

§ 3.5 词法分析程序的实现

§ 3.5.1 词法分析程序的编写

正则文法  状态转换图  程序框图

1、单词符号的正则文法

G[<单词符号>]:

<单词符号> \rightarrow <标识符> ε | <无符号整数> ε |

<标识符> \rightarrow 字母 | <标识符>字母 | <标识符>数字

<无符号整数> \rightarrow 数字 | <无符号整数>数字

<单字符分界符> \rightarrow + | - | * | , | ; | (|)

<双字符分界符> \rightarrow <大于>= | <小于>= | <小于>>
| <等于>= | <冒号>= | <斜竖> *

<小于> \rightarrow <

<等于> \rightarrow =

<大于> \rightarrow >

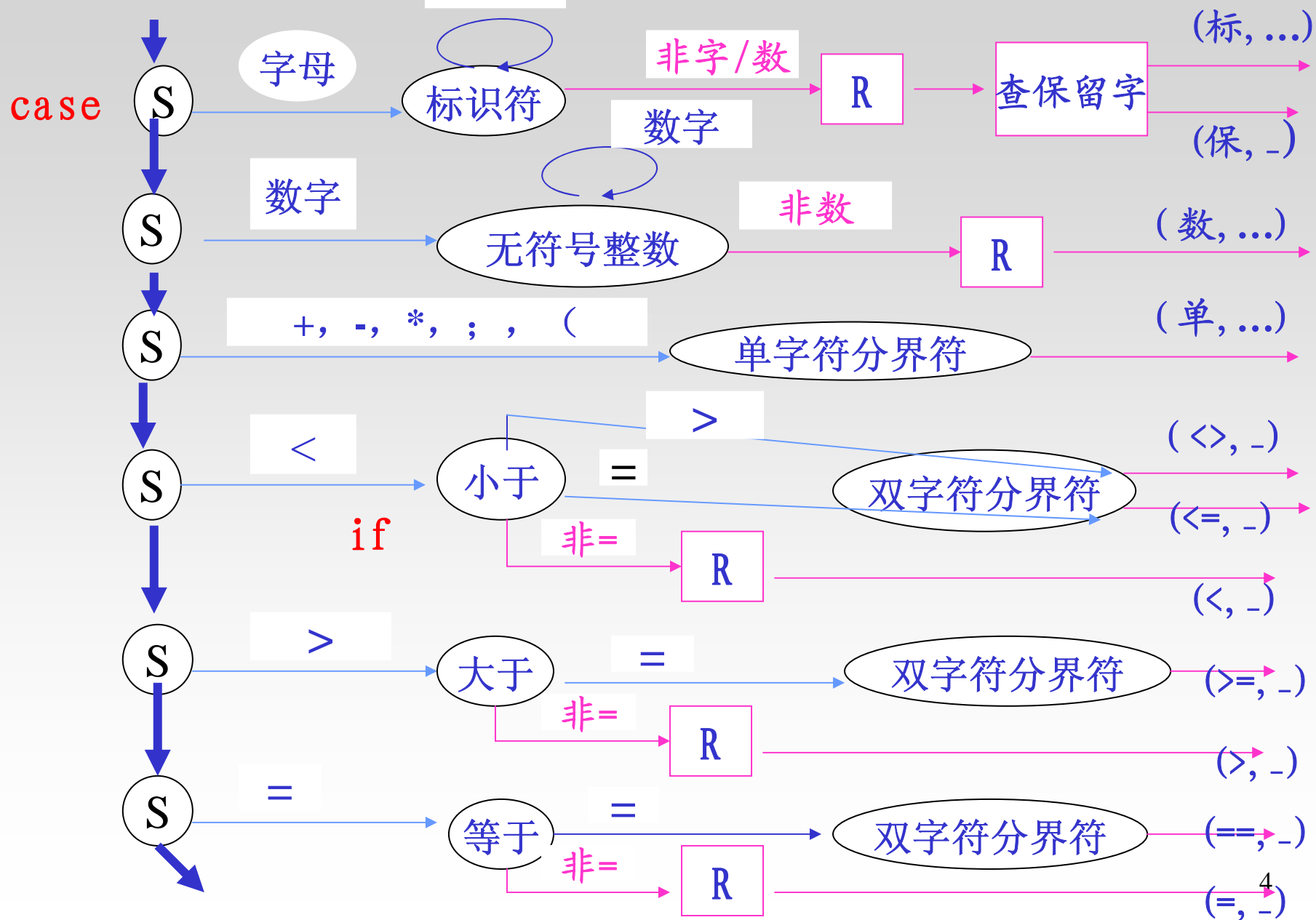
<冒号> \rightarrow :

