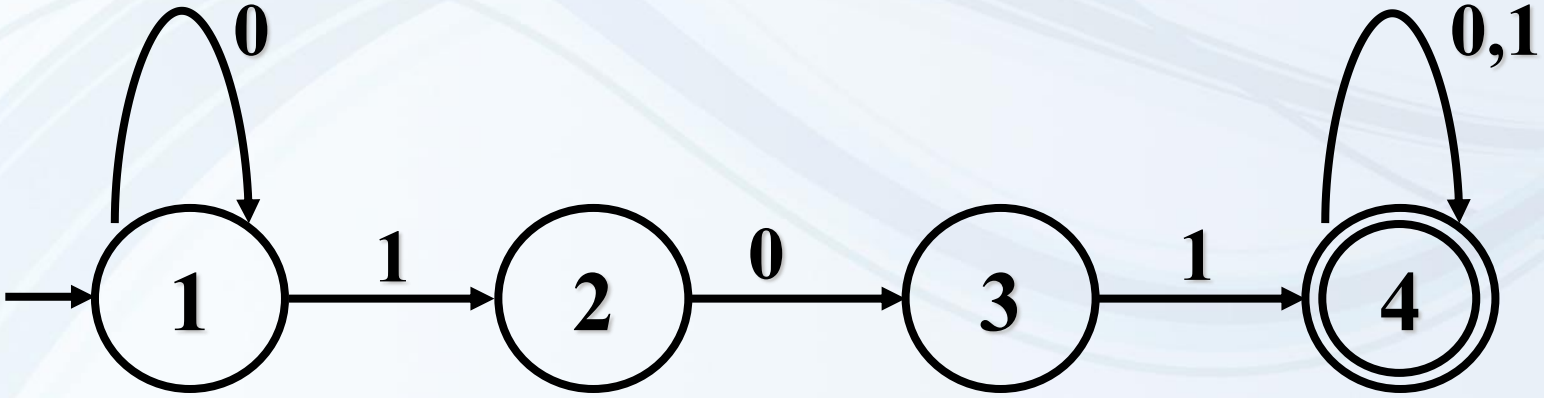
3.4 词法单元的识别

❖状态转换图(*Transition Diagram*)用来描述和识别记号:

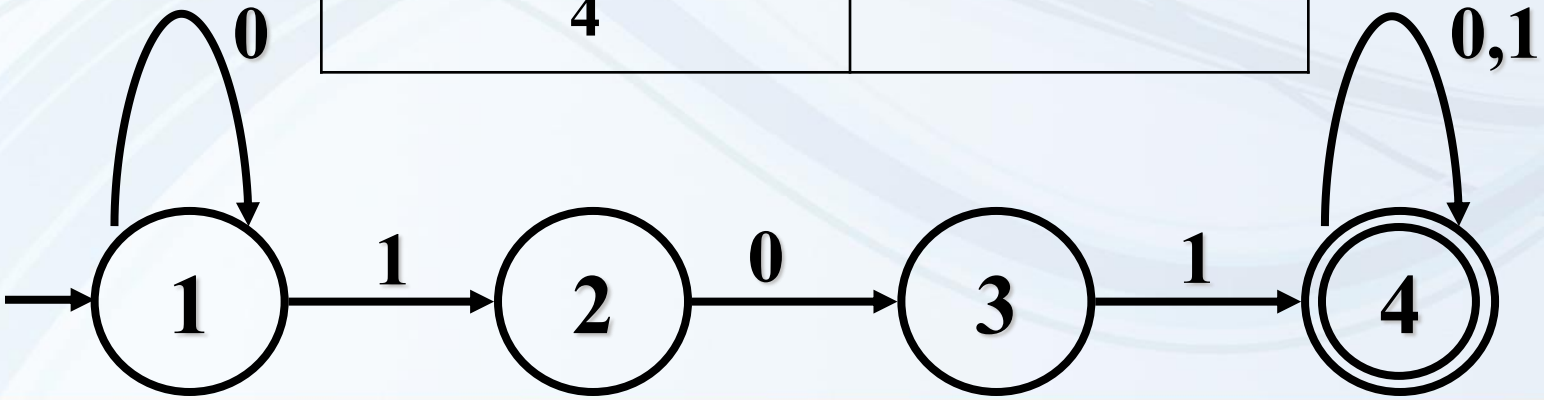
- ➢状态*States*: 圆圈
- ➢动作*Actions*: 箭头(含文字)
- ➢初态*Start State*:模式的开始，表示为无源箭头的目标
- ➢终态*Final State(s)*:模式的结束，用双圈表示

