# 4 语法分析程序设计及实现

## 4.1 SLR文法的构造

4.1.1 SLR原文法

源程序设计语言 G[<程序>]

<程序>→<变量说明><BEGIN> <语句表> <END>.

<变量说明>→VAR<变量表>:<类型>；|<空>

<变量表>→<变量表>,<变量>|<变量>

<类型>→INTEGER

<语句表>→<语句> | <语句>;<语句表>

<语句>→<赋值语句>|<条件语句>|<WHILE语句>|<复合语句>

<赋值语句>→<变量>:=<算术表达式>

<条件语句>→IF<关系表达式>THEN<语句>ELSE<语句>

<WHILE语句>→WHILE<关系表达式>DO<语句>

<复合语句>→BEGIN<语句表>END

<算术表达式>→<项>|<算术表达式>+<项>|<算术表达式>-<项>

<项>→<因式>|<项>\*<因式>|<项>/<因式>

<因式>→<变量>|<整数>|(<算术表达式>)

<关系表达式>→<算术表达式><关系符><算术表达式>

注释：从接连出现的/\*到下一次接连出现的\*/之间的任何文字都是注释。

从某行接连出现的//到该行的结尾的任何文字都是注释。

白空格：两个单词之间的任何空格，制表符，回车，换行等都是白空格，除了用来分隔单词以外，没有意义。

限制条件：标识符的长度不大于8，整数不大于65535。

4.1.2 构造SLR文法

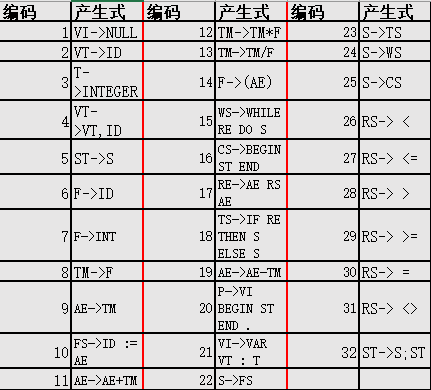
（1）将文法的产生式左部的非终结符单词化及编码，如表4-1所示；

表4-1 非终结符分类及编码



1. 将文法的产生式编码，如表4-2所示；

表4-2 产生式编码



（3）构造语法树

由于涉及部分产生式涉及递归，所以需要具体问题具体分析。下面给出一程序，根据前面得到的产生式语法构造语法树，模拟其语法分析过程。

假设程序如下程序：

VAR

X,Y,Z,A,B,C,D:INTEGER;

BEGIN

WHILE (A+1)>B DO

IF C>D THEN

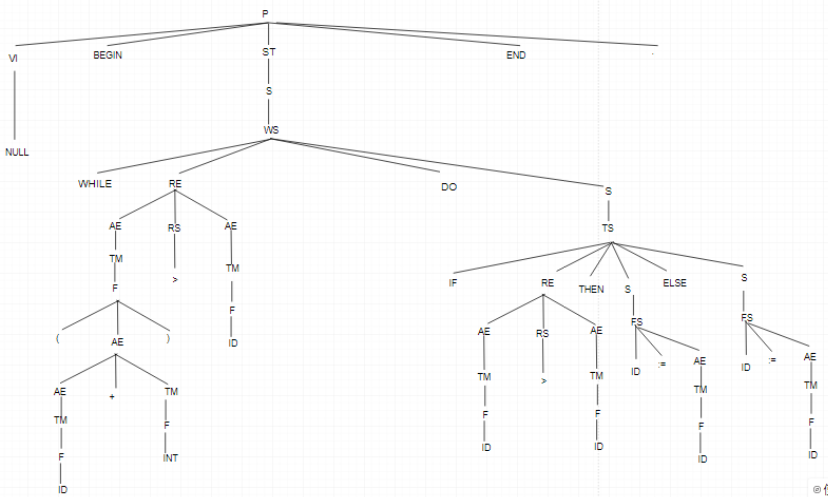
X:=Y+Z

ELSE

Y:=X+Z

END.

