北京大学计算机学院

2024年秋季学期　　《编译原理》

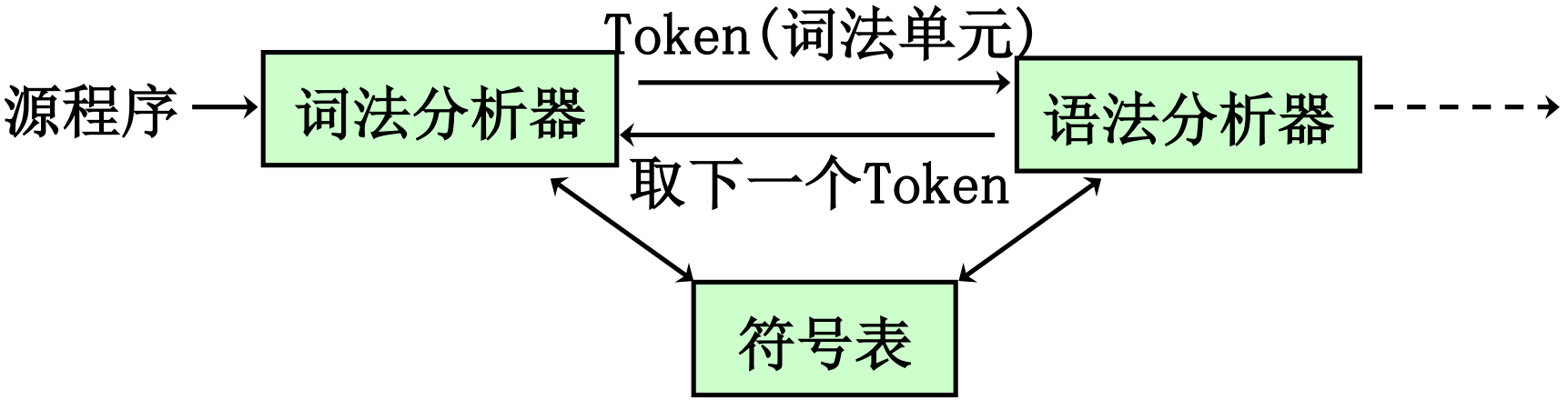
第3章 词法分析（2）

Lexical Analysis

【对应教材 3.3- 3.5】

上节内容回顾

- 词法分析器的作用



- 词法单元的描述方法
  - 字母表、符号串和语言
  - 正则集合、正则表达式和正则定义

Review Questions

- 写一个正则表达式，表示所有能被5整除的十进制数。
  - (0|1|…|9)*(0|5)
- 写一个正则表达式，表示所有能被5整除的不包含前导0的十进制数。
  - 0 | 5 | (1|2|…|9) (0|1|…|9)*(0|5)
- 写一个正则表达式，表示所有能被5整除的二进制数。
  - (0|1(10)*(0|11)(01*01|01*00(10)*(0|11))*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

