第三章 词法分析

§3.1 设计扫描器时应考虑的问题

§3.2 正则文法和状态转换图

§3.3 有限自动机

§3.4 正则表达式和正规集

§3.5 词法分析程序的实现

§3.5 词法分析程序的实现

§3.5.1 词法分析程序的编写

根据各类单词的描述和定义，手工构造词法程序

§3.5 .2 词法分析程序的自动生成

各类单词的正规式+语义处理工作，通过词法分析程序的构造程序进行加工，自动生成词法分析程序（更加抽象）

§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> → <标识符> ε | <无符号整数> ε | .....

<标识符>→字母 | <标识符>字母 | <标识符>数字

<无符号整数>→数字 | <无符号整数>数字

<单字符分界符> →+ | － | * | , | ; | (|)

<双字符分界符>→<大于>= | <小于>= | <小于>>

| <等于>= | <冒号> = | <斜竖> *

<小于>→<

<等于>→=

<大于>→>

<冒号> →:

<斜竖> →/

2、单词的类别编码

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

几点重要限制说明

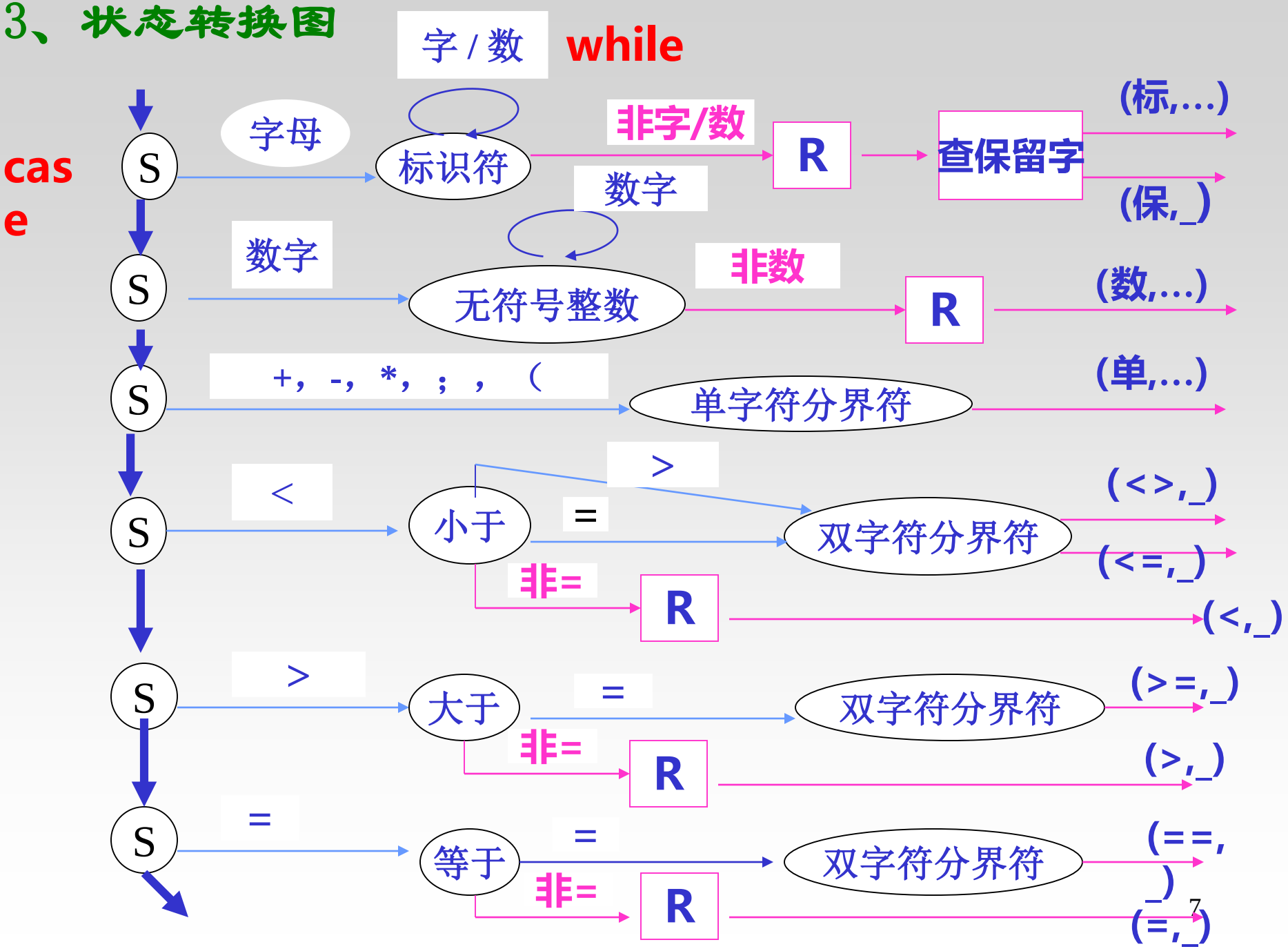
- 所有关键字都是保留字;用户不能用它们作自己的标识符
- 关键字作为特殊的标识符来处理;不用特殊的状态图来识别，只要查保留字表。
- 如果关键字、标识符和常数(或标号)之间没有确定的运算符或界符作间隔，则必须使用一个空白符作间隔。

