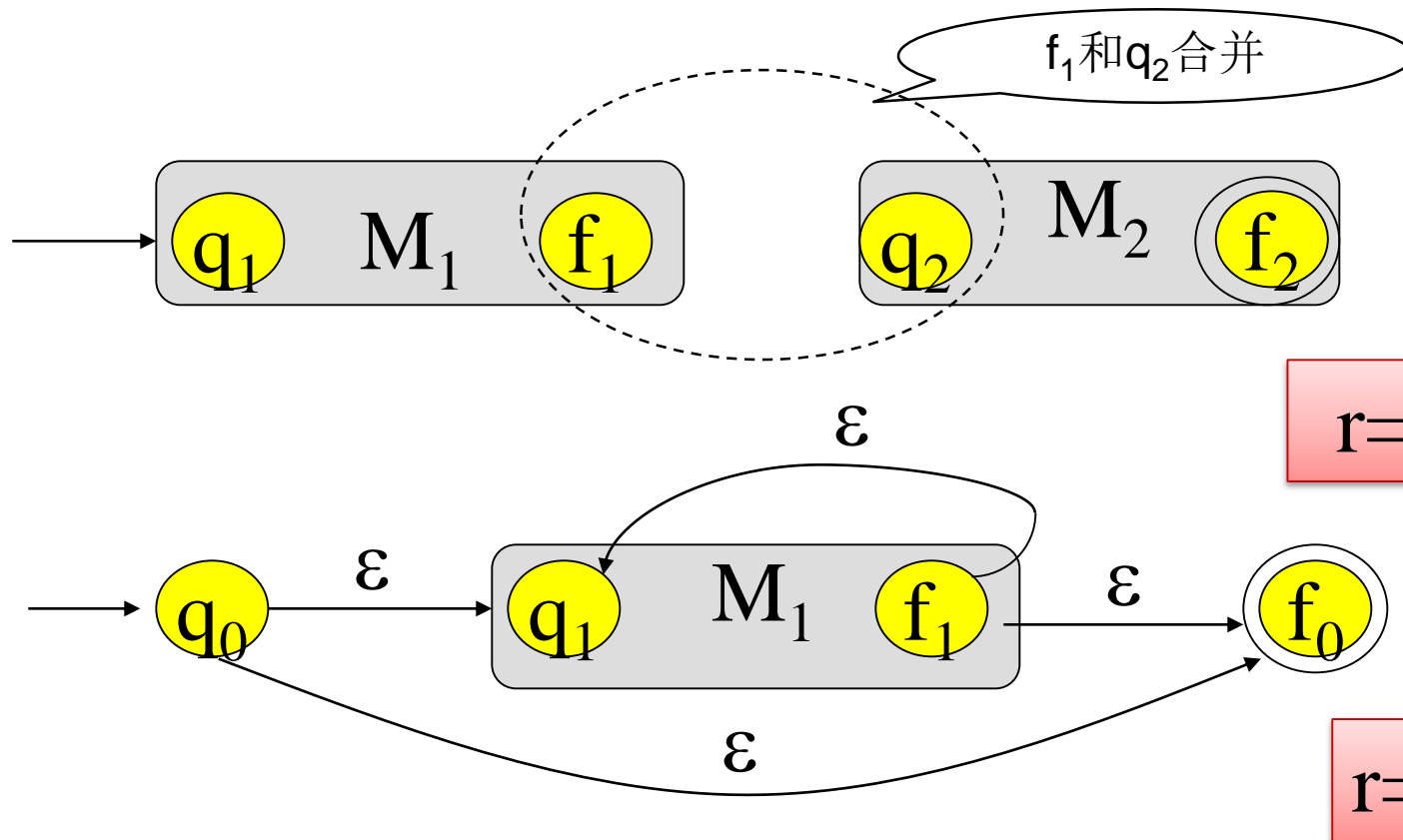
设结论对少于 $i(i \geq 1)$ 个运算的正则表达式 $r$ 成立。  
 当 $r$ 有 $i$ 个运算时，有三种情况：

情况1  $r=r_1 | r_2$     情况2  $r=r_1 r_2$     情况3  $r=r_1^*$

有  $M_1=(\Sigma_1, Q_1, q_1, F_1, \delta_1)$ ,  $M_2=(\Sigma_2, Q_2, q_2, F_2, \delta_2)$