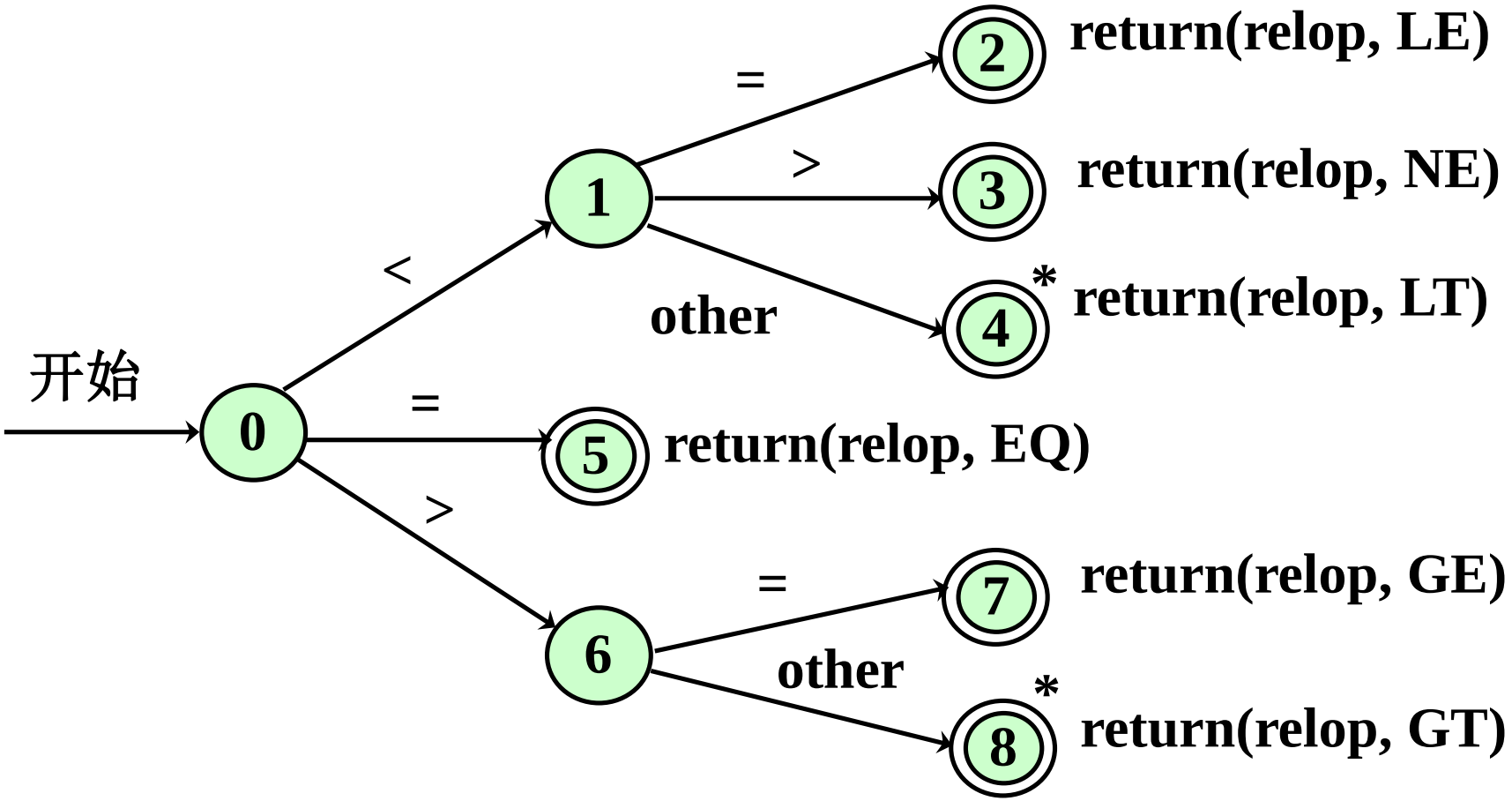
如何识别词法单元？

- 一般有两种方式：
  - 借助状态转换图（有限自动机的图形表示）手工构造词法分析器
  - 通过LEX自动生成词法分析器。
    - 正则表达式
      ⇒ NFA
      ⇒ DFA
      ⇒ minDFA
      ⇒词法分析器

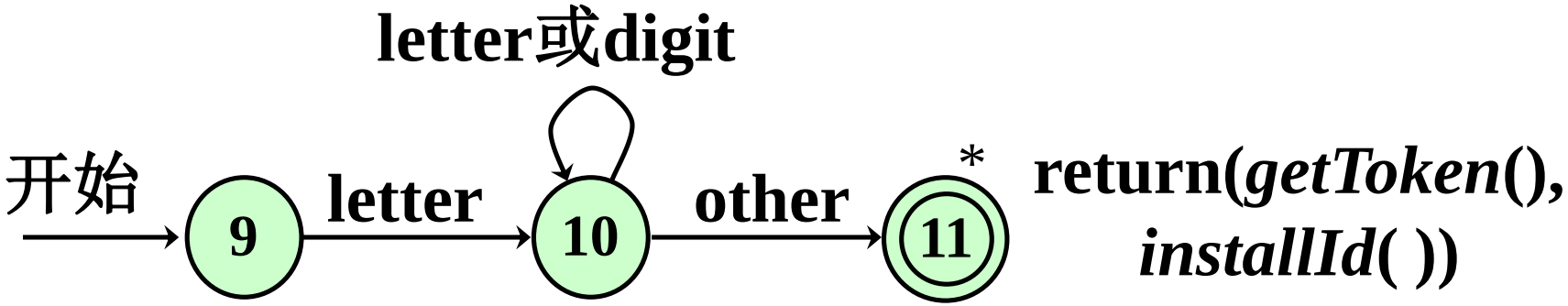
状态转换图

- 状态转换图(transition diagram)
  - 状态(state)：表示在识别词素时可能出现的情况
    - 状态看作是已处理部分的总结
    - 某些状态为接受状态或最终状态，表明已找到词素
    - 加上*的接受状态表示最后读入的符号不在词素中
    - 开始状态（初始状态）：用“开始”边表示
  - 边(edge)：从一个状态指向另一个状态；边的标号是一个或多个符号
    - 当前状态为s，下一个输入符号为a，就沿着从s离开，标号为a的边到达下一个状态

