重庆大学
CHONGQING UNIVERSITY

第三讲 词法分析

重庆大学 计算机学院 张敏

本章内容

基础知识：

- 程序设计语言
- 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
- 词法分析器的构造

简介

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简介

3.1词法分析程序的任务

输入

L	i	n	e	=	8	0	;

输出

⟨ id(25) , ‘Line’⟩
⟨ =(36), ——— ⟩
⟨ num(27), ‘80’⟩
⟨ ;(45), ——— ⟩

3.1词法分析程序的任务

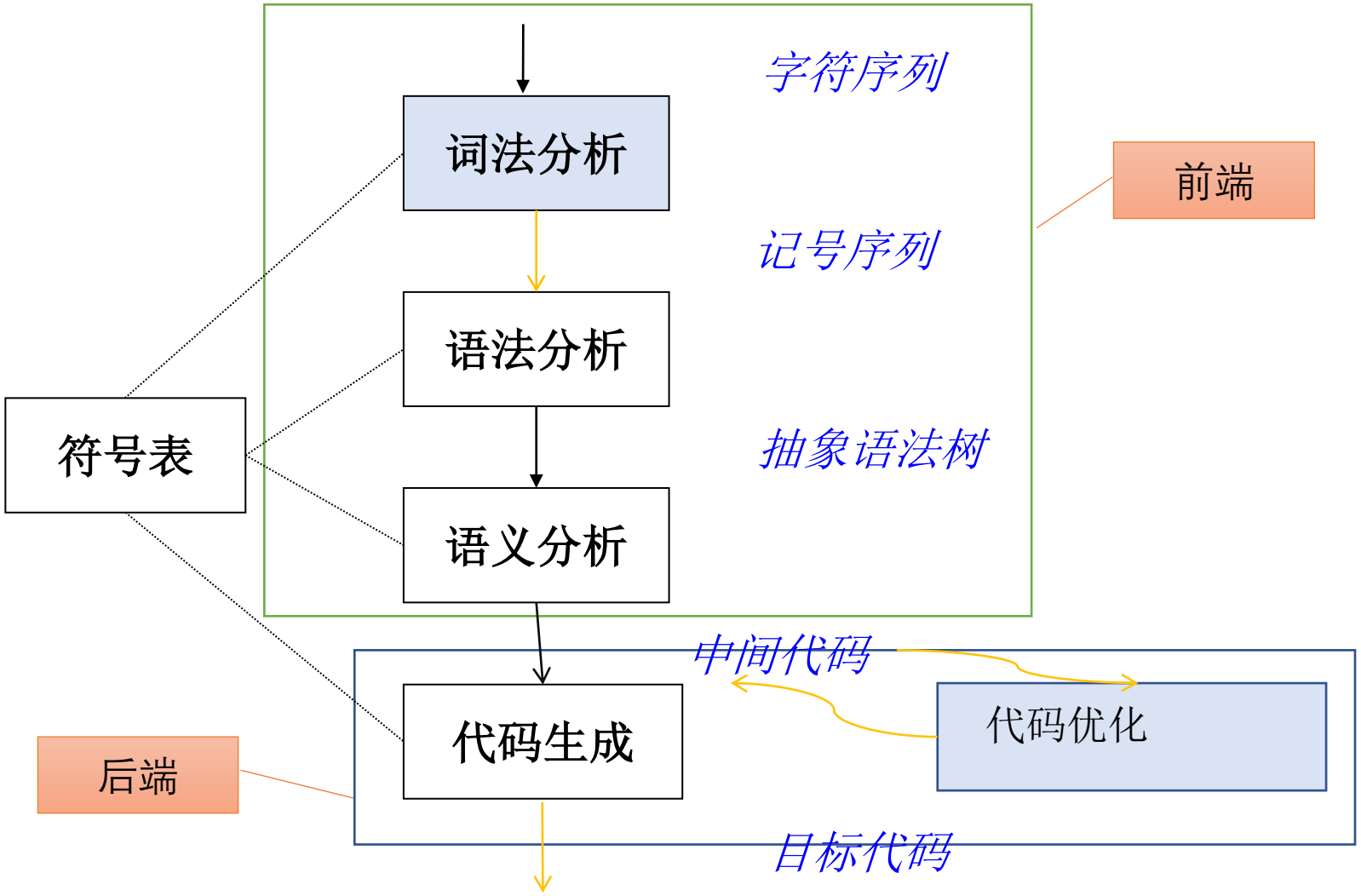
词法分析程序的作用

- 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
- 其他辅助任务：

词法检查
创建符号表(需要的话)
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词法分析程序的任务



3.1词法分析程序的任务

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

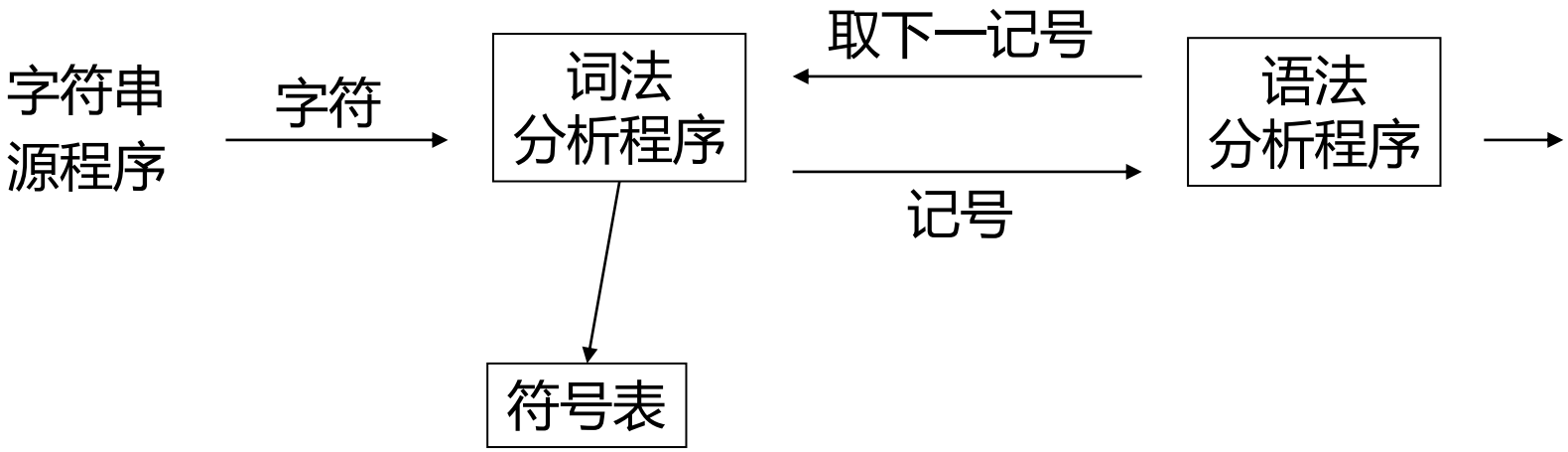
3.1词法分析程序的任务

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

3.1词法分析程序的任务

词法分析程序作为语法分析程序的子程序

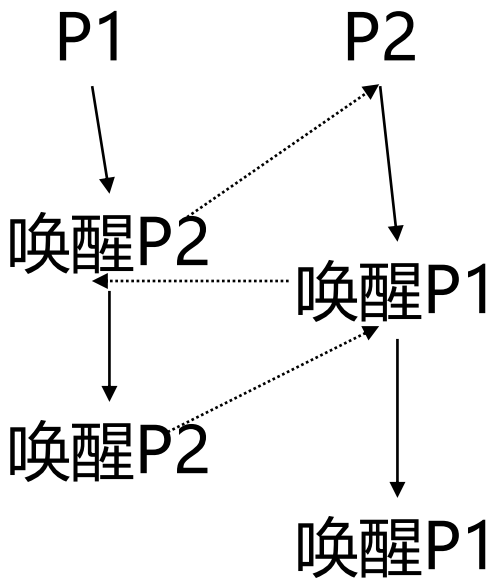


- 避免了中间文件
- 省去了取送符号的工作
- 有利于提高编译程序的效率

3.1词法分析程序的任务

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。



课堂讨论

- 分离词法程序的优点?
- 设计一个词法分析器，分为几步来实现?
- 词法分析器的结构如何设计?
- 需要用到哪些辅助的数据结构及方法?

