实验三. 自上而下语法分析

设计思想

根据对自下而上语法分析的理论知识的学习，可以知道自下而上语法分析的两种实现方法：算符优先分析法和LR分析法，本实验采用后者LR分析法

本实验对PL0文法的表达式文法进行设计自下而上语法分析，表达式巴斯克范式如下：

文法的初始化

$$
\begin{aligned}
&< \text{表达式} >::= [+|-] < \text{项} > \{< \text{加法运算符} >< \text{项} >\}\\
&< \text{项} >::=< \text{因子} > \{< \text{乘法运算符} >< \text{因子} >\}\\
&< \text{因子} >::=< \text{标识符} > | < \text{无符号整数} > |'('< \text{表达式} >')'\\
&< \text{加法运算符} >::= +|-\\
&< \text{乘法运算符} >::= *|/
\end{aligned}
$$

对以上文字描述的文法可以给出对应的英文表示，以及对应的非终结符的 $First$集合 和 $Follow$集合 ：

$$
\begin{aligned}
&< Expression >::= [+|-] < Term > \{< Addop >< Term >\}\\
&< Term >::=< Factor > \{< Mulop >< Factor >\}\\
&< Factor >::=< Ident > | < UnsignInt > |(< Expression >)\\
&< Addop >::= +|-\\
&< Mulop >::= *|/
\end{aligned}
$$

该文法的非终结符结合以及终结符集合如下：

$$
\begin{aligned}
V_T &= \{+, -, *, /, (,), Ident, UnsignInt\}\\
V_N &= \{Expression, Term, Addop, Factor, Mulop\}
\end{aligned}
$$

$First$集合 :

$$
\begin{aligned}
&First(Expression) = \{+, -, Ident, UnsignInt, (\}\\
&First(Term) = \{Ident, UnsignInt, (\}\\
&First(Factor) = \{Ident, UnsignInt, (\}\\
&First(Addop) = \{+, -\}\\
&First(Mulop) = \{*, /\}
\end{aligned}
$$

$Follow$集合 :

$$
\begin{aligned}
&Follow(S') = \{\#\}\\
&Follow(Expression) = \{\#, +, -,)\}\\
&Follow(Term) = \{\#, +, -, *, /,)\}\\
&Follow(Factor) = \{\#, +, -, *, /,)\}
\end{aligned}
$$

将其简化一下文法：

$$
\begin{aligned}
&E \to T|+T|-T\\
&E \to E+T|E-T\\
&T \to F\\
&T \to T*F|T/F\\
&F \to i|u|(E)
\end{aligned}
$$

对应的终结符和非终结符集合如下：

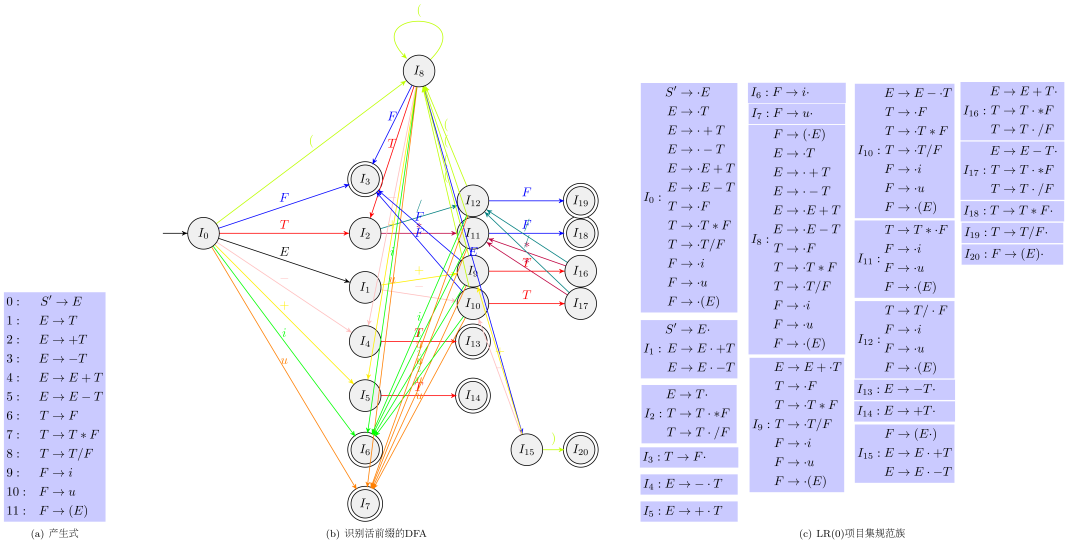
$$V_T = \{+, -, *, /, (,), i, u\}$$
$$V_N = \{S', E, T, F\}$$

拓广文法

文法的拓广，将文法 $G(S)$ 拓广为 $G'(S')$ ，并规定每一个产生式的编号，以便后续的方便使用：

$$\begin{aligned}
0: &\quad S' \to E \\
1: &\quad E \to T \\
2: &\quad E \to +T \\
3: &\quad E \to -T \\
4: &\quad E \to E + T \\
5: &\quad E \to E - T \\
6: &\quad T \to F \\
7: &\quad T \to T * F \\
8: &\quad T \to T/F \\
9: &\quad F \to i \\
10: &\quad F \to u \\
11: &\quad F \to (E)
\end{aligned}$$