关系算符的转换图

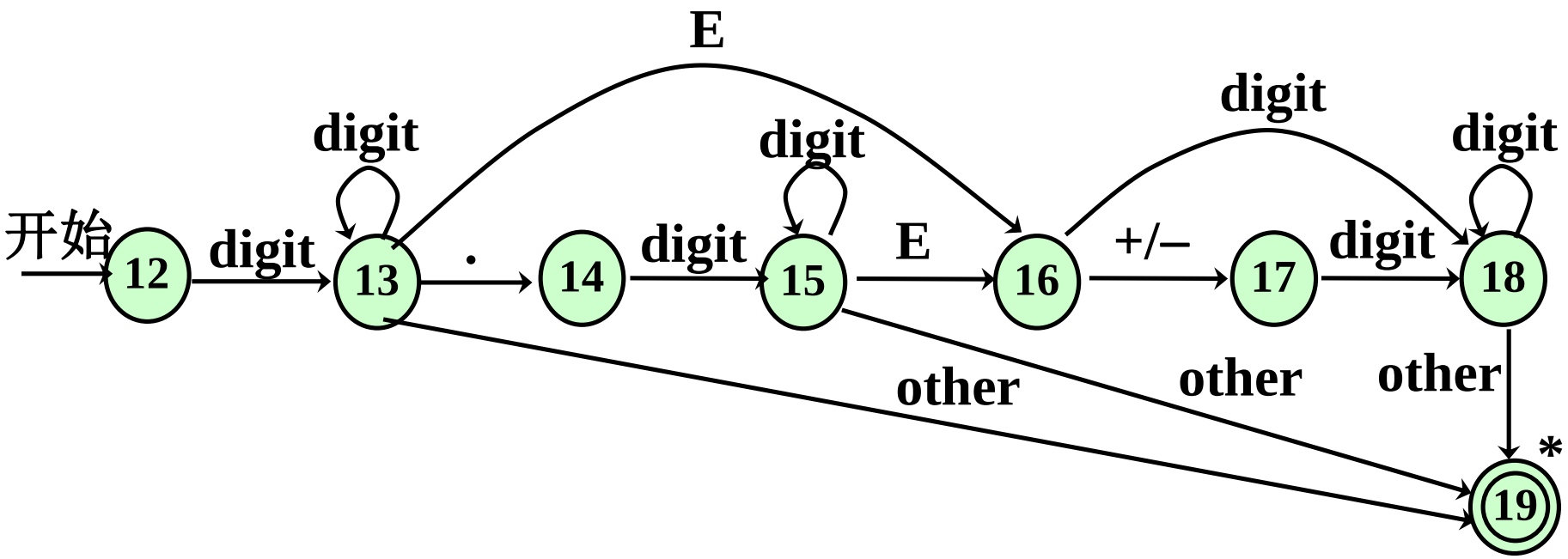

开始
0
<
=
>
1
2
3
4
5
6
7
8
=
>
other
=
other
*
*
return(relop, LE)
return(relop, NE)
return(relop, LT)
return(relop, EQ)
return(relop, GE)
return(relop, GT)

标识符和保留字的转换图



无符号数的转换图

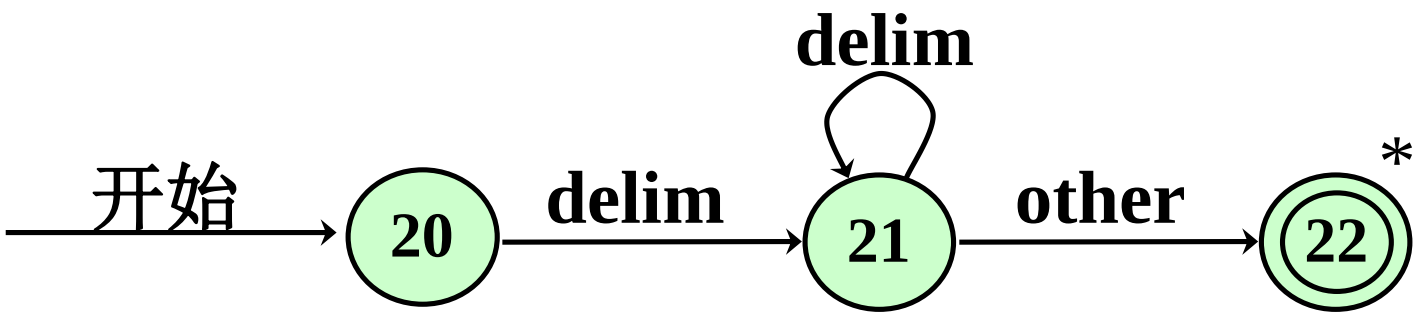
number → digit+ (.digit+)? (E (+ | –)? digit+)?



识别空白的转换图

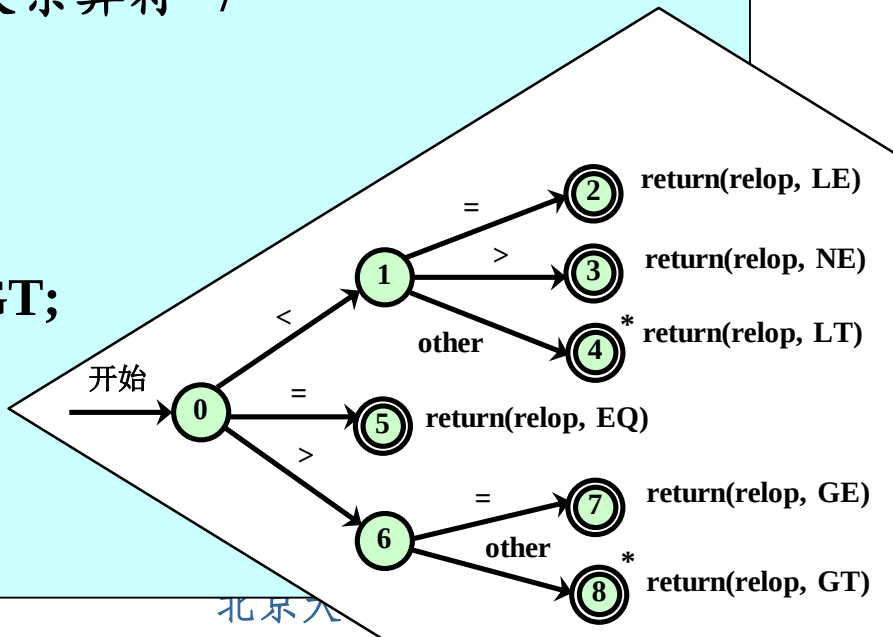
delim → blank | tab | newline

ws → delim$^+$



手动编写词法分析程序：以relop为例

```
TOKEN getRelop ( )
{    TOKEN retToken = new ( RELOP ) ;
     while ( 1 ) {    /* 反复读入字符，直到return或 遇到错误 */
          switch (state) {
           case 0 : c = nextChar ( ) ;
                    if ( c == ' < ' ) state = 1 ;
                    else if ( c == ' = ' ) state = 5 ;
                    else if ( c == ' > ' ) state = 6 ;
                    else fail ( ) ;      /* 非关系算符 */
                    break ;
           case 1 :  ……
              ……
           case 8 : retract ( );
                    retToken.attribute = GT;
                    return (retToken);
          }
     }
}
```