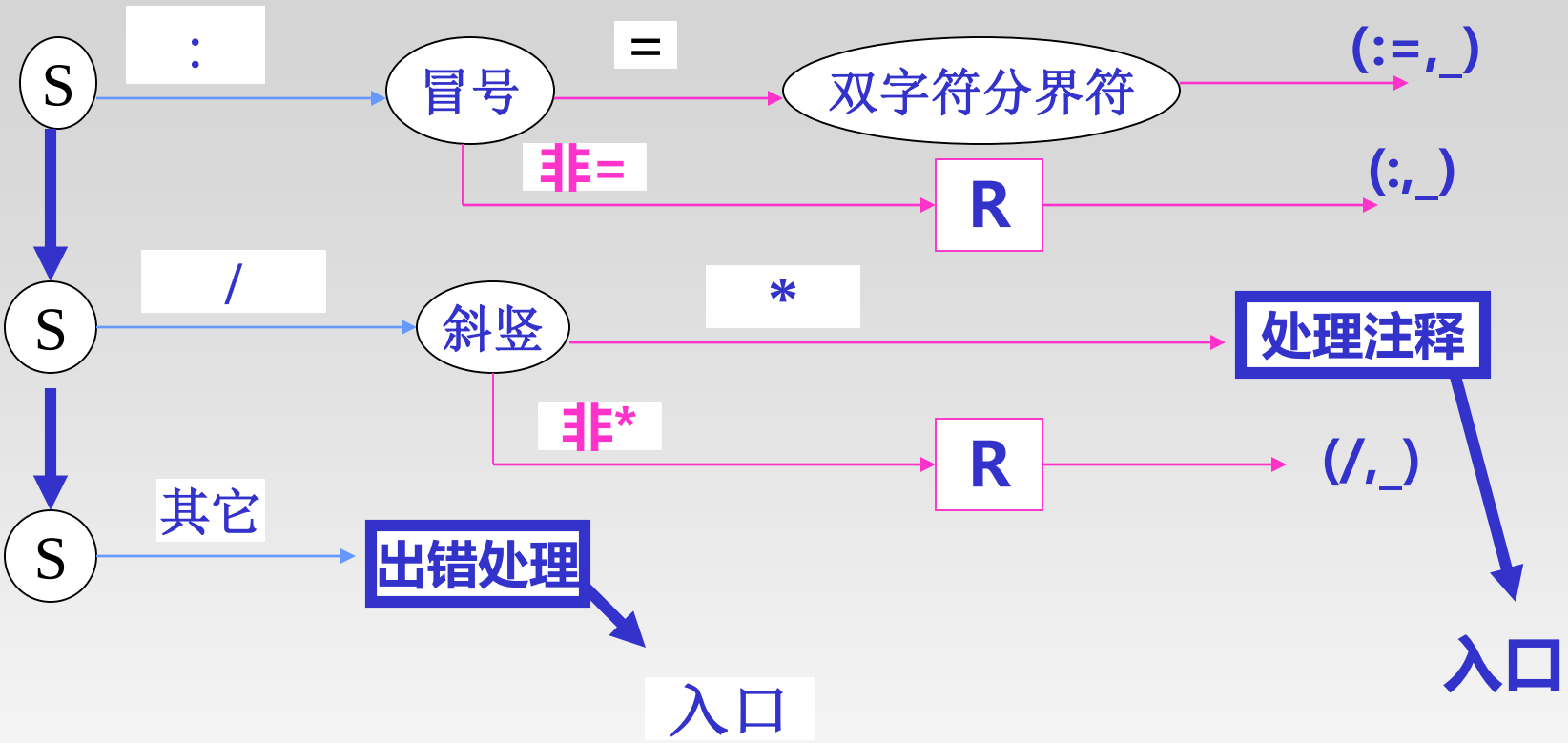
```
DO99K=1, 10
```

要写成

```
DO 99 K=1, 10
```

3、状态转换图


字 / 数
while
case
S
字母
标识符
非字/数
R
查保留字
(标,...)
(保,_)
数字
数字
无符号整数
非数
R
(数,...)
+，-，*，；，（
单字符分界符
(单,...)
<
小于
=
>
双字符分界符
(<>,_)
(<=,_)
非=
R
(<,_)
>
大于
=
双字符分界符
(>=,_)
非=
R
(>,_)
=
等于
=
双字符分界符
(==,_)
非=
R
(=,_)


S
:
冒号
=
双字符分界符
(:=,_)
非=
R
(:,_)
S
/
斜竖
*
处理注释
非*
R
(/,_)
S
其它
出错处理
入口