构造识别活前缀的DFA



(a) 产生式　(b) 识别活前缀的DFA　(c) LR(0)项目集规范族

Figure 1: 整个编码执行流程

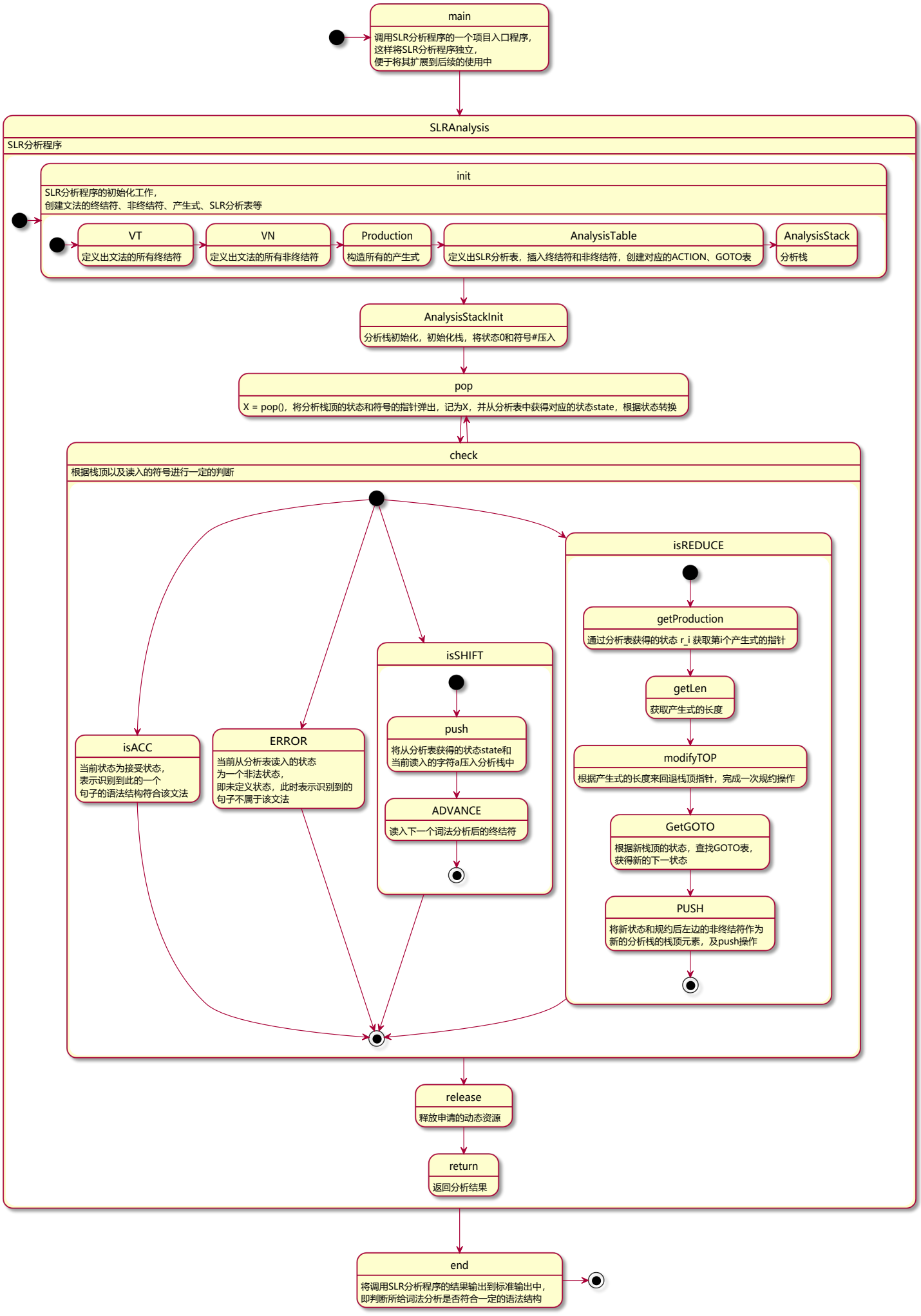
由此DFA可以看出，某些状态存在“规约-移进”冲突，所以可以通过SLR分析法的方式来尝试构造分析表，构造出此时构造出的分析表显然没有冲突：

SLR分析表

	+	−	*	/	(	)	i	u	#	E	T	F
0	s5	s4			s8		s6	s7		1	2	3
1	s9	s10							acc			
2	r1	r1	s11	s12		r1			r1			
3	r6	r6	r6	r6		r6			r6			
4											13	
5											14	
6	r9	r9	r9	r9		r9			r9			
7	r10	r10	r10	r10		r10			r10			
8	s5	s4			s8		s6	s7		15	2	3
9					s8		s6	s7			16	3
10					s8		s6	s7			17	3
11					s8		s6	s7				18
12					s8		s6	s7				19
13	r3	r3				r3			r3			
14	r2	r2				r2			r2			
15	s9	s10				s20						
16	r4	r4	s11	s12		r4			r4			
17	r5	r5	s11	s12		r5			r5			
18	r7	r7	r7	r7		r7			r7			
19	r8	r8	r8	r8		r8			r8			
20	r11	r11	r11	r11		r11			r11			