语法树构造如下图所示：

图 4-1 语法树模拟语法分析过程

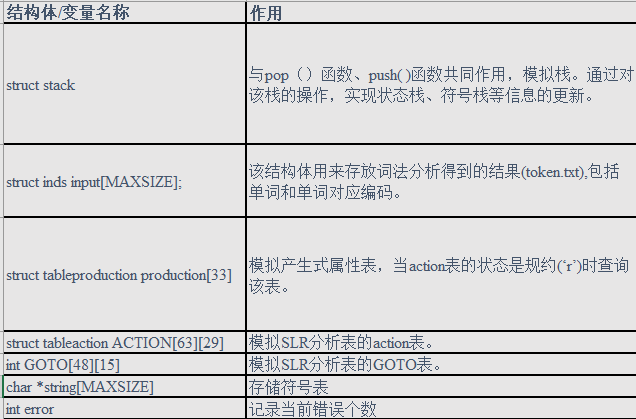
（4） 构造DFA（见附录一）

（5） SLR文法（见附录二）。

4.2 语法分析程序设计与实现

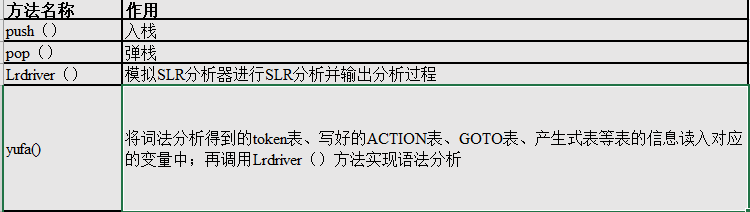
4.2.1设计结构体和主要变量（见表4-3）

表4-3 结构体和主要变量表



4.2.2 主要方法及作用（见表4-4）

表4-4 主要方法和作用表



## 4.3 主要模块的算法

（1）语句存放数组

考虑到程序内的单词是需要依次读取的，为了提高我们的提取效率，这里选择使用一个数组来存放这些单词和对应的编码。考虑到目标程序内的单词数量不确定，我们尽可能让数组的最大长度大一些，用了500。

#define MAXSIZE 500

struct inds

{

char \*instr; //字符串

int sty; //类型

};

struct inds input[MAXSIZE];

（2）SLR分析表的存储结构

struct tableaction //构造action分析表

{

char c; //动作

int s; //状态

}ACTION[63][29]={0};

int GOTO[48][15]={0}; //构造goto表

（3）语句合法性判断

根据 当前栈顶元素内的存储状态变量的值 和 余留串当前指向的单词对应编码 作为ACTION[X][Y]的X、Y坐标，查询ACTION[X][Y]的动作：

a. 若状态是“A”，则接受，SLR分析结束。

b. 若状态是“error”（即空值）则输出报错信息。

c. 若状态是“S”则根据ACTION[X][Y]的状态进行对应的入栈操作。

d. 若状态是“R”，则根据ACTION[X][Y]的状态为下标查询产生式表，根据对应产生式的成员word\_count值将栈回退word\_count位，同时获得其word\_number值（左边产生式的编码值），与回退后的栈顶元素的状态变量的值为下标查询GOTO表，由左边产生式的编码值、GOTO表的对应内容获得下一步信息，进而执行入栈操作。

下面举例子说明，假设程序前六个单词为“var X , Y , Z”。

首先将单词存入input[ ] 数组，并初始化栈（加入#），如图4-2所示

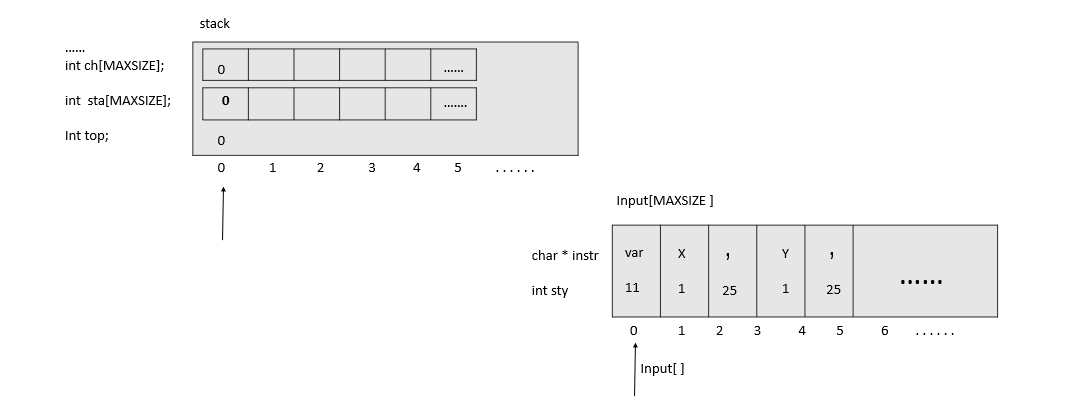


图4-2 初始化栈

这时，根据栈stack的元素sta[0] 、和input[ 0] 的元素sty的值作为action表坐标查action 表，此时即为action [ 0] [ 11]。查表发现，action表该坐标下状态是“S”，故进行入栈操作，结果如图4-3所示。

