



第3章 词法分析



重庆大学 葛亮

基础知识：PASCAL、C语言、正规表达式

正规文法、有限自动机

知识点：词法分析器的作用、地位

记号、模式

词法分析器的状态转换图

词法分析

简介

- 3.1 词法分析程序与语法分析程序的关系
- 3.2 词法分析程序的输入与输出
- 3.3 记号的描述和识别
- 3.4 词法分析程序的设计与实现
- 3.5 软件工具LEX

小结

简介

- 词法分析任务由词法分析程序完成

- 本章内容安排

讨论手工设计并实现词法分析程序的方法和步骤

- ◆ 词法分析程序的作用
- ◆ 词法分析程序的地位
- ◆ 源程序的输入与词法分析程序的输出
- ◆ 单词符号的描述及识别
- ◆ 词法分析程序的设计与实现

词法分析程序自动生成工具LEX简介

词法分析程序的作用

■ 词法分析程序的作用：

扫描源程序字符流

按照源语言的词法规则识别出各类单词符号

产生用于语法分析的记号序列

词法检查

创建符号表(需要的话)

把识别出来的标识符插入符号表中

与用户接口的一些任务：

- 跳过源程序中的注释和空白
- 把错误信息和源程序联系起来

3.1 词法分析程序与语法分析程序的关系

■ 词法分析程序与语法分析程序之间的三种关系

- ◆ 词法分析程序作为独立的一遍
- ◆ 词法分析程序作为语法分析程序的子程序
- ◆ 词法分析程序与语法分析程序作为协同程序

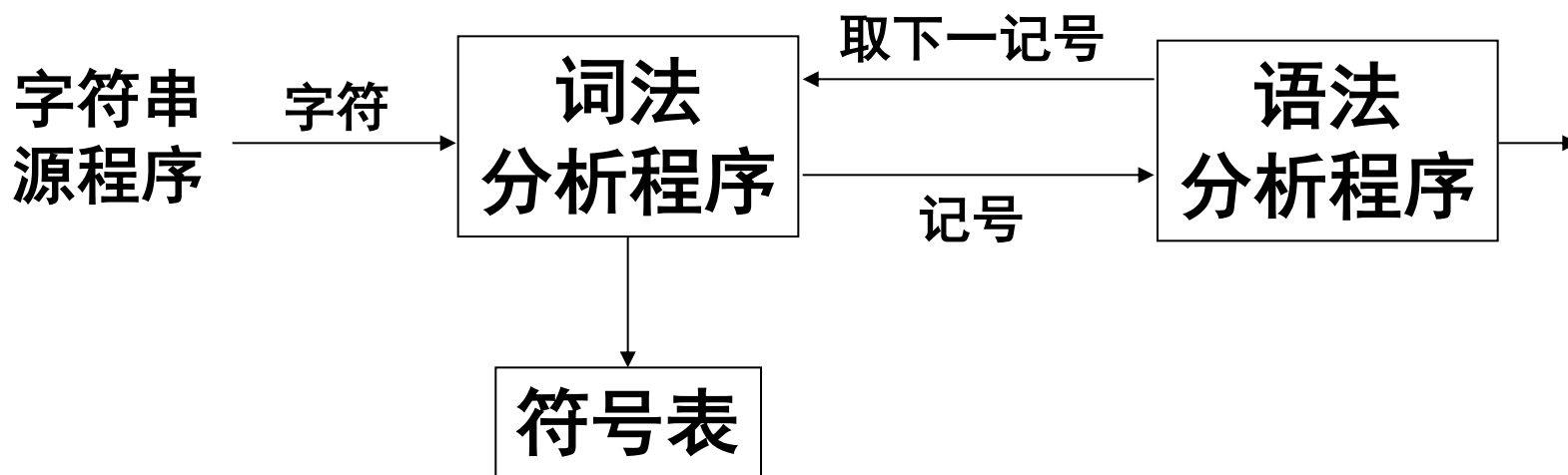
词法分析程序作为独立的一遍



■ 输出放入一个中间文件

磁盘文件
内存文件

词法分析程序作为语法分析程序的子程序



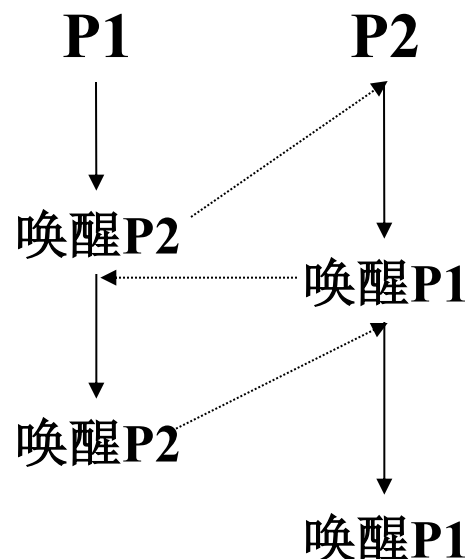
- 避免了中间文件
- 省去了取送符号的工作
- 有利于提高编译程序的效率

词法分析程序与语法分析程序作为协同程序

■ 协同程序：

如果两个或两个以上的程序，它们之间交叉地执行，这些程序称为协同程序。

词法分析程序与语法分析程序在同一遍中，以生产者和消费者的关系同步运行。



分离词法分析程序的好处

■ 可以简化设计

- ◆ 词法程序很容易识别并去除空格、注释，使语法分析程序致力于语法分析，结构清晰，易于实现。

■ 可以改进编译程序的效率

- ◆ 利用专门的读字符和处理记号的技术构造更有效的词法分析程序。

■ 可以加强编译程序的可移植性

- ◆ 在词法分析程序中处理特殊的或非标准的符号。

3.2 词法分析程序的输入与输出

- 一、词法分析程序的实现方法
- 二、设置缓冲区的必要性
- 三、配对缓冲区
- 四、词法分析程序的输出

一、词法分析程序的实现方法

■ 利用词法分析程序自动生成器

- ◆ 从基于正规表达式的规范说明自动生成词法分析程序。
- ◆ 生成器提供用于源程序字符流读入和缓冲的若干子程序

■ 利用传统的系统程序设计语言来编写

- ◆ 利用该语言所具有的输入/输出能力来处理读入操作

■ 利用汇编语言来编写

- ◆ 直接管理源程序字符流的读入

二、设置缓冲区的必要性

- 为了得到某一个单词符号的确切性质，需要超前扫描若干个字符。

- 有合法的FORTRAN语句：

DO 99 K = 1, 10



DO 99 K = 1. 10



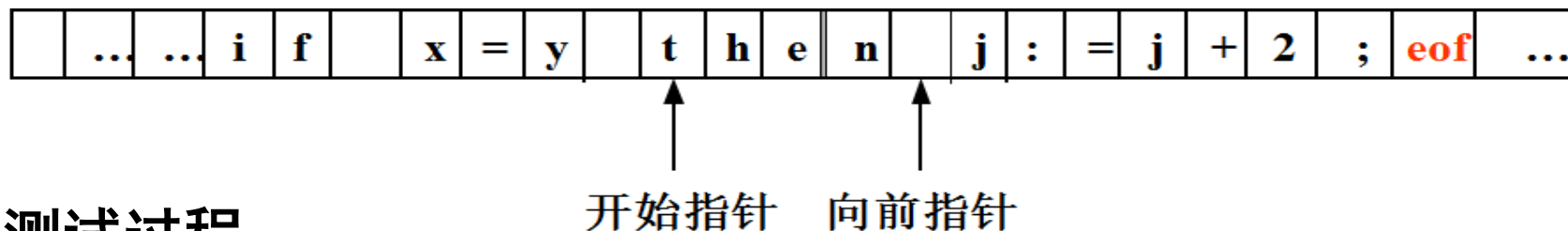
- 为了区别这两个语句，必须超前扫描到等号后的第一个分界符处。

```
do 100 counter=initial,endnumber,increment
...      ! counter计数器，从初始值开始，每循环一次加增量大小。
...      ! endnumber计数器的终止，当计数器大于终止数时，循环停止
...      ! increment 增量数，每次循环，给计数器加一个增量数
100      continue
```