3.2 词法分析程序的输入与输出

3.2.1 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

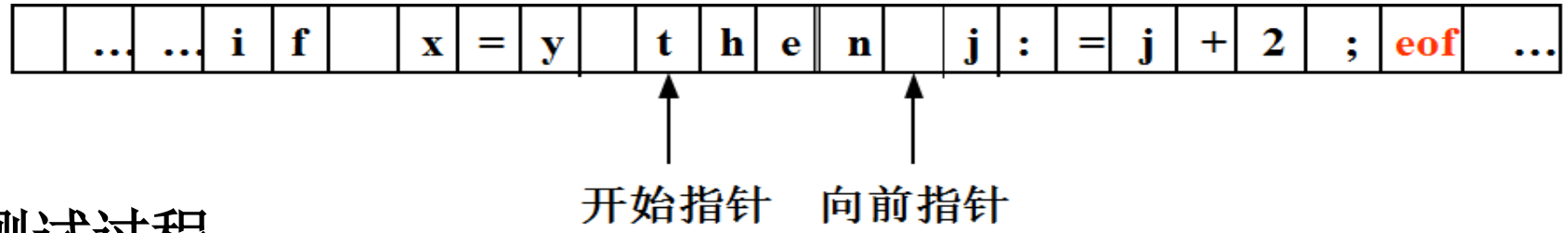
3.2 词法分析程序的输入与输出

3.2.2.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。
- 合法的FORTRAN语句：
  DO 99 K=1,10　和　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。