词法单元的自动识别

- 首先通过正则表达式来描述词法单元的模式
- 基本目标：判断一个串s是否属于一个正则表达式R表示的语言

$$s \in L(R)$$

- 在现实中，还要能够连续识别多个不同类别的词法单元

```
if (a == b) …
```

词法自动识别过程

(1) 分别为每一类词法单元写出正则表达式R_i

(2) 构造一个正则表达式R来匹配所有的词法单元

$$R = R_1 \mid R_2 \mid \ldots \mid R_k$$

(3) 设输入为$x_1x_2\ldots x_n$, 对$1 \le i \le n$，检查是否

$$x_1\ldots x_i \in L(R)$$

(4) 如果匹配成功，则存在j，使得

$$x_1\ldots x_i \in L(R_j)$$

(5) 把$x_1\ldots x_i$从输入中移走，继续执行（3）

匹配过程中需要解决的问题

- 如何确定匹配的长度？
  - 有可能多个前缀都可以产生匹配
  - 解决办法：匹配最长可能的串
- 选择哪个正则表达式来匹配？
  - 有可能多个正则表达式都可以匹配
  - 解决办法：排在前面的正则表达式优先匹配
- 如果所有正则表达式都不能匹配怎么办？
  - 怎么报错？
  - 解决办法：可以构造一个ERROR正则表达式，放到所有表达式在后面，用来报告错误信息

Quiz：选择题

- 使用如下的词法描述，在识别字符串“dictatorial”的过程中会如何进行分割？

  dict (1)
  dictator (2)
  [a-z]* (3)
  dictatorial (4)

  a) 4
  b) 3
  c) 1, 3
  d) 2, 3

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

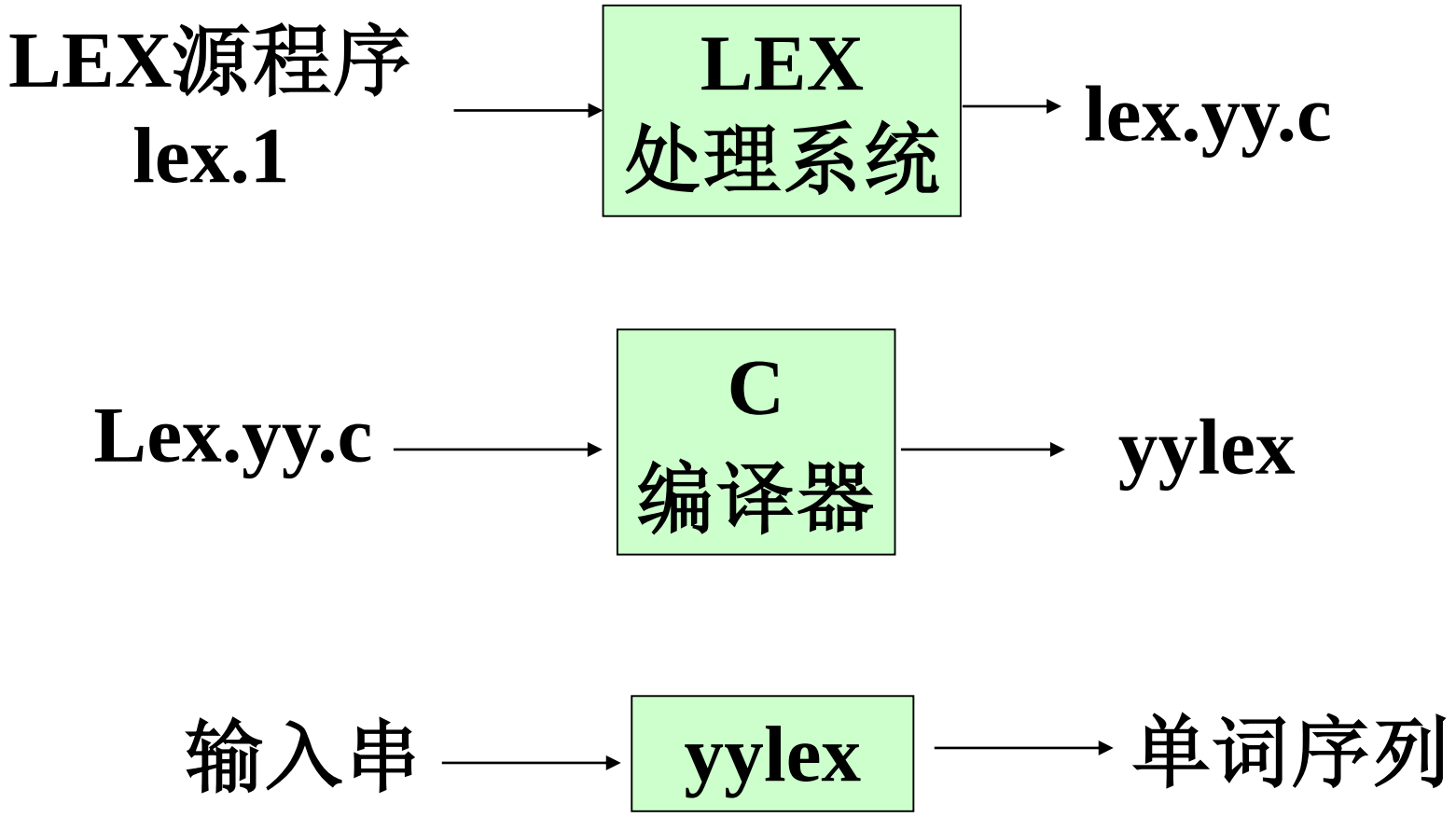
Lex 简介

- 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⇒ NFA ⇒DFA ⇒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

```
%{
      /* def init ions of manifest constant s
      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} l{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 () ; rettirn(ID) ;}
{number}  {yylval = (int) installNum() ; return (NUMBER) ;}

" < "     {yylval = LT ; return (RELOP) ;}
" <="     {yylval = LE ; return (RELOP) ;}
" = "     {yylval = EQ ; return (RELOP) j}
" <>"     {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*/
```