入口

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,就把扫描指示器回腿一个字符位置(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  identifer*/
{
   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* 回退功能/
c=lookup(TOKEN);// 查保留字，赋类别值，否则置零
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

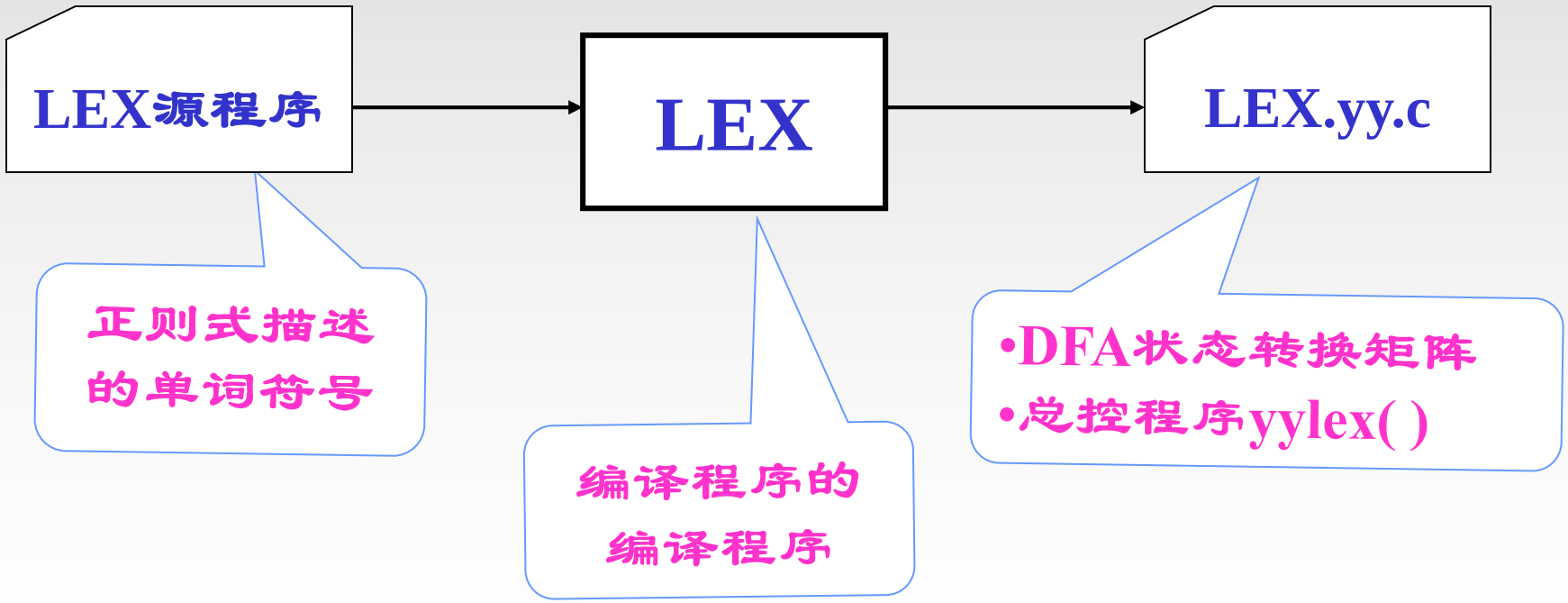
LEX:美国Bell实验室用C语言研制的一个UNIX操作系统下的词法分析程序自动生成工具.

