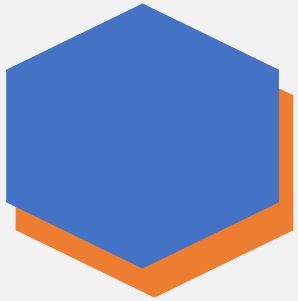


# 编译技术

计算机学院计算机系

刘佰龙

liubailong@cumt.edu.cn

# 03 词法分析

# 3 词法分析

3.1 词法分析的功能

3.2 单词的描述工具

3.3 词法分析程序的设计与实现

3.4 词法分析程序的自动生成

# 词法分析程序的设计与实现

词法规则 → 正规表达式 → 状态转换图（最小DFA）

1.构造识别单词的状态转换图

(1)对程序语言的单词按种类分别构造对应的状态转换图.

(2)对各类转换图合并，构成一个能识别语言所有单词的状态转换图.

2.编程实现状态转换图

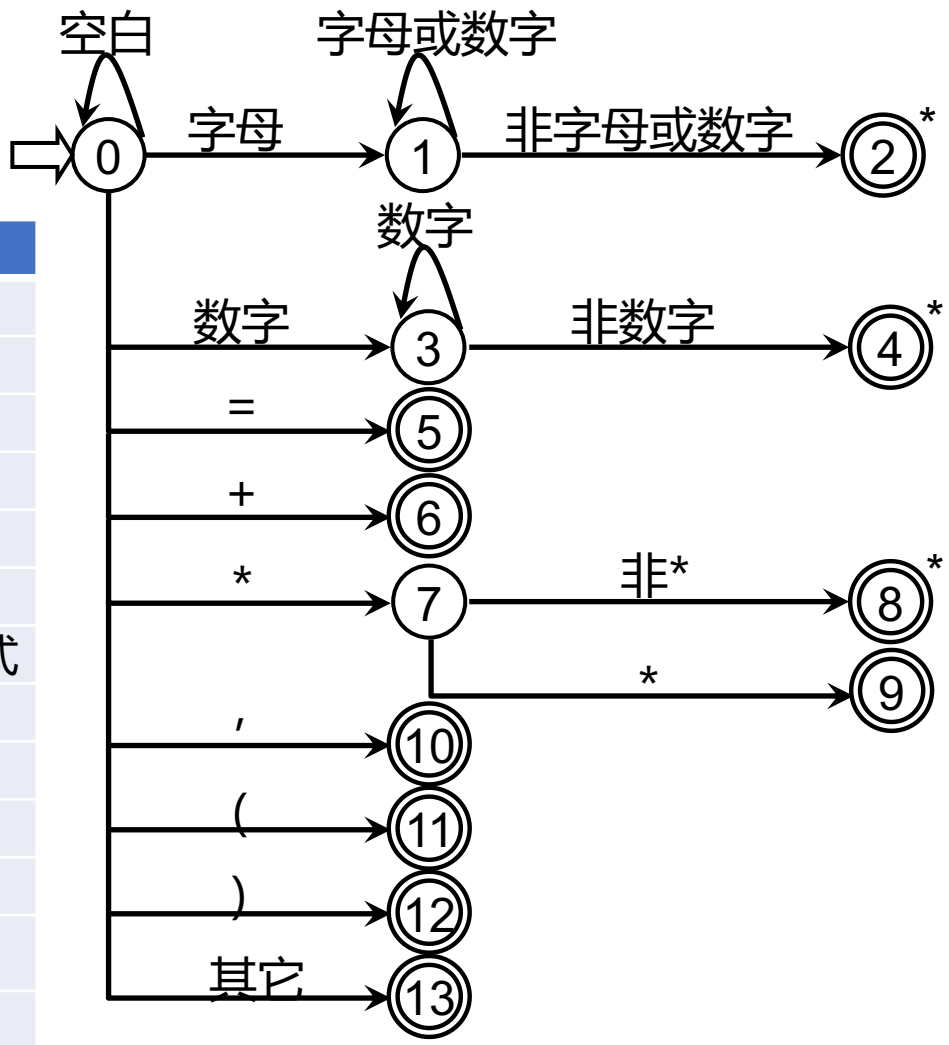
# 词法分析器的设计示例

- 一个简单的程序设计语言
  - 单词表

| 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 |
| 常数(数) | 7 | $INT | 标准二进制形式 |
| = | 8 | $ASSIGN | - |
| + | 9 | $PLUS | - |
| * | 10 | $STAR | - |
| ** | 11 | $POWER | - |
| , | 12 | $COMMA | - |
| ( | 13 | $LPAR | - |
| ) | 14 | $RPAR | - |

# 词法分析器的设计示例

- 设计状态转换图




| 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 |
| 常数(数) | 7 | $INT | 标准二进制形式 |
| = | 8 | $ASSIGN | - |
| + | 9 | $PLUS | - |
| * | 10 | $STAR | - |
| ** | 11 | $POWER | - |
| , | 12 | $COMMA | - |
| ( | 13 | $LPAR | - |
| ) | 14 | $RPAR | - |