Lex中的冲突解决方法

- 冲突：多个输入前缀与某个模式相匹配，或者一个前缀和多个模式匹配
- Lex解决冲突的方法
  - 多个前缀可能匹配时，选择最长的前缀
    - E.g, 保证词法分析器把 <= 当作一个词法单元
  - 某个前缀和多个模式匹配时，选择列在前面的模式
    - E.g, 如果保留字的规则在标识符的规则之前，词法分析器将识别出保留字

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

有限自动机（Finite Automata）

- 有限自动机是词法分析器生成工具（Lex）的关键技术
  - 正则表达式→有限自动机→词法分析程序
- 有限自动机与状态转换图类似，但是有限自动机是识别器，只能对每个可能的输入串简单地回答“yes”or“no”。
- 有限自动机可以分为两类：
  - 确定的有限自动机（DFA）
  - 不确定的有限自动机（NFA）

确定的有限自动机 (Deterministic FA)

定义 一个确定的有限自动机 M (记作DFA M)

是一个五元组 $M=(\Sigma, Q, q_0, F, \delta)$, 其中

（1） Σ是一个有限字母表, 它的每个元素称为一个输入符号

（2） Q 是一个有限状态集合

（3） $q_0 \in Q$, q_0 称为开始状态

（4） $F \subseteq Q$, F 称为终止状态（或接受状态）集合

（5） δ是一个从 $Q \times \Sigma$ 到 Q 的单值映射 (称为转换函数)

$$\delta(q, a)=q' \quad (q, q' \in Q, a \in \Sigma)$$

表示当前状态为q, 输入符号为a 时, 自动机 M 将转换到下一个状态q′， q′ 称为q的一个后继。

DFA的表示形式

例： 设DFA M＝({a,b}, {0,1,2,3}, 0, {3}, δ)

其中δ:

δ(0,a)＝1， δ(1,a)＝3

δ(2,a)＝1， δ(3,a)＝3

δ(0,b)＝2， δ(1,b)＝2

δ(2,b)＝3， δ(3,b)＝3

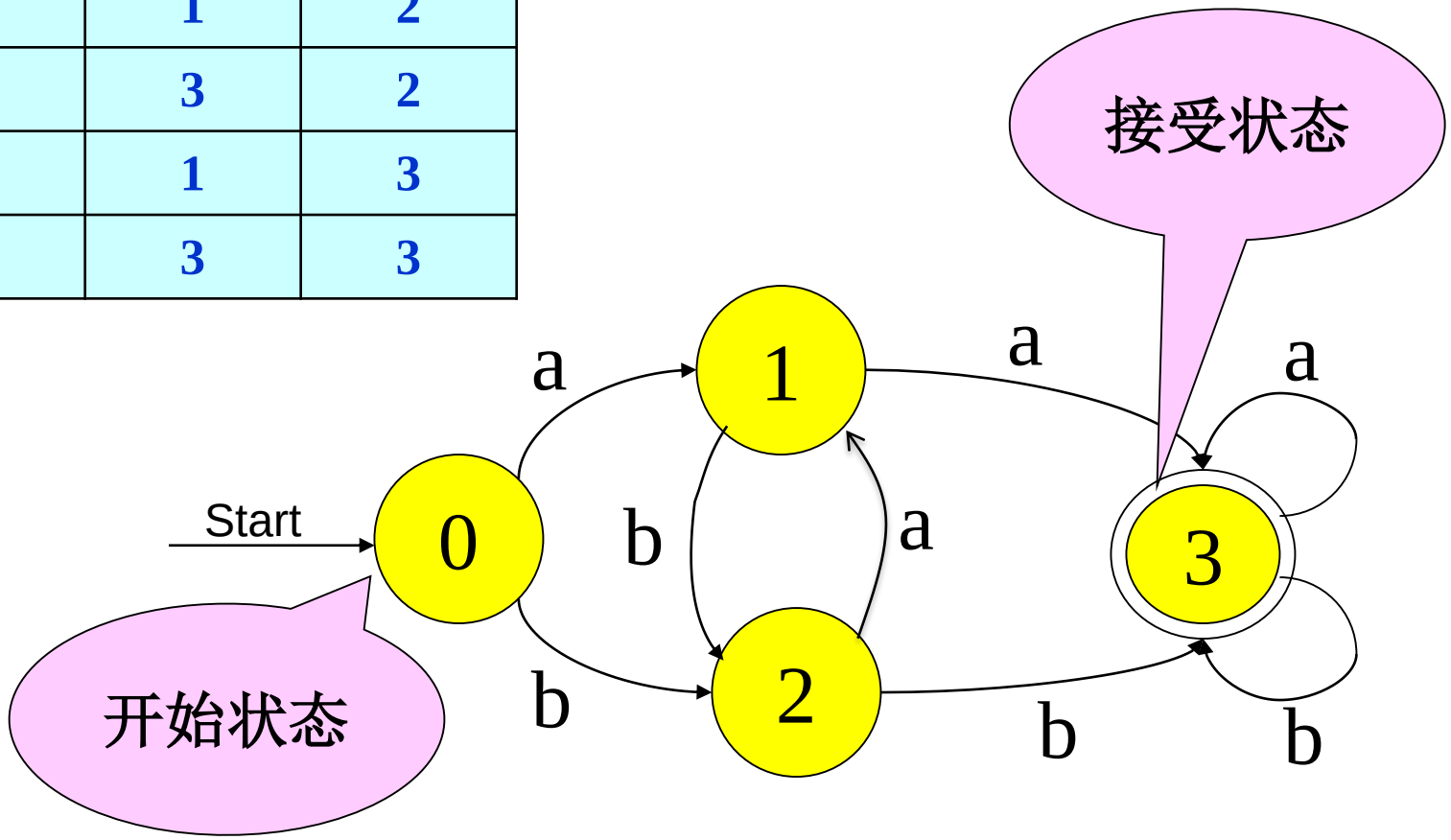
可以使用转移矩阵来表示（易存储）

状态 \ 输入字符	a	b
0	1	2
1	3	2
2	1	3
3	3	3

所谓确定的自动机，其确定性表现在状态转换函数是单值函数！

使用状态转换图来表示

状态 \ 输入	a	b
0	1	2
1	3	2
2	1	3
3	3	3



DFA M 接受的语言

如果对所有 $w \in \Sigma^*$，以下述方式递归地扩展δ的定义

$$\delta(q, \varepsilon) = q$$
$$\delta(q, wa) = \delta(\delta(q, w), a)$$

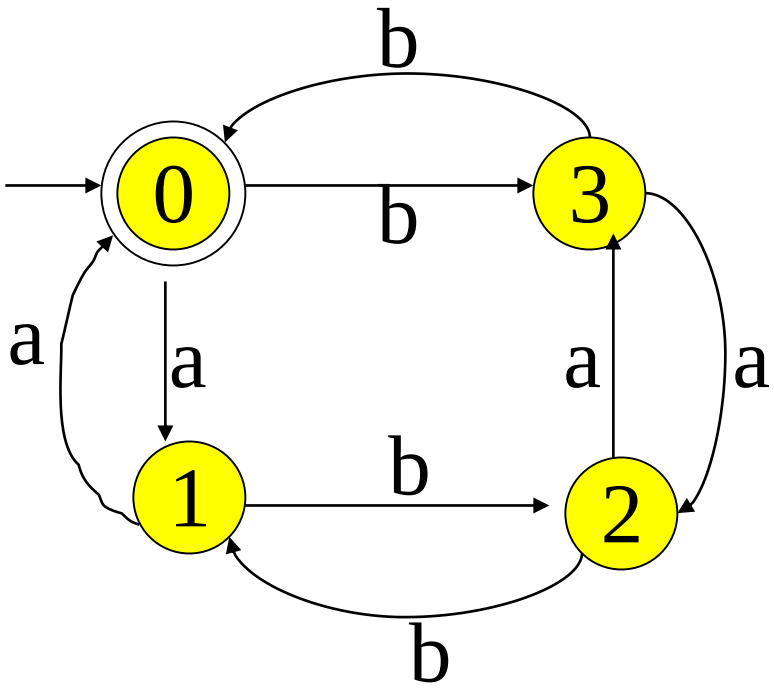
对任何 $a \in \Sigma, q \in Q$，则有

$$L(M) = \{w \mid w \in \Sigma^*，若存在 q \in F，使 \delta(q_0, w) = q\}$$

对于前面例子中的 DFA M 和 w = baa，有

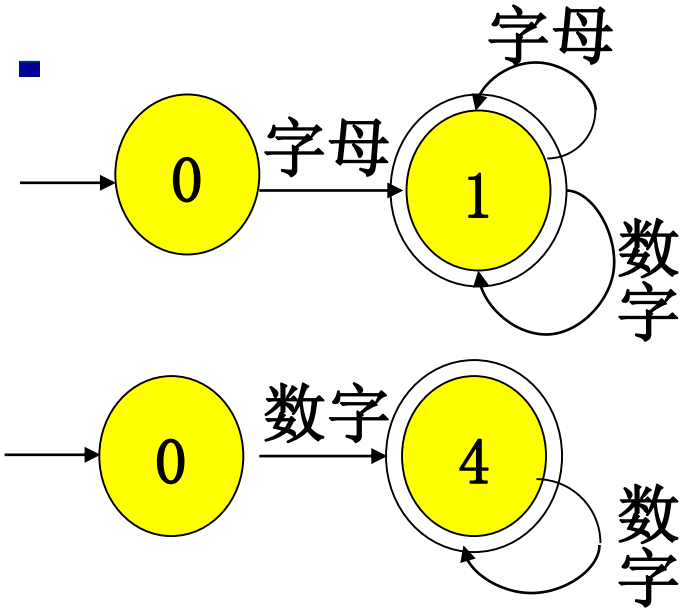
$$\delta(0, baa) = \delta(2, aa) = \delta(1, a) = 3$$