得到分析表后，就可以根据LR分析的程序框架完成总控程序等的编写，由上一次的 自上而下的SLR分析表法 的代码可以简单的修改一些代码，便可实现整个分析程序的编写。

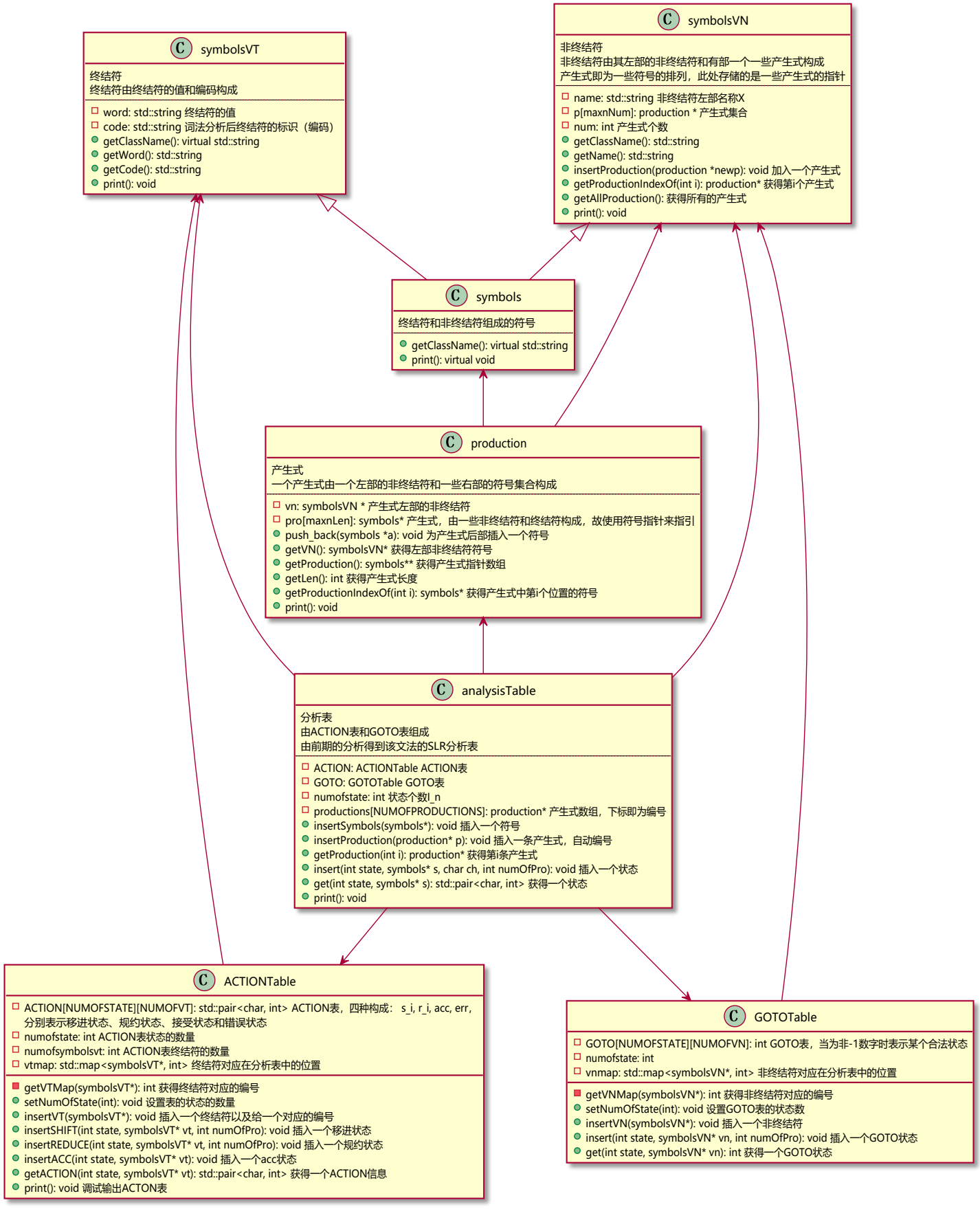
算法流程


main
调用SLR分析程序的一个项目入口程序，
这样将SLR分析程序独立，
便于将其扩展到后续的使用中
SLRAnalysis
SLR分析程序
init
SLR分析程序的初始化工作，
创建文法的终结符、非终结符、产生式、SLR分析表等
VT
定义出文法的所有终结符
VN
定义出文法的所有非终结符
Production
构造所有的产生式
AnalysisTable
定义出SLR分析表，插入终结符和非终结符，创建对应的ACTION、GOTO表
AnalysisStack
分析栈
AnalysisStackInit
分析栈初始化，初始化栈，将状态0和符号#压入
pop
X = pop()，将分析栈顶的状态和符号的指针弹出，记为X，并从分析表中获得对应的状态state，根据状态转换
check
根据栈顶以及读入的符号进行一定的判断
isACC
当前状态为接受状态，
表示识别到此的一个
句子的语法结构符合该文法
ERROR
当前从分析表读入的状态
为一个非法状态，
即未定义状态，此时表示识别到的
句子不属于该文法
isSHIFT
push
将从分析表获得的状态state和
当前读入的字符a压入分析栈中
ADVANCE
读入下一个词法分析后的终结符
isREDUCE
getProduction
通过分析表获得的状态 r_i 获取第i个产生式的指针
getLen
获取产生式的长度
modifyTOP
根据产生式的长度来回退栈顶指针，完成一次规约操作
GetGOTO
根据新栈顶的状态，查找GOTO表，
获得新的下一状态
PUSH
将新状态和规约后左边的非终结符作为
新的分析栈的栈顶元素，及push操作
release
释放申请的动态资源
return
返回分析结果
end
将调用SLR分析程序的结果输出到标准输出中，
即判断所给词法分析是否符合一定的语法结构

源程序

整个SLR分析程序是由上一实验的预测分析程序修改而来，根据SLR分析的所需，简单的修改了分析表的构成，即由ACTION和GOTO表组成，将原来的分析总控程序中的具体的执行流程根据LR分析法来修改，其他内容保持不变即可：

整个LR分析程序所设计到的类以及相互的关系如下：



项目地址

基础符号类

基础符号类作为一个符号类的基类，主要用途为使用基类指针来指引子类的终结符或非终结符，简化后续的操作。

```
// symbols.h
#ifndef symbols_h
#define symbols_h
#include<iostream>
/*
终结符和非终结符的一个共同的基类
这样可以通过基类来引用终结符和非终结符

*/
class symbols
{
private:
    /* data */
public:
    symbols(){};
    virtual ~symbols(){};
    virtual std::string getClassName(){
        return "symbols";
    }
    virtual void print(){};
};

#endif /*symbols_h*/
```

终结符类

终结符类继承至基础符号类，终结符由终结符的值和编码构成。

```
// symbolsVT.h
#ifndef symbolsVT_h
#define symbolsVT_h
#include<iostream>
#include"symbols.h"
/*
一个终结符
终结符由终结符的值和编码构成

*/
class symbolsVT : public symbols{
private:
    std::string word;   //终结符的值
    std::string code;   //词法分析后终结符的标识（编码）
public:
    symbolsVT(){}
    symbolsVT(std::string W, std::string C):word(W), code(C) {
        std::cerr<< "V_T: " << word << " " << code << " created..." << std::endl;
    }
    ~symbolsVT() {}
    std::string getClassName();
    std::string getWord();
    std::string getCode();
    void print();
};

#endif /*symbolsVT_h*/
```

```cpp
// symbolsVT.cpp
#include<iostream>
#include"symbolsVT.h"

std::string symbolsVT::getClassName(){
    return "VT";
}
std::string symbolsVT::getWord(){
    return word;
}
std::string symbolsVT::getCode(){
    return code;
}
void symbolsVT::print(){
    std::cerr << "VT: " << word << " " << code << std::endl;
}
```