变量名称是以字母开头再加上1到5位字母或数字构成

三、配对缓冲区

- 把一个缓冲器分为大小相同的两半，每半各含N个字符，一般N=1KB或4KB。



- 测试过程：

IF (向前指针在左半区的终点) {
 读入字符串，填充右半区;
 向前指针前移一个位置;
}

ELSE IF (向前指针在右半区的终点) {
 读入字符串，填充左半区;
 向前指针移到缓冲区的开始位置;
}

ELSE 向前指针前移一个位置;

每半区带有结束标记的缓冲器



↑ ↑
开始指针 向前指针

■ 测试过程:

向前指针前移一个位置;

IF (向前指针指向 **eof**) {

IF (向前指针在左半区的终点) {

 读入字符串, 填充右半区;

 向前指针前移一个位置;

 };

ELSE IF (向前指针在右半区的终点) {

 读入字符串, 填充左半区;

 向前指针指向缓冲区的开始位置;

 };

ELSE 终止词法分析;

}

四、词法分析程序的输出——记号

■ 记号、模式和单词

- ◆ 记号：是指某一类单词符号的种别编码，如标识符的记号为**id**，数的记号为**num**等。
- ◆ 模式：是指某一类单词符号的构词规则，如标识符的模式是“由字母开头的字母数字串”。
- ◆ 单词：是指某一类单词符号的一个特例，如**position**是标识符。

■ 记号的属性

记号的属性

- 词法分析程序在识别出一个记号后，要把与之有关的信息作为它的属性保留下来。
- 记号影响语法分析的决策，属性影响记号的翻译。
- 在词法分析阶段，对记号只能确定一种属性
 - ◆ 标识符：单词在符号表中入口的指针
 - ◆ 常数：它所表示的值
 - ◆ 关键字：（一符一种、或一类一种）
 - ◆ 运算符：（一符一种、或一类一种）
 - ◆ 分界符：（一符一种、或一类一种）

total:=total+rate*4 的词法分析结果

<id, 指向标识符**total**在符号表中的入口的指针>

<assign_op, - >

<id, 指向标识符**total**在符号表中的入口的指针>

<plus_op, - >

<id, 指向标识符**rate**在符号表中的入口的指针>

<mul_op, - >

<num, 整数值**4**>

3.3 记号的描述和识别

- 识别单词是按照记号的模式进行的，一种记号的模式匹配一类单词的集合。
 - ◆ 为设计词法程序，对模式要给出规范、系统的描述。
 - 正规表达式和正规文法是描述模式的重要工具。
 - ◆ 二者具有同等表达能力
 - ◆ 正规表达式：清晰、简洁
 - ◆ 正规文法：便于识别
-
- 一、词法与正规文法
 - 二、记号的文法
 - 三、状态转换图与记号的识别

一、词法与正规文法

- 把源语言的文法 G 分解为若干子文法：

G_0 、
 G_1 、 G_2 、 \dots 、 G_n

语法

词法

- 词法：描述语言的标识符、常数、运算符和标点符号等记号的文法
—— 正规文法
- 语法：借助于记号来描述语言结构的文法
—— 上下文无关文法

二、记号的文法

- 标识符
- 常数
 - ◆ 整数
 - ◆ 无符号数
- 运算符
- 分界符
- 关键字

标识符

- 假设标识符定义为“由字母打头的、由字母或数字组成的符号串”

- 描述标识符集合的正规表达式：

letter(letter|digit)*

- 表示标识符集合的正规定义式：

letter \rightarrow **A|B|...|Z|a|b|...|z**

digit \rightarrow **0|1|...|9**

id \rightarrow letter(letter|digit)*

把正规定义式转换为相应的正规文法

$$\begin{aligned} & (\text{letter} \mid \text{digit})^* \\ &= \varepsilon \mid (\text{letter} \mid \text{digit})^+ \\ &= \varepsilon \mid (\text{letter} \mid \text{digit})(\text{letter} \mid \text{digit})^* \\ &= \varepsilon \mid \text{letter}(\text{letter} \mid \text{digit})^* \mid \text{digit}(\text{letter} \mid \text{digit})^* \\ &= \varepsilon \mid (\text{A} \mid \dots \mid \text{Z} \mid \text{a} \mid \dots \mid \text{z})(\text{letter} \mid \text{digit})^* \\ &\quad \mid (0 \mid \dots \mid 9)(\text{letter} \mid \text{digit})^* \\ &= \varepsilon \mid \text{A}(\text{letter} \mid \text{digit})^* \mid \dots \mid \text{Z}(\text{letter} \mid \text{digit})^* \\ &\quad \mid \text{a}(\text{letter} \mid \text{digit})^* \mid \dots \mid \text{z}(\text{letter} \mid \text{digit})^* \\ &\quad \mid 0(\text{letter} \mid \text{digit})^* \mid \dots \mid 9(\text{letter} \mid \text{digit})^* \end{aligned}$$

标识符的正规文法

$id \rightarrow A\ rid \mid \dots \mid Z\ rid \mid a\ rid \mid \dots \mid z\ rid$

$rid \rightarrow \varepsilon \mid A\ rid \mid B\ rid \mid \dots \mid Z\ rid$

$\mid a\ rid \mid b\ rid \mid \dots \mid z\ rid$

$\mid 0\ rid \mid 1\ rid \mid \dots \mid 9\ rid$

■ 一般写作:

$id \rightarrow \text{letter}\ rid$

$rid \rightarrow \varepsilon \mid \text{letter}\ rid \mid \text{digit}\ rid$

常数——整数

- 描述整数结构的正规表达式为：

$$(\text{digit})^+$$

- 对此正规表达式进行等价变换：

$$(\text{digit})^+ = \text{digit}(\text{digit})^*$$

$$(\text{digit})^* = \varepsilon \mid \text{digit}(\text{digit})^*$$

- 整数的正规文法：

$$\text{digits} \rightarrow \text{digit remainder}$$

$$\text{remainder} \rightarrow \varepsilon \mid \text{digit remainder}$$

常数——无符号数

- 无符号数的正规表达式为：

$(\text{digit})^+ (.\text{(digit)}^+)? (\text{E}(+|-)?(\text{digit})^+)?$

- 正规定义式为

$\text{digit} \rightarrow \mathbf{0|1|...|9}$

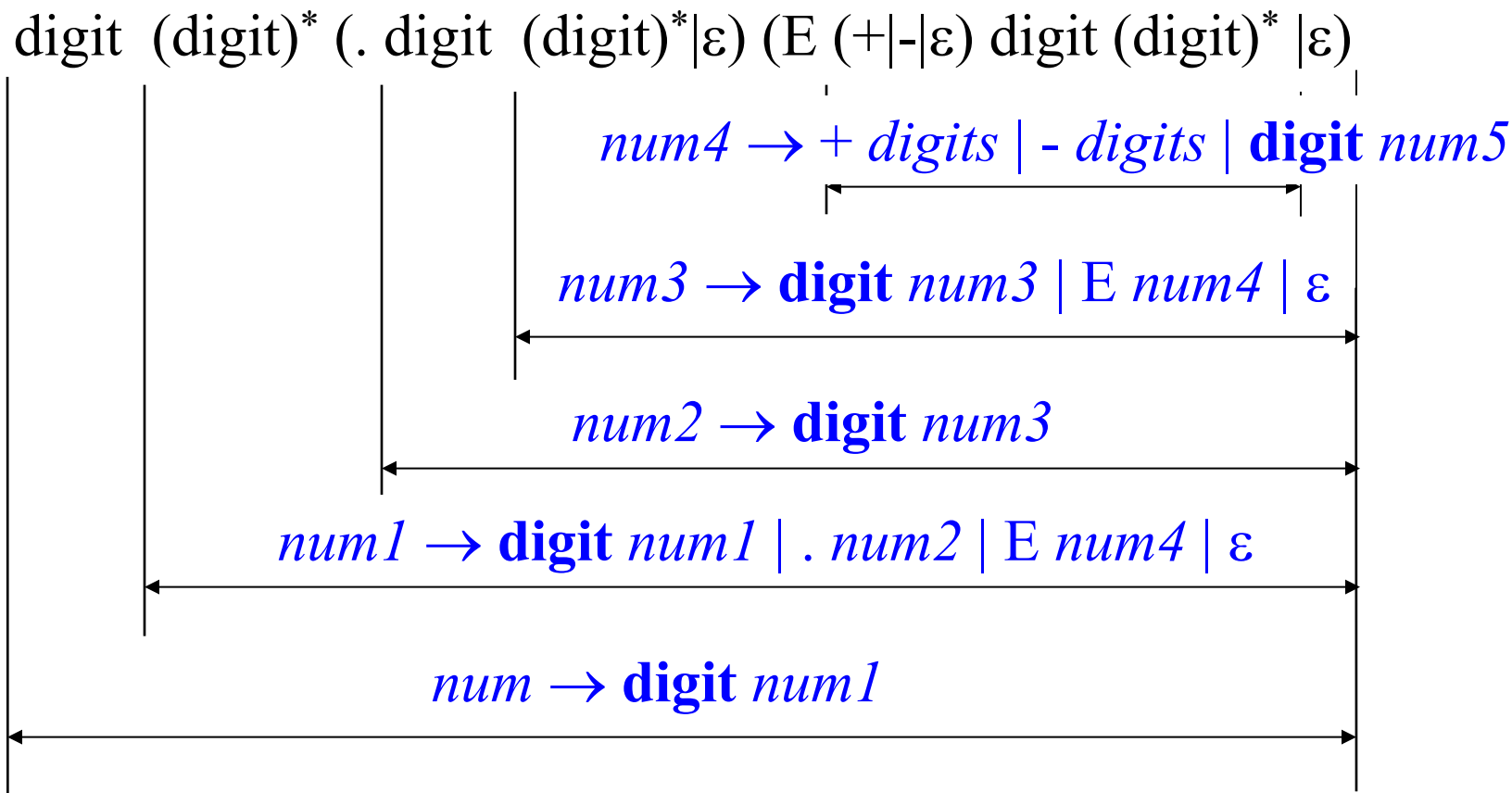
$\text{digits} \rightarrow \text{digit}^+$

$\text{optional_fraction} \rightarrow (. \text{digits})?$

$\text{optional_exponent} \rightarrow (\mathbf{E(+|-)?} \text{digits})?$

$\text{num} \rightarrow \text{digits optional_fraction optional_exponent}$

无符号数分析图



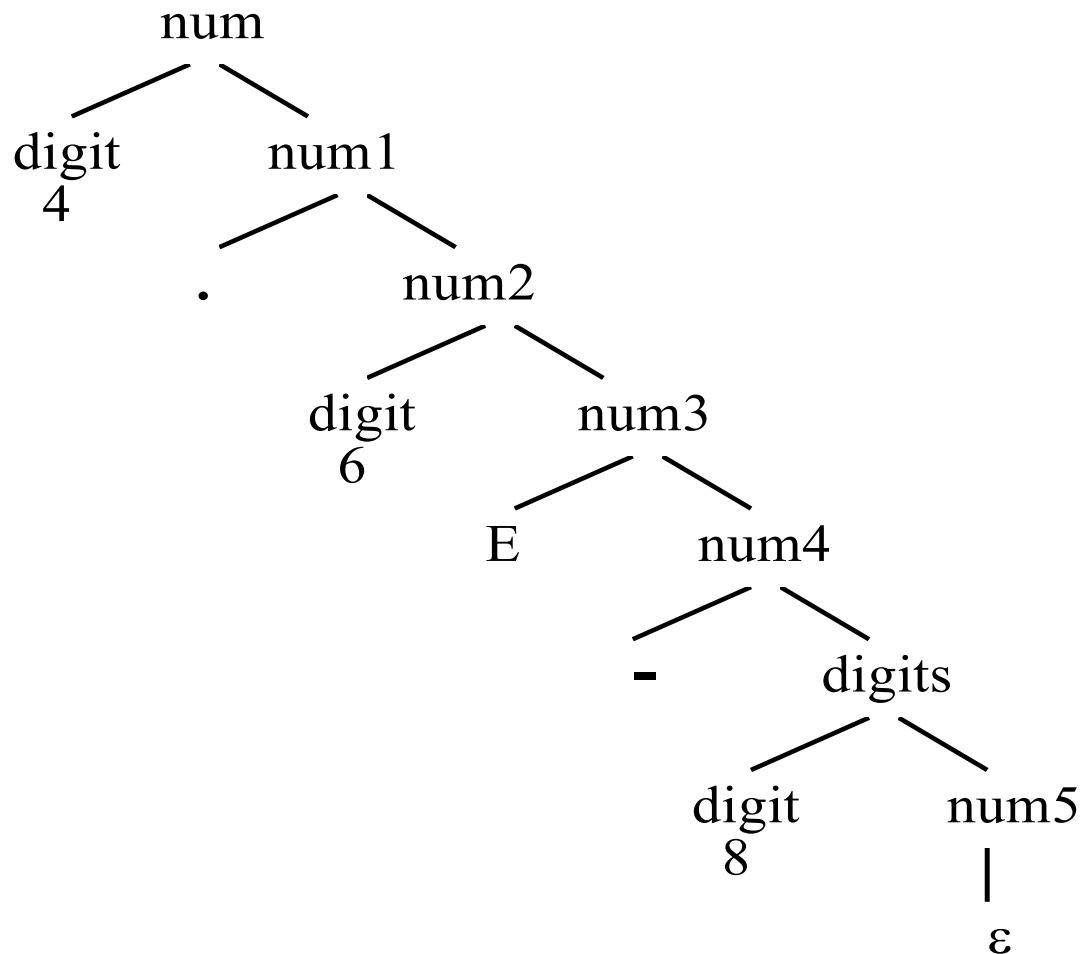
num5 表示 $(\text{digit})^*$
 digits 表示 $(\text{digit})^+$

$\text{digits} \rightarrow \text{digit } num5$
 $num5 \rightarrow \text{digit } num5 \mid \epsilon$

无符号数的正规文法

$num \rightarrow digit\ num1$
 $num1 \rightarrow digit\ num1 \mid .\ num2 \mid E\ num4 \mid \varepsilon$
 $num2 \rightarrow digit\ num3$
 $num3 \rightarrow digit\ num3 \mid E\ num4 \mid \varepsilon$
 $num4 \rightarrow +\ digits \mid -\ digits \mid digit\ num5$
 $digits \rightarrow digit\ num5$
 $num5 \rightarrow digit\ num5 \mid \varepsilon$

无符号数 4.6E-8 的分析树



运算符

- 关系运算符的正规表达式为：

$\lt \mid \leq \mid = \mid \lt \gt \mid \gt = \mid \gt$

- 正规定义式：

$\text{relop} \rightarrow \lt \mid \leq \mid = \mid \lt \gt \mid \gt = \mid \gt$

- 关系运算符的正规文法：

$\text{relop} \rightarrow \lt \mid \lt \text{equal} \mid = \mid \lt \text{greater} \mid \gt \mid \gt \text{equal}$