3.2.3 配对缓冲区

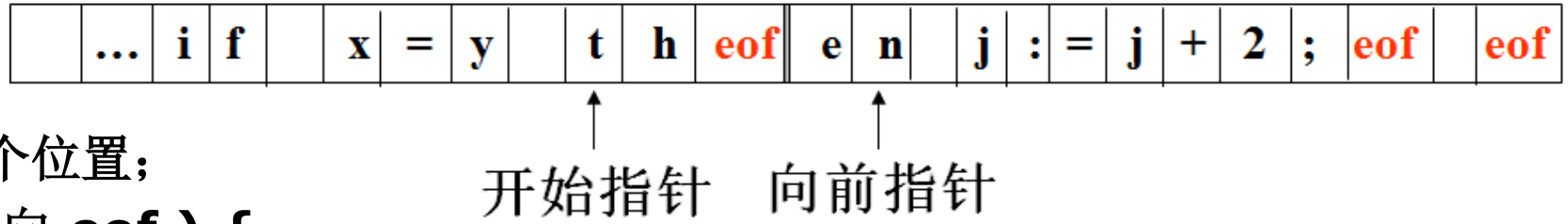
- 把一个缓冲区分为大小相同的两半，每半各含N个字符，一般N=1KB或4KB。



n 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



- 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;
}
```

3.2.4 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。

3.2.4 词法分析程序的输出——记号

记号的机内表示

机内表示，是在语法分析程序和语义分析程序之间交换的表示形式，而不是给人来阅读的。机内表示形式通常用二元形式（token字）

词法分析器的输出：

（词类编码，记号自身的属性值）

3.2.4 词法分析程序的输出——记号

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>

3.2.4 词法分析程序的输出——记号

词类编码原则:

界符和运算符:一符一码

关键字:可分成一类，也可以一个关键字 分成一类,一字一码。

常数:可统归一类，也可按类型（整型、实型、布尔型等），　每个类型的常数划分成一类,一类型一码

标识符:所有标识符分为一类，一类一码

对于关键字、界符、运算符来说，它们的词类编码就可以表示其完整的信息，故对于这类记号，其记号自身的属性值通常为空；

而对于标识符，词类编码所反映的信息不够充分，标识符的具体特性还要通过记号自身的属性进行互相区分；

标识符的记号自身的属性常用其在符号表中的入口指针来表示；

对于常数，　其记号自身的属性常用其在常数表中的入口指针来表示。

表3.1 记号词类编码

单词	词类编码	单词	词类编码	单词	词类编码
and	1	procedure	16	*	31
array	2	programe	17	+	32
begin	3	read	18	,	33
bool	4	real	19	——	34
call	5	then	20	/	35
char	6	TRUE	21	=	36
do	7	var	22	<	37
else	8	while	23	<=	38
end	9	write	24	>	39
FALSE	10	标识符	25	>=	40
for	11	整常数	26	<>	41
if	12	实常数	27	==	42
integer	13	‘	28	[	43
not	14	(	29	]	44
or	15	)	30	.	45

示例

以语句子a=b+c*d　为例，假设按表3.1为记号编码，词法分析后的结果为：

Token字
<25, >
<36, -------- >
<25, >
<32, -------- >
<25, >
<31, -------- >
<25, >

符号表	
a	25
b	25
c	25
d	25

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

3.3.1 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、…、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

语法：借助于记号来描述语言的结构的文法

3.3.2 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

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

标识符的正规文法

id → A *rid*|...|Z *rid*|a *rid*|...|z *rid*
rid → ε |A *rid*|B *rid*|...|Z *rid*
|a *rid*|b *rid*|...|z *rid*
|0 *rid*|1 *rid*|...|9 *rid*

一般写作:

id → letter *rid*
rid → ε | letter *rid* | digit *rid*

示例：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：

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

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

示例：常数—无符号数

无符号数的正规表达式为：

$(digit)^+ (.(digit)^+)? (E(+|-)?(digit)^+)?$

正规定义式为

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

$digits \rightarrow digit^+$

$optional\_fraction \rightarrow (.digits)?$

$optional\_exponent \rightarrow (E(+|-)?digits)?$

$num \rightarrow digits\ optional\_fraction\ optional\_exponent$

把正规定义式转换为正规文法

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

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

$$=\text{digit}(\text{digit})^{*}(.\text{digit}(\text{digit})^{*}|\varepsilon)(E(+|-|\varepsilon)\text{digit}(\text{digit})^{*}|\varepsilon)$$

num1 表示无符号数的第一个数字之后的部分

num2 表示小数点以后的部分

num3 表示小数点后第一个数字以后的部分

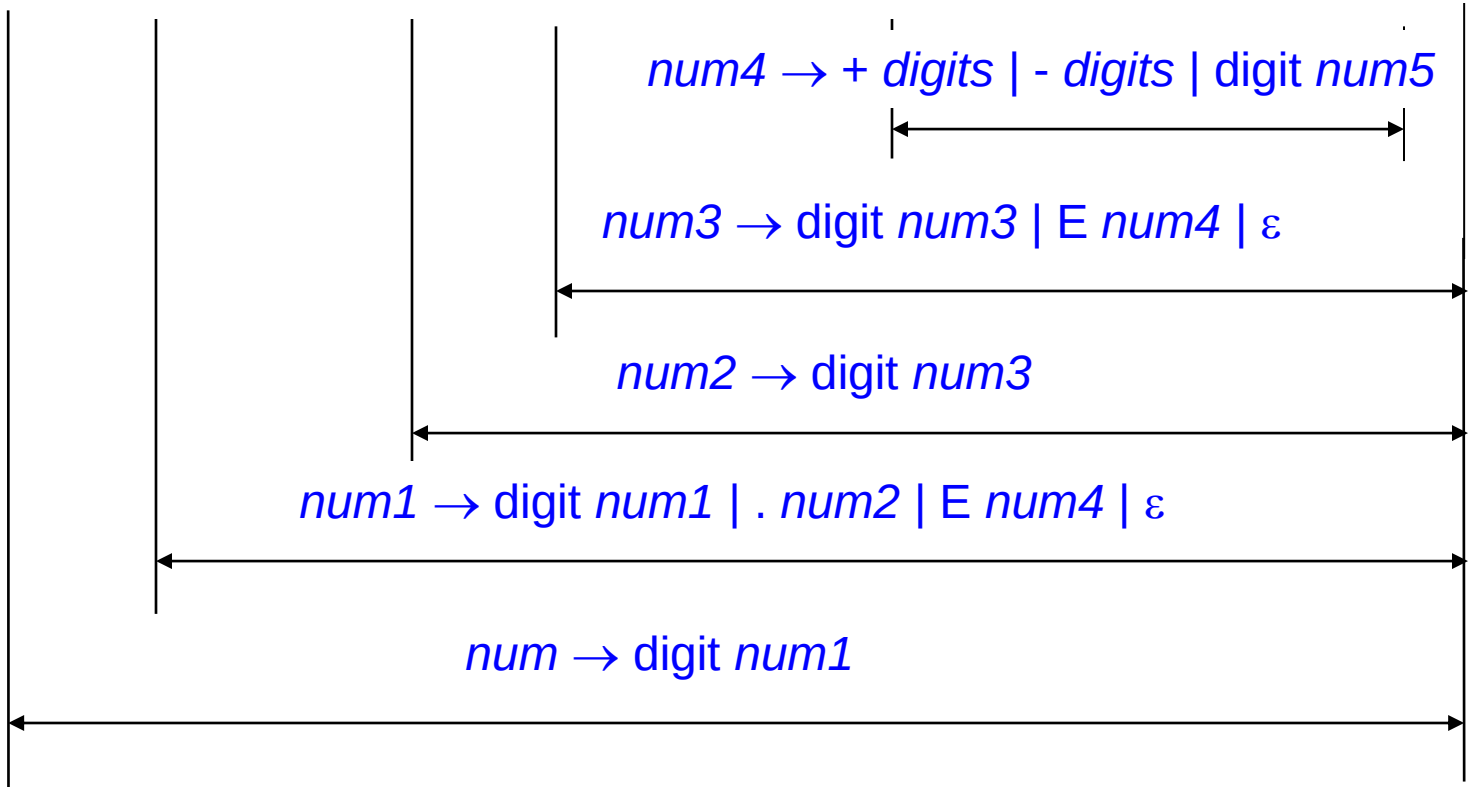
num4 表示E之后的部分

num5 表示$(\text{digit})^{*}$

digits 表示$(\text{digit})^{+}$

无符号数分析图

digit (digit)* (. digit (digit)*|ε) (E (+|-|ε) digit (digit)* |ε)



num5 表示(digit)*

digits 表示(digit)+

digits → digit num5

num5 → digit num5 | ε

无符号数的正规文法

num → digit *num1*

num1 → digit *num1* | . *num2* | E *num4* | ε

num2 → digit *num3*

num3 → digit *num3* | E *num4* | ε

num4 → + *digits* | - *digits* | digit *num5*

digits → digit *num5*

num5 → digit *num5* | ε

示例：运算符

- 关系运算符的正规表达式为：

  < |<= | = | <> | >= | >

- 正规定义式：

  relop →< |<= | = | <> | >= | >

- 关系运算符的正规文法：

  relop → < | <*equal* | = |<*greater* | > | >*equal*

  greater → >

  equal → =

3.3.3 记号的识别

状态转换图：

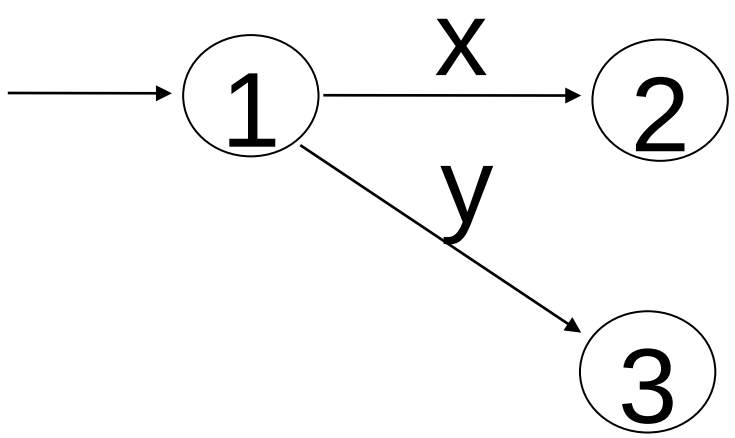
利用状态转换图识别记号，为线性文法构造相应的状态转换图。

状态集合的构成

状态之间边的形成

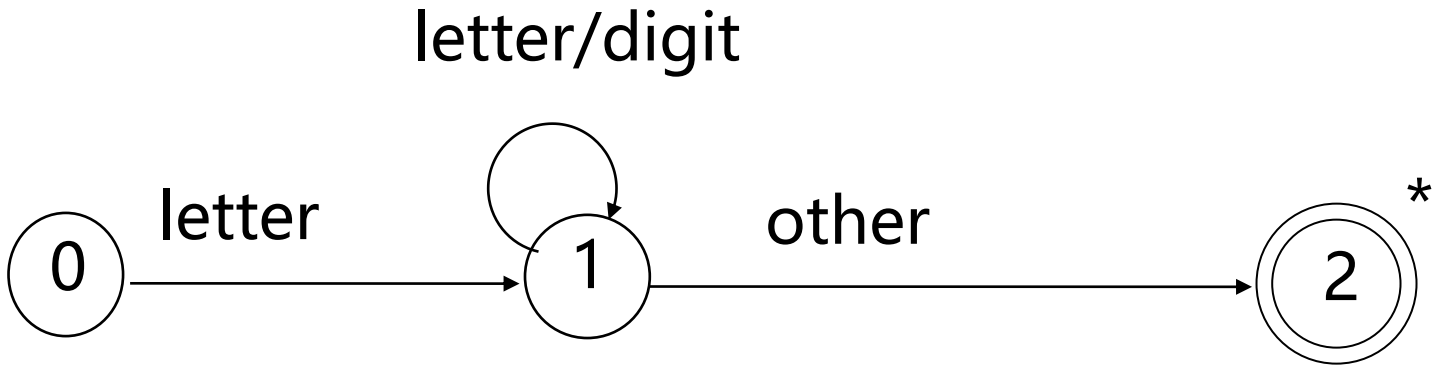
3.3.3.1 状态转换图

- 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。

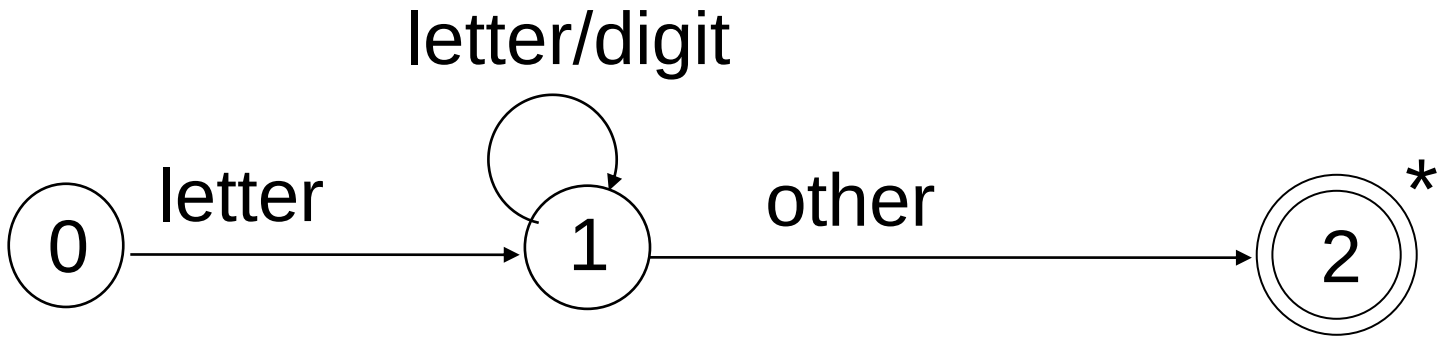


示例：标识符的状态转换图

- 标识符的文法产生式：
  id → letter *rid*
  rid → ε | letter *rid* | digit *rid*
- 标识符的状态转换图



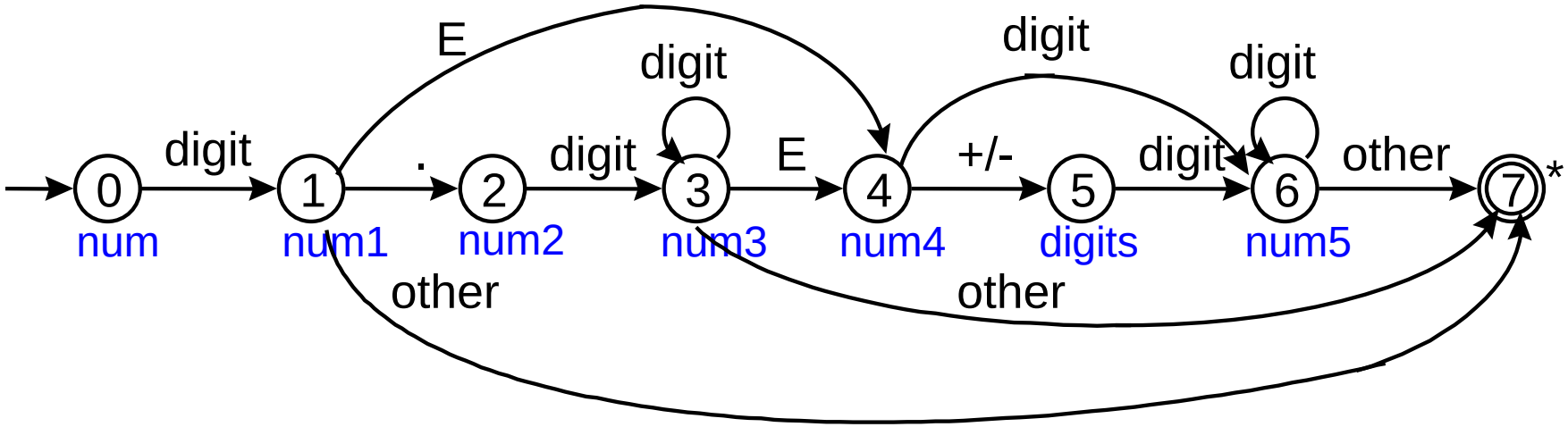
利用状态转换图识别记号



- 语句 DO99K=1.10 中标识符 DO99K 的识别过程

无符号数的右线性文法的状态转换图

num → digit *num1*
num1 → digit *num1* | . *num2* | E *num4* | ε
num2 → digit *num3*
num3 → digit *num3* | E *num4* | ε
num4 → + *digits* | - *digits* | digit *num5*
digits → digit *num5*
num5 → digit *num5* | ε



3.4 词法分析程序的设计与实现

手工编码实现法

✓ 相对复杂、且容易出错

✓ 目前非常流行的实现方法

GCC、LLVM…

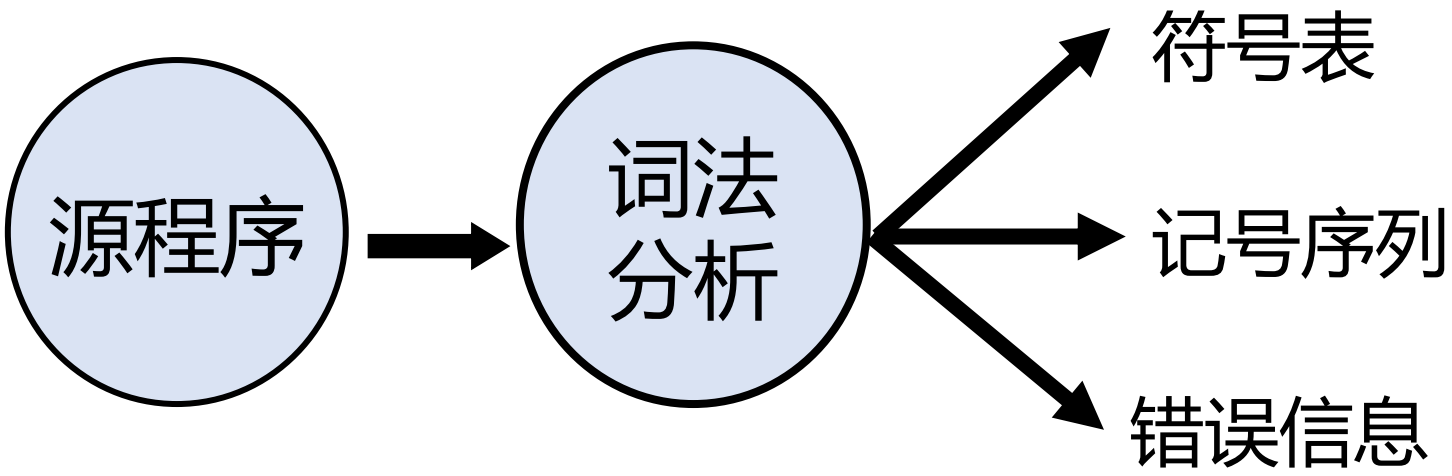
词法分析生成器

✓可快速原型、代码量较少

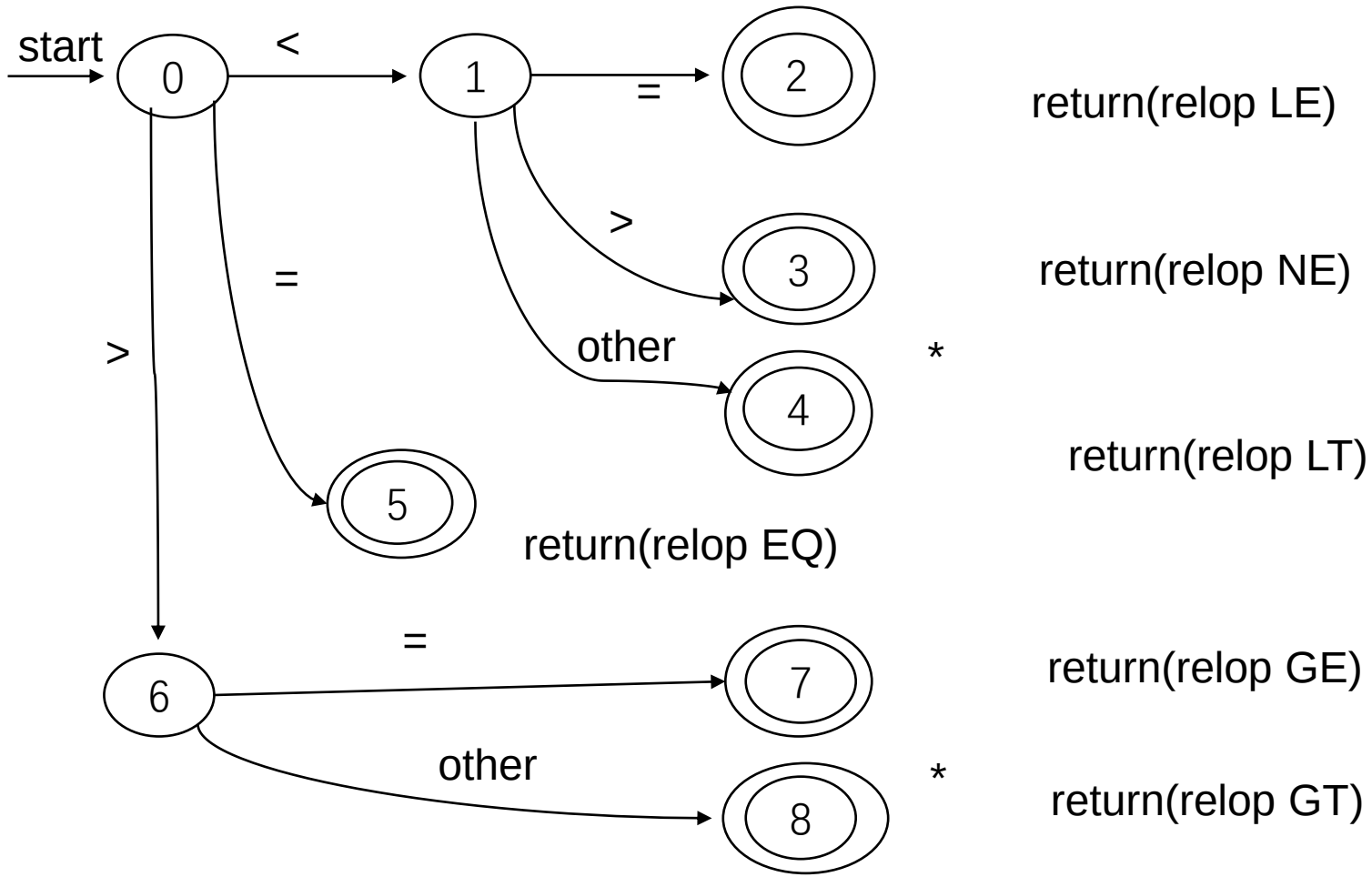
✓较难控制细节

3.4.1 手工构造词法分析器

- 词法分析的设计
- 如何高效的实现？算法和数据结构



3.4.1.1 转移图构造法


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

3.4.1.1 转移图构造法