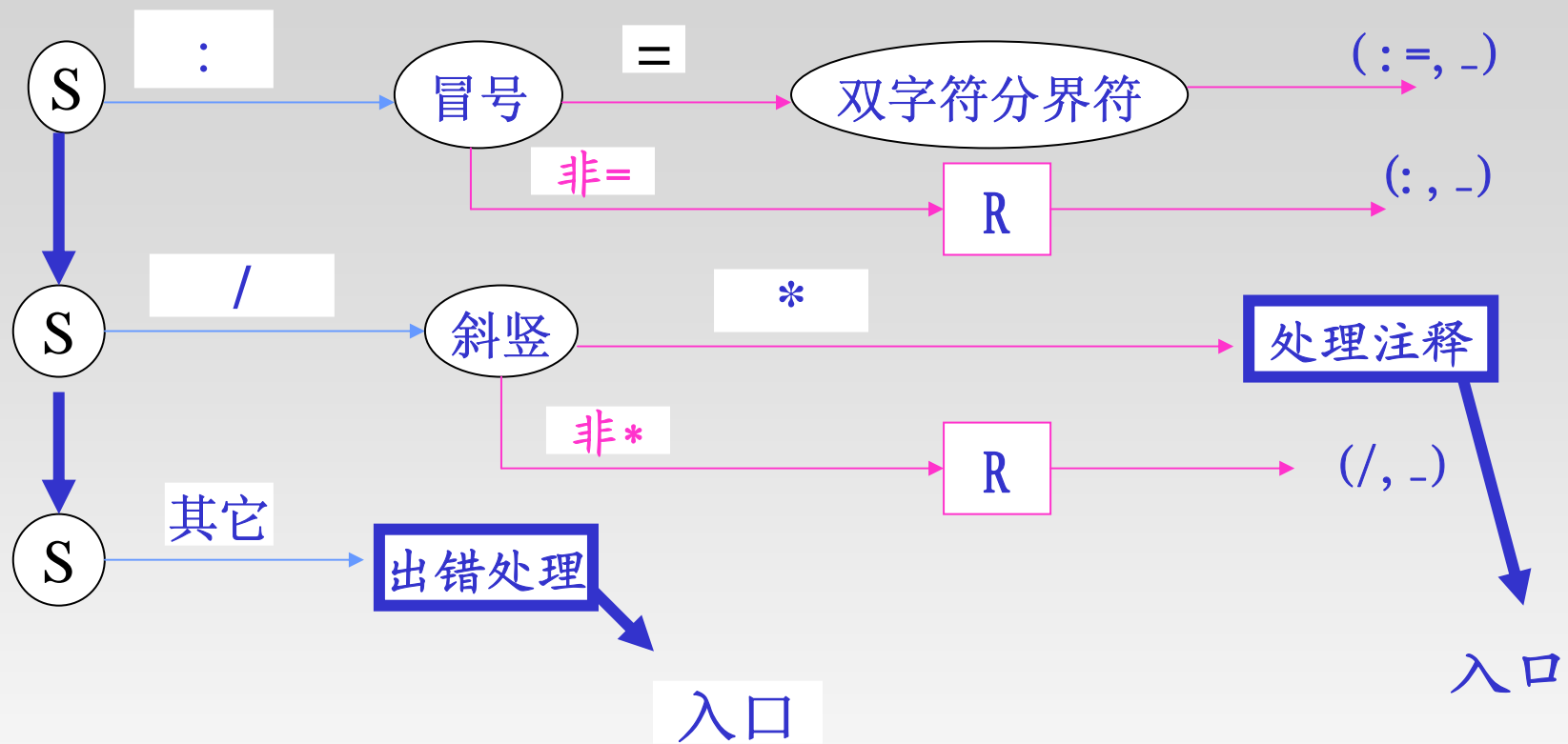
<斜竖> \rightarrow /

2、单词的类别编码

单词符号	类别编码	类别码的助记符
begin	1	BEGIN
end	2	END
if	3	IF
then	4	THEN
else	5	ELSE
标识符	6	ID
整数	7	INT
<	8	LT
<=	9	LE
=	10	EQ

3、状态转换图





4、程序的编写

语义变量及语义函数

函数GETCHAR 每调用一次,就把扫描指示器当前所指示的源程序字符送入变量ch,然后把扫描器前推一个字符位置.