$\text{greater} \rightarrow \gt$

$\text{equal} \rightarrow =$

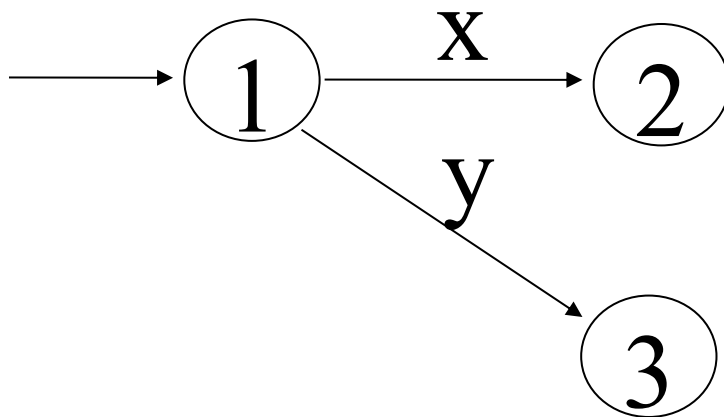
三、状态转换图与记号的识别

- 状态转换图
- 利用状态转换图识别记号
- 为线性文法构造相应的状态转换图
 - ◆ 状态集合的构成
 - ◆ 状态之间边的形成

状态转换图

■ 状态转换图是一张有限的方向图

- ◆ 图中结点代表状态，用圆圈表示。
- ◆ 状态之间用有向边连接。
- ◆ 边上的标记代表在射出结状态下，可能出现的输入符号或字符类。



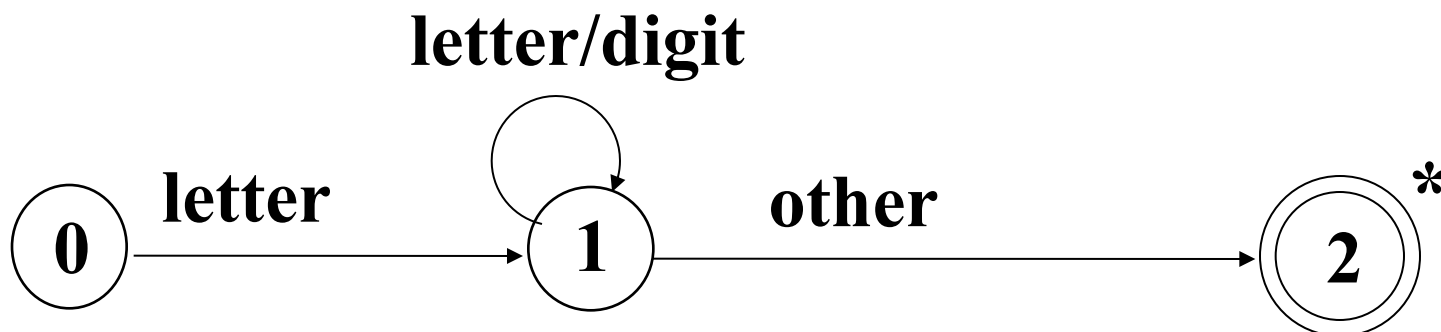
标识符的状态转换图

- 标识符的文法产生式：

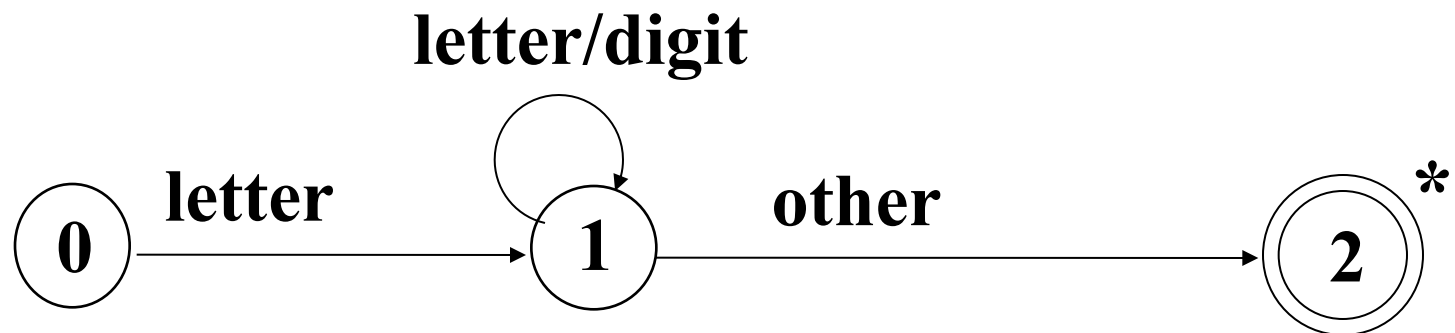
$id \rightarrow \text{letter } rid$

$rid \rightarrow \varepsilon \mid \text{letter } rid \mid \text{digit } rid$

- 标识符的状态转换图



利用状态转换图识别记号



- 语句 `D099K=1.10` 中标识符 `D099K` 的识别过程

为线性文法构造相应的状态转换图

■ 状态集合的构成

- ◆ 对文法 G 的每一个非终结符号设置一个对应的状态
- ◆ 文法的开始符号对应的状态称为初态
- ◆ 增加一个新的状态，称为终态。

■ 状态之间边的形成

- ◆ 对产生式 $A \rightarrow aB$ ，从 A 状态到 B 状态画一条标记为 a 的边
- ◆ 对产生式 $A \rightarrow a$ ，从 A 状态到终态画一条标记为 a 的边
- ◆ 对产生式 $A \rightarrow \varepsilon$ ，从 A 状态到终态画一条标记为 ε 的边

无符号数的右线性文法的状态转换图

$num \rightarrow digit\ num1$

$num1 \rightarrow digit\ num1 \mid \cdot\ num2 \mid E\ num4 \mid \varepsilon$

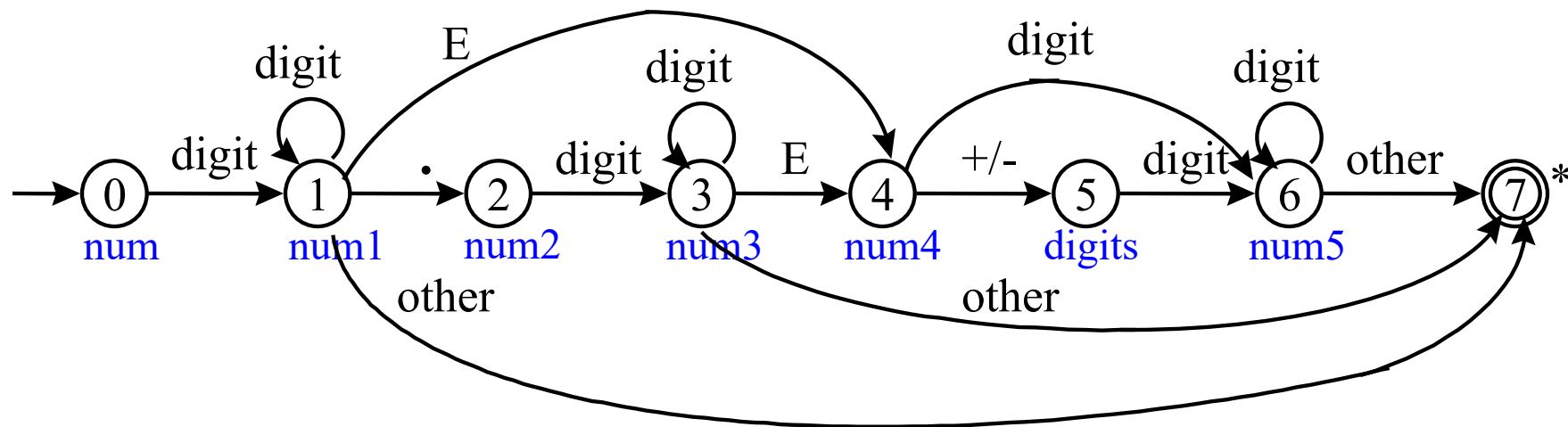
$num2 \rightarrow digit\ num3$

$num3 \rightarrow digit\ num3 \mid E\ num4 \mid \varepsilon$

$num4 \rightarrow +\ digits \mid -\ digits \mid digit\ num5$

$digits \rightarrow digit\ num5$

$num5 \rightarrow digit\ num5 \mid \varepsilon$



3.4 词法分析程序的设计与实现

一、文法及状态转换图

1. 语言说明
2. 记号的正规文法
3. 状态转换图

二、词法分析程序的构造

三、词法分析程序的实现

1. 输出形式
2. 设计全局变量和过程
3. 编制词法分析程序

一、文法及状态转换图

■ 语言说明

标识符：以字母开头的、后跟字母或数字组成的符号串。

保留字：标识符的子集。

无符号数：同**PASCAL**语言中的无符号数。

关系运算符：<、<=、=、<>、>=、>。

标点符号：+、-、*、/、(、)、:、'、; 等。

赋值号：:=

注释标记：以 '/*' 开始，以 '*/' 结束。

单词符号间的分隔符：空格

记号的正规文法

■ 标识符的文法

$id \rightarrow \text{letter } rid$

$rid \rightarrow \varepsilon \mid \text{letter } rid \mid \text{digit } rid$

■ 无符号整数的文法

$digits \rightarrow \text{digit } remainder$

$remainder \rightarrow \varepsilon \mid \text{digit } remainder$

记号的正规文法(续1)