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}

%% //分割符号

{识别规则}

%% //分割符号

{辅助函数部分}

(1)正则式辅助定义式 （宏定义）

D_i R_i

其中：

①R_i是正则表达式，D_i是其简名

②R_i中只能出现V_t中字符及

D_1，D_2，…，D_{i-1}

例如：digit [0—9]

alnum ({alpha} | {digit})

(2)识别规则

P_1 $\{A_1\}$

P_2 $\{A_2\}$

：

P_n $\{A_n\}$

其中：

①P_i是定义在V_t及$\{D_1, D_2, \ldots D_n\}$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 //辅助定义

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

digit →0 | 1 | 2 | ……. | 9

%%

RECOGNITION RULES //识别规则

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

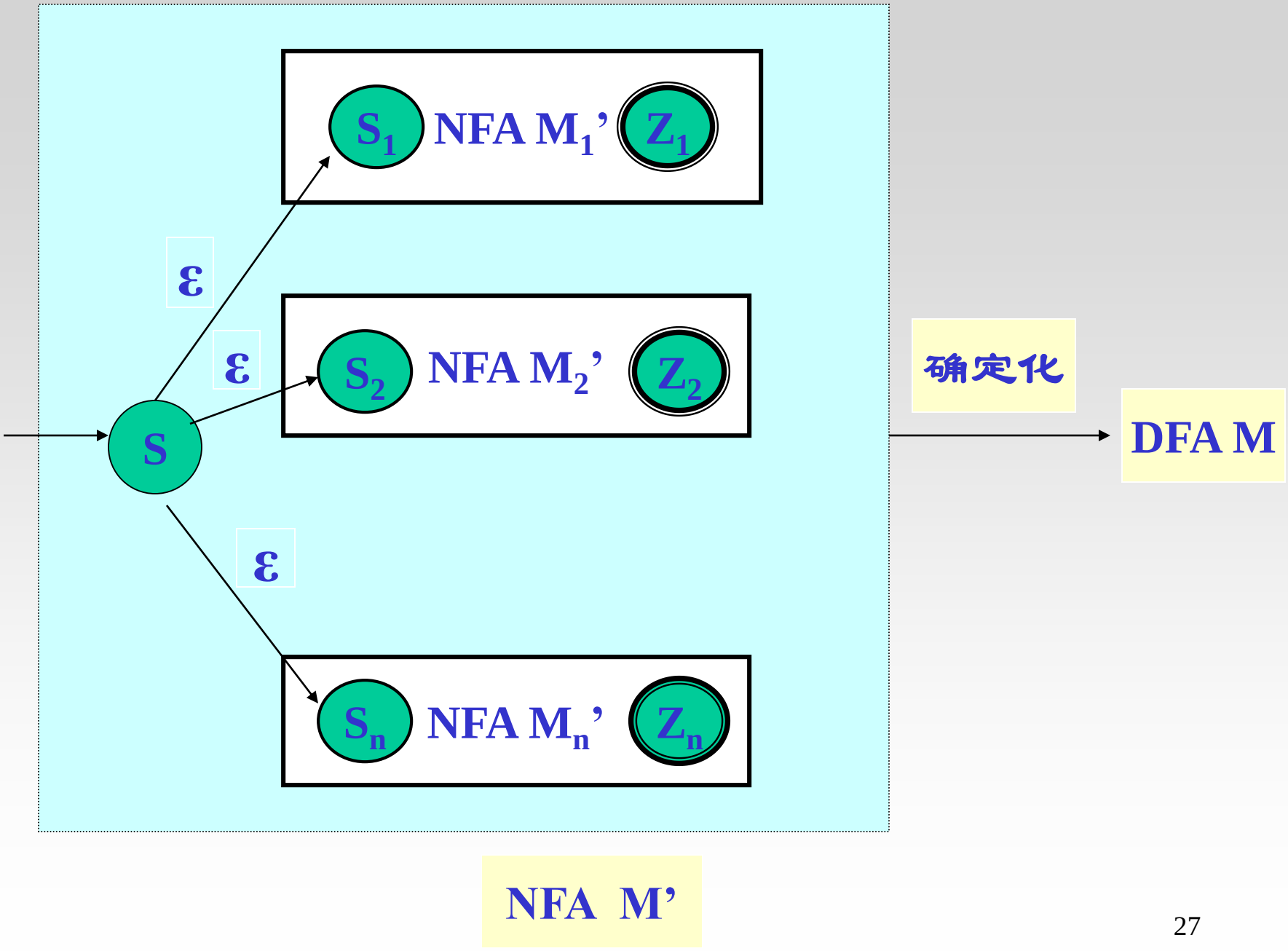
(1)工作过程

①由每条正则式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


S1
NFA M1'
Z1
ε
ε
S2
NFA M2'
Z2
确定化
S
DFA M
ε
Sn
NFA Mn'
Zn
NFA M'