还可以用什么描述记号？

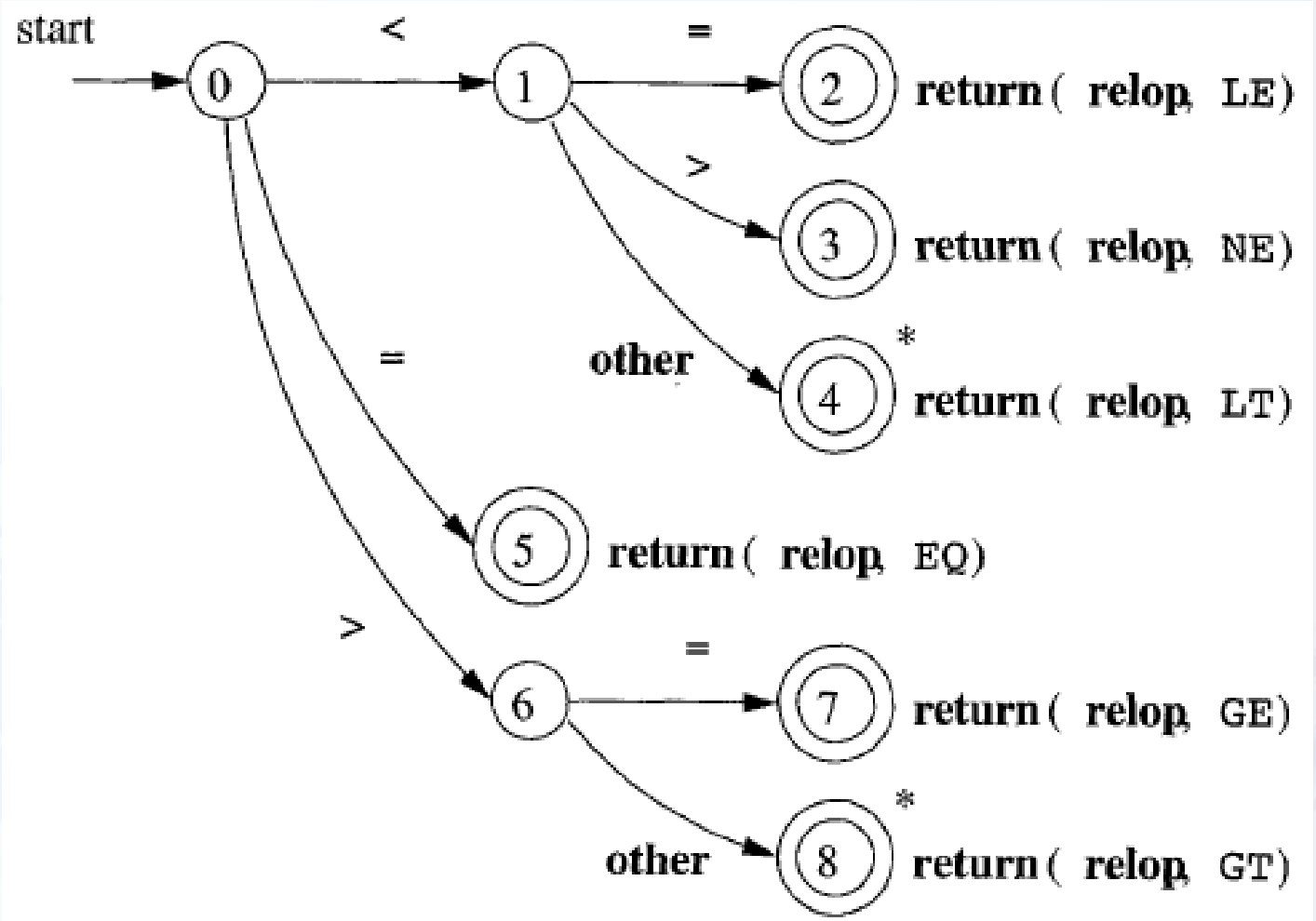


状态转换图例

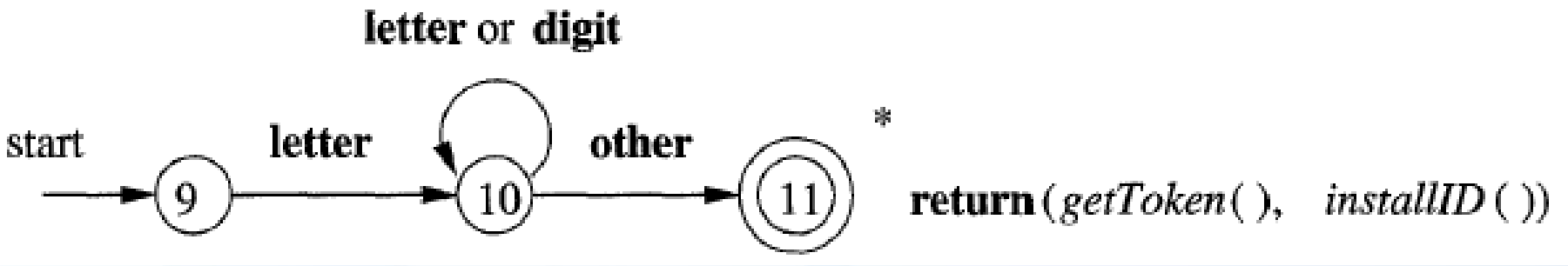
state	input
1	00101
1	0101
1	101
2	01
3	1
4	