■ 无符号数的文法

$num \rightarrow digit\ num1$

$num1 \rightarrow digit\ num1 \mid .\ num2 \mid E\ num4 \mid \varepsilon$

$num2 \rightarrow digit\ num3$

$num3 \rightarrow digit\ num3 \mid E\ num4 \mid \varepsilon$

$num4 \rightarrow +\ digits \mid -\ digits \mid digit\ num5$

$digits \rightarrow digit\ num5$

$num5 \rightarrow digit\ num5 \mid \varepsilon$

■ 关系运算符的文法

$relop \rightarrow < \mid <equal \mid = \mid <greater \mid > \mid >equal$

$greater \rightarrow >$

$equal \rightarrow =$

记号的正规文法(续2)

■ 赋值号的文法

assign_op* → :*equal

***equal* → =**

■ 标点符号的文法

***single* → + | - | * | / | (|) | : | ' | ;**

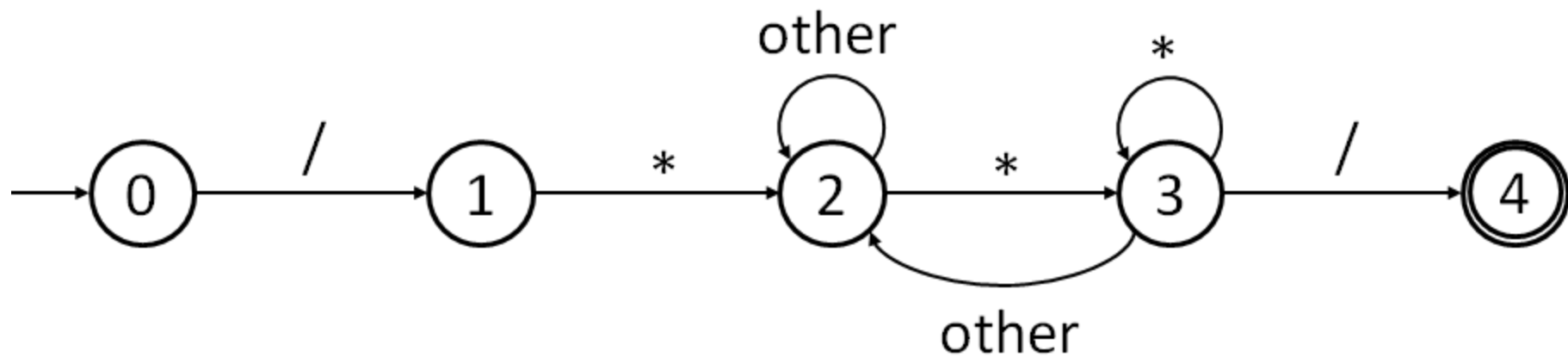
■ 注释头符号的文法

note* → / *star

***star* → ***

Liang GE
CAU

识别注释的DFA



二、词法分析程序的构造

- 把语义动作添加到状态转换图中，使每一个状态都对应一小段程序，就可以构造出相应的词法分析程序。
- 如果某一状态有若干条射出边，则程序段：读一个字符，根据读到的字符，选择标记与之匹配的边到达下一个状态，即程序控制转去执行下一个状态对应的语句序列。
- 在状态0，首先要读进一个字符。若读入的字符是一个空格（包括blank、tab、enter）就跳过它，继续读字符，直到读进一个非空字符为止。接下来的工作就是根据所读进的非空字符转相应的程序段进行处理。
- 在标识符状态，识别并组合出一个标识符之后，还必须加入一些动作，如查关键字表，以确定识别出的单词符号是关键字还是用户自定义标识符，并输出相应的记号。
- 在“<”状态，若读进的下一个字符是“=”，则输出关系运算符“<=”；若读进的下一个字符是“>”，则输出关系运算符“<>”；否则输出关系运算符“<”。

三、词法分析程序的实现

- 输出形式
- 设计全局变量和过程
- 编制词法分析程序

输出形式

- 利用翻译表，将识别出的单词的记号以二元式的形式加以输出
- 二元式的形式：
 <记号，属性>
- 翻译表：

翻译表

正规表达式	记号	属性
if	if	-
then	then	-
else	else	-
id	id	符号表入口指针
num	num	常数表入口指针 / val
<	relop	LT
<=	relop	LE
=	relop	EQ
<>	relop	NE
>	relop	GT
>=	relop	GE
:=	assign-op	-
+	+	-
-	-	-
*	*	-
/	/	-
((-
))	-
,	,	-
;	;	-
:	:	-

设计全局变量和过程

- (1) **state**: 整型变量, 当前状态指示。
- (2) **C**: 字符变量, 存放当前读入的字符。
- (3) **token**: 字符数组, 存放当前正在识别的单词字符串。
- (4) **buffer**: 字符数组, 输入缓冲区。
- (5) **forward**: 字符指针, 向前指针。
- (6) **lexemebegin**: 字符指针, 指向buffer中当前单词的开始位置。
- (7) **get_char**: 过程, 每调用一次, 根据forward的指示从buffer中读一个字符, 并把它放入变量C中, 然后, 移动forward, 使之指向下一个字符。
- (8) **get_nbc**: 过程, 检查C中的字符是否为空格, 若是, 则反复调用过程get_char, 直到C中进入一个非空字符为止。
- (9) **cat**: 过程, 把C中的字符连接在token中的字符串后面。
- (10) **iskey**: 整型变量, 值为-1, 表示识别出的单词是用户自定义标识符, 否则, 表示识别出的单词是关键字, 其值为关键字的记号。

设计全局变量和过程（续）

- (11) **letter**: 布尔函数, 判断C中的字符是否为字母, 若是则返回true, 否则返回false。
- (12) **digit**: 布尔函数, 判断C中的字符是否为数字, 若是则返回true, 否则返回false。
- (13) **retract**: 过程, 向前指针forward后退一个字符。
- (14) **reserve**: 函数, 根据token中的单词查关键字表, 若token中的单词是关键字, 则返回值该关键字的记号, 否则, 返回值“-1”。
- (15) **STol**: 过程, 将token中的字符串转换成整数。
- (16) **SToF**: 过程, 将token中的字符串转换成浮点数。
- (17) **table_insert**: 函数, 将识别出来的标识符（即token中的单词）插入符号表, 返回该单词在符号表中的位置指针。
- (18) **error**: 过程, 对发现的错误进行相应的处理。
- (19) **return**: 过程, 将识别出来的单词的记号返回给调用程序。

Liang GE
CAU

词法分析程序（类C语言描述）

```
state=0;
DO {
  SWITCH ( state ) {
    CASE 0:  // 初始状态
      token=' ';  get_char();  get_nbc();
      SWITCH ( C ) {
        CASE 'a': CASE 'b': ... CASE 'z': state=1; break;  //设置标识符状态
        CASE '0': CASE '1': ... CASE '9': state=2; break;  //设置常数状态
        CASE '<': state=8; break;  //设置 '<' 状态
        CASE '>': state=9; break;  //设置 '>' 状态
        CASE ':': state=10; break;  //设置 ':' 状态
        CASE '/': state=11; break;  //设置 '/' 状态
        CASE '=': state=0; return(relop, EQ); break;  //返回 '=' 的记号
        CASE '+': state=0; return('+', -); break;  //返回 '+' 的记号
        CASE '-': state=0; return('-', -); break;  //返回 '-' 的记号
        CASE '*': state=0; return('*', -); break;  //返回 '*' 的记号
        CASE '(': state=0; return('(', -); break;  //返回 '(' 的记号
        CASE ')': state=0; return(')', -); break;  //返回 ')' 的记号
        CASE ';': state=0; return(';', -); break;  //返回 ';' 的记号
        CASE '\\': state=0; return('\\', -); break;  //返回 '\\' 的记号
        default: state=13; break;  //设置错误状态
      };
    break;
  }
}
```

词法分析程序 (续1)

CASE 1: // 标识符状态

```
cat();
```

```
get_char();
```

```
IF ( letter() || digit() ) state=1;
```

```
ELSE {
```

```
    retract();
```

```
    state=0;
```

```
    iskey=reserve(); // 查关键字表
```

```
    IF ( iskey!=-1 ) return (iskey, -); // 识别出的是关键字
```

```
    ELSE { // 识别出的是用户自定义标识符
```