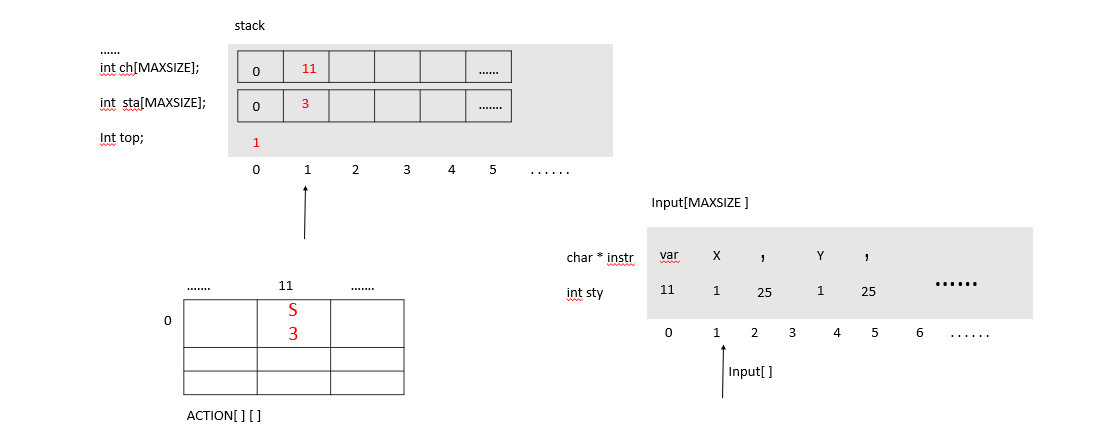


图4-3 移进1

这时，栈中的状态栈就是03，符号栈就是011，栈顶元素为1。下一步对单词“，”进行操作，过程类似上一步，在此不再赘述，结果见图4-4。

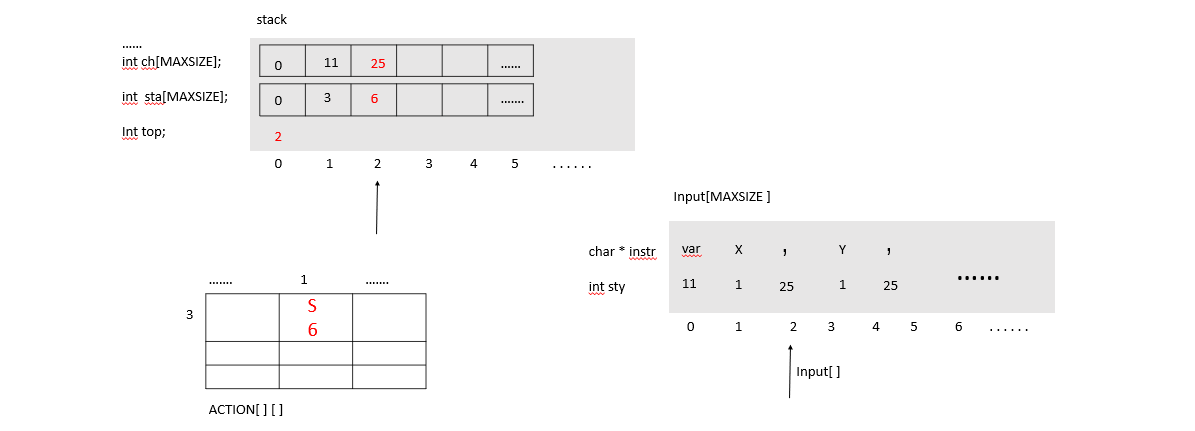


图4-4 移进2

这时，栈中的状态栈就是036，符号栈就是01125，栈顶元素为2。

下一步对单词“Y”进行操作。查询发现ACTION表该坐标下状态是“r”,所以要进行规约操作，先由ACTION表该坐标下的状态值查询产生式表，获取需要弹栈个数和GOTO表的纵坐标，其中，产生式表下的word\_count的值即为栈需要弹栈的元素个数。弹栈后以当前栈的状态栈为GOTO表横坐标，即可查询GOTO表，得到下一步的状态。最后进行入栈操作。入栈后栈中的状态栈就是035，符号栈就是01131，栈顶元素为2，详情见图4-5 。

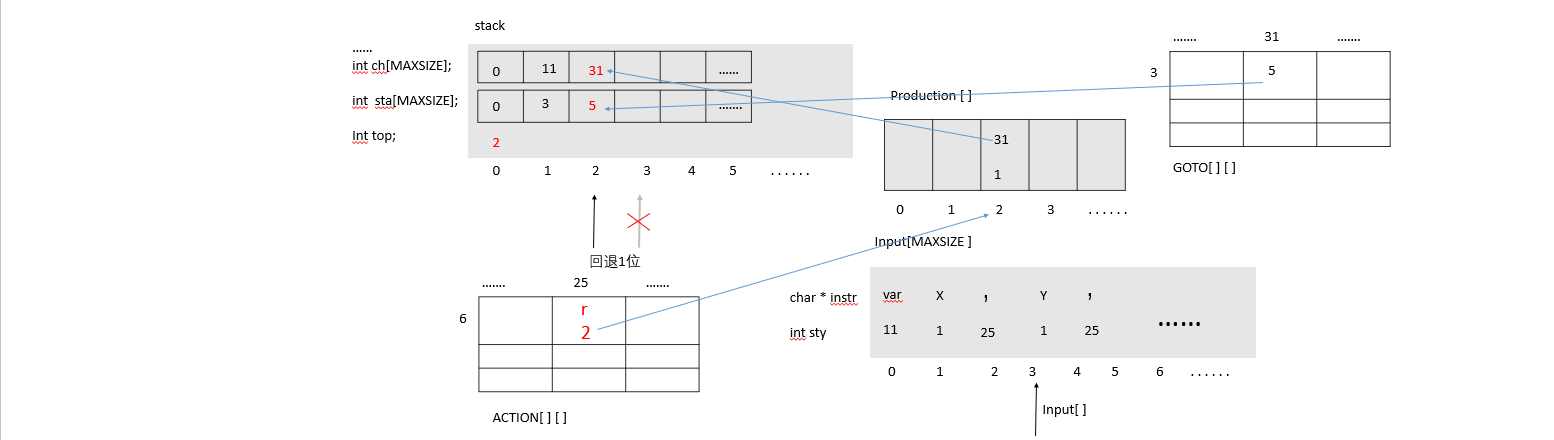


图4-5 规约

以上便是模拟SLR分析表的移进、规约的实现过程。下面是相关代码：

void LRdriver(stack\* st) //SLR分析器驱动

{

int l;