```
Token nextToken    ( )
        c = getchar ();
        switch (c)
            case '<': c = getChar () ;
                        switch (c)
                            case '=' : return LE;
                            case '>' : return NE;
                            default: rollback(); return LT;
          case '=': return EQ;
          case '>': c = nextChar () ;
                        switch (c) …
```

3.4.1.1 手工构造词法分析器实例1

词法规则

状态转换图

词法分析程序

3.4.1.1 手工构造词法分析器实例1

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

3.4.1.1 手工构造词法分析器实例1

记号的正规文法

标识符的文法

id → letter *rid*

rid → ε | letter *rid* | digit *rid*

无符号整数的文法

digits → digit *remainder*

remainder → ε | digit *remainder*

无符号数的文法

num → digit *num1*

num1 → digit *num1* | . *num2* | E *num4* | ε

num2 → digit *num3*

num3 → digit *num3* | E *num4* | ε

num4 → + *digits* | - *digits* |digit *num5*

digits → digit *num5*

num5 → digit *num5* | ε

3.4.1.1 手工构造词法分析器实例1

记号的正规文法（续）

关系运算符的文法

relop → < | < *equal* | = | < *greater* | > | > *equal*

greater → >

equal → =

赋值号的文法

assign_op → :*equal*

equal → =

标点符号的文法

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

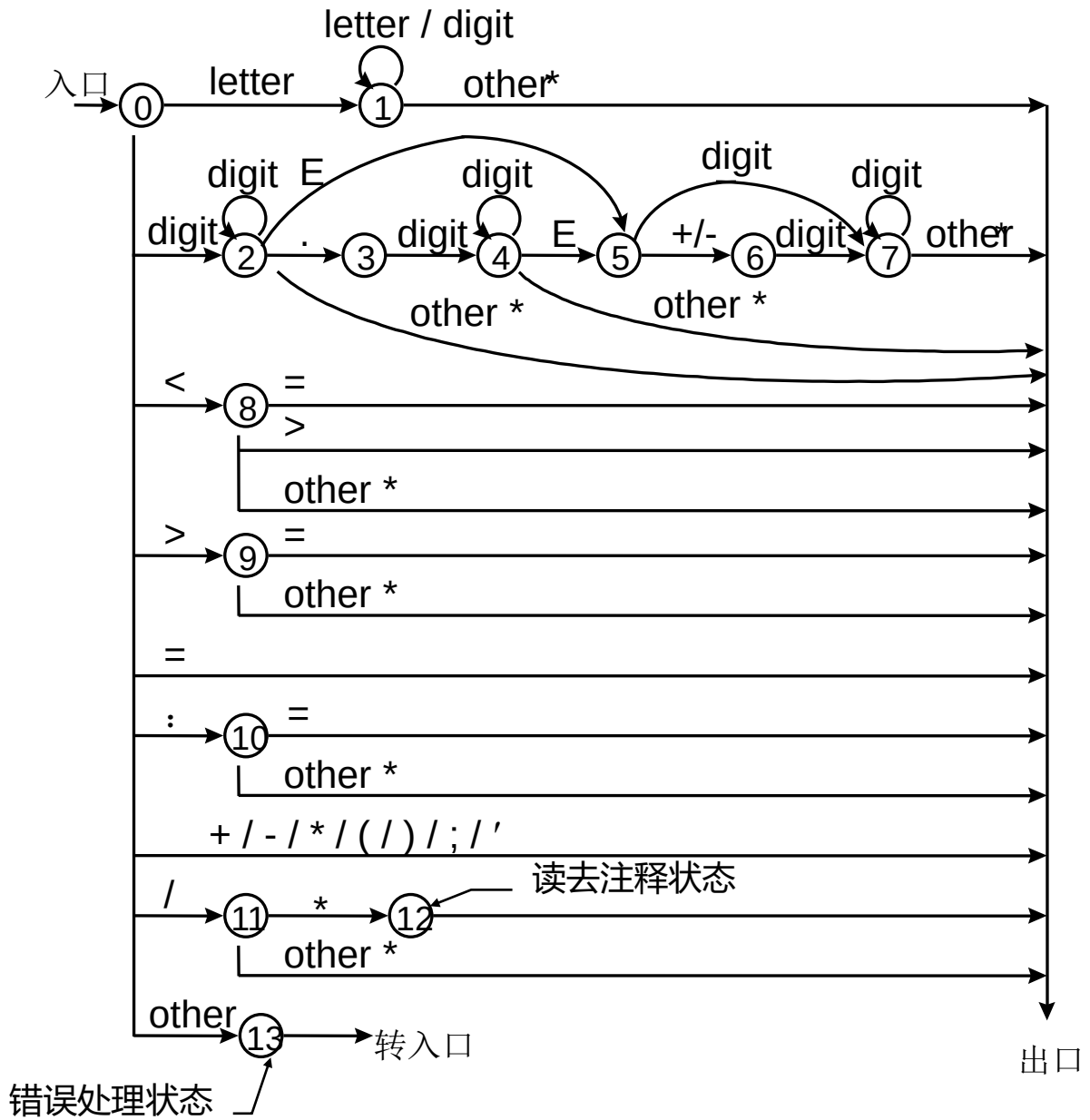
注释头符号的文法

note → / *star*

star → *

3.4.1.1 手工构造词法分析器实例1

编制词法分析程序



3.4.1.1 手工构造词法分析器实例1

- 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：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“<”。

3.4.1.1 手工构造词法分析器实例1

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>

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	-
+	+	-
-	-	-
*	*	-
/	/	-
(	(	-
)	)	-
'	'	-
;	;	-
:	:	-

3.4.1.1 手工构造词法分析器实例1

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。

3.4.1.1 手工构造词法分析器实例1

设计全局变量和过程

(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)SToI：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。

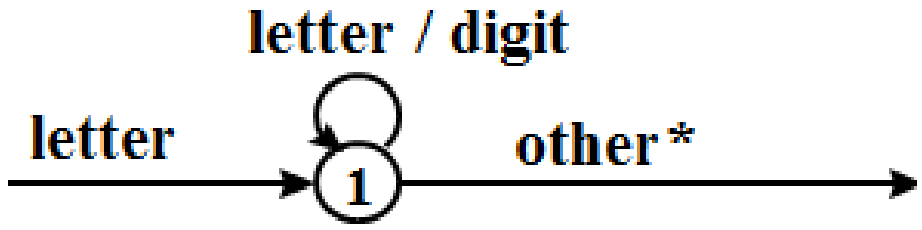
3.4.1.1 手工构造词法分析器实例1

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;
```