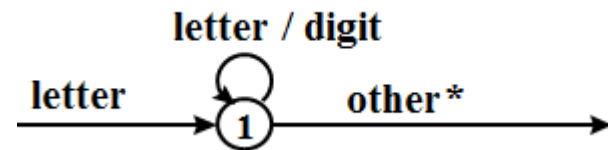
```
        identry=table_insert(); // 返回该标识符在符号表的入口指针
```

```
        return(ID, identry);
```

```
    };
```

```
};
```

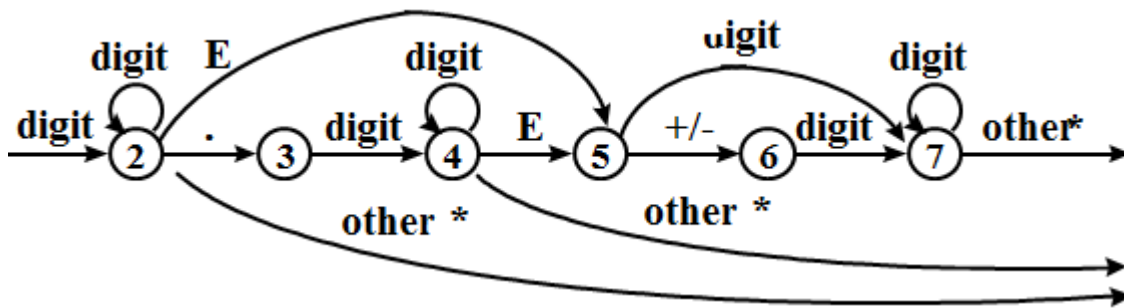
```
break;
```



词法分析程序（续2）

CASE 2: // 常数状态

```
cat();
get_char();
SWITCH ( C ) {
    CASE '0':
    CASE '1':
        ⋮
    CASE '9': state=2; break;
    CASE '.': state=3; break;
    CASE 'E': state=5; break;
    DEFAULT: // 识别出整常数
        retract();
        state=0;
        return(NUM, STol(token)); // 返回整数
        break;
};
break;
```



3.5 软件工具 LEX

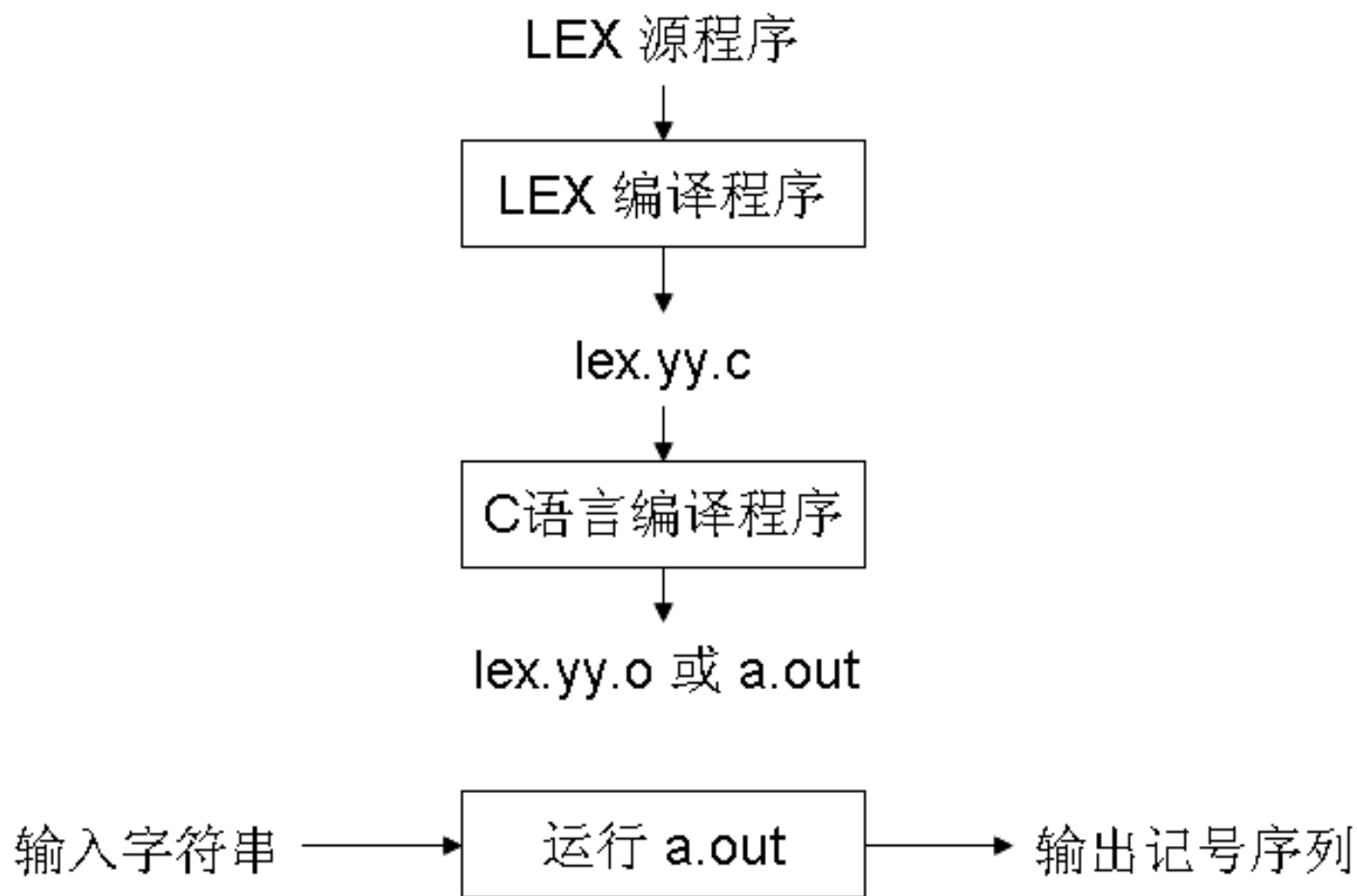
■ 词法分析程序自动生成工具

一、LEX使用流程

二、LEX源程序结构

三、LEX工作原理

一、LEX使用流程



二、LEX源程序结构

一个LEX源程序由三部分组成：

1. 说明部分
2. 翻译规则
3. 辅助过程

说明部分

%%

翻译规则

%%

辅助过程

1. 说明部分

- 包括：变量说明、标识符常量说明、正规定义
- 正规定义中的名字可在翻译规则中用作正规表达式的成分
- C语言的说明必须用分界符 “ %{ ”和 “%} ”括起来。

◆ 比如：

```
%{  
    #include <stdio.h>  
    #include <stdlib.h>  
    #include <string.h>  
    #include <ctype.h>  
    #include "y.tab.h"  
    typedef char * YYSTYPE;  
    char * yylval;  
%}
```


2. 翻译规则

■ 形式：

$P_1 \quad \{ \text{动作}_1 \}$

$P_2 \quad \{ \text{动作}_2 \}$

...

$P_n \quad \{ \text{动作}_n \}$

■ P_i 是一个正规表达式，描述一种记号的模式。

■ 动作 i 是C语言的程序语句，表示当一个串匹配模式 P_i 时，词法分析器应执行的动作。

P_i书写中可能用到的规则

- (1) 转义字符: " \ [] ^ - ? . * + | () \$ / { } % < >
 - ◆ 具有特殊含义, 不能用来匹配自身。如果需要匹配的话, 可以通过引号(")或者转义符号(\)来指示。比如: C"++"和C\+\+ 都可以匹配C++。
- (2) 通配符: . 可以匹配任何一个字符。
 - ◆ 如: a.c匹配任何以a开头、以c结尾的长度为3的字符串。
- (3) 字符集: 用方括号 "[" 和 "]" 指定的字符构成一个字符集。
 - ◆ 如, [abc]表示一个字符集, 可以匹配a、b或c中的任意一个字符。
 - ◆ 使用 " - " 可以指定范围。比如: [A-Za-z]。
- (4) 重复: " *" 表示任意次重复 (可以是零次),
" + " 表示至少一次的重复,
" ? " 表示零次或者一次。
 - ◆ 如: a+相当于aa*, a*相当于a+| ϵ , a?相当于a| ϵ 。
- (5) 选择和分组: " | " 表示二者则一; 括号 "(" 和 ")" 表示分组, 括号内的组合被看作是一个原子。
 - ◆ 如: x(ab|cd)y 匹配xaby或者xcdy。

3. 辅助过程

- 对翻译规则的补充
- 翻译规则部分中某些动作需要调用的过程，如果不是C语言的库函数，则要在此给出具体的定义。
- 这些过程也可以存入另外的程序文件中，单独编译，然后和词法分析器连接装配在一起。

LEX源程序举例

■ 正规定义式：

if → if

then → then

else → else

relop → < | <= | = | <> | > | >=

id → letter(letter|digit)*

num → digit⁺(.digit⁺)?(E(+|-)?digit⁺)?