TS = st->sta[st->top]; //栈顶状态号（TopStat）

IS = input[intop].sty; //当前输入单词的编码(InpSym)

while ((ACTION[TS][IS].c != 'A') && (!error))//A为acc

{ /\*输出分析过程\*/

printf("\n");

for (l = 0; l <= st->top; l++)

printf("%d-", st->sta[l]);

if (l >= 12)

printf("\t");

else if (l >= 9 && l <= 11)

printf("\t\t");

else if (l >= 7 && l <= 8)

printf("\t\t\t");

else if (l >= 4 && l <= 6)

printf("\t\t\t\t");

else

printf("\t\t\t\t\t");

for (l = 0; l <= st->top; l++)

printf("%d-", st->ch[l]);

if (l >= 10 && l <= 20)

printf("\t\t");

else if (l >= 7 && l <= 9)

printf("\t\t\t");

else if (l >= 4 && l <= 6)

printf("\t\t\t\t");

else

printf("\t\t\t\t\t");

for (l = intop; input[l].sty != 100; l++)

printf("%-s", input[l].instr);

if (ACTION[TS][IS].c == '\0') //分析动作为ERROR

{

error++;

}

else if (ACTION[TS][IS].c == 'S') //分析动作为移进（s）

{

// 当前单词 单词对应编码 下一步状态

push(st, input[intop].instr, input[intop].sty, ACTION[TS][IS].s, 0, 0, 0);

//栈指向下一个（即可读长度加1）

intop++;

color(FOREGROUND\_INTENSITY | FOREGROUND\_GREEN);

printf("\ts%d", ACTION[TS][IS].s); //输出 : s\*

SetConsoleTextAttribute(GetStdHandle(STD\_OUTPUT\_HANDLE), FOREGROUND\_INTENSITY | FOREGROUND\_INTENSITY);

}

else if (ACTION[TS][IS].c == 'r') //分析动作为归约（r）

{

//弹栈（弹栈元素个数由产生式表的属性：word\_count决定）

pop(st, production[ACTION[TS][IS].s].word\_count);

PL = production[ACTION[TS][IS].s].word\_number;//以动作表当前状态值为下标去查产生式表，得到产生式左边的单词对应的值

// 当前状态栈值 产生式左边单词对应的值

NS = GOTO[(st->sta[st->top])][production[ACTION[TS][IS].s].word\_number - 29];//查goto表 ，

// 单词 状态

push(st, input[intop].instr, PL, NS, production[ACTION[TS][IS].s].Pro\_place, production[ACTION[TS][IS].s].Pro\_TC, production[ACTION[TS][IS].s].Pro\_FC);

color(FOREGROUND\_INTENSITY | FOREGROUND\_RED);

printf("\tr%d", ACTION[TS][IS].s);

SetConsoleTextAttribute(GetStdHandle(STD\_OUTPUT\_HANDLE), FOREGROUND\_INTENSITY | FOREGROUND\_INTENSITY);

}

TS = st->sta[st->top];

IS = input[intop].sty;

}

if (!error)

{

printf("\n");

for (l = 0; l <= st->top; l++)

printf("%d", st->sta[l]);

printf("\t\t");

for (l = 0; l <= st->top; l++)

printf("%d", st->ch[l]);

printf("\t");

for (l = intop; input[l].sty != 100; l++)

printf("%s", input[l].instr);

color(FOREGROUND\_INTENSITY | FOREGROUND\_RED);

printf("\tAccepted!\n");

SetConsoleTextAttribute(GetStdHandle(STD\_OUTPUT\_HANDLE), FOREGROUND\_INTENSITY | FOREGROUND\_INTENSITY);

}

else {

color(FOREGROUND\_INTENSITY | FOREGROUND\_RED);

printf("\nIt must be an error here,Please check it again!\n");