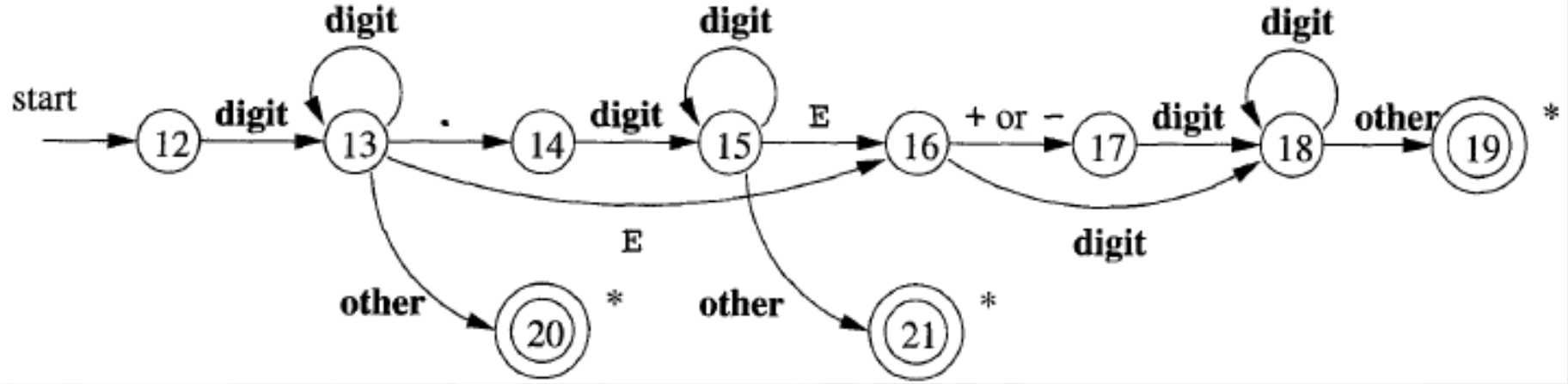
词法单元relop的状态转换图



id和关键字的状态转换图

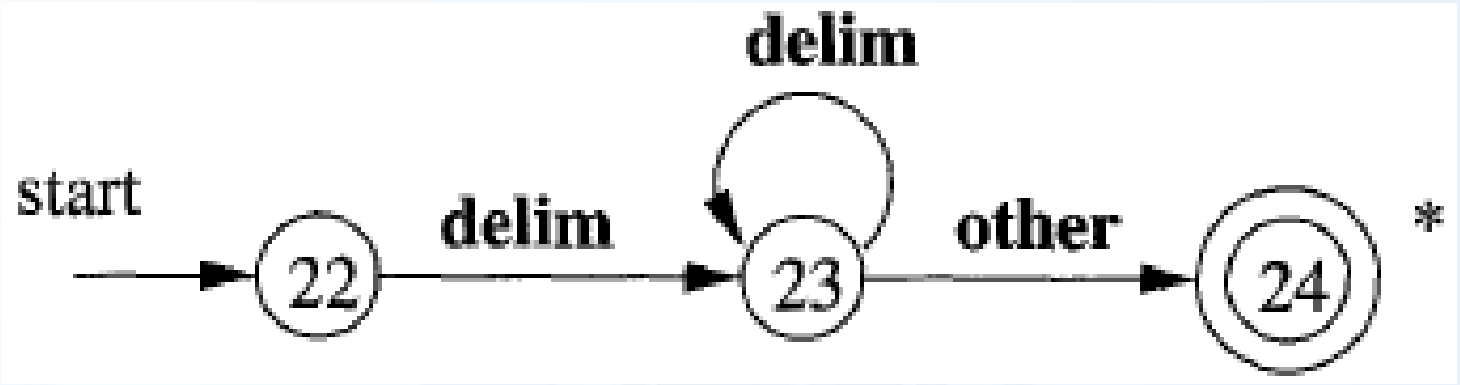