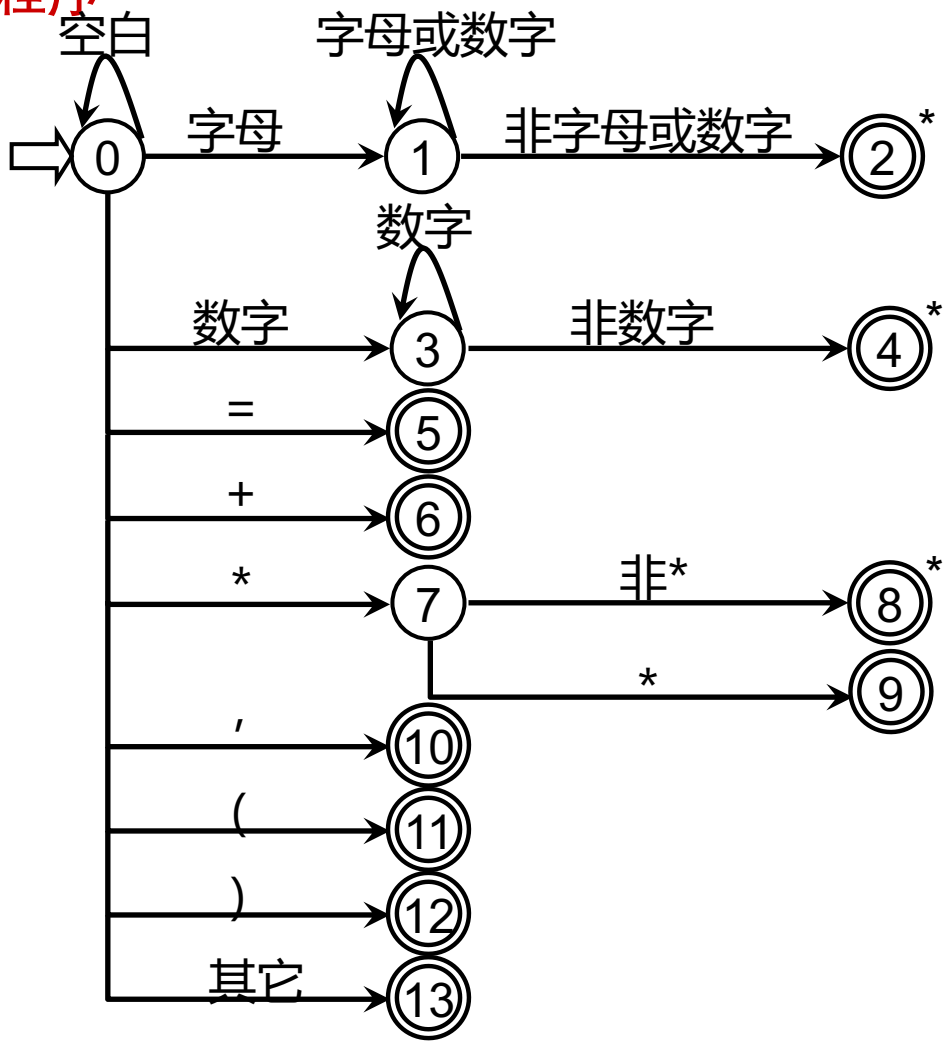
# 几点限制——不必使用超前搜索

- 所有基本字都是保留字;用户不能用它们作自己的标识符
- 基本字作为特殊的标识符来处理，使用保留字表
- 如果基本字、标识符和常数(或标号)之间没有确定的运算符或界符作间隔，则必须使用一个空白符作间隔

  DO99K=1，10
  要写成　DO 99 K=1，10

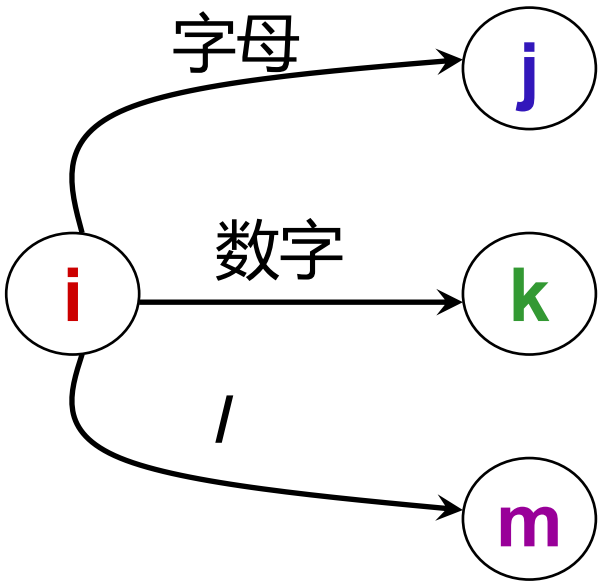
# 词法分析器的设计示例

- 状态转换图的代码实现
  - 每个状态结点对应一小段程序



# 状态转换图的实现

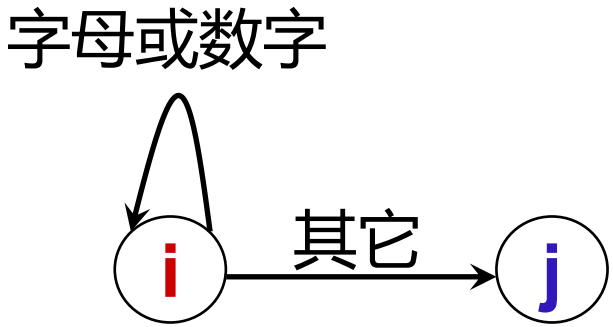
- 不含回路的分叉结点
  - 可用一个CASE语句或一组IF-THEN-ELSE语句实现




```
GetChar( );
if (IsLetter( ))
    {...状态j的对应程序段...;}
else if (IsDigit( ))
    {...状态k的对应程序段...;}
else if (ch='/')
    {...状态m的对应程序段...;}
else
    {...错误处理...;}
```

# 状态转换图的实现

- 含回路的状态结点
  - 对应一段由WHILE结构和IF语句构成的程序




```
GetChar( );
while (IsLetter( )or IsDigit( ))
      GetChar( );
...状态j的对应程序段...
```

# 状态转换图的实现

- 终态结点
  - 表示识别出某种单词符号，对应返回语句


RETURN (C，VAL)