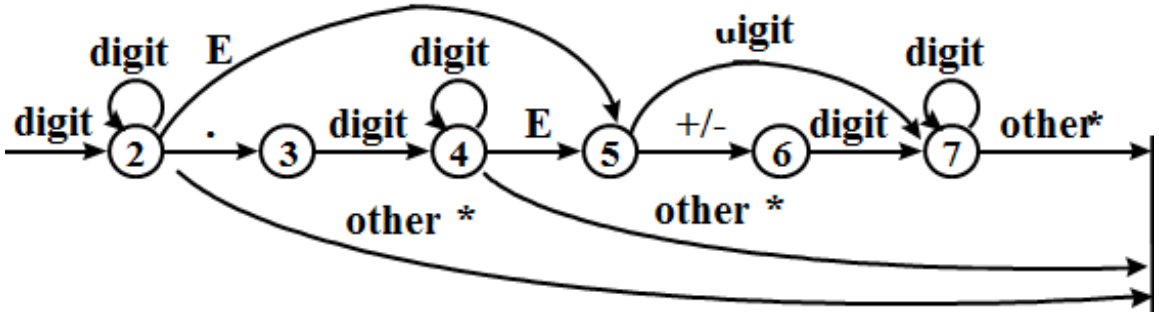
词法分析程序--类C语言描述

```
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;
```