字符数组TOKEN:用来依此存放一个单词词文中的各个字符.

函数CAT:每次调用,就把当前ch中的字符拼接于TOKEN中所存字符串的右边.

函数LOOKUP: 每调用一次,就以TOKEN中的字符串查保留字,若查到,就将相应的关键字的类别码赋给整型变量c,否则将c置为零.

函数RETRACT:每调用一次,就把扫描指示器回退一个字符位置(即退回多读的那个字符)

函数OUT:一般仅在进入终态时调用此函数,调用的形式为OUT(c,VAL).其中,实参c为相应单词的类别码或助记符;当所识别的单词为标识符和整数时,实参VAL为TOKEN(即词文分别为字母和数字串),对于其余种类的单词,VAL均为空

串.函数OUT的功能是,在送出一个单词的内部表示之后,返回到调用该词法分析程序的那个程序.

```
#define ID 6  
#define INT 7  
#define LT 8  
#define LE 9  
#define EQ 10  
#define NE 11  
#define GT 12
```



```
#define GE 13
char TOKEN[20];
extern int lookup(char *);
extern void out(int, char*);
extern report_example(FILE *fp)
{
    char ch;
    int I, c;
    ch=fgetc(fp);
    if (isalpha(ch)) /*it must be a identifier*/
```

```
{  
    TOKEN[0]=ch;  
    ch=fgetc(fp); i=1;  
    while(isalnum(ch))  
    {  
        TOKEN[i]=ch;i++;  
        ch=fgetc(fp);  
    }  
}
```

```
TOKEN[I]='\0'  
fseek(fp,-1,1);    /*retract*/  
c=lookup(TOKEN);  
if(c==0) out(ID,TOKEN);  
    else out(c, “ ”);  
}  
else  
if(isdigit(ch))
```

```
{  
    TOKEN[0]=ch;  
    ch=fgetc(fp);i=1;  
    while(isdigit(ch))  
    {  
        TOKEN[i]=ch;i++;  
        ch=fgetc(fp);  
    }  
    TOKEN[i]='\0';  
    fseek(fp,-1,1);  
    out(INT,TOKEN);  
}
```

```
}  
else  
    switch(ch)  
    {  
        case '<': ch=fgetc(fp);  
            if(ch == '=') out(LE," ");  
            else if (ch == '>')out (NE," ");  
            else  
            {  
                fseek(fp,-1,1);  
                out(LT," ");  
            }  
        break;
```

```
case '=': out(EQ," ");break;
case '>': ch=fgetc(fp);
           if(ch== '=')out(GE," ");
           else
           {
               fseek(fp,-1,1);
               out(GT," ");
           }
           break;
default:report_error();
        break;
    }
    treturn
}
```