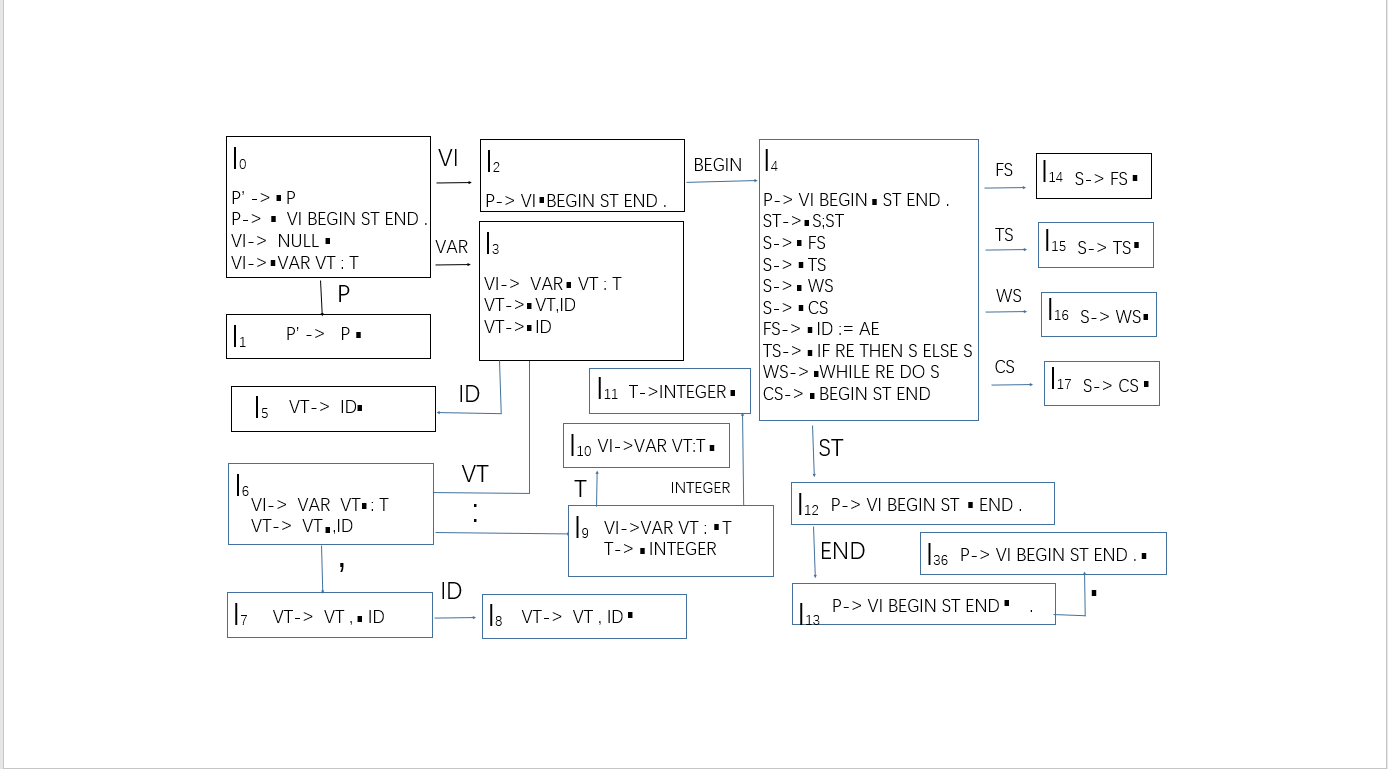
SetConsoleTextAttribute(GetStdHandle(STD\_OUTPUT\_HANDLE), FOREGROUND\_INTENSITY | FOREGROUND\_INTENSITY);