非终结符类

非终结符类与终结符类一样继承至基础符号类，非终结符由其左部的非终结符和有部一个一些产生式构成，产生式即为一些符号的排列，此处存储的是一些产生式的指针。

```cpp
// symbolsVN.h
#ifndef symbolsVN_h
#define symbolsVN_h
#include<iostream>
#include"symbols.h"
#include"production.h"
const int maxnNum = 5;     // 一个终结符所有的产生式的数量
class production;
/*
非终结符
非终结符由其左部的非终结符和有部一个一些产生式构成
产生式即为一些符号的排列，此处存储的是一些产生式的指针
*/

class symbolsVN: public symbols{
private:
    std::string name;                                           // 非终结符左部名称X
    production *p[maxnNum];                                     // 产生式集合
    int num;
public:
    symbolsVN();
    symbolsVN(std::string);
    ~symbolsVN() {}
    std::string getClassName();
    std::string getName();
    void insertProduction(production *newp);    // 加入一个产生式
    production* getProductionIndexOf(int i);    // 获得第i个产生式
    production** getAllProduction();                    // 获得所有的产生式
    void print();
};

#endif /*symbolsVN_h*/
```

```cpp
// symbolsVN.cpp
#include<iostream>
#include"symbolsVN.h"

symbolsVN::symbolsVN(){
    num = 0;
}
symbolsVN::symbolsVN(std::string n):name(n){
    symbolsVN();
    std::cerr << "V_N: " << name << " created..." << std::endl;
}

std::string symbolsVN::getClassName(){
    return "VN";
}
std::string symbolsVN::getName(){
    return name;
}
void symbolsVN::insertProduction(production *newp){
    p[num++] = newp;
    return;
}
production* symbolsVN::getProductionIndexOf(int i){
    if(i >= num){
        std::cerr << "index overflow..." << std::endl;
        return NULL;
    }
    return p[i];
}
production** symbolsVN::getAllProduction(){
    return p;
}
void symbolsVN::print(){
    std::cerr << "VN: " << name << std::endl;
    std::cerr << "ALL production: " << std::endl;
    for(int i = 0; i < num; ++i){
        std::cerr << name << " \\to ";
        p[i]->print();
    }
    std::cerr << std::endl;
}
```

产生式类

一个产生式由一个左部的非终结符和一些右部的符号集合构成。

```h
// production.h
#ifndef production_h
#define production_h
#include<iostream>
#include"symbols.h"
#include"symbolsVT.h"
#include"symbolsVN.h"
/*
产生式
一个产生式由一个左部的非终结符和一些右部的符号集合构成
*/

const int maxnLen = 10;        // 一个产生式的右部符号的数量
class symbolsVN;
class production
{
private:
    symbolsVN *vn;                                    // 产生式左部的非终结符
    symbols *pro[maxnLen];                        // 产生式，由一些非终结符和终结符构成，故使用符号指针来指引
    int len;
public:
    production();
    production(symbolsVN *v);
    ~production(){}
    void push_back(symbols *a);         // 为产生式后部插入一个符号
    symbolsVN* getVN();                           // 获得左部非终结符符号
    symbols** getProduction();          // 获得产生式指针数组
    int getLen();                                     // 获得产生式长度
    symbols* getProductionIndexOf(int i); // 获得产生式中第i个位置的符号
    void print();
};

#endif /*production_h*/
```

```cpp
// production.cpp
#include<iostream>
#include"production.h"
#include"symbolsVT.h"
#include"symbolsVN.h"

production::production(){
    len = 0;
}
production::production(symbolsVN *v){
    vn = v;
    production();
    std::cerr << "A production of " << vn->getName() << " has created..." << std::endl;
}
void production::push_back(symbols *a){
    pro[len++] = a;
}
symbolsVN* production::getVN(){
    return vn;
}
symbols** production::getProduction(){
    return pro;
}
symbols* production::getProductionIndexOf(int i){
    if(i >= len){
        std::cerr << "index Overflow..." << std::endl;
        return NULL;
    }
    return pro[i];
}
int production::getLen(){
    return len;
}
void production::print(){
    std::cerr << vn->getName() << "->";
    for(int i = 0; i < len; ++i){
        if(pro[i]->getClassName() == "VT"){
            std::cerr << ((symbolsVT*)pro[i])->getWord();
        }
        else{
            std::cerr << ((symbolsVN*)pro[i])->getName();
        }
    }
    // std::cerr << std::endl;
}
```