且 $L(M_1)=L(r_1)$ ,  $L(M_2)=L(r_2)$ ，由 $M_1$ 和 $M_2$ 构造 $M$ ,使得  $L(M)=L(r)$ 。构造方法图示如下：





由此可以证明：假定知道 $r$ 的计算顺序，对于任意正则表达式 $r$ ，可以构造一个FA  $M$ ，使得  $L(M)=L(r)$ 。



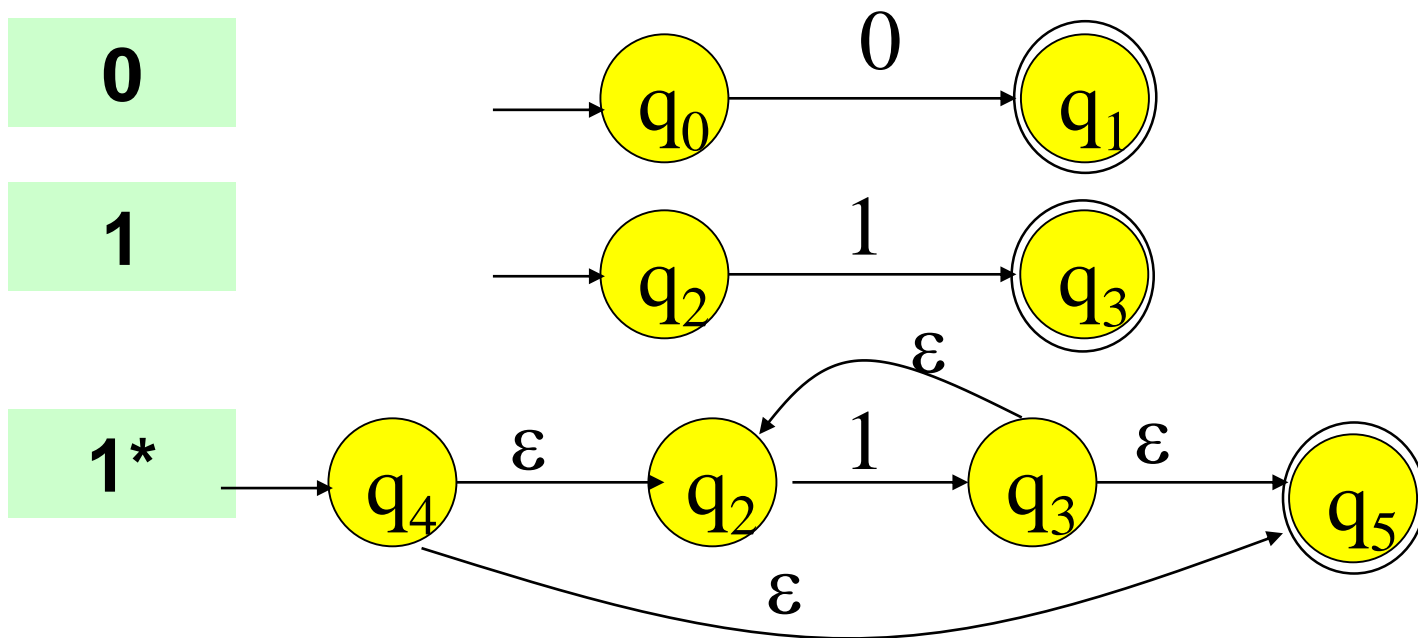
# 转换得到的NFA的特性

- 状态数量最多为 $r$ 中的运算符和运算符分量总数的两倍
  - 因为每个步骤只引入两个状态
- 有且只有一个开始状态和一个接受状态
- 除接受状态之外，每个状态要么有一条标号不为 $\epsilon$ 的出边，要么有两条标号为 $\epsilon$ 的出边。