相应的 LEX 源程序 框架

```
/* 声明部分 */
```

```
%{
```

```
#include <stdio.h>
```

```
! /* C语言描述的符号常量的定义，如LT、LE、EQ、NE、GT、  
    GE、IF、THEN、ELSE、ID、NUMBER、RELOP */
```

```
extern yylval, yytext, yyleng;
```

```
%}
```

```
/* 正规定义式 */
```

```
delim      [ \t\n]
```

```
ws         {delim}+
```

```
letter     [A-Za-z]
```

```
digit      [0-9]
```

```
id         {letter}({letter}|{digit})*
```

```
num        {digit}+(\.{digit}+)?(E[+-]?{digit}+)?
```

```
%%
```

相应的 LEX 源程序 框架

/* 规则部分 */

```
{ws}      { /* 没有动作, 也不返回 */ }  
if        { return(IF); }  
then      { return(THEN); }  
else      { return(ELSE); }  
{id}      { yylval=install_id(); return(ID); }  
{num}     { yylval=install_num(); 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); }  
%%
```

如果没有return语句, 则, 处理完整个输入之后才会返回!!

相应的 LEX 源程序 框架

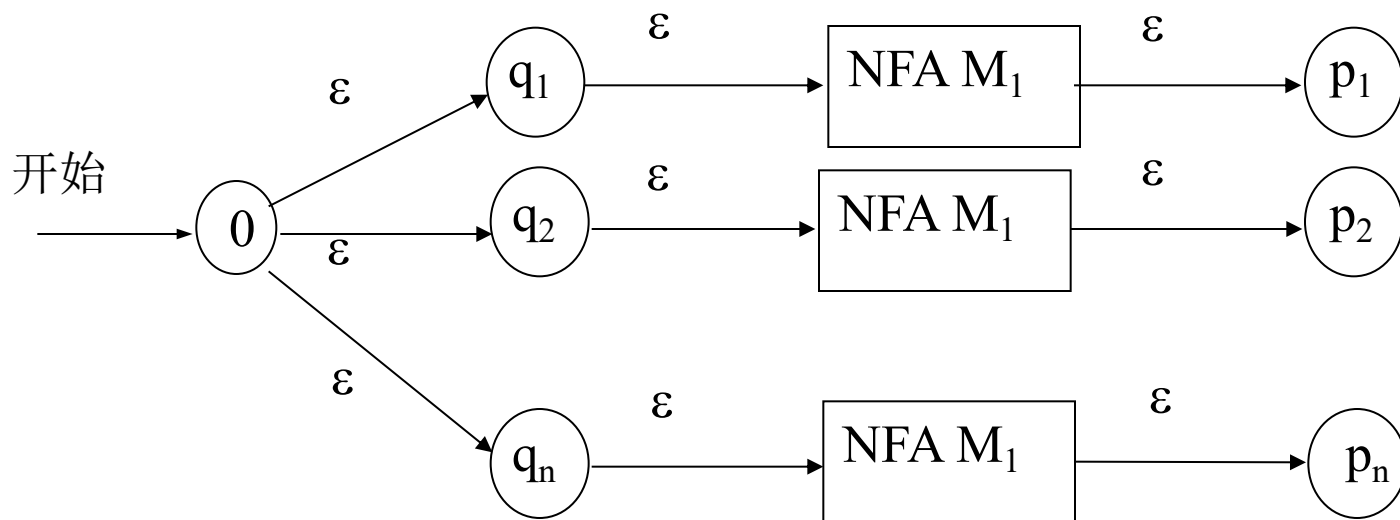
```
/* 辅助过程 */  
int install_id() {  
    ! /* 把单词插入符号表并返回该单词在符号表中的位置  
       yytext指向该单词的第一个字符  
       yyleng给出它的长度 */  
}  
int num_val() {  
    ! /* 将识别出的无符号数字字符串转换成数值型返回。 */  
}
```

三、LEX的工作原理

1. LEX的工作过程
2. 处理二义性问题的两条规则
3. LEX工作过程举例
4. 控制执行程序

1. 工作过程

- 扫描每一条翻译规则 P_i ，为之构造一个非确定的有限自动机NFA M_i
- 将各条翻译规则对应的NFA M_i 合并为一个新的NFA M



- 将NFA M 确定化为DFA D ，并生成该DFA D 的状态转换矩阵和控制执行程序。

2. 识别单词时的二义性处理

- 最长匹配原则
 - ◆ 在识别单词符号过程中，当有几个规则看来都适用时，则实施最长匹配的那个规则
- 优先匹配原则
 - ◆ 如有几条规则可以同时匹配一字符串，并且匹配的长度相同，则实施最上面的规则。
- 如果有一些内容不匹配任何规则，则LEX将会把它拷贝到标准输出

3. 工作过程举例

%%

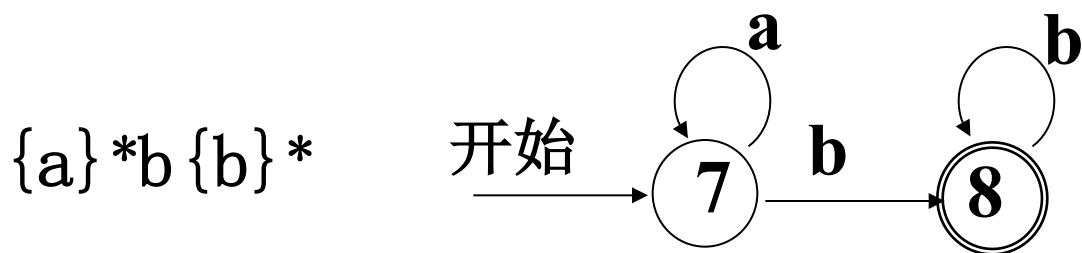
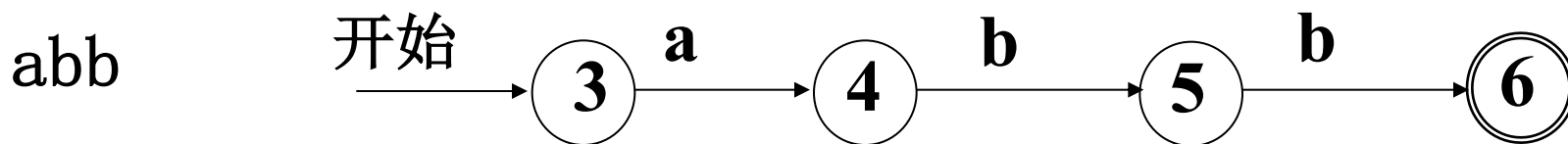
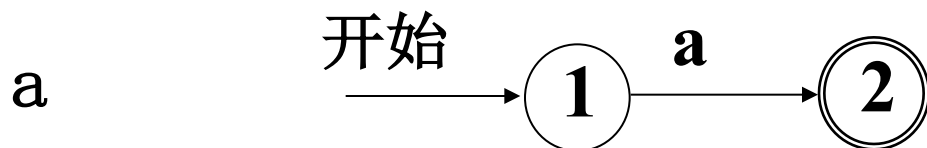
a { }