其中，C为单词种别，VAL为单词自身值

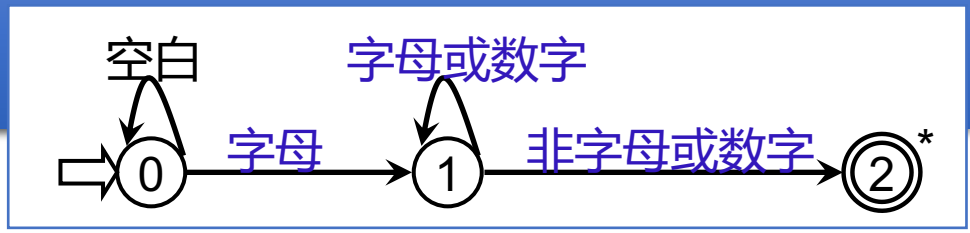
# 状态转换图的实现

- 全局变量与过程
  - **ch** 字符变量，存放最新读入的源程序字符
  - **strToken** 字符数组，存放构成单词符号的字符串
  - **GetChar** 子程序过程，把下一个字符读入到 **ch** 中
  - **GetBC** 子程序过程，跳过空白符，直至 **ch** 中读入一非空白符
  - **Concat** 子程序，把**ch**中的字符连接到 **strToken**

# 状态转换图的实现

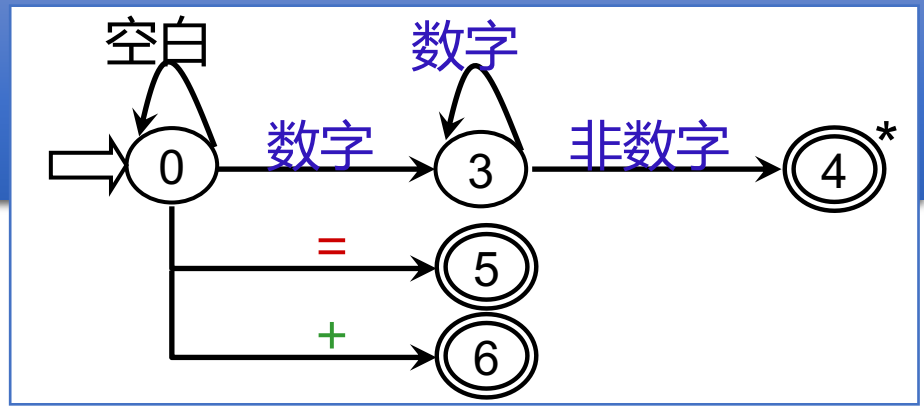
- 全局变量与过程
  - **IsLetter**和 **IsDisgital** 布尔函数，判断ch中字符是否为字母和数字
  - **Reserve** 整型函数，对于 strToken 中的字符串查找保留字表，若它是保留字则给出它的编码，否则回送0
  - **Retract** 子程序，把搜索指针回调一个字符位置
  - **InsertId** 整型函数，将strToken中的标识符插入符号表，返回符号表指针
  - **InsertConst** 整型函数过程，将strToken中的常数插入常数表，返回常数表指针

# 词法分析器的实现



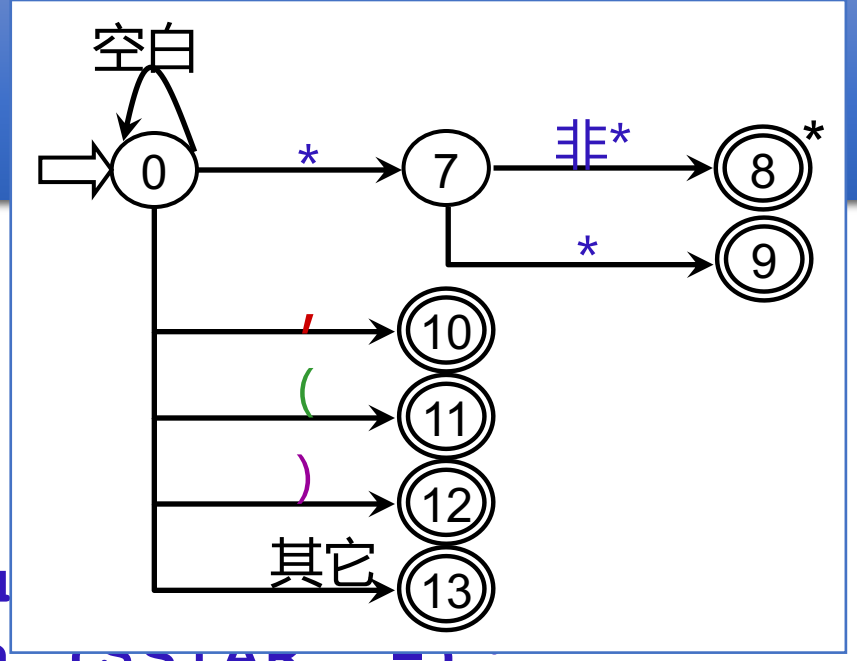

```
int code, value;
strToken := " ";      /*置strToken为空串*/
GetChar(); GetBC();
if (IsLetter())
begin
      while (IsLetter() or IsDigit())
      begin
            Concat(); GetChar();
      end
      Retract();
      code := Reserve();
      if (code = 0)
      begin
            value := InsertId(strToken);
            return ($ID, value);
      end
      else
            return (code, -);
end
```

# 词法分析器的实现




```
else if (IsDigit())
begin
      while (IsDigit())
      begin
            Concat( ); GetChar( );
      end
      Retract();
      value := InsertConst(strToken);
      return($INT, value);
end
else if (ch ='=') return ($ASSIGN, -);
else if (ch ='+') return ($PLUS, -);
```