§ 3.5 .2 词法分析程序的自动生成

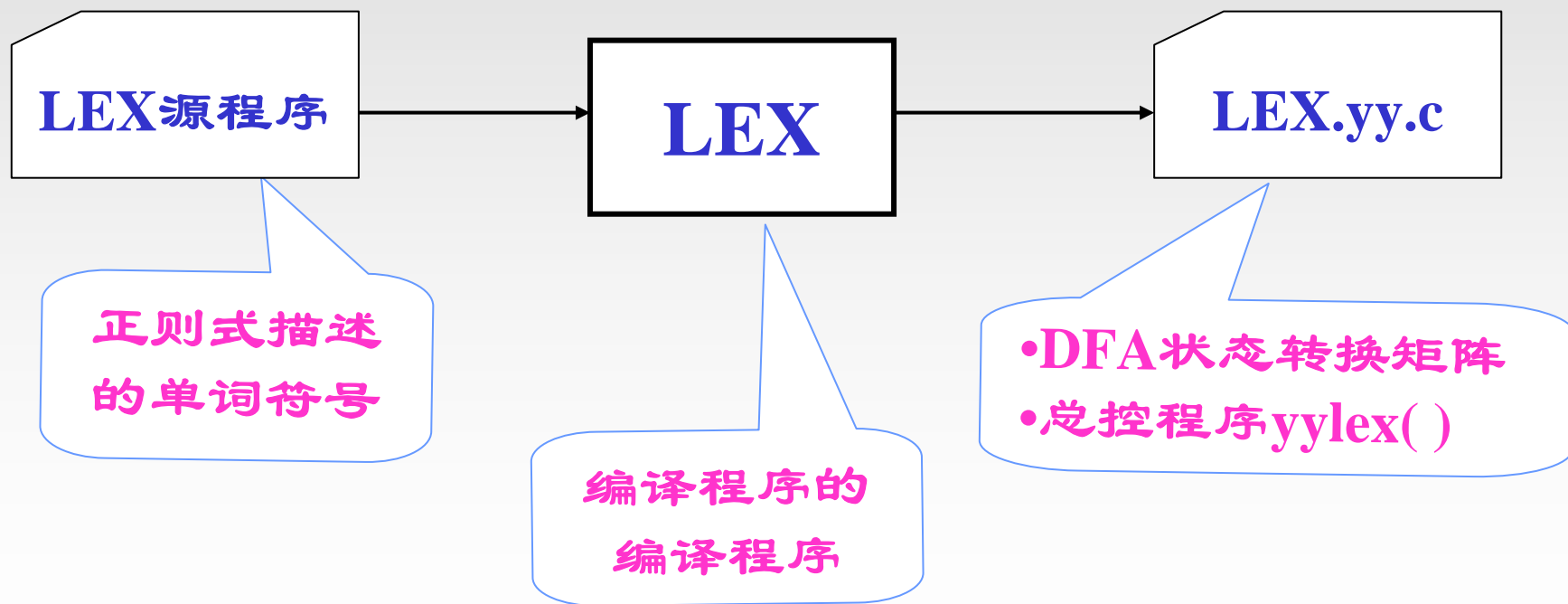
LEX:美国Bell实验室用C语言研制的一个
UNIX操作系统下的词法分析程序自
动生成工具。

主要内容

- LEX工具
- LEX源程序
- LEX的实现

1、LEX概述

LEX是一个词法分析器的自动产生系统。



2、LEX源程序 (C语言的风格)

形式

{正则式辅助定义式}

%%

{识别规则}

%%

{辅助函数部分}

(1)正则式辅助定义式

$$D_i \rightarrow R_i$$

其中：

① R_i 是正则表达式， D_i 是其简名

② R_i 中只能出现 V_t 中字符及

$$D_1, D_2, \dots, D_{i-1}$$

(2)识别规则

$$\begin{array}{ll} P_1 & \{A_1\} \\ P_2 & \{A_2\} \\ & \vdots \\ P_n & \{A_n\} \end{array}$$

其中：

- ① P_i 是定义在 V_t 及 $\{D_1, D_2, \dots, D_n\}$ 上的正则式，也称词形。
- ② $\{A_i\}$ 为与 P_i 对应的语义动作（一段程序），识别出词形 P_i 后，词法分析器应采取的动作。