分析表

分析表，由ACTION表和GOTO表构成，具体的状态内容由前期的DFA分析等得到：

```h
// analysisTable.h
#ifndef analysisTable_h
#define analysisTable_h
#include<map>
#include"symbols.h"
#include"symbolsVN.h"
#include"symbolsVT.h"
#include"production.h"
const int NUMOFVT = 20;
const int NUMOFVN = 20;
const int NUMOFSTATE = 30;
const int NUMOFPRODUCTIONS = 30;
/*
分析表子程序
由前期的分析得到该文法的分析表，由ACTION表和GOTO表构成
*/
class ACTIONTable{
private:
    std::pair<char, int> ACTION[NUMOFSTATE][NUMOFVT];
    int numofstate;                                          // ACTION表状态的数量
    int numofsymbolsvt;                                      // ACTION表终结符的数量
    std::map<symbolsVT*, int> vtmap;    // 终结符对应在分析表中的位置
    int getVTMap(symbolsVT*);                    // 获得终结符对应的编号
public:
    ACTIONTable();
    ACTIONTable(int);
    ~ACTIONTable(){}
    void setNumOfState(int);                     // GOTO状态数量
    void insertVT(symbolsVT*);                   // 插入一个终结符以及给一个对应的编号
    void insertSHIFT(int state, symbolsVT* vt, int numOfPro);       // 插入一个移进状态
    void insertREDUCE(int state, symbolsVT* vt, int numOfPro);      // 插入一个规约状态
    void insertACC(int state, symbolsVT* vt);                       // 插入一个acc状态
    std::pair<char, int> getACTION(int state, symbolsVT* vt);       // 获得一个ACTION信息
    void print();
};
class GOTOTable{
private:
    int GOTO[NUMOFSTATE][NUMOFVN];
    int numofstate;                                          // GOTO状态数量
    int numofsymbolsvn;
    std::map<symbolsVN*, int> vnmap;    // 非终结符对应在分析表中的位置
    int getVNMap(symbolsVN*);                    // 获得非终结符对应的编号
public:
    GOTOTable();
    GOTOTable(int);
    ~GOTOTable(){}
    void setNumOfState(int);                     // 设置GOTO表的状态数
    void insertVN(symbolsVN*);          // 插入一个非终结符
    void insert(int state, symbolsVN* vn, int numOfPro);    // 插入一个GOTO状态
    int get(int state, symbolsVN* vn);                      // 获得一个GOTO状态
    void print();
};
class analysisTable
{
private:
    ACTIONTable ACTION;                 // ACTION表
    GOTOTable GOTO;                     // GOTO表
    int numofstate;                     // 状态个数I_n
    int numofpro;                       // 产生式数量
    production* productions[NUMOFPRODUCTIONS];  // 产生式数组，下标即为编号
public:
    analysisTable(int ns);
    // analysisTable(int, int, int);
    ~analysisTable() {}
    void insertSymbols(symbols*);               // 插入一个符号
    void insertProduction(production* p);       // 插入一条产生式，自动编号
    production* getProduction(int i);   // 获得第i条产生式
    void insert(int state, symbols* s, char ch, int numOfPro);  // 插入一个状态
    std::pair<char, int> get(int state, symbols* s);    // 获得一个状态
    void print();
};

#endif /*analysisTable_h*/
```

```cpp
// analysisTable.cpp
#include<iostream>
#include<algorithm>
#include<string.h>
#include"analysisTable.h"

ACTIONTable::ACTIONTable(){
    numofsymbolsvt = 0;
    vtmap.clear();
    std::pair<char, int> init = std::make_pair('e', -1);
    // fill(begin(ACTION), end(ACTION), std::make_pair('e', -1));
    for(int i = 0; i < NUMOFSTATE; ++i)
        for(int j = 0; j < NUMOFVT; ++j)
            ACTION[i][j] = init;
    std::cerr << "ACTIONTable has created..." << std::endl;
}
void ACTIONTable::setNumOfState(int ns){
    numofstate = ns;
}
int ACTIONTable::getVTMap(symbolsVT* vt){
    if(vtmap.find(vt) != vtmap.end())return vtmap[vt];
    return -1;
}
void ACTIONTable::insertVT(symbolsVT* vt){
    vtmap[vt] = numofsymbolsvt++;
}
void ACTIONTable::insertSHIFT(int state, symbolsVT* vt, int numOfPro){
    int nvt = getVTMap(vt);
    if(state < numofstate && ~nvt){
        ACTION[state][nvt] = std::make_pair('s', numOfPro);
    }
}
void ACTIONTable::insertREDUCE(int state, symbolsVT* vt, int numOfPro){
    int nvt = getVTMap(vt);
    if(state < numofstate && ~nvt){
        ACTION[state][nvt] = std::make_pair('r', numOfPro);
    }
}
void ACTIONTable::insertACC(int state, symbolsVT* vt){
    int nvt = getVTMap(vt);
    if(state < numofstate && ~nvt){
        ACTION[state][nvt] = std::make_pair('a', 0x3f3f3f3f);
    }
}
std::pair<char, int> ACTIONTable::getACTION(int state, symbolsVT* vt){
    int nvt = getVTMap(vt);
    if(state < numofstate && ~nvt){
        return ACTION[state][nvt];
    }
    return std::make_pair('e', -1);
}
void ACTIONTable::print(){
    std::cerr << "ACTION:" << std::endl;
    std::cerr << numofsymbolsvt << std::endl;
    std::cerr << "\t";
    // for(auto i: vtmap)std::cerr << i.first->getWord() << "\t";
    for(std::map<symbolsVT*, int>::iterator i = vtmap.begin(); i != vtmap.end(); ++i)std::cerr << i->first->getWord() << "\t";
    std::cerr << std::endl;
    for(int i = 0; i < numofstate; ++i){
        std::cerr << i << ": \t";
        for(int j = 0; j < numofsymbolsvt; ++j){
            if(~ACTION[i][j].second)
                std::cerr << ACTION[i][j].first << ACTION[i][j].second << " \t";
            else
                std::cerr << "   \t";
        }
        std::cerr << std::endl;
    }
    std::cerr << std::endl;
}


GOTOTable::GOTOTable(){
    numofsymbolsvn = 0;
    vnmap.clear();
```

```
    memset(GOTO, -1, sizeof GOTO);
    std::cerr << "GOTOTable has created..." << std::endl;
}
void GOTOTable::setNumOfState(int ns){
    numofstate = ns;
}
void GOTOTable::insertVN(symbolsVN* vn){
    vnmap[vn] = numofsymbolsvn++;
}
int GOTOTable::getVNMap(symbolsVN* vn){
    if(vnmap.find(vn) != vnmap.end())return vnmap[vn];
    return -1;
}
void GOTOTable::insert(int state, symbolsVN* vn, int numOfPro){
    int nvn = getVNMap(vn);
    if(state < numofstate && ~nvn){
        GOTO[state][nvn] = numOfPro;
    }
}
int GOTOTable::get(int state, symbolsVN* vn){
    int nvn = getVNMap(vn);
    if(state < numofstate && ~nvn){
        return GOTO[state][nvn];
    }
    return -1;
}
void GOTOTable::print(){
    std::cerr << "GOTO:" << std::endl;
    std::cerr << numofsymbolsvn << std::endl;
    std::cerr << "\t";
    // for(auto i: vnmap)std::cerr << i.first->getName() << "\t";
    for(std::map<symbolsVN*, int>::iterator i = vnmap.begin(); i != vnmap.end(); ++i)std::cerr << i->first->getName() << "\t";
    std::cerr << std::endl;
    for(int i = 0; i < numofstate; ++i){
        std::cerr << i << ": \t";
        for(int j = 0; j < numofsymbolsvn; ++j){
            if(~GOTO[i][j])
                std::cerr << GOTO[i][j] << "\t";
            else
                std::cerr << "   \t";
        }
        std::cerr << std::endl;
    }
    std::cerr << std::endl;
}


analysisTable::analysisTable(int ns):numofstate(ns){
    ACTION.setNumOfState(numofstate);
    GOTO.setNumOfState(numofstate);
    numofpro = 0;
    std::cerr << "An AnalysisTable has created..." << std::endl;
}
void analysisTable::insertSymbols(symbols* s){
    if(s->getClassName() == "VT"){
        ACTION.insertVT((symbolsVT*)(s));
    }
    else if(s->getClassName() == "VN"){
        GOTO.insertVN((symbolsVN*)(s));
    }
}
void analysisTable::insertProduction(production* p){
    productions[numofpro++] = p;
}
production* analysisTable::getProduction(int i){
    if(i < numofpro)return productions[i];
    // return nullptr;
    return NULL;
}
void analysisTable::insert(int state, symbols* s, char ch, int numOfPro){
    if(s->getClassName() == "VT"){
        if(ch == 'a'){
            ACTION.insertACC(state, (symbolsVT*)(s));
        }
        else if(ch == 's'){
            ACTION.insertSHIFT(state, (symbolsVT*)(s), numOfPro);
```

```
        }
        else if(ch == 'r'){
            ACTION.insertREDUCE(state, (symbolsVT*)(s), numOfPro);
        }
    }
    else if(s->getClassName() == "VN"){
        GOTO.insert(state, (symbolsVN*)(s), numOfPro);
    }
}
std::pair<char, int> analysisTable::get(int state, symbols* s){
    if(s->getClassName() == "VT"){
        return ACTION.getACTION(state, (symbolsVT*)(s));
    }
    else if(s->getClassName() == "VN"){
        return std::make_pair('g', GOTO.get(state, (symbolsVN*)(s)));
    }
    return std::make_pair('e', -1);
}
void analysisTable::print(){
    std::cerr << "analysisTable: " << std::endl;
    ACTION.print();
    GOTO.print();
    std::cerr << std::endl;
}
```