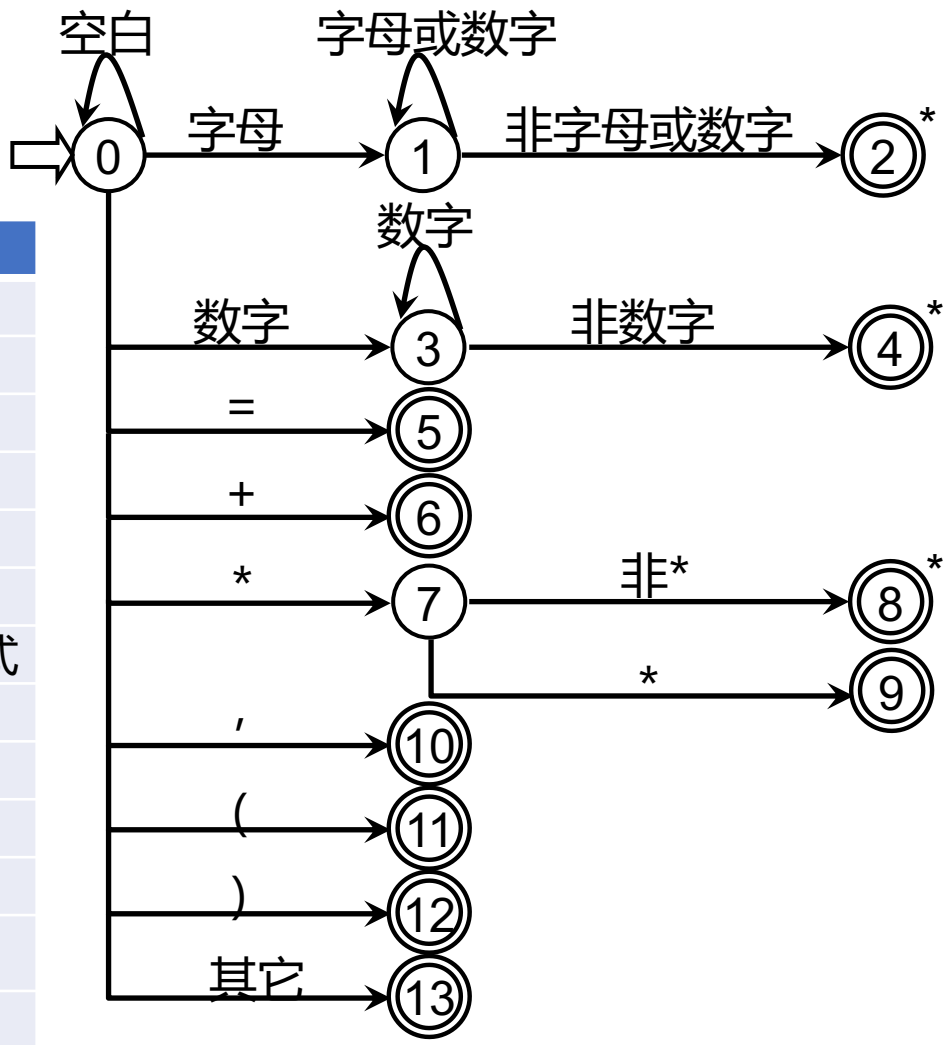
# 词法分析器的实现




```
else if (ch ='*')
begin
      GetChar();
      if (ch ='*') retu
      Retract(); return ($STAR, -);
end
else if (ch =',') return ($COMMA, -);
else if (ch ='(') return ($LPAR, -);
else if (ch =')') return ($RPAR, -);
else ProcError( );        /* 错误处理*/
```

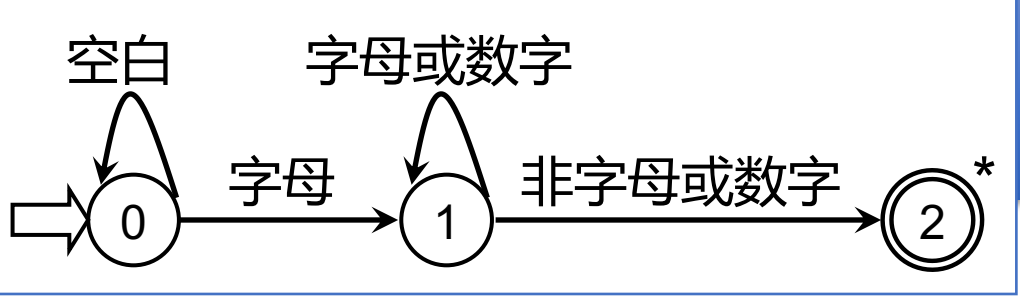
# 词法分析器的设计示例

- 设计状态转换图




| 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 |
| 常数(数) | 7 | $INT | 标准二进制形式 |
| = | 8 | $ASSIGN | - |
| + | 9 | $PLUS | - |
| * | 10 | $STAR | - |
| ** | 11 | $POWER | - |
| , | 12 | $COMMA | - |
| ( | 13 | $LPAR | - |
| ) | 14 | $RPAR | - |

# 将状态图的代码一般化




- 变量curState用于保存现有的状态
- 用二维数组表示状态图：stateTrans[state][ch]

只是个框架，还有很多细节需要考虑！是否有自动的方法产生词法分析程序？

```
curState = 初态
GetChar();
while( stateTrans[curState][ch]有定义){
    //存在后继状态，读入、拼接
    Concat();
    //转换入下一状态，读入下一字符
    curState= stateTrans[curState][ch];
    if curState是终态 then 返回strToken中的单词
    GetChar( );
}
```