从状态转换图看，从开始状态出发，沿任一条路径到达接受状态，这条路径上的弧上的标记符号连接起来构成的符号串被DFA M接受。



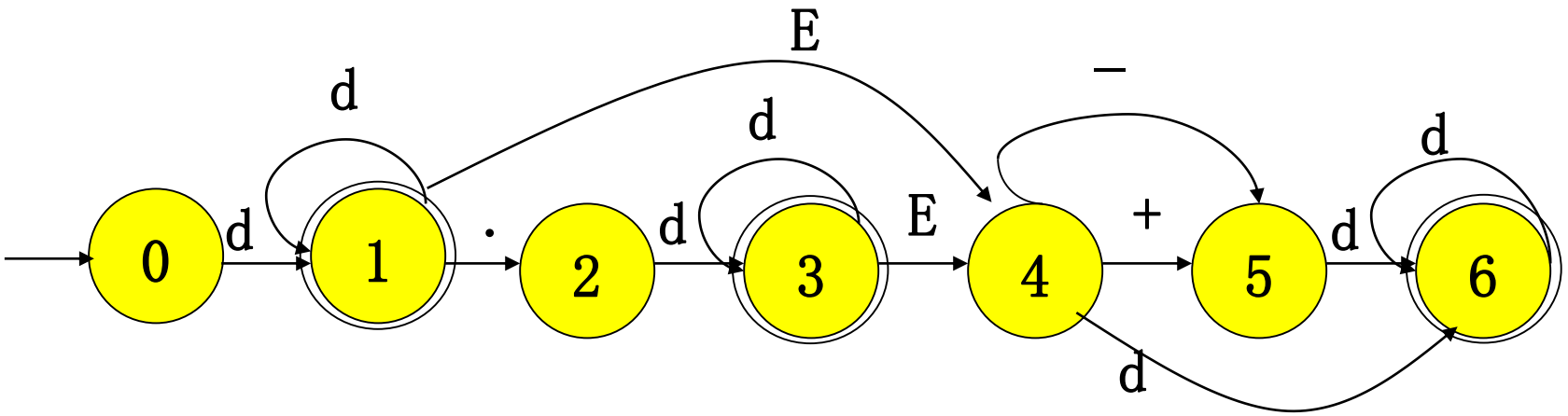
偶数个a，偶数个b的{a,b}串集合

DFA示例-1



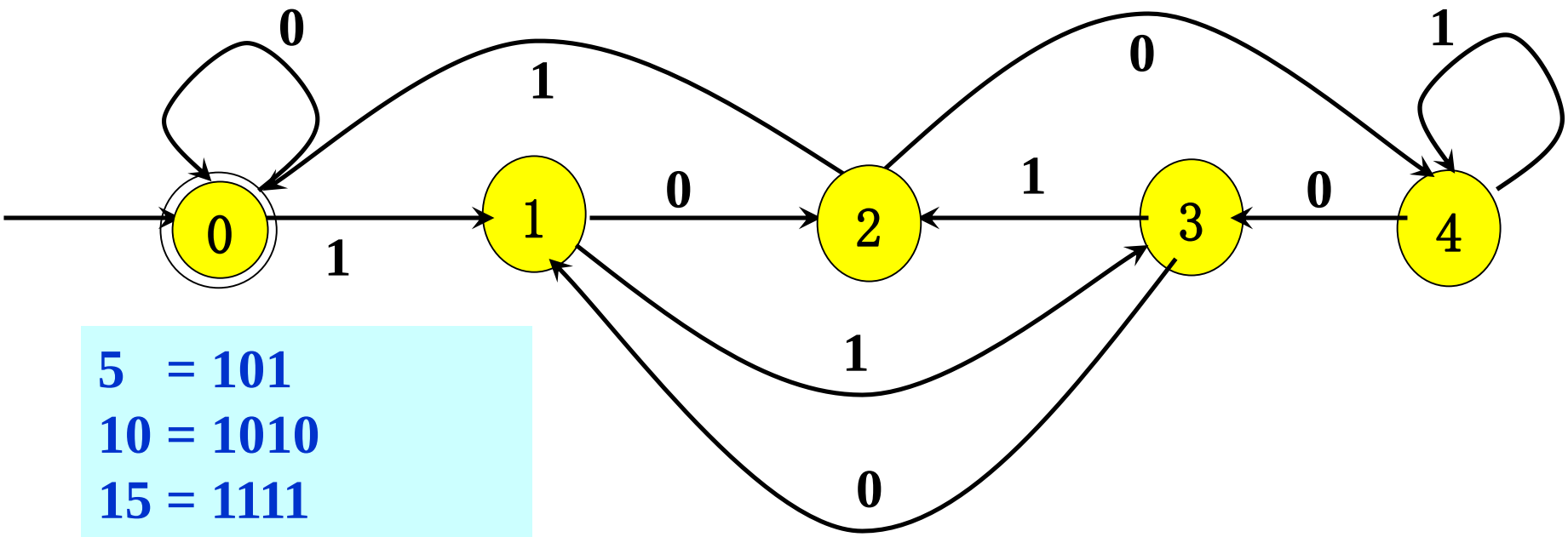
Pascal 标识符

Pascal整数和实数



DFA示例-2

识别Σ ={0,1}上能被能5整除的二进制数



5 = 101
10 = 1010
15 = 1111
65 = 1000001

Quiz

- 画一个DFA，表示所有能被32整除的二进制数

- 画一个DFA，表示所有除32余1的二进制数

非确定的带ε_转移的有限自动机NFA

定义：非确定的带ε_转移的有限自动机NFA M是一个五元组

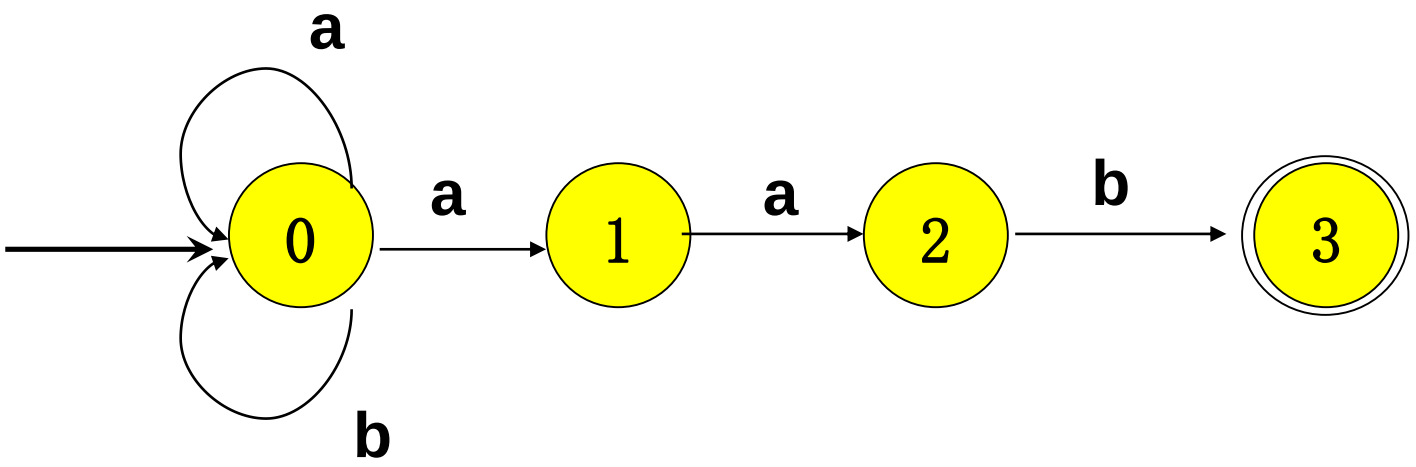
$$M=(\Sigma, Q, q_0, F, \delta)$$

其中Σ, Q, q_0, F的意义和DFA的定义一样，而δ是一个从$Q\times(\Sigma\cup\{\varepsilon\})$到Q的子集的映射，即

$$\delta: Q\times(\Sigma\cup\{\varepsilon\}) \rightarrow 2^Q$$