无符号数字的状态转换图



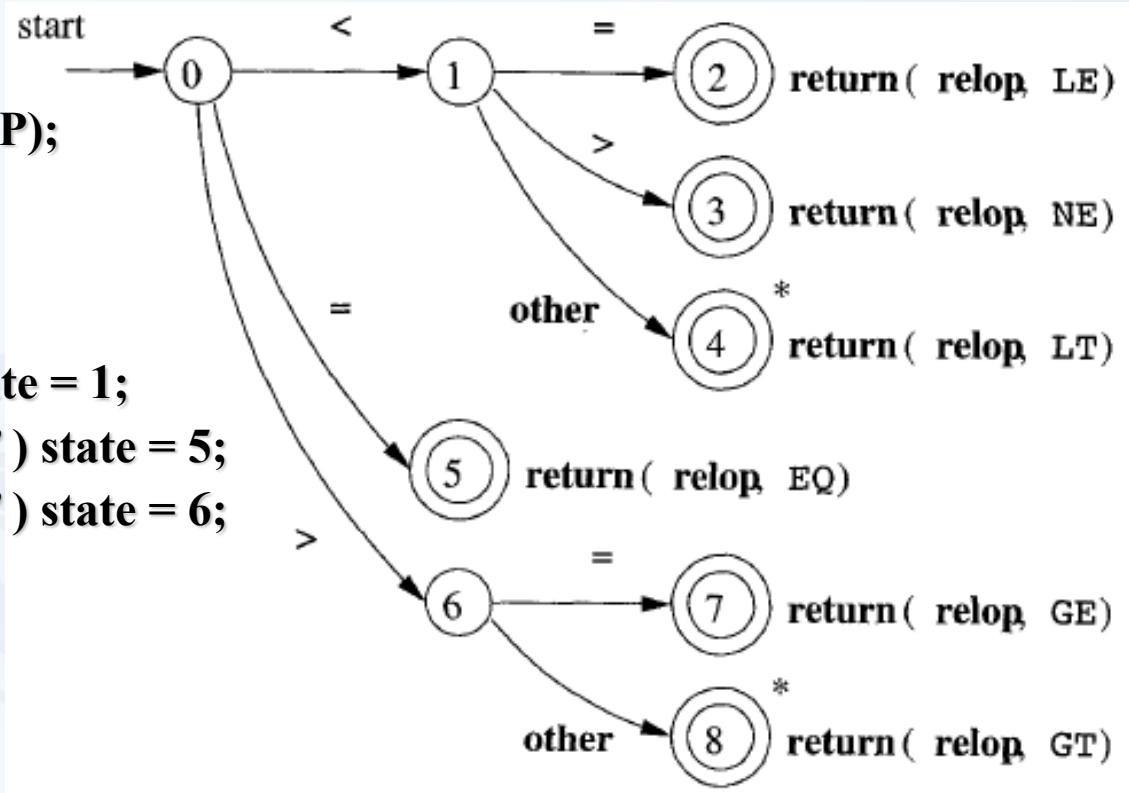
$digit^{+}(.digit^{+})?(E(+|–)?digit^{+})?$