LR分析总控程序

根据LR分析程序编写的总控程序，包括初始化、读入、分析以及释放资源等：

```cpp
// SLRAnalysis.cpp
#include<iostream>
#include<cstdio>
#include<string.h>
#include"symbols.h"
#include"symbolsVN.cpp"
#include"symbolsVT.cpp"
#include"production.cpp"
#include"analysisTable.cpp"
const int maxnAnalysisStack = 1e2 + 5;

// 定义出文法的所有终结符
symbolsVT* PLUS = new symbolsVT("+", "plus");
symbolsVT* MINUS = new symbolsVT("-", "minus");
symbolsVT* times = new symbolsVT("*", "times");
symbolsVT* slash = new symbolsVT("/", "slash");
symbolsVT* lparen = new symbolsVT("(", "lapren");
symbolsVT* rparen = new symbolsVT(")", "rparen");
symbolsVT* ident = new symbolsVT("i", "ident");
symbolsVT* unsignint = new symbolsVT("u", "unsignint");
symbolsVT* END = new symbolsVT("#", "end");
symbolsVT* epslion = new symbolsVT("e", "epslion");
// 定义出文法的所有非终结符
symbolsVN* Sdot = new symbolsVN("S'");
symbolsVN* E = new symbolsVN("E");
symbolsVN* T = new symbolsVN("T");
symbolsVN* F = new symbolsVN("F");

// 构造所有的产生式
production* Sdotproduction[1];
production* Eporduction[5];
production* Tproduction[3];
production* Fproduction[3];

// 定义出预测分析表
analysisTable AnalysisTable(21);

// 分析栈
std::pair<int, symbols*> analysisStack[maxnAnalysisStack];
int top;
void init(){
    // 初始化所有变量
        // 根据文法的不同，得到的分析表的结构也不同，此时初始化部分也不同

    // 定义出预测分析表
        // 为预测分析表插入终结符、非终结符
    AnalysisTable.insertSymbols(PLUS);
    AnalysisTable.insertSymbols(MINUS);
    AnalysisTable.insertSymbols(times);
    AnalysisTable.insertSymbols(slash);
    AnalysisTable.insertSymbols(lparen);
    AnalysisTable.insertSymbols(rparen);
    AnalysisTable.insertSymbols(ident);
    AnalysisTable.insertSymbols(unsignint);
    AnalysisTable.insertSymbols(END);

    AnalysisTable.insertSymbols(Sdot);
    AnalysisTable.insertSymbols(E);
    AnalysisTable.insertSymbols(T);
    AnalysisTable.insertSymbols(F);

    // 根据文法定义E的三条产生式，同理处理其他的产生式
    for(int i = 0; i < 1; ++i)Sdotproduction[i] = new production(Sdot);
    Sdotproduction[0]->push_back(E);
    Sdotproduction[0]->print();


    for(int i = 0; i < 5; ++i)Eporduction[i] = new production(E);

    Eporduction[0]->push_back(T);
    Eporduction[1]->push_back(PLUS); Eporduction[1]->push_back(T);
    Eporduction[2]->push_back(MINUS); Eporduction[2]->push_back(T);
    Eporduction[3]->push_back(E); Eporduction[3]->push_back(PLUS); Eporduction[3]->push_back(T);
    Eporduction[4]->push_back(E); Eporduction[4]->push_back(MINUS); Eporduction[4]->push_back(T);
    for(int i = 0; i < 5; ++i)E->insertProduction(Eporduction[i]);
```

```
    for(int i = 0; i < 5; ++i)Eporduction[i]->print();

    for(int i = 0; i < 3; ++i)Tproduction[i] = new production(T);
    Tproduction[0]->push_back(F);
    Tproduction[1]->push_back(T); Tproduction[1]->push_back(times); Tproduction[1]->push_back(F);
    Tproduction[2]->push_back(T); Tproduction[2]->push_back(slash); Tproduction[2]->push_back(F);
    for(int i = 0; i < 3; ++i)T->insertProduction(Tproduction[i]);
    for(int i = 0; i < 3; ++i)Tproduction[i]->print();


    for(int i = 0; i < 3; ++i)Fproduction[i] = new production(F);
    Fproduction[0]->push_back(ident);
    Fproduction[1]->push_back(unsignint);
    Fproduction[2]->push_back(lparen); Fproduction[2]->push_back(E); Fproduction[2]->push_back(rparen);
    for(int i = 0; i < 3; ++i)F->insertProduction(Fproduction[i]);
    for(int i = 0; i < 3; ++i)Fproduction[i]->print();

    for(int i = 0; i < 1; ++i)AnalysisTable.insertProduction(Sdotproduction[i]);
    for(int i = 0; i < 5; ++i)AnalysisTable.insertProduction(Eporduction[i]);
    for(int i = 0; i < 3; ++i)AnalysisTable.insertProduction(Tproduction[i]);
    for(int i = 0; i < 3; ++i)AnalysisTable.insertProduction(Fproduction[i]);

    // 给出LR分析表
    AnalysisTable.insert(0, PLUS, 's', 5);
        AnalysisTable.insert(0, MINUS, 's', 4);
        AnalysisTable.insert(0, lparen, 's', 8);
        AnalysisTable.insert(0, ident, 's', 6);
        AnalysisTable.insert(0, unsignint, 's', 7);
        AnalysisTable.insert(0, E, ' ', 1);
        AnalysisTable.insert(0, T, ' ', 2);
        AnalysisTable.insert(0, F, ' ', 3);
    AnalysisTable.insert(1, PLUS, 's', 9);
        AnalysisTable.insert(1, MINUS, 's', 10);
        AnalysisTable.insert(1, END, 'a', -1);
    AnalysisTable.insert(2, PLUS, 'r', 1);
        AnalysisTable.insert(2, MINUS, 'r', 1);
        AnalysisTable.insert(2, times, 's', 11);
        AnalysisTable.insert(2, slash, 's', 12);