例：构造与下列正则式

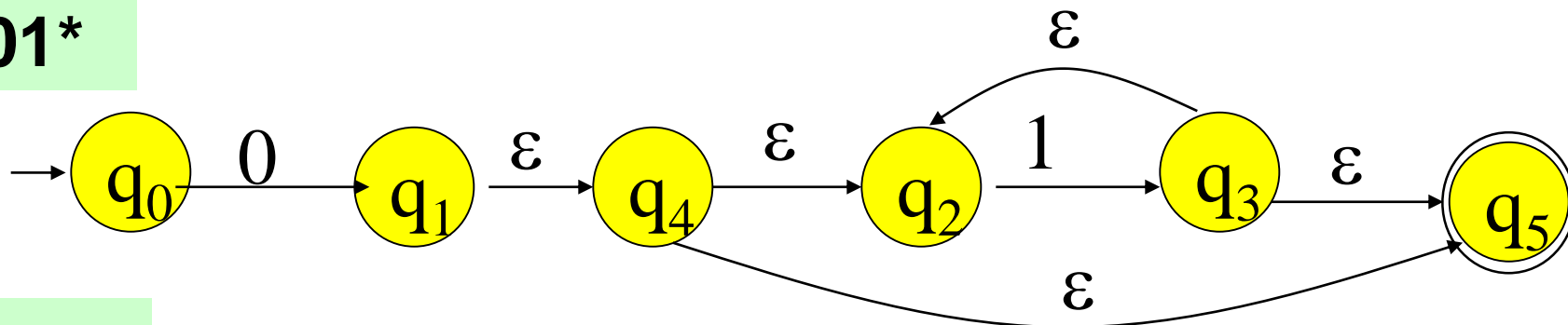
$$r = 01^* | 1$$

等价的有限自动机。

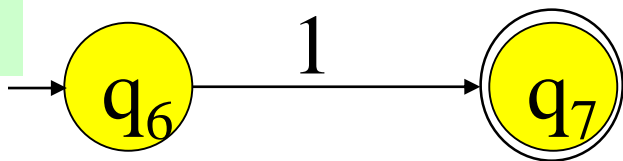




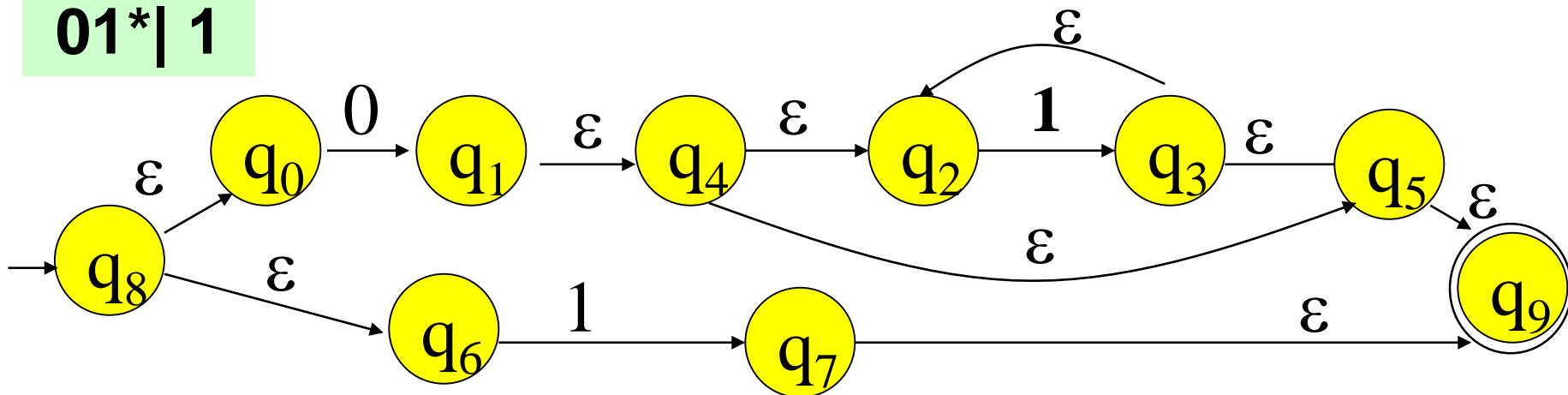