空白符的状态转换图



relop的状态转换图的实现

```
TOKEN getRelop( ){
   TOKEN retToken = new(RELOP);
   while(1) {
    switch(state) {
          case 0:    c = nextChar();
                     if ( c == '<' ) state = 1;
                     else  if ( c == '=' ) state = 5;
                     else  if ( c == '>' ) state = 6;
                     else  fail( ) ;
                     break;
          case 1: . . .
          ...
          case 8:  retract( ) ;  retToken.attribute = GT;    return(retToken) ;
          }
    }
}
```



匹配失败的处理

```
int state = 0,  start = 0;
int fail( )
{    start = state;     forward = lexeme beginning;
     switch (start) {
        case 0:   start = 9;  break;
        case 9:   start = 12; break;
        case 12:  start = 22; break;
        case 22:  start = 25; break;
        case 25:  recover( );  break;
        default:   /* compiler error */
     }
     return start;
}
```

给出识别正则式所描述语言的状态转换图

- ab
- a|b
- (a|b)*
- (a|b)$^+$
- a(a|b)*a
- a(a|b)*a?
- a?(a|b)*a
- a*b*c*
- (a|b)*a(a|b)(a|b)
- aaa | bbb
- (a|b)*(aa|bb)(a|b)*

（1）首先关注连接操作

（2）扩展表示要转化

（3）连续闭包要分离

（4）或操作逐步求精

3.5 词法分析器生成工具Lex

- 功能：将待识别模式的形式化描述（正则式）转换成状态转换图，生成代码，并存放到文件lex.yy.c中。
- Lex语言：Lex工具的输入表示方法
- Lex的使用
  - 用Lex语言写一个输入文件，描述要生成的词法分析器
  - Lex编译器将lex文件转换成C语言程序
  - 编译C程序,即得词法分析器

Lex程序的结构

declarations　　//声明

%%

translation rules　　//转换规则

%%

auxiliary procedures　//辅助过程

一个Lex程序

```
delim       [ \t\n]
ws          {delim}+
letter      [A-Za-z]
digit       [0-9]
id          {letter} ({letter} | {digit}) *
number      {digit}+(\. {digit}+)? (E[+-]?{digit}+)?
%%
{ws}        { }
if          {return(IF);}
then        {return(THEN);}
else        {return(ELSE);}
```

一个Lex程序(续）

```
{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( ){  }
int installNum( ) {  }
```

给出scanner对输入串ababcbacaabaababaa的输出：

a^+b*a	{printf(“1%s\n”, yytext);}
$(ab)^+c?$	{printf(“2%s\n”, yytext);}
aa	{printf(“3%s\n”, yytext);}
(a|b)*c	{printf(“4%s\n”, yytext);}