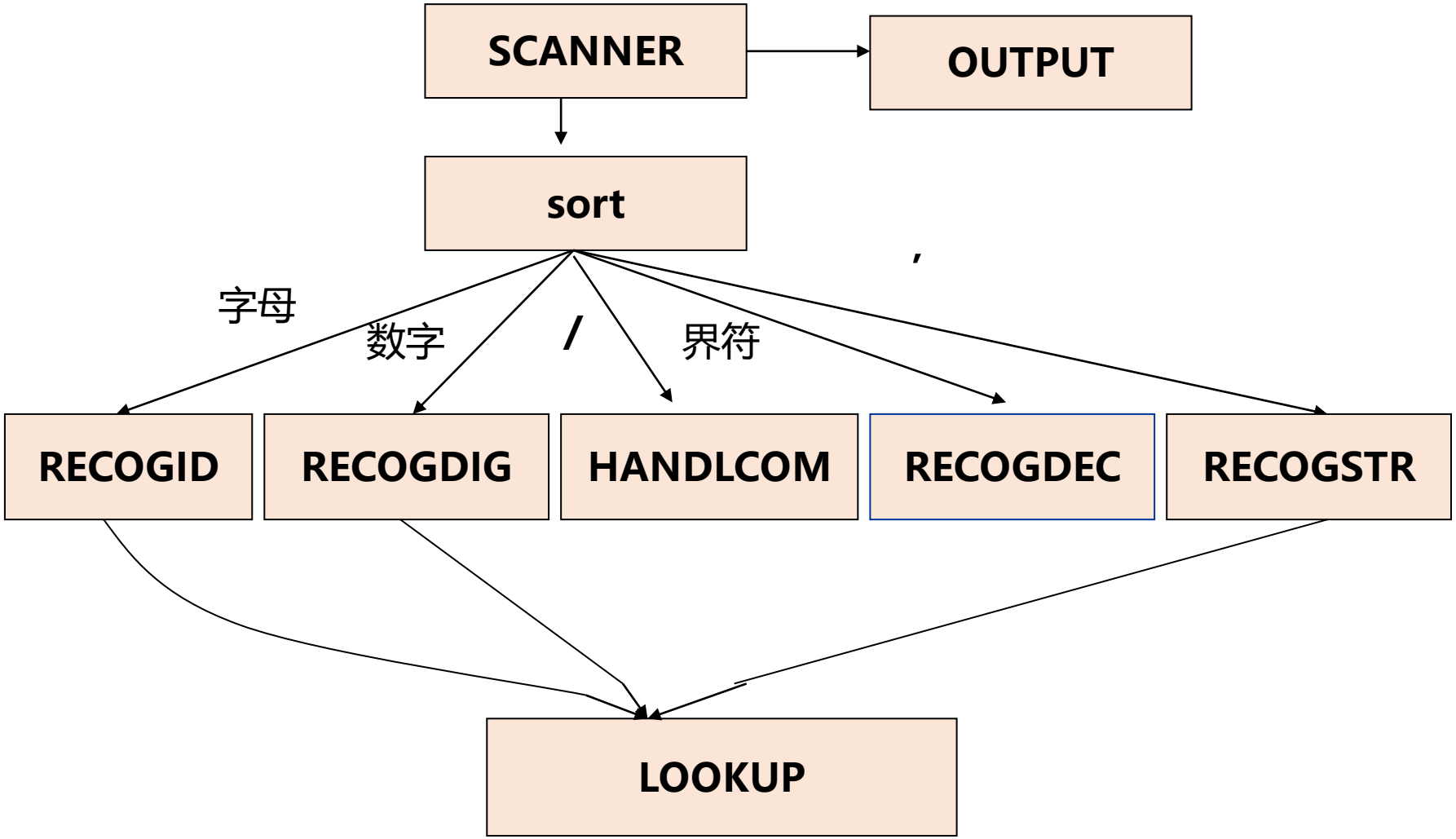
词法分析程序--类C语言描述

```
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, STOI(token));      // 返回整数
      break;
   };
   break;
```



3.4.1.2 手工构造词法分析器实例2

类pascal语言simple语言词法分析器



3.4.1.2 手工构造词法分析器实例2

```
program SCANNER;
Begin  initiate符号表,字符串表,行,列计数器;
Open 源文件,TOKEN文件,打印机文件;
Repeat
    FIRSTCH(CH);
    if CH!=EOL then
          call SORT(CH)
    else    RDLINE;
until CH=EOF;
把符号表,字符串表做成文件;
close源文件,TOKEN文件;
call OUTPUTR;
```

模块SCANNER：

词法分析器主控

3.4.1.2 手工构造词法分析器实例2

记号分类模块(SORT)输入：

CH内含记号首符；

```
procedure SORT(CH);
{   case CH of ‘字母’ :
          ‘字母’ :   call RECOGID(CH,TOKEN);
           ‘/’:       call HANDLECOM(CH,TOKEN);
          ‘数字’ :   call RECOGDIG(CH,TOKEN);
          ‘‘‘          call RECOGSTR(CH,TOKEN);
          otherwise call RECOGDEL(CH,TOKEN);
    end case;
    write TOKEN into TOKEN文件;
    Return   }
```

模块SORT：

词法分析器主控

3.4.1.2 手工构造词法分析器实例2

记号分类模块(SORT)输入：

CH内含记号首符；

```
procedure SORT(CH);
{   case CH of '字母' :
          '字母'  :   call RECOGID(CH,TOKEN);
           '/':       call HANDLECOM(CH,TOKEN);
          '数字'  :   call RECOGDIG(CH,TOKEN);
          ''''        call RECOGSTR(CH,TOKEN);
          otherwise call RECOGDEL(CH,TOKEN);
    end case;
    write TOKEN into TOKEN文件;
    Return   }
```

模块SORT：

词法分析器主控

3.4.1.2 手工构造词法分析器实例2

```
procedure HANDLECOM(TOKEN);
{   call GETCH(CH);
if CH!='*' then
  {     列计数-1;
        TOKEN=('/'的识别码,_);
        return   };
TOKEN='-1';
GETCH(CH);
while 列计数<=行长-1  do
   {  CH1:=CH;
      call GETCH(CH);
      if CH1='*' and CH='/'     then    TOKEN:=' ';   }
if TOKEN!=' ' then call PRINTERR('注解未完');
TOKEN:=' ';
return   }
```

处理注解(HANDLECOM);

输入：　'/';进入该模块之前已扫描了一个字符　'/'

输出：　'/'的TOKEN字或空TOKEN字;

3.4.1.2 手工构造词法分析器实例2

识别界限符(RECOGDEL)