**$01^*$**



**1**



**$01^* | 1$**

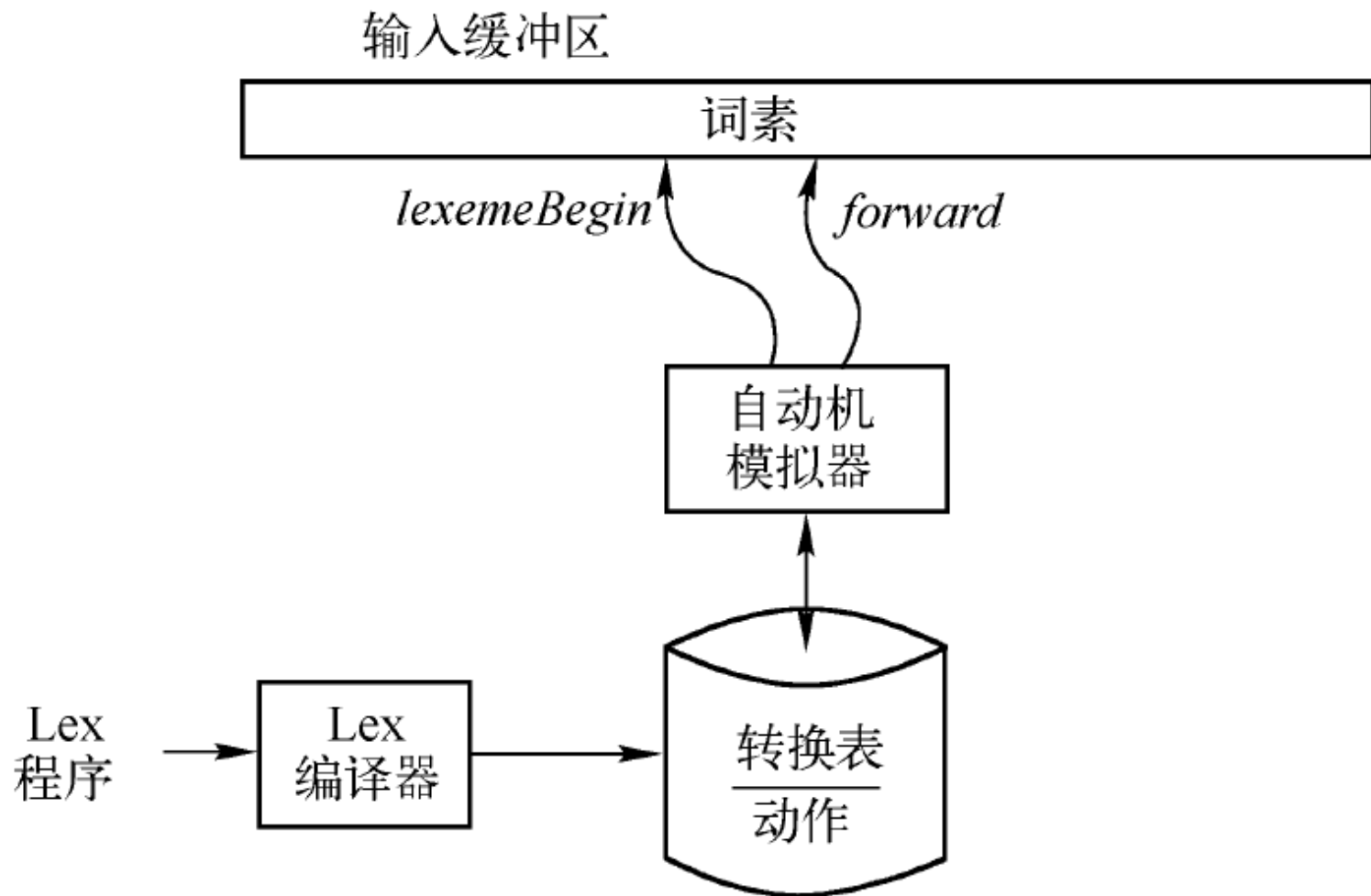




# 内容提要

- 词法分析器的作用
- 词法单元的规约
  - 串和语言；正则表达式、正则定义
- 词法单元的识别
- 词法分析器生成工具—LEX
- 有限自动机 (Finite Automata)
- 正则表达式到有限自动机
- 词法分析器生成工具的设计