        AnalysisTable.insert(2, rparen, 'r', 1);
        AnalysisTable.insert(2, END, 'r', 1);
    AnalysisTable.insert(3, PLUS, 'r', 6);
        AnalysisTable.insert(3, MINUS, 'r', 6);
        AnalysisTable.insert(3, times, 'r', 6);
        AnalysisTable.insert(3, slash, 'r', 6);
        AnalysisTable.insert(3, rparen, 'r', 6);
        AnalysisTable.insert(3, END, 'r', 6);
    AnalysisTable.insert(4, T, ' ', 13);
    AnalysisTable.insert(5, T, ' ', 14);
    AnalysisTable.insert(6, PLUS, 'r', 9);
        AnalysisTable.insert(6, MINUS, 'r', 9);
        AnalysisTable.insert(6, times, 'r', 9);
        AnalysisTable.insert(6, slash, 'r', 9);
        AnalysisTable.insert(6, rparen, 'r', 9);
        AnalysisTable.insert(6, END, 'r', 9);
    AnalysisTable.insert(7, PLUS, 'r', 10);
        AnalysisTable.insert(7, MINUS, 'r', 10);
        AnalysisTable.insert(7, times, 'r', 10);
        AnalysisTable.insert(7, slash, 'r', 10);
        AnalysisTable.insert(7, rparen, 'r', 10);
        AnalysisTable.insert(7, END, 'r', 10);
    AnalysisTable.insert(8, PLUS, 's', 5);
        AnalysisTable.insert(8, MINUS, 's', 4);
        AnalysisTable.insert(8, lparen, 's', 8);
        AnalysisTable.insert(8, ident, 's', 6);
        AnalysisTable.insert(8, unsignint, 's', 7);
        AnalysisTable.insert(8, E, ' ', 15);
        AnalysisTable.insert(8, T, ' ', 2);
        AnalysisTable.insert(8, F, ' ', 3);
    AnalysisTable.insert(9, lparen, 's', 8);
        AnalysisTable.insert(9, ident, 's', 6);
        AnalysisTable.insert(9, unsignint, 's', 7);
        AnalysisTable.insert(9, T, ' ', 16);
        AnalysisTable.insert(9, F, ' ', 3);
    AnalysisTable.insert(10, lparen, 's', 8);
        AnalysisTable.insert(10, ident, 's', 6);
        AnalysisTable.insert(10, unsignint, 's', 7);
```

```
        AnalysisTable.insert(10, T, ' ', 17);
        AnalysisTable.insert(10, F, ' ', 3);
    AnalysisTable.insert(11, lparen, 's', 8);
        AnalysisTable.insert(11, ident, 's', 6);
        AnalysisTable.insert(11, unsignint, 's', 7);
        AnalysisTable.insert(11, F, ' ', 18);
    AnalysisTable.insert(12, lparen, 's', 8);
        AnalysisTable.insert(12, ident, 's', 6);
        AnalysisTable.insert(12, unsignint, 's', 7);
        AnalysisTable.insert(12, F, ' ', 19);
    AnalysisTable.insert(13, PLUS, 'r', 3);
        AnalysisTable.insert(13, MINUS, 'r', 3);
        AnalysisTable.insert(13, rparen, 'r', 3);
        AnalysisTable.insert(13, END, 'r', 3);
    AnalysisTable.insert(14, PLUS, 'r', 2);
        AnalysisTable.insert(14, MINUS, 'r', 2);
        AnalysisTable.insert(14, rparen, 'r', 2);
        AnalysisTable.insert(14, END, 'r', 2);
    AnalysisTable.insert(15, PLUS, 's', 9);
        AnalysisTable.insert(15, MINUS, 's', 10);
        AnalysisTable.insert(15, rparen, 's', 20);
    AnalysisTable.insert(16, PLUS, 'r', 4);
        AnalysisTable.insert(16, MINUS, 'r', 4);
        AnalysisTable.insert(16, times, 's', 11);
        AnalysisTable.insert(16, slash, 's', 12);
        AnalysisTable.insert(16, rparen, 'r', 4);
        AnalysisTable.insert(16, END, 'r', 4);
    AnalysisTable.insert(17, PLUS, 'r', 5);
        AnalysisTable.insert(17, MINUS, 'r', 5);
        AnalysisTable.insert(17, times, 's', 11);
        AnalysisTable.insert(17, slash, 's', 12);
        AnalysisTable.insert(17, rparen, 'r', 5);
        AnalysisTable.insert(17, END, 'r', 5);
    AnalysisTable.insert(18, PLUS, 'r', 7);
        AnalysisTable.insert(18, MINUS, 'r', 7);
        AnalysisTable.insert(18, times, 'r', 7);
        AnalysisTable.insert(18, slash, 'r', 7);
        AnalysisTable.insert(18, rparen, 'r', 7);
        AnalysisTable.insert(18, END, 'r', 7);
    AnalysisTable.insert(19, PLUS, 'r', 8);
        AnalysisTable.insert(19, MINUS, 'r', 8);
        AnalysisTable.insert(19, times, 'r', 8);
        AnalysisTable.insert(19, slash, 'r', 8);
        AnalysisTable.insert(19, rparen, 'r', 8);
        AnalysisTable.insert(19, END, 'r', 8);
    AnalysisTable.insert(20, PLUS, 'r', 11);
        AnalysisTable.insert(20, MINUS, 'r', 11);
        AnalysisTable.insert(20, times, 'r', 11);
        AnalysisTable.insert(20, slash, 'r', 11);
        AnalysisTable.insert(20, rparen, 'r', 11);
        AnalysisTable.insert(20, END, 'r', 11);
    AnalysisTable.print();

        // 初始化分析栈
    top = -1;
}
void release(){
        // 释放所有的动态申请的资源
    delete PLUS;
        delete MINUS;
        delete times;