优先原则：

（1）匹配最长串的规则优先

（2）长度相同时，在前的规则优先

a b a b c b a c a a b a a b a b a a

a^+b*a

$(ab)^+c?$

aa

(a|b)*c

给出scanner对输入串ababcbacaabaababaa的输出：

Step1:	输出 2 ababc
a^+b^*a	aba
$(ab)^+c?$	ababc
$(a\|b)^*c$	ababc
Step2:	输出 4 bac
Step3:	输出1 aaba
a^+b^*a	aaba
aa	aa
Step 4:	输出 2 abab
a^+b^*a	aba
$(ab)^+c?$	abab
Step 5:	输出 1 aa

```
a+b*a     {printf("1%s\n", yytext}
(ab)+c?   {printf("2%s\n", yytext}
aa        {printf("3%s\n", yytext}
(a|b)*c   {printf("4%s\n", yytext}
```

```
2 ababc
4 bac
1 aaba
2 abab
1 aa
```

例

输入abbbaacc，以下哪些模式组可以识别出ab/bb/a/acc?多选

(A)
ab
b^+
ac*

(B)
b^+
ab*
ac*

(C)
a(b|c*)
b^+

(D)
c*
b^+
ab
ac*

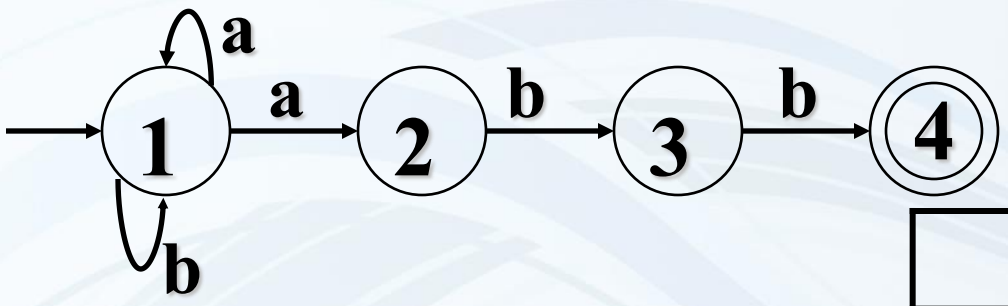
3.6 有穷自动机

- 不确定的有穷自动机 (NFA, *Non-Deterministic Finite Automata*)是一个五元组，它包括:
  - 有穷状态集合S　　状态集
  - 输入符号集合Σ　　字母表
  - 转换函数*move*　　转移函数
    - *move*(state, symbol) → *set of states*
  - 一个开始状态 $s_0 \in S$　　开始状态(初态)
  - 一个接受状态集合$F \subseteq S$　　终止状态集(终态)