和DFA类似，NFA M也可以用状态转换图表示，也可以定义NFA M接受的语言

NFA示例-1

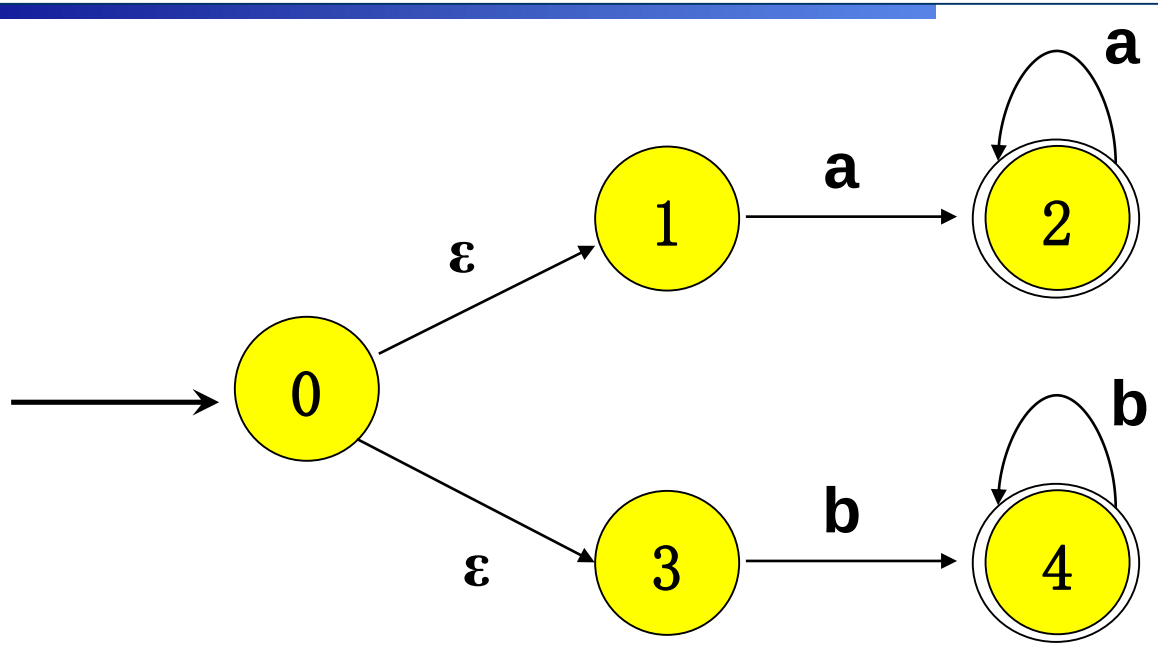


NFA 表示的语言：

(a|b)*aab

可不可以用DFA来表示？

NFA示例-2



NFA 表示的语言：

aa*|bb*

可不可以用DFA来表示？

关于NFA的说明

- NFA接受的字符串和语言
  - 如果在NFA中存在一个从开始状态到接受状态的路径，该路径上的符号序列构成的字符串是w，那么称该NFA可以接受字符串w
    - 一个字符串在NFA中可能对应不同的接受路径
    - NFA接受的字符串可能存在其他不能接受的路径
    - 如果在某个状态对于输入字符a不存在可用的转移动作，那么不能通过该路径接受当前的字符串
  - 一个NFA M接受的所有字符串的集合构成该NFA所接受的语言L(M)
- DFA是NFA的一种特例
  - DFA的表达能力与NFA是等价的

例 题 1

- 给出接受 (a|b)*a(a|b)(a|b) 的DFA。

例 题 2

- 指出下面的正则表达式描述的语言，并画出接受该语言的最简DFA的状态转换图。

  (1|01)* 0*

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25日交作业
- Ex. 3.6.2 (本 Ex. 3.5.2) (1,2,3,6,9小题)