# 词法分析器设计过程

词法规则 → 正规表达式 → 状态转换图（最小DFA）

→ 合并状态转换图

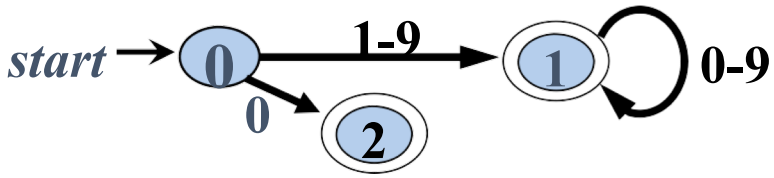
合并状态转换图 → 编程实现状态转换图

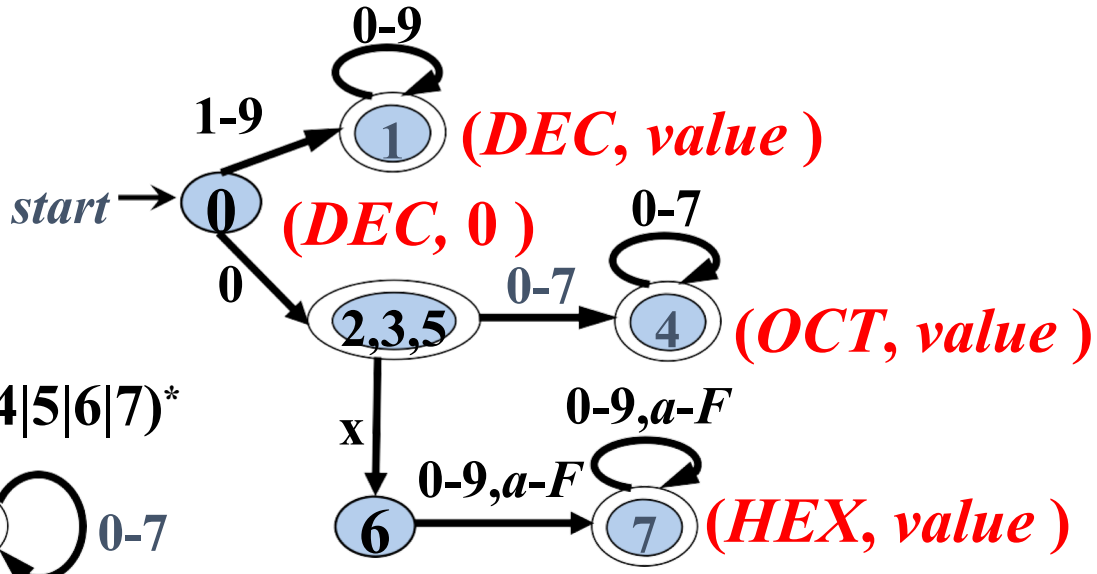
文法、正规表达式、有限自动机

有限自动机(最小化？)