NFA的两种表示法

状态转换图：包括状态(圆圈)、边、初态、终态

状态转换表：描述各个状态下，对应于各个输入的状态迁移



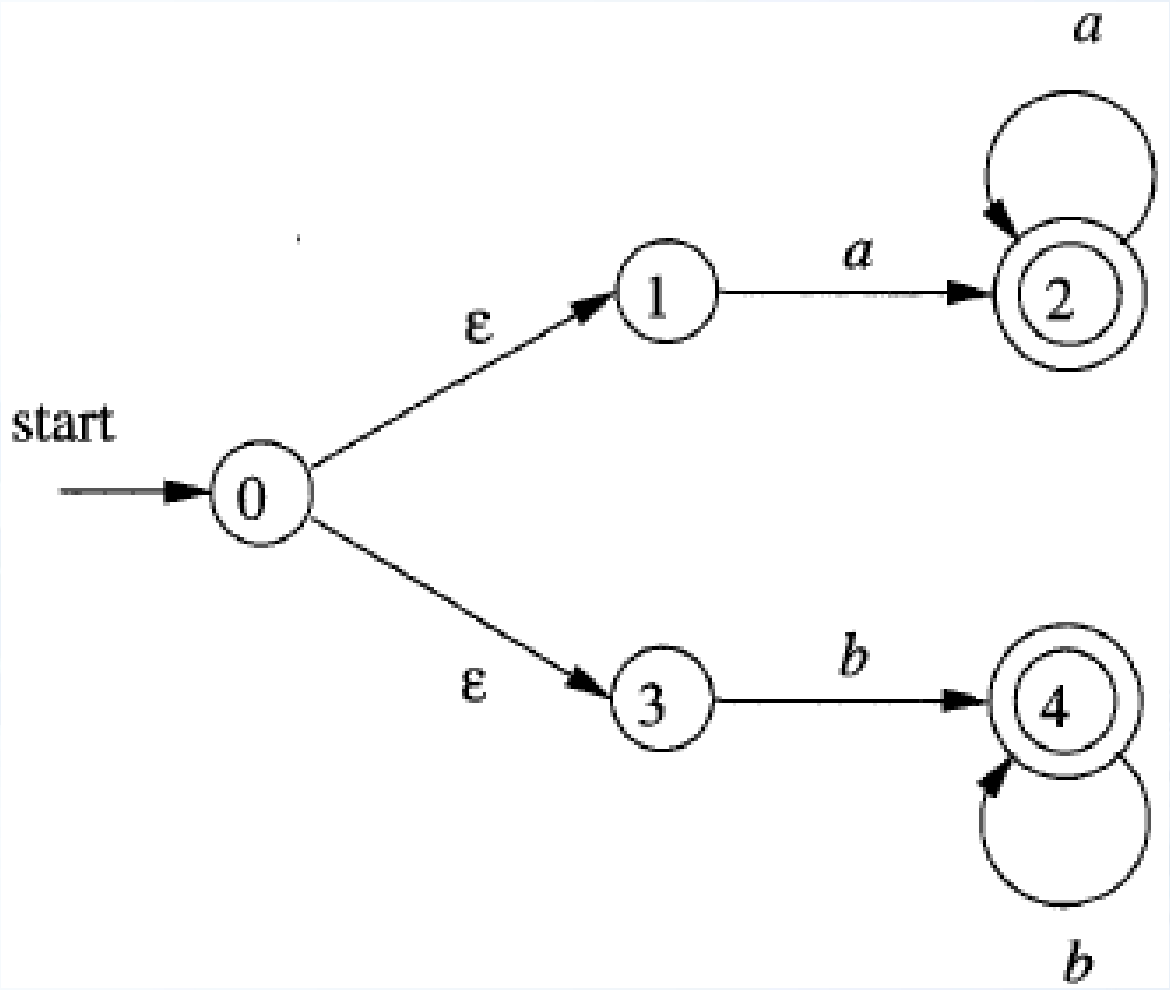
状态转换图中无 ε 边时，状态转换表中可以忽略 ε 列。

	a	b	ε
1	{1,2}	{1}	Ø
2	Ø	{3}	Ø
3	Ø	{4}	Ø
4	Ø	Ø	Ø

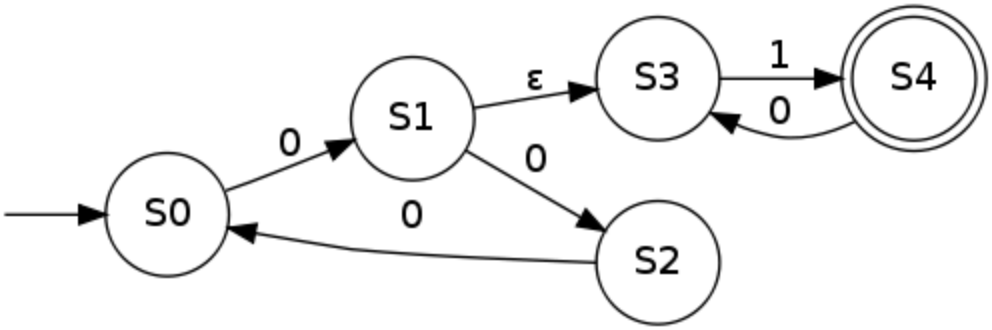
输入串的接受

- 一个NFA接受(*accept*)输入串x, 当且仅当状态转换图中存在一条从初态到某个终态的路径，该路径上各条边的符号组成串x。
- 一个NFA定义（接受）的语言是其接受的所有串的集合。

自动机中输入串的接受（例）



（多选）哪些正则式生成的语言与所给NFA识别之语言相同?



A (000)*(01)+
B 0(000)*1(01)*
C (000)*(10)+
D 0(000)*(01)*
E 0(00)*(10)