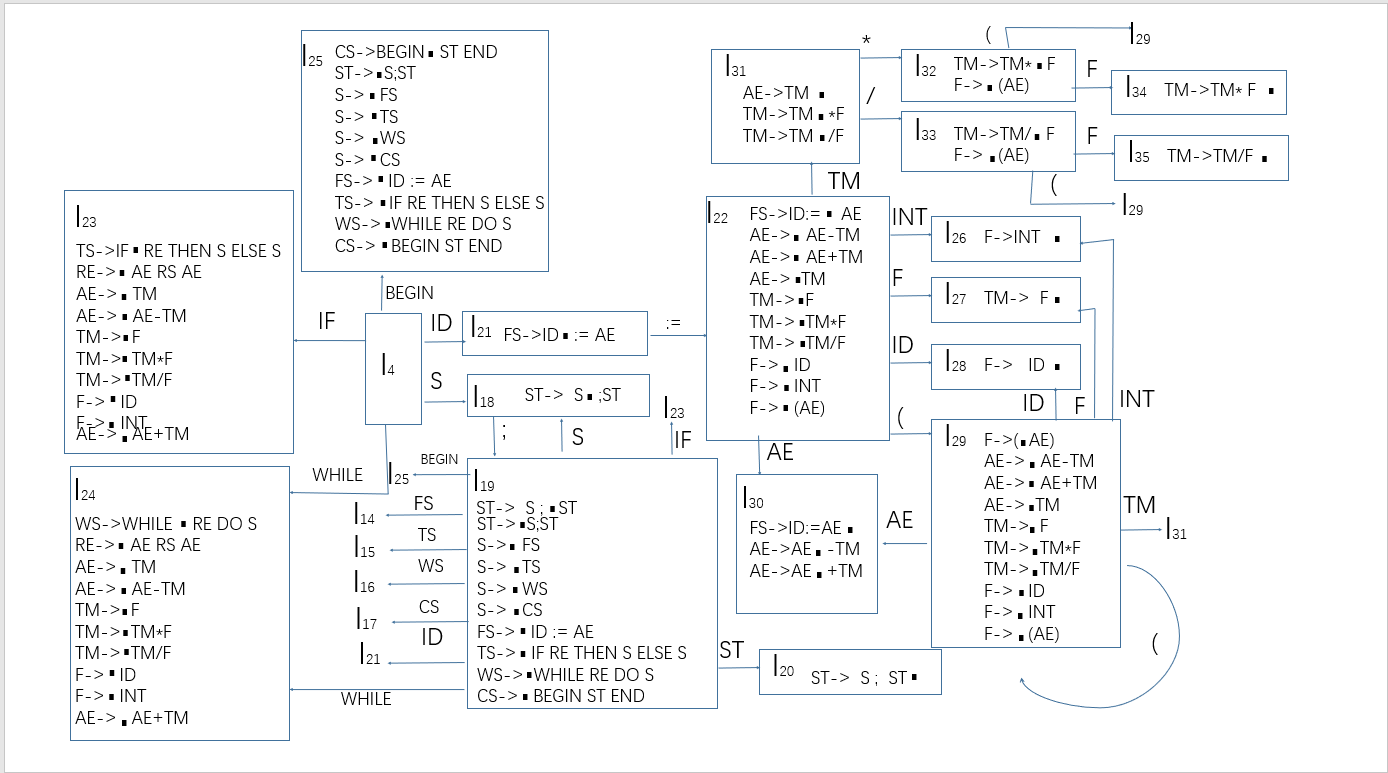
}

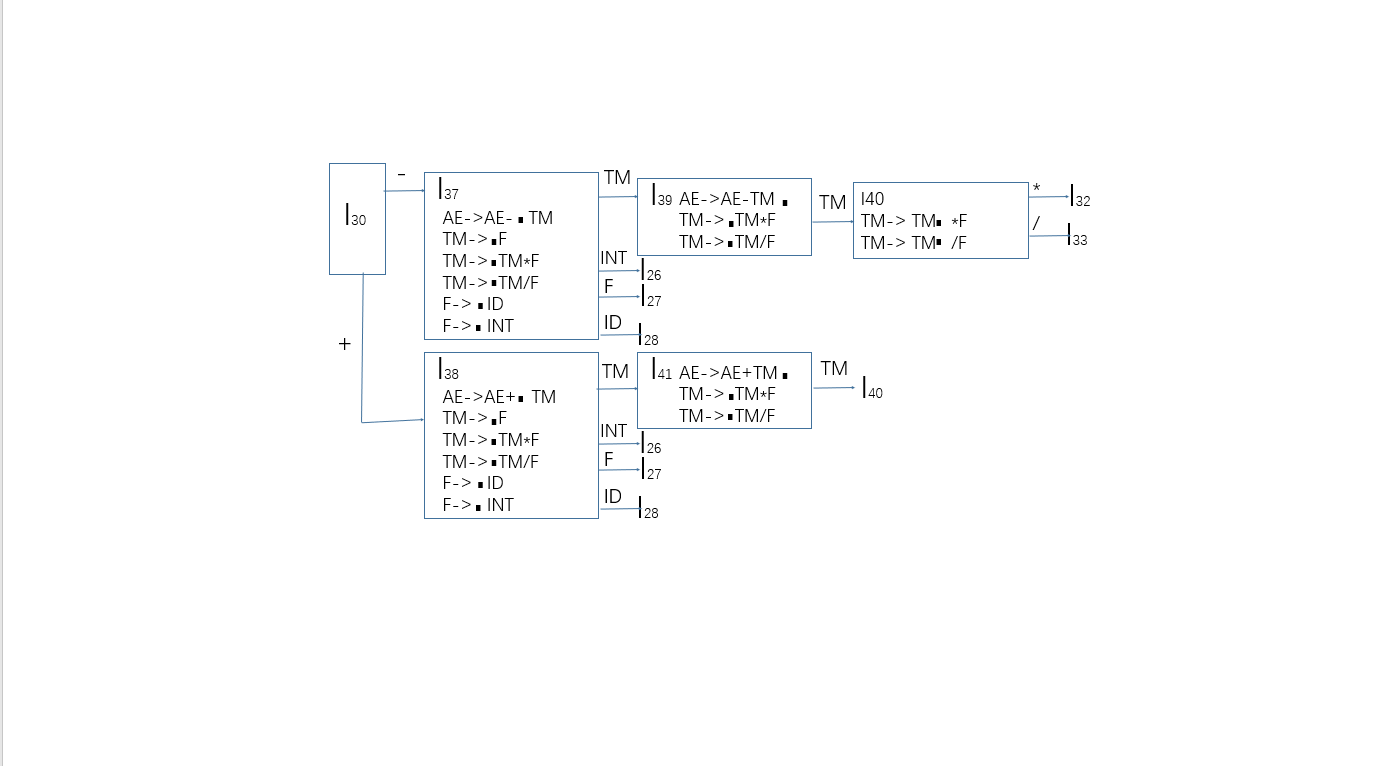
};