abb { }

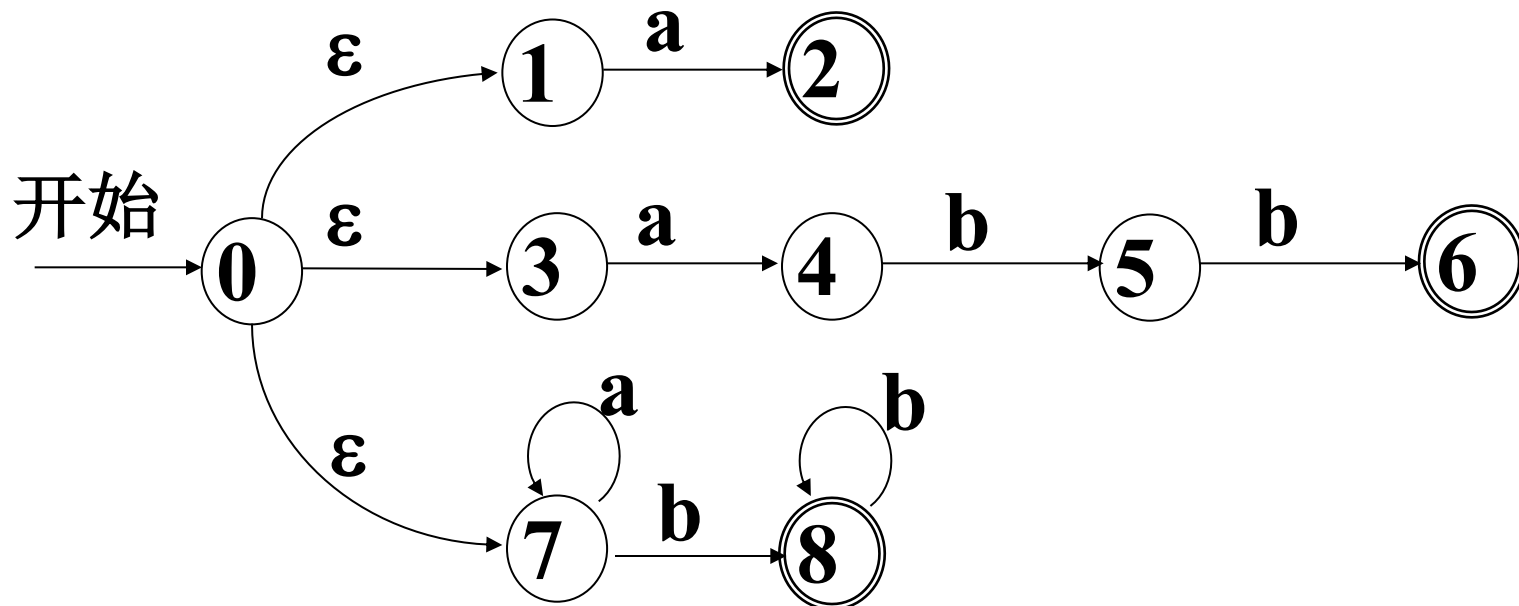
{a}*b{b}* { }

%%

读LEX源程序，分别生成非确定的有限自动机



合并为一个NFA M

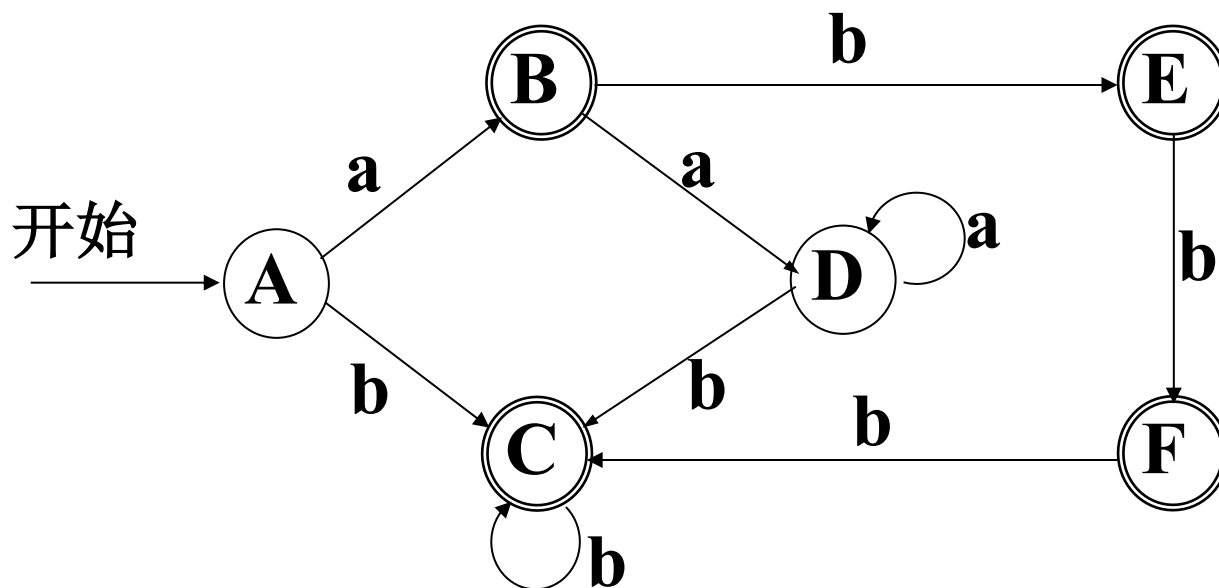


将该NFA M确定化为DFA D

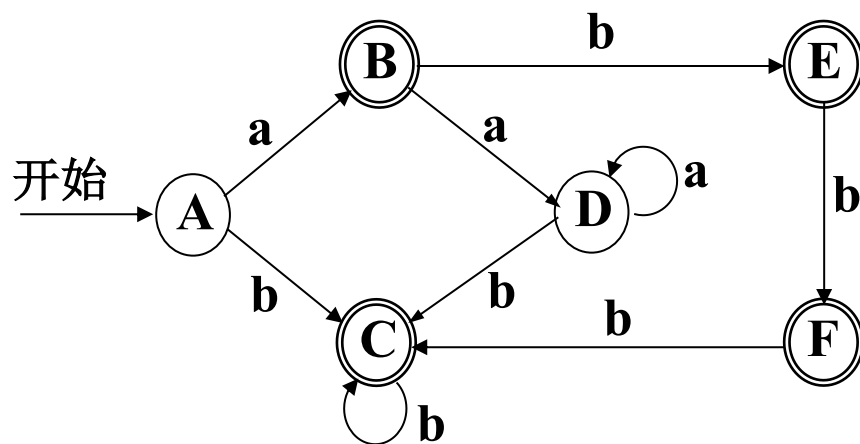
DFA $D = (\{a, b\}, \{A, B, C, D, E, F\}, A, \{B, C, E, F\}, \varphi)$

其中: $A = \{0, 1, 3, 7\}$ $B = \{2, 4, 7\}$ $C = \{8\}$

$D = \{7\}$ $E = \{5, 8\}$ $F = \{6, 8\}$



4. 控制执行程序



$A = \{0, 1, 3, 7\}$

$B = \{2, 4, 7\}$

$C = \{8\}$

$D = \{7\}$

$E = \{5, 8\}$

$F = \{6, 8\}$

输入字符串为 aba...

读入

状态

—

A

a

B

b

E

a

—

小 结

- 词法分析器的作用
- 与语法分析器的关系
 - ◆ 独立、子程序、协同程序
- 配对缓冲区
 - ◆ 必要性、算法
- 记号
 - ◆ 记号、模式、单词
 - ◆ 属性
 - ◆ 二元式形式 <记号, 属性>
 - ◆ 描述：正规表达式、正规文法
 - ◆ 识别：状态转换图

小结 (续)

■ 词法分析器的设计与实现

- ◆ 各类单词符号的正规表达式
- ◆ 各类单词符号的正规文法
- ◆ 构造与文法相应的状态转换图
- ◆ 合并为词法分析器的状态转换图
- ◆ 增加语义动作，构造词法分析器的程序框图
- ◆ 确定输出形式、设计翻译表
- ◆ 定义变量和过程
- ◆ 编码实现

■ LEX工具

- ◆ LEX源程序的结构
- ◆ LEX工具的使用