(3)辅助函数部分：给出用户所需要的其他操作，是对识别规则的补充，识别规则中某些动作需要调用的过程，如不是C的库函数，则要给出具体的定义。

讨论：

- ①识别规则是LEX的核心，完全决定了词法分析器的功能，该分析器只能识别词形为 P_i 的单词符号。
- ②LEX是借助其宿主语言C完成工作的。

AUXILIARY DEFINITIONS

letter → **A | B | | Z | a | b | | z**

digit → **0 | 1 | 2 | | 9**

% %

RECOGNITION RULES

1 BEGIN {return(1,_)}

2 END {return(2,_)}

3 IF {return(3,_)}

4 THEN {return(4,_)}

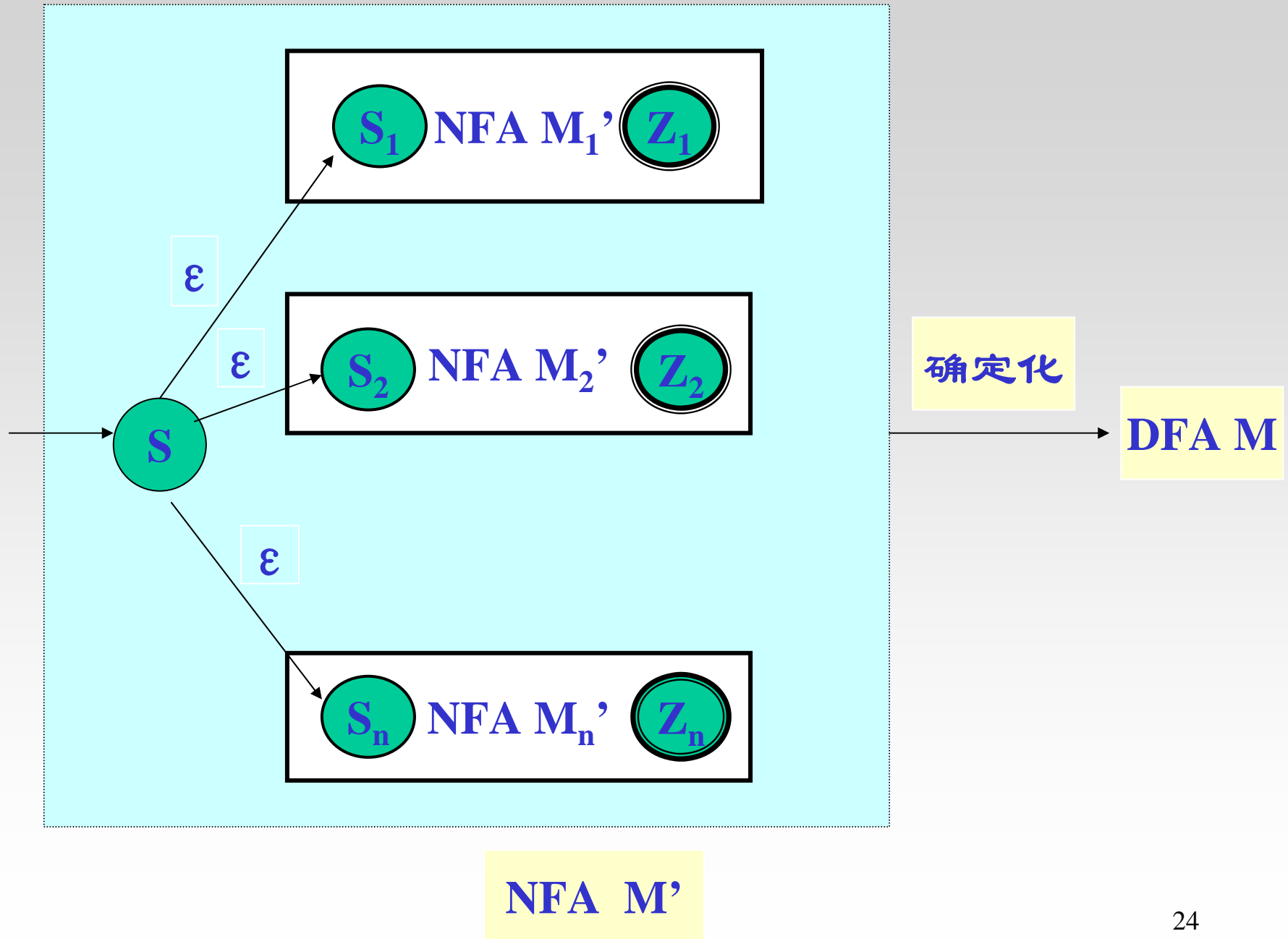
5 ELSE {return(5,_)}

6 letter{letter digit}	{ return(20,TOKEN)}
7 digit{digit}	{return(21,DTB)}
8 <	{ }
9 <=	{ }
.....	
% %	

3、LEX的实现

(1)工作过程

- ①由每条 P_i 构造一NFA M_i'
- ②将各 M_i' 合并成一新的NFA M_i'
加入 ε 弧
- ③将NFA M_i' 确定化为DFA M



2、词法分析器(DFA M)

LEX.yy.c { 状态转换矩阵
总控程序(yylex())



3、处理二义性问题的两个原则

begin $P_1(\text{begin})$, $P_6(\text{letter}(\text{letter} \mid \text{digit})^*)$ 匹配

\leq $P_8(<)$, $P_9(\leq)$ 匹配

原则:

(1) 最长匹配原则:

能匹配最多字符的规则优先 $\leq P_9$

(2) 优先匹配原则:

满足最长匹配的情况下,一个字符串可

与若干 P_i 匹配, P_i 在前的优先. **Begin** P_1

例：用C语言编写的词法分析程序

- `#include<stdio.h>`
- `#include<ctype.h>`
- `char * progame;`
- `# define NUMBER 400`
- `# define COMMENT 401`
- `# define TEXT 402`
- `#define COMMAND 403`

- `main(argc argv)`
- `ing argc;`
- `char * argv[];`
- `{`
- `int val;`
- `while(val=lexer())prints(“value is
%d\n”,val);`
- `}`