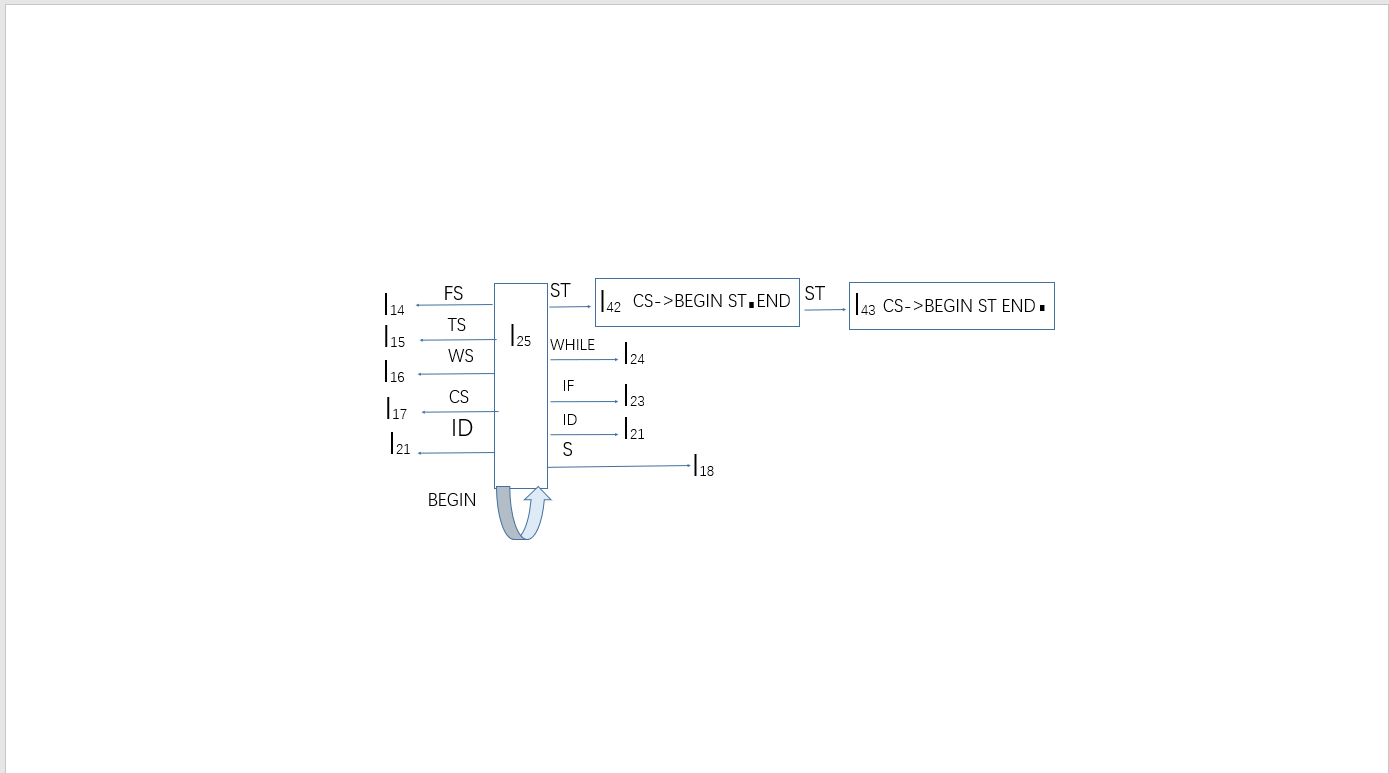
附录一

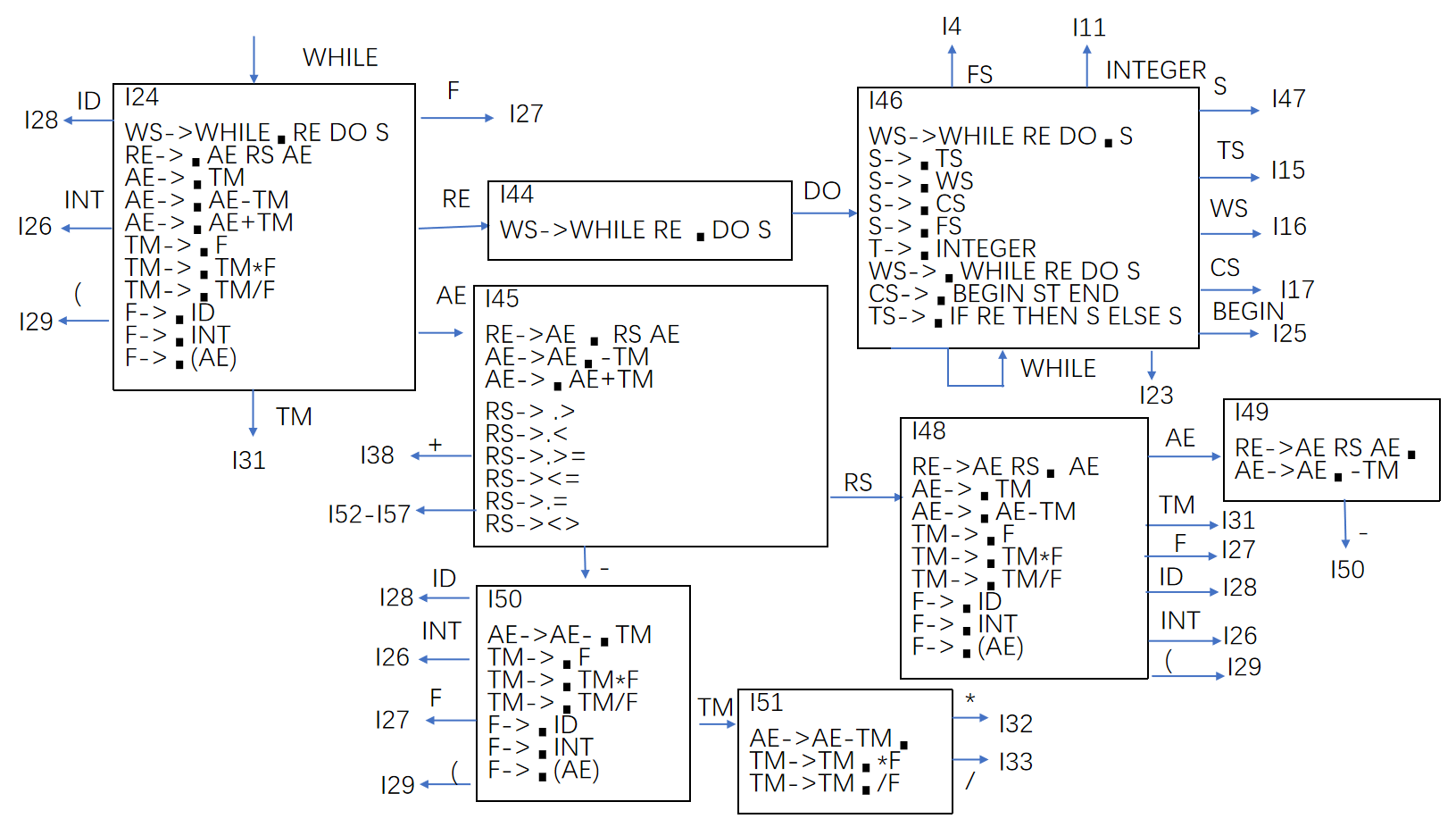