输入：CH内含单界限符；

输出：各种界符的TOKEN字；

```
procedure RECOGDEL(CH,TOKEN);
{ case CH of
    '+':  TOKEN:=('+'的种别码,_);
     ')':  TOKEN:=(')'的种别码,_);
    '<':   {       call GETCH(CH);
                    if  CH='='then  TOKEN:=('<='的种别码,_)
                   else   if CH='>' then TOKEN:=('<>'的种别码,_)
                          else {    列计数-1;TOKEN:=('<'的种别码,_)  }
              }
    ……..
    endcase;
   return    }
```

识别界限符；

输入：CH中含界限符的首字母；

输出：TOKEN(二元式形式);

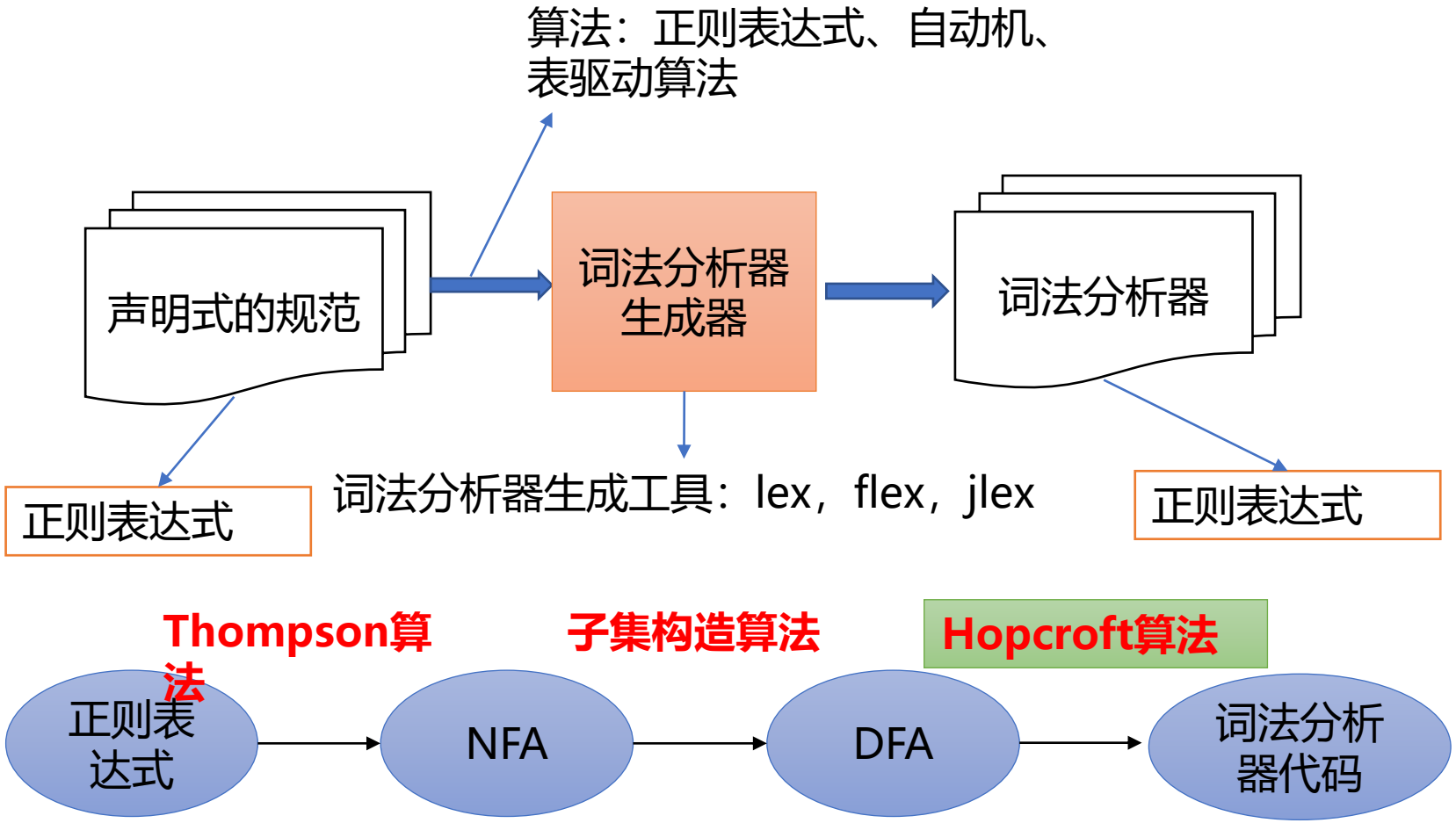
3.4.1.4 符号表

符号表用来存放在程序中出现的各种标识符及其语义属性.
符号表的结构如何设计？在具体编码实现时，可以用什么数据结构来实现?