- **lexer()**
- **{**
- **int c;**
- **while ((c=getchar())== ' ' || c== '\t ') ;**
- **if (c==EOF) return 0;**
- **if (c== '.' || isdigit (c)) { /*数字*/**
- **while ((c=getchar())!=EOF && isdigit (c));**
- **if (c== '.') while ((c=getchar())!=EOF &&**
- **isdigit (c));**
- **ungetc (c, stdin);**
- **return NUMBER;**
- **}**

- **if (C== '# '){ /*注释***
- **while ((c=getchar())!=EOF && c!= '\n ');**
- **ungetc(c, stdin);**
- **return COMMENT;**
- **If (c== ' " '){ /*文本*/**
- **while((c=getchar())!=EOF**
- **&&c!= ' " ' &&c!= '\n ');**
- **If (c== '\n ') ungentic(c,stdin);**
- **Return TEXT;**
- **}**

```
If(isalpha( c)){ /*检查是否是一个命令*/  
    while ((c=getchar( ))!=EOF &&isalnum(c ));  
        ungetc(c,stdin);  
        return COMMAND;  
    }  
Return c;  
}
```

- 例：用LEX编写的同样的词法分析程序
- % {
- #define NUMBER 400
- #define COMMENT 401
- #define TEXT 402
- #define COMMAND 403
- % }
- % %

[\t] ;