DFA构造表

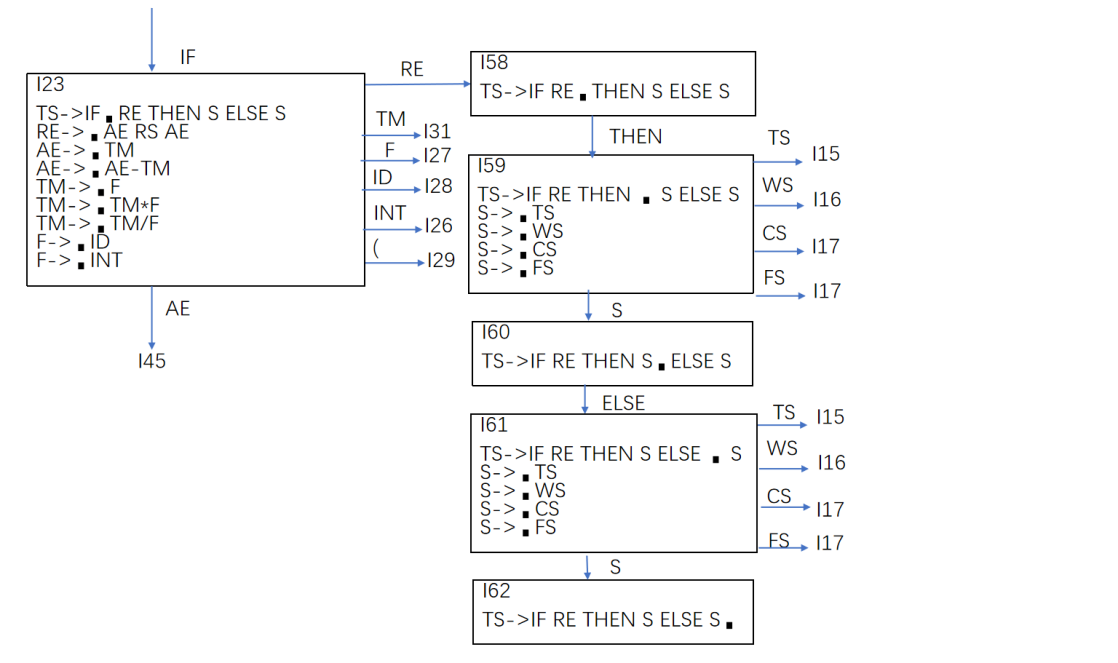












附录二

SLR文法分析表