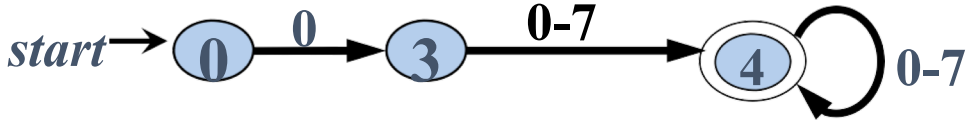
# 识别各进制无符号整数的DFA

*DEC* → (1|...|9)(0|...|9)* |0

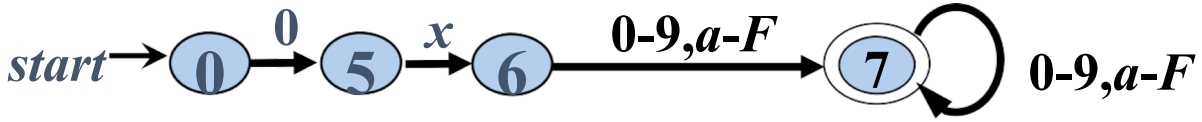


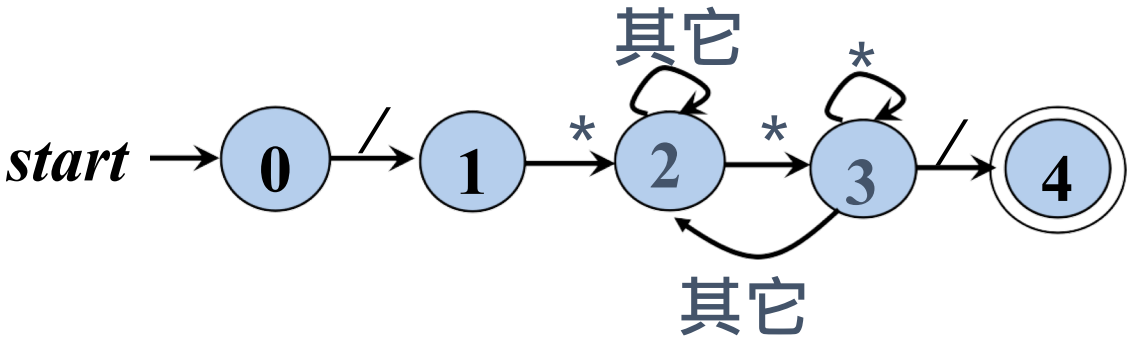

*OCT* → 0(0|1|2|3|4|5|6|7)(0|1|2|3|4|5|6|7)*



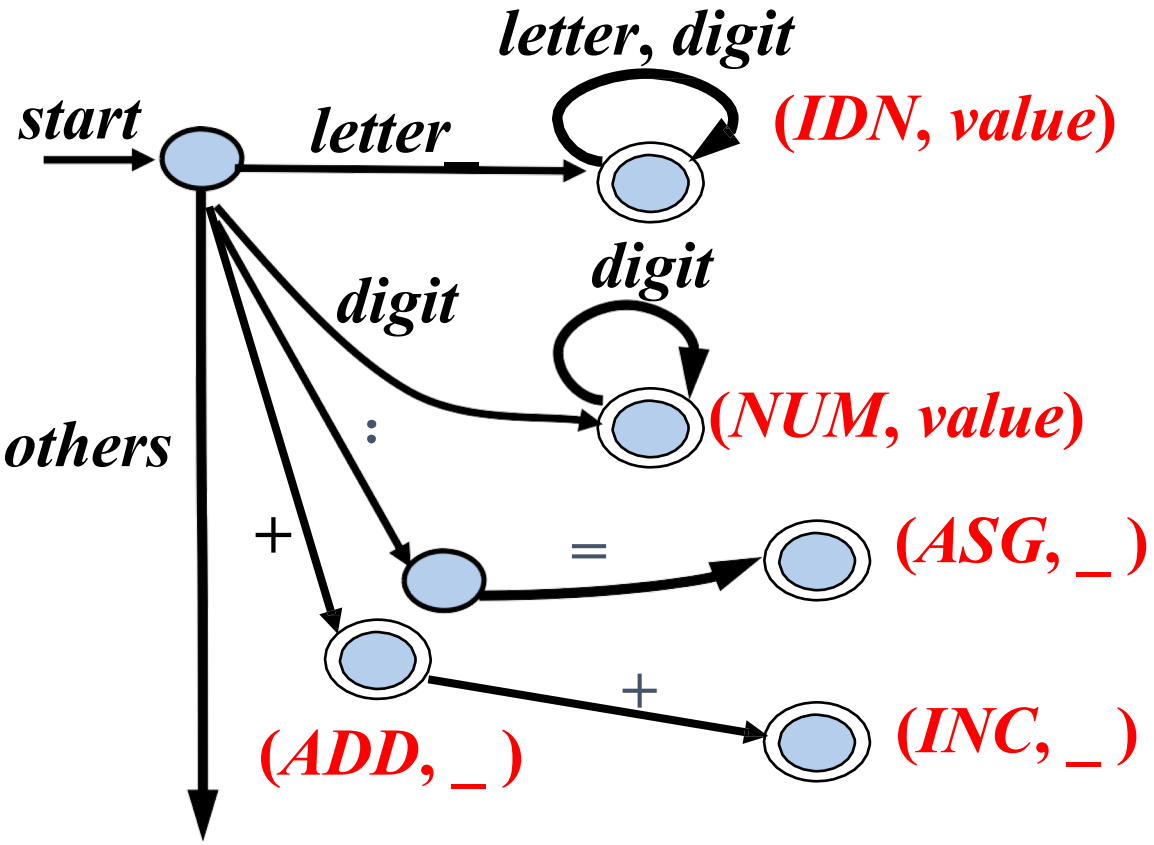

*HEX* → 0*x*(0|1|...|9|*a*|...|*f*|*A*|…|*F*)(0|...|9|*a*|...|*f* |*A*|…|*F*)*



# 识别注释的 DFA



# 识别 Token的 DFA



# 词法分析阶段的错误处理

## 词法分析阶段可检测错误的类型

- 单词拼写错误
  - 例：*int i* = 0*x*3*G*; *float j* =1.05*e*;
- 非法字符
  - 例：~ @

## 词法错误检测

- 如果当前状态与当前输入符号在转换表对应项中的信息为空，而当前状态又不是终止状态，则调用错误处理程序

# 错误处理

- 查找已扫描字符串中最后一个对应于某终态的字符
- 如果找到了，将该字符与其前面的字符识别成一个单词。 然后将输入指针退回到该字符，扫描器重新回到初始状态，继续识别下一个单词
- 如果没找到，则确定出错，采用错误恢复策略

# 错误恢复策略

最简单的错误恢复策略 :“ 恐慌模式 (*panic mode*)” 恢 复

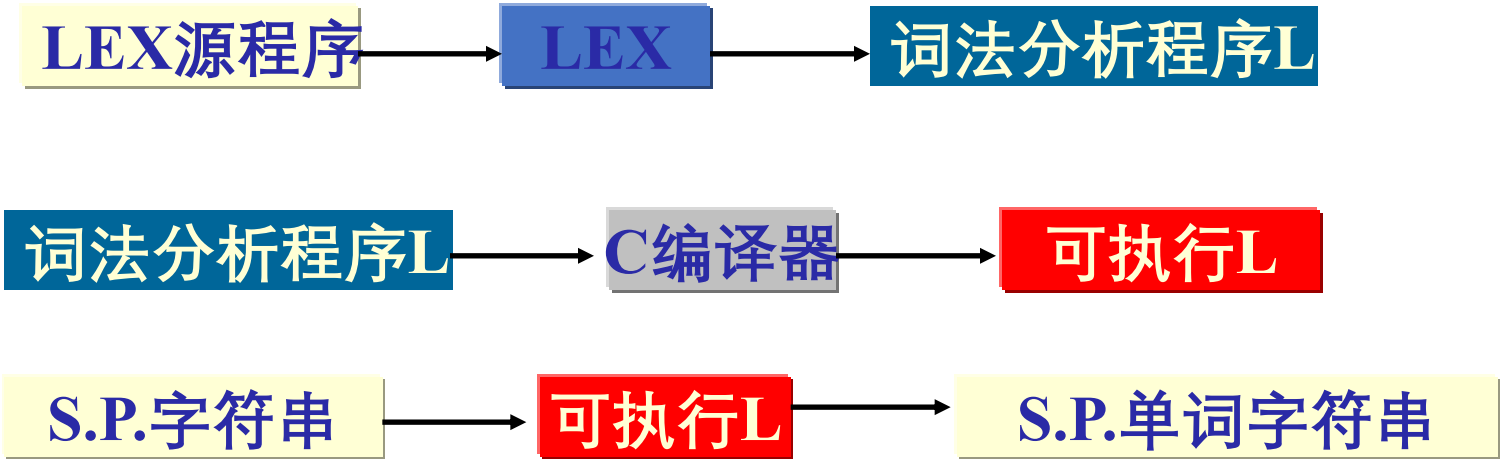
从剩余的输入中不断删除字符，直到词法分析器能够在剩余输入的开头发现一个正确的字符为止