# 词法分析器生成工具的设计





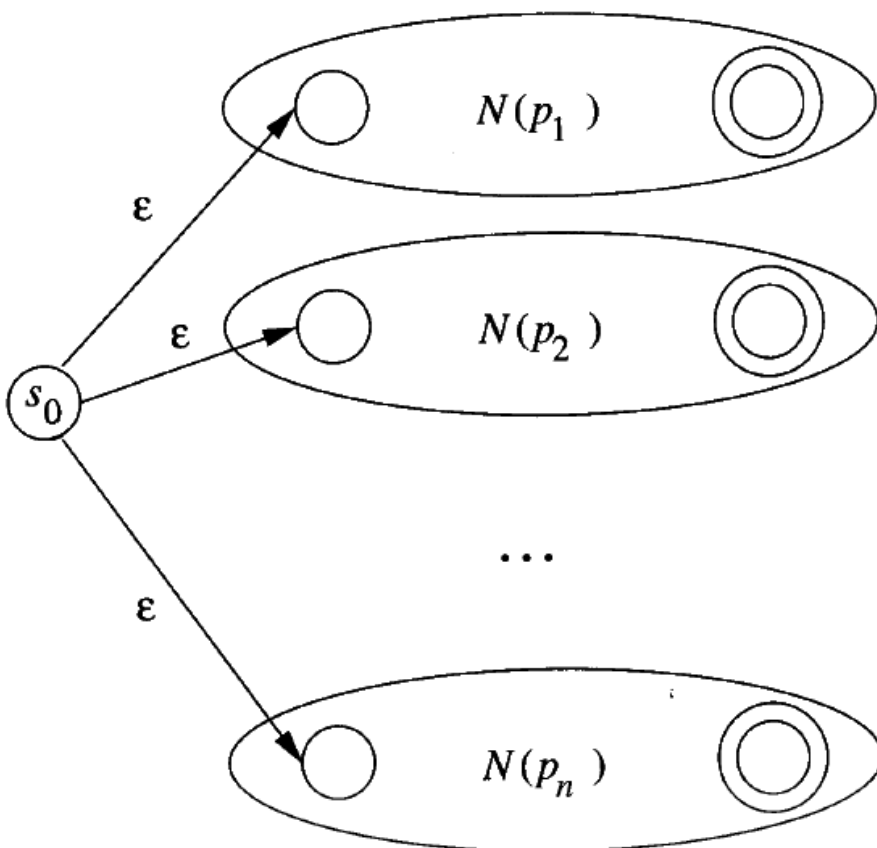
# 词法分析器生成工具的功能

- 生成的词法分析器中包含一个模拟有限自动机的模块
- 其余部分由生成工具根据词法规则的描述自动生成，包括
  - 自动机的转换表
  - 和动作相关的代码，适当的时候由模拟器调用。
- 构造自动机时
  - 首先构造出各个模式对应的NFA
  - 然后将这些NFA合并成为一个NFA
  - （根据需要）进行确定化

# NFA合并的方法

## □ 合并方法：

- 引入新的开始状态，并引入从这个开始状态到各个原开始状态的 $\epsilon$ 转换。
- 得到的NFA所接受的语言是原来各个NFA的语言的并集。
- 不同的接受状态可代表不同的模式。
- 不仅判断输入前缀是否NFA的语言，还需要知道对应于哪个模式





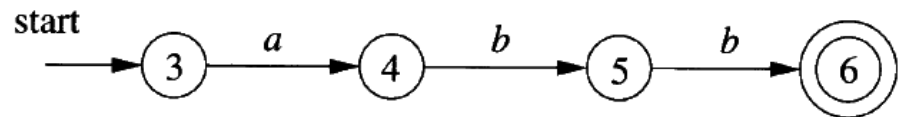
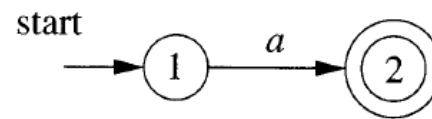
# NFA到DFA的转换

- 对得到的NFA进行确定化，得到DFA。
  - 对得到的DFA的状态进行最小化
- 一个DFA的接受状态对应于NFA状态的集合，其中至少包括一个NFA接受状态
  - 如果其中包括多个对应于不同模式的NFA接受状态，则表示当前的输入前缀对应于多个模式，存在冲突。
- 找出第一个这样的模式，将这个模式作为这个DFA接受状态的输出。