[0-9]+ |

[0-9]+\.[0-9]+ |

\.[0-9]+ {return NUMBER;}

#.* {return COMMENT;}

\ "[^\" \n]*\" {return TEXT;}

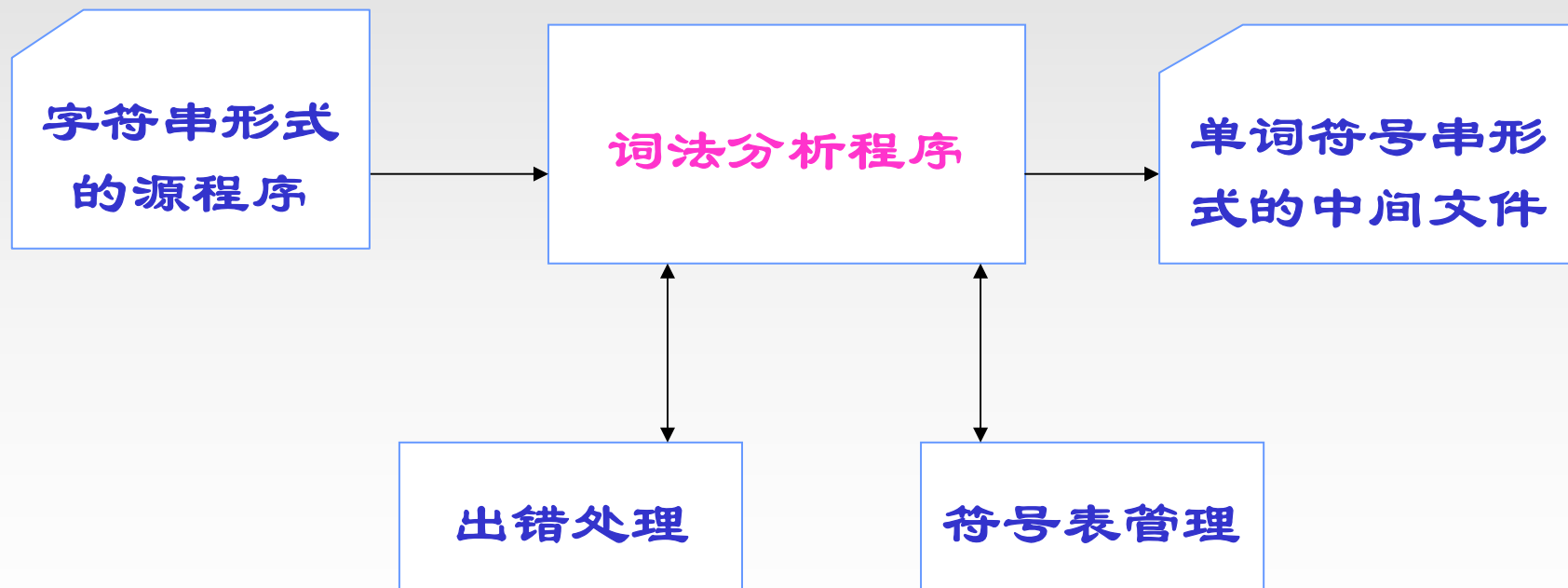
[a-zA-Z][a-zA-Z0-9]+ {return COMMAND;}

\n {return '\n '};}

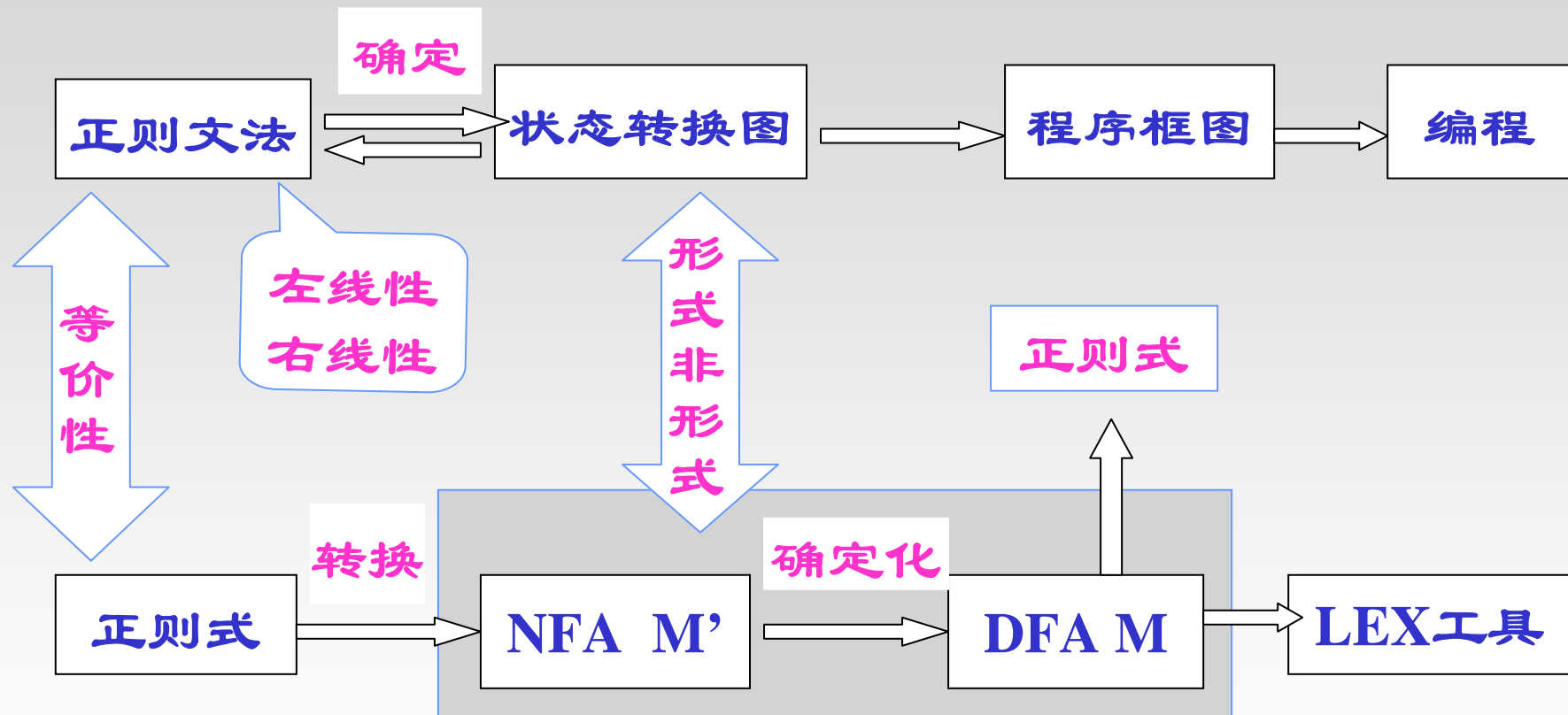
% %

```
#include <stdio.h>
Main(argc,argv)
Int argc;
Char *argv[];
{
Int val;
While(val=yylex( ))printf(“value is%d\n”,val);
}
```

词法分析总结



词法分析程序的手工编写



词法分析程序的自动生成

第三章作业

P_{97}

3-3,

3-6,

3-7,

3-8,

3-9,

3-12(3)(4)

3-13,

3-20,

3-22(1)(2)