# 3 词法分析

# 词法分析程序的自动生成器—LEX（LEXICAL Analyzer Generator）

LEX是1972年在贝尔实验室的UNIX上首先实现
FLEX是1984年GNU工程推出，是LEX的扩充，并兼容LEX

原理：



# LEX源程序

一个LEX源程序主要由三个部分组成:

1. 辅助定义式(说明部分)
2. 识别规则
3. 用户子程序

各部分之间用%%隔开，同时%%在最左边.

# 辅助定义式

## 如下形式的LEX语句：

$D_1$ $R_1$
$D_2$ $R_2$
：
：
$D_n$ $R_n$

其中：$R_1, R_2$，……，$R_n$ 为正则表达式。
$D_1, D_2$，……，$D_n$ 为正则表达式名字，称简名字

# 例子

例：C++标识符

letter　A|B|……|Z|_

digit　0|1| ……|9

iden　letter(letter|digit)*

# 识别规则

## 一串如下形式的LEX语句：

$$
\begin{array}{ll}
P_1 & \{A_1\} \\
P_2 & \{A_2\} \\
\quad\vdots & \\
\quad\vdots & \\
P_m & \{A_m\}
\end{array}
$$

$P_i$：定义在$\Sigma \cup \{D_1, D_2, \cdots D_n\}$上的正规表达式，也称词形。

$\{A_i\}$：$A_i$为语句序列，它指出，在识别出词形为$P_i$的单词以后，词法分析器所应作的动作。

其基本动作是返回单词的类别编码和其属性。

# 用户子程序

**定义模式动作需要的函数或包括主函数**

**在这三部分中一和三为可选项，但是二是必须的**

除辅助定义之外定义的部分必须用符号%{和%}括起来，其内容为：

- 所对应语言的库文件
- 外部变量
- 外部函数和函数原型

# 识别某语言单词符号的LEX源程序举例

```
AUXILIARY    DEFINITIONS   /*辅助定义*/
    letter   [A—z]
    digit    [0—9]
%%
RECOGNIYION RULES          /*识别规则*/
 “BEGIN”                   {RETURN(1,—) }
 “END”                     {RETURN(2,—) }
 “FOR”                     {RETURN(3,—) }
```