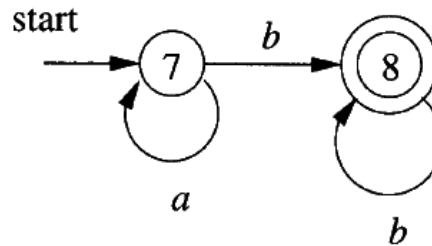
# 示例（1）

## □ 假设有三个模式

- $a \{A1\}$
- $abb \{A2\}$
- $a^*b^+ \{A3\}$



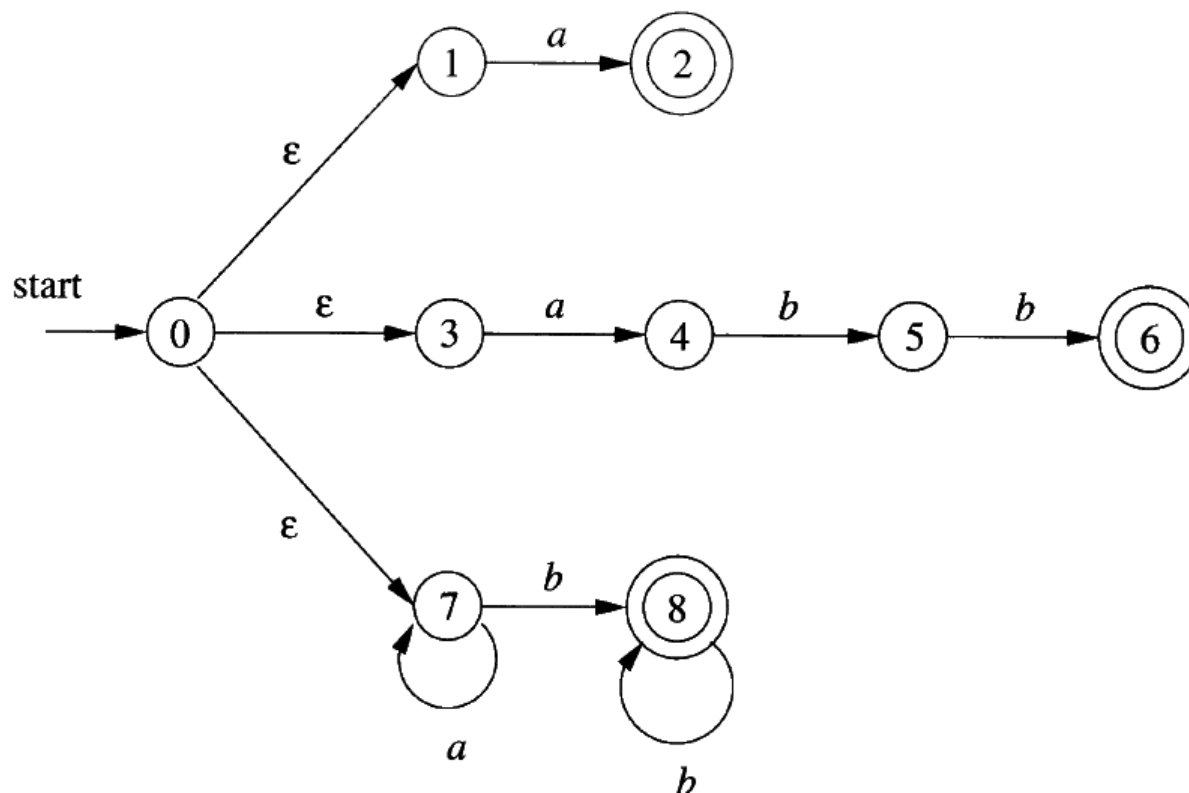
## □ 构造各模式的NFA如右



# 示例 (2)

## □ 合并NFA

- 2: 模式1
- 6: 模式2
- 8: 模式3

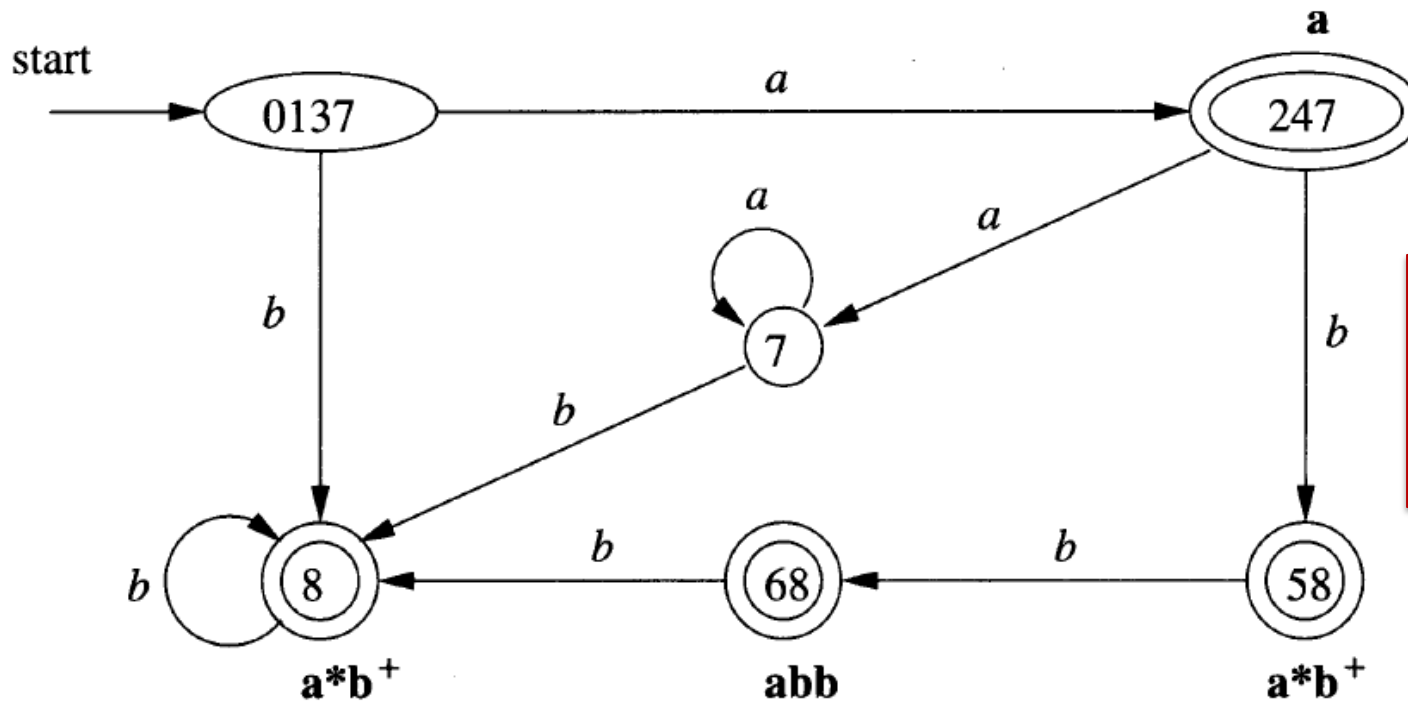




# 示例 (3)

## □ 确定化得到如下DFA

- DFA状态68对应NFA状态集合{6,8}, 对应的模式是abb (第二个模式), 而不是 $a^*b^+$  (第三个模式)



是否需要  
对DFA进  
行最小化?



# 运行的方式

- 模拟DFA，不断读入输入字符串中的字符
- 直到某一时刻没有后继为止（不是到达某个接受状态）
  - 注意：根据教材的定义，DFA总是有后继的。
  - 这里是指DFA进入了死状态，即永远不可能到达接受状态的状态。
  - 这样可以找到最长可能的词素。
- 回头查找最后的接受状态，执行相应的动作
  - 如果查不到，报词法错
  - 在回退时，需要同时回退读入的字符



# 本章小结

- 词法规则通常可以使用正则表达式来描述。
  - LEX中使用正则表达式来自动生成词法分析器
- 有限自动机（FA）可以用来描述词法规则。
  - DFA、NFA
  - NFA 到 DFA 的转换
  - DFA 的最小化
- 正则表达式到有限自动机的转换
- 词法分析器生成工具Lex
  - Lex的工作原理

# 作业

注：在括号中标注“本”的为本科教学版对应的习题编号。



- 9月27日交
- NFA->DFA
  - Ex. 3.7.1 (本 Ex. 3.6.1) 2-3小题
- 正则表达式 -> DFA
  - Ex. 3.7.3 (本 Ex. 3.6.3) 3-4小题（对得到DFA进行最小化）