4、词法分析器(DFA M)

LEX.yy.c
- 状态转换矩阵
- 总控程序(yylex( ))

5、处理二义性问题的两个原则

begin P_1(begin) , P_6(letter(letter | digit)*)匹配

<= P_8(<) , P_9(<=)匹配

原则:

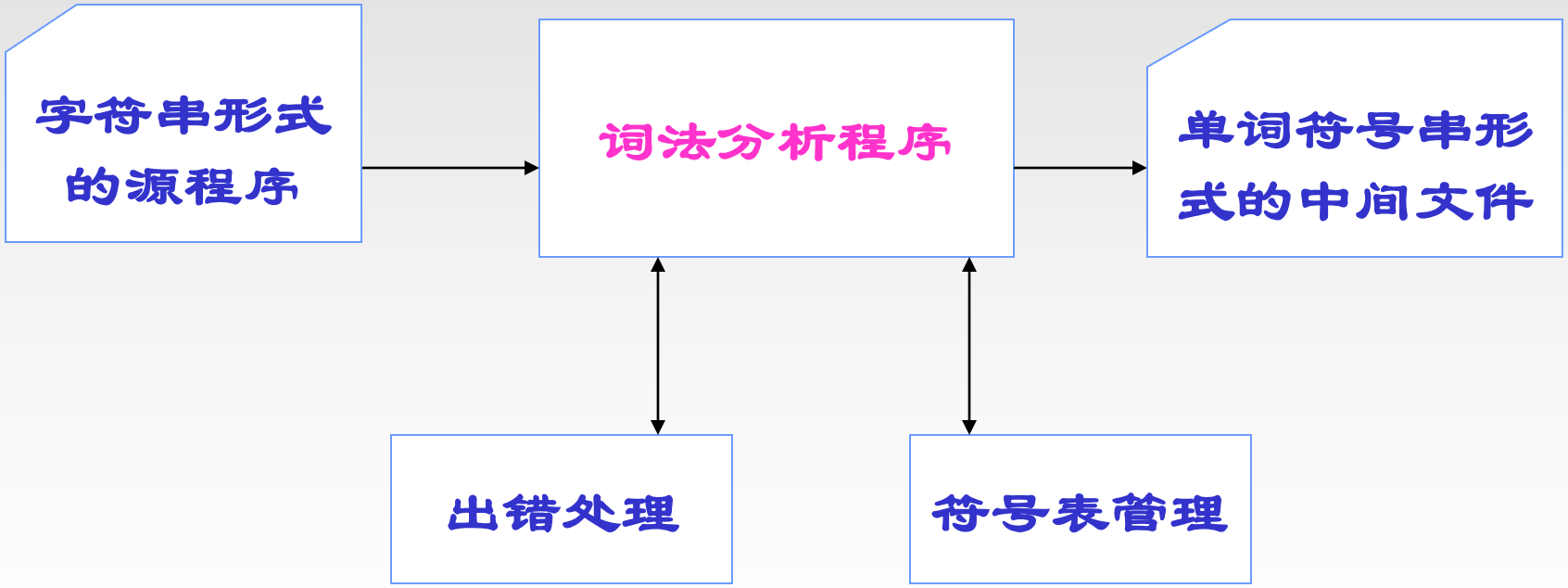
(1) 最长匹配原则:

能匹配最多字符的规则优先 <= P_9

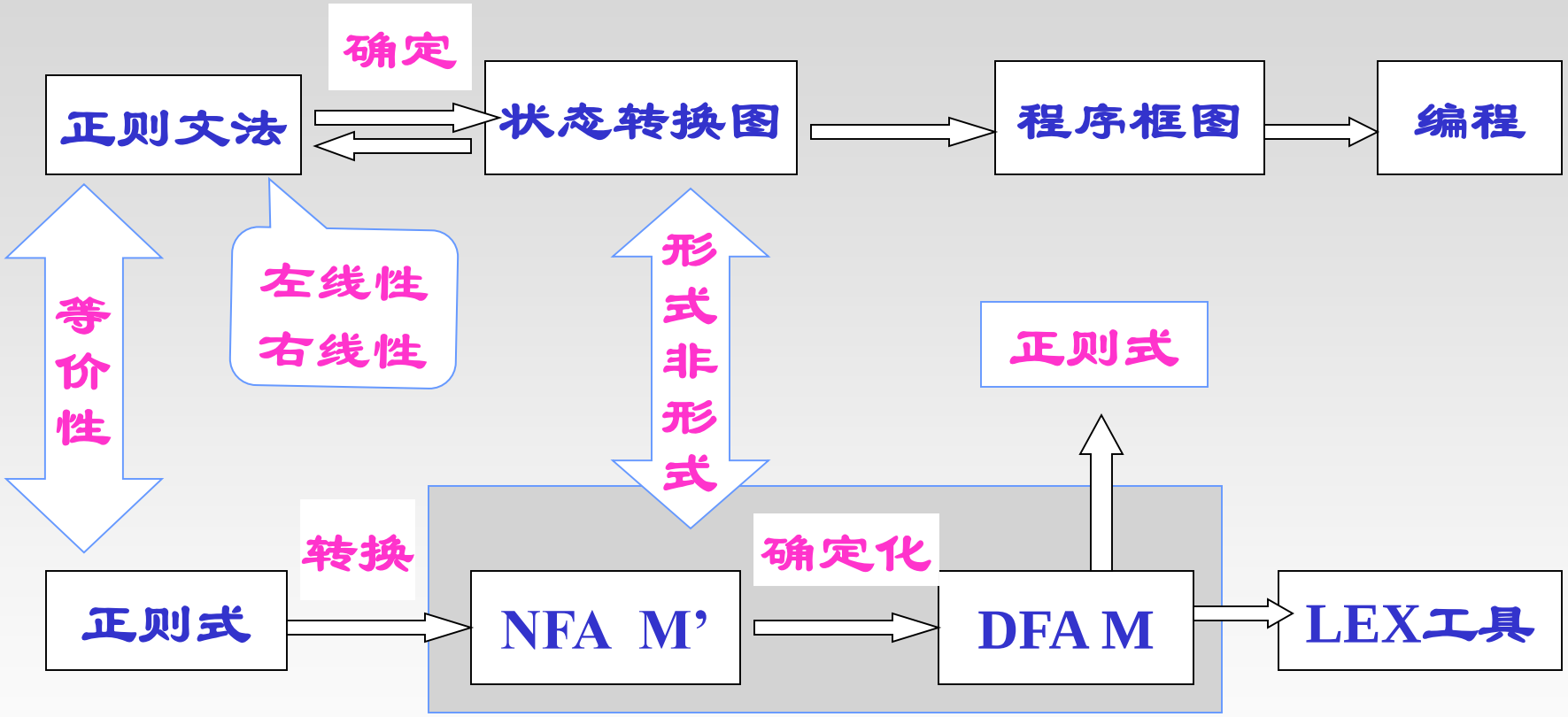
(2) 优先匹配原则:

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

词法分析总结



词法分析程序的手工编写

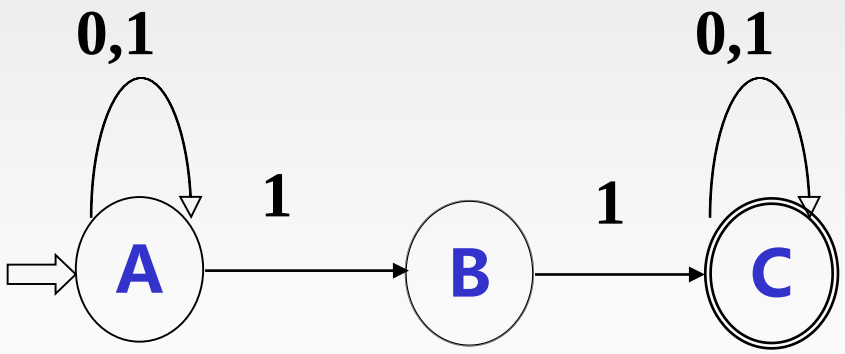


词法分析程序的自动生成

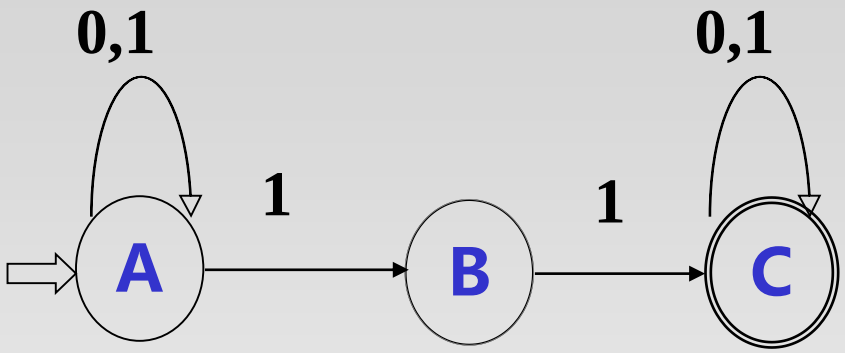
思考题（一）

- 已知有穷自动机如图：

1. 表示的语言是什么？（可用L={W | ...}来描述）
2. 写出该语言的正规式和正规文法
3. 构造识别该语言的DFA



思考题（二）



解：

(1) L={W |W ∈{0,1},并且W至少有两个连续的1}

(2) 正则式为 (0|1)*11(0|1)*

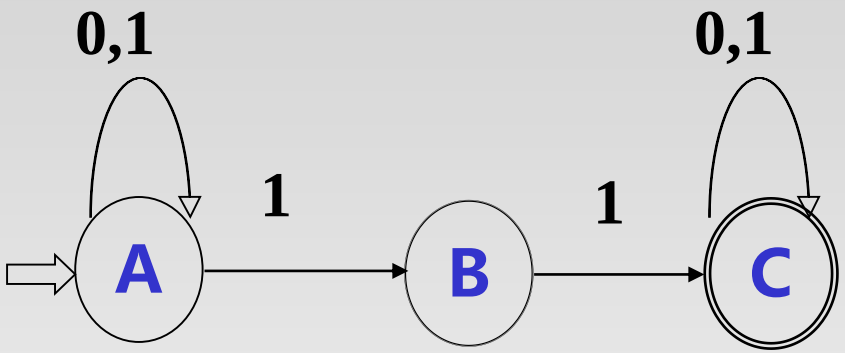
正则文法G(Z)为：

A→0A|1A|1B

B →1C|1

C →0C|1C|0|1

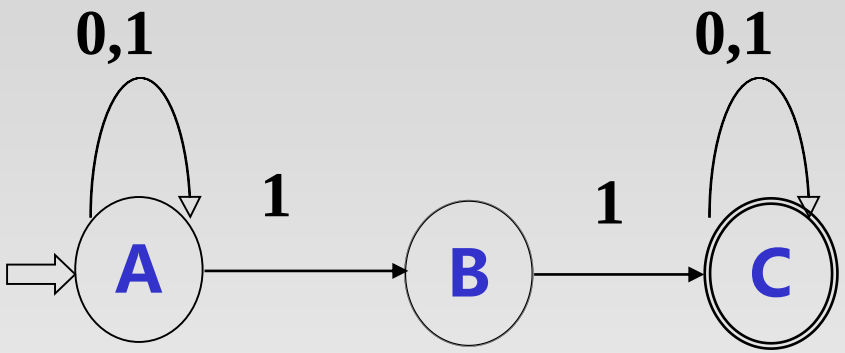
思考题（二）



(3) 将图中有限自动机确定化：

首先从初态A出发：

思考题（二）



I	0	1
(1){A}	(1){A}	(2) {A,B}
(2){A,B}	(1){A}	(3){A,B,C}
(3){A,B,C}	(4){A,C}	(3){A,B,C}
(4){A,C}	(4){A,C}	(3){A,B,C}

由此可画出DFA

思考题（二）

- 某操作系统下合法的文件名规则为：
  device:name.extention，其中第一部分（device:）和第三部分（.extention）可缺省，若device、name和extention都是由字母组成，长度不限，但至少一位。画出识别这种文件名的DFA。

解法：直接画出DFA

或者分解一下：

1. 写出识别这种文件名的正则式；
2. 画出其对应的NFA；
3. 将上述NFA确定化为等价的DFA.

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)