RETURN是LEX过程，该过程将单词传给语法分析程序

RETURN（C，LEXVAL）

其中C为单词类别编码

LEXVAL：

标识符：TOKEN（字符数组）

整常数：DTB（数值转换函数，将TOKEN中的数字串转换二进制值）

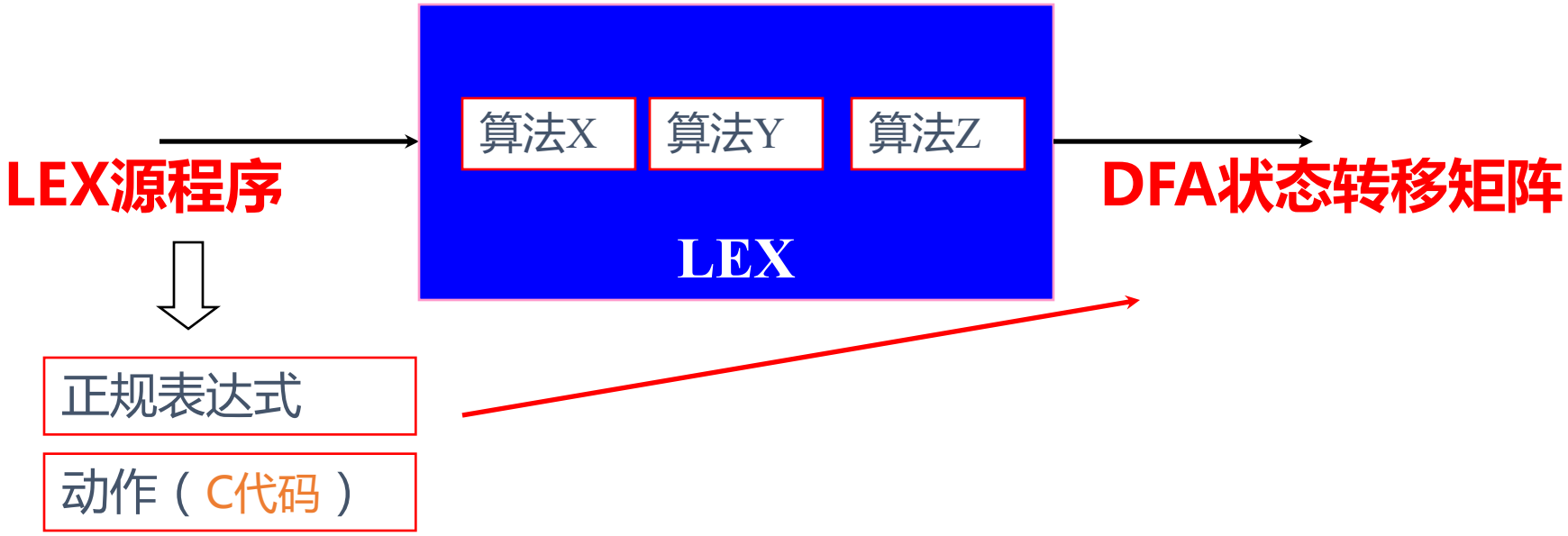
其他单词：无定义。

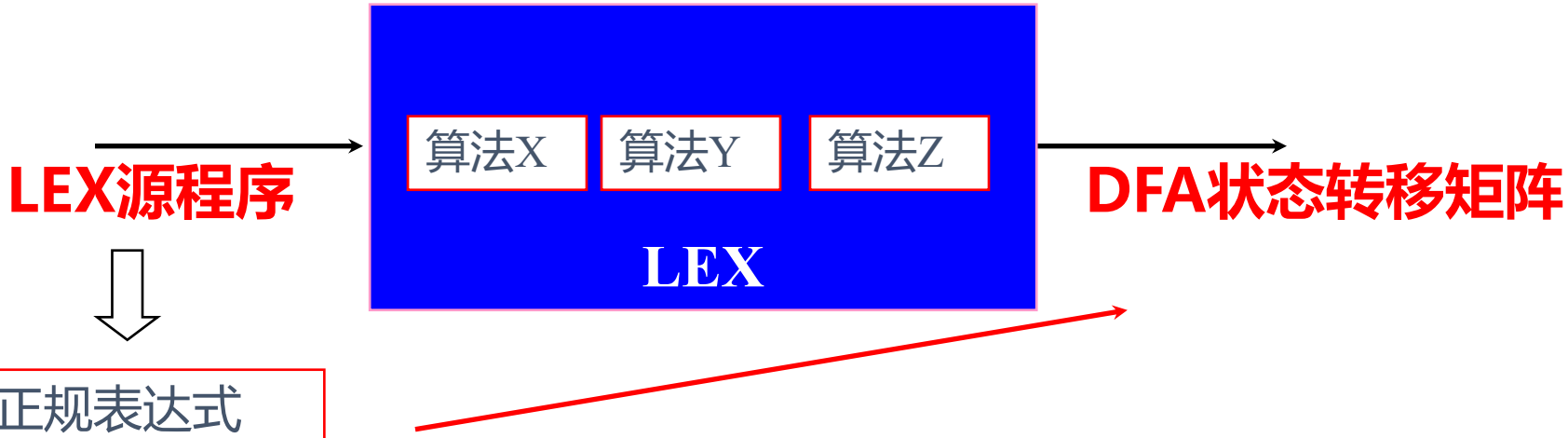
| | |
|---|---|
| “DO” | {RETURN(4,–) } |
| “IF” | {RETURN(5,–) } |
| “THEN” | {RETURN(6,–) } |
| “ELSE ” | {RETURN(7,–) } |
| {letter}({letter} \| {digit})* | {RETURN(8,TOKEN) } |
| {digit}({digit})* | {RETURN(9,DTB ) } |
| “:” | {RETURN(10,–) } |
| “+” | {RETURN(11,–) } |
| “*” | {RETURN(12,–) } |

| | |
|---|---|
| “;” | {RETURN(13,–) } |
| “ ( ” | {RETURN(14,–) } |
| “ ) ” | {RETURN(15,–) } |
| “:=” | {RETURN(16,–) } |
| “=” | {RETURN(17,–) } |

# LEX的实现

**LEX的功能是根据LEX源程序构造一个词法分析程序，该词法分析器实质上是一个有限自动机。**






LEX生成的词法分析程序有两部分组成:

词法分析程序

| 状态转换矩阵(DFA) |
| --- |
| 控制执行程序 |

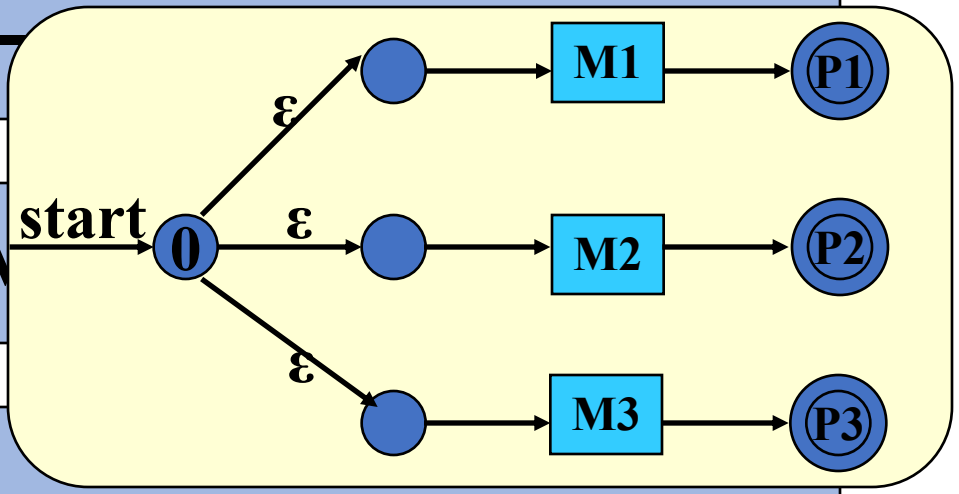
**∴LEX的功能是根据LEX源程序生成状态转换矩阵和控制程序**

# LEX的处理过程：

· 扫描每条识别规则Pi构造一

¨将各条规则的有穷自动机N

·¨确定化　　NFA⇒DFA




生成该DFA的状态转换矩阵和控制执行程序

# 小结

- 正规式⇔有限自动机⇔正规文法（右线性）
- 词法规则→有限自动机→确定的有限自动机→编程实现DFA

词法分析中为了描述一类单词，使用了正则表达式(正规文法，有限自动机)
为了描述一类语法，使用上下文无关文法。