3.5 词法分析器的自动生成

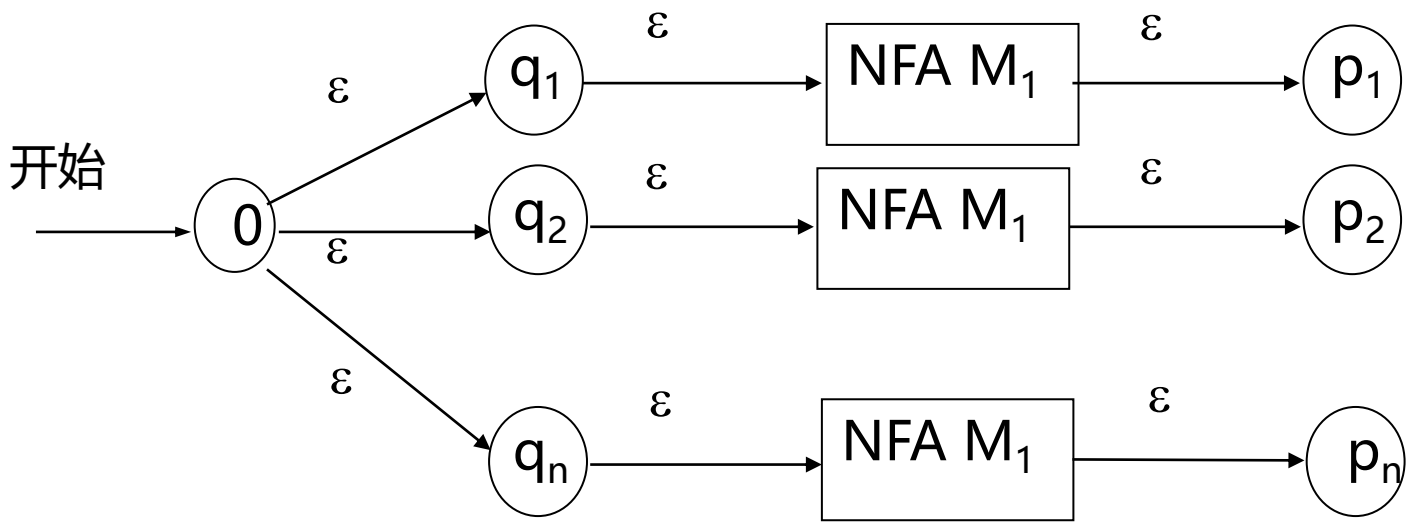
3.5 词法分析器的自动生成

3.5.1 词法分析器的自动生成原理



3.5.1 词法分析器的自动生成原理

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



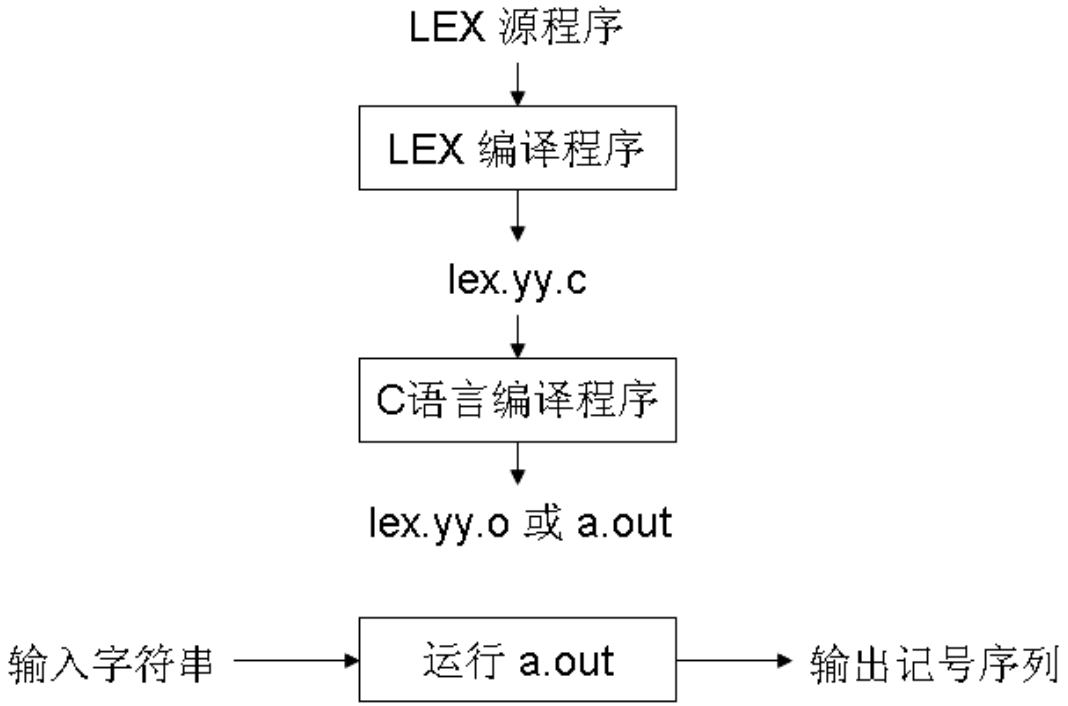
将NFA M确定化为DFA D，并生成该DFA D的状态转换矩阵和控制执行程序。

3.5.2 词法分析器生成器LEX简介

- LEX使用流程
- LEX源程序结构
- LEX工作原理

3.5.2 词法分析器生成器LEX简介

3.5.2.1 LEX使用流程



3.5.2.2 LEX源程序结构

一个LEX源程序由三部分组成：

1、声明部分

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。

2. 转换规则

形式：

P1　　{ 动作1 }

…

Pn　　{ 动作n }

Pi 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Pi时，词法分析器应执行的动作。

3. 辅助函数

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。

声明部分
%%
转换规则
%%
辅助函数

3.5.2.2 LEX源程序结构

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。

3.5.2.2 LEX源程序结构

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。

3.5.2.2 LEX源程序结构

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}+)?
%%
```

3.5.2.2 LEX源程序结构

相应的LEX源程序框架

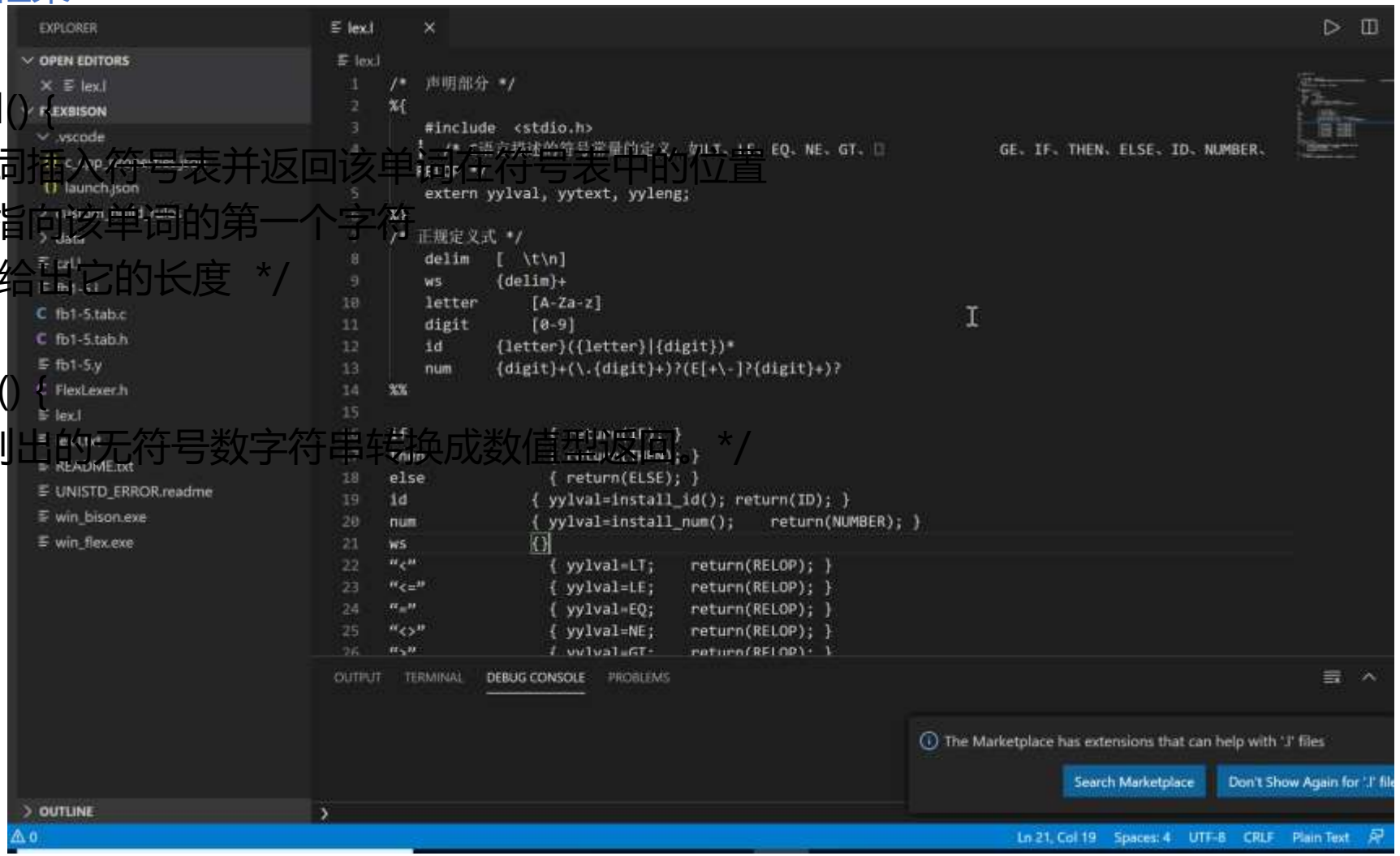
```
/* 规则部分 */
    {ws}        {/* 没有动作，也不返回 */}
    if           { return(IF); }
    then         { return(THEN); }
    else         { return(ELSE); }
    {id}        { yylval=install_id(); return(ID); }
    {num}       { yylval=install_num();    return(NUMBER); }
    “<”         { yylval=LT;    return(RELOP); }
    “<=”        { 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！！

3.5.2.2 LEX源程序结构

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符串转换成数值型返回  */
    }
```



3.5.2.3 其他词法分析器自动生成器

- Flex & Bison
  win-flex：
  http://gnuwin32.sourceforge.net/packages/flex.htm
- JavaCC(Java Compiler Compiler)
  https://javacc.github.io/javacc/
- Antlr (ANother Tool for Language Recognition)
  https://www.antlr.org/

课后作业 1

1、构造C语言的词类编码表

2、编写程序，对C语言中十进制、八进制及十六进制数进行识别。

输入：数字,例如0177777、0X2A

输出：记号（作业1的词类编码表）

课后拓展

1、阅读资料flex&bison中flex相关内容
2、试用一种词法分析器生成工具

课后作业 2

采用一种词法分析器自动生成工具，实现C语言的词法分析器，要求待分析的语言为C语言的子集，至少包含：

整数/字符常数/布尔常数
算术运算/布尔运算
If-then-else 语句
While 语句
注释
分隔符