        delete slash;
        delete lparen;
        delete rparen;
        delete ident;
        delete unsignint;
        delete END;
        delete epslion;
    delete E;
        delete T;
        delete F;
    for(int i = 0; i < 1; ++i)delete Sdotproduction[i];
    for(int i = 0; i < 5; ++i)delete Eporduction[i];
    for(int i = 0; i < 3; ++i)delete Tproduction[i];
    for(int i = 0; i < 3; ++i)delete Fproduction[i];
}
```

```
std::string word, code;
// char word[10], code[10];
char ch;
symbolsVT* a;
void ADVANCE(){
        // 读入一个词法分析的结果项，同时给出对应的终结符a
    // if(scanf("(%s,%s)", code, word) != -1){
    std::cin >> ch;
    if(!std::cin.eof()){
    // if(scanf("%c", &ch) != -1){
        std::getline(std::cin, code, ',');
        std::getline(std::cin, word);
        word.resize(word.size() - 1);
        // std::cin >> ch;
        std::cerr << word << " " << code << std::endl;
        if(code == "plus")a = PLUS;
        else if(code == "minus") a = MINUS;
        else if(code == "times") a = times;
        else if(code == "slash") a = slash;
        else if(code == "lparen") a = lparen;
        else if(code == "rparen") a = rparen;
        else if(code == "ident") a = ident;
        else if(code == "number") a = unsignint;
    }
    else{
        a = END;
    // if(std::cin.eof() == EOF){
        std::cerr << "hhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhhh" << std::endl;
    }
    std::cerr << word << "_____________" << code << std::endl;
}
bool SLRAnalysis(){
        // 预测分析程序的总控程序
    init();
    bool grammer = true;                // 表示句子是否符合一定的文法
    bool flag = true;                   // 总控程序的运行标志
    analysisStack[++top] = std::make_pair(0, (symbols*)END);  // 初始化栈，将状态0和符号#压入
    std::pair<int, symbols*> X;                                 // 定义一个公共变量：状态和符号的指针
    production *p;                              // 定义一个产生式的指针
    std::pair<char, int> state; // 从分析表中获得的状态信息
    ADVANCE();                                  // 读入一个词法分析的结果项
    while(flag){
                //*************************************************************//
                // 调试信息：状态栈和符号栈的中内容
        std::cerr << std::endl << std::endl;
        std::cerr << "================" << std::endl;
        a->print();
        std::cerr << "stack: " << std::endl;;
        for(int i = 0; i <= top; ++i){
            std::cerr << analysisStack[i].first << " ";
        }
        std::cerr << std::endl;
        for(int i = 0; i <= top; ++i){
            if(analysisStack[i].second->getClassName() == "VT")std::cerr << ((symbolsVT*)(analysisStack[i].second))->getWord() << "
            else std::cerr << ((symbolsVN*)analysisStack[i].second)->getName() << " ";
        }
        std::cerr << std::endl << "================" << std::endl;
        std::cerr << std::endl;
                //*************************************************************//

        X = analysisStack[top]; // 得到分析栈的栈顶元素，pop操作
        state = AnalysisTable.get(X.first, a);  // 根据栈顶的状态以及分析表中的变化情况来获得下一转换的状态s_i, r_i, acc, i等等
        std::cerr << state.first << "  " << state.second << std::endl;
        if(state.first == 's'){         // 如果是移进状态
            analysisStack[++top] = std::make_pair(state.second, a);
            ADVANCE();
            std::cerr << "One SHIFT..." << std::endl << std::endl;;
        }
        else if(state.first == 'r'){    // 如果是规约状态
            p = AnalysisTable.getProduction(state.second);      // 获得第i个产生式
            p->print();
            int len = p->getLen();
            top -= len;                                 // 将栈顶的符号按照产生式来规约
            X = analysisStack[top];             // 获得此时的栈顶元素，据此来获得GOTO表的下一状态
            analysisStack[++top] = std::make_pair(AnalysisTable.get(X.first, p->getVN()).second, p->getVN());
            std::cerr << "One REDUCE..." << std::endl << std::endl;;
```

```
        }
        else if(state.first == 'a'){    // 如果是acc状态
            std::cerr << "ACC!!!" << std::endl << std::endl;
            flag = false;
        }
        else{                                                   // 到达分析表的其他状态，错误
            grammer = false;
            flag = false;
        }
    }

    release();          // 释放资源
    return grammer;     // 返回结果，true表示句子符合一定的语法
}
```

程序入口

项目调用测试入口：

```
// main.cpp
#include<bits/stdc++.h>
#include"SLRAnalysis.cpp"
using namespace std;
typedef long long ll;
const int mod = 1e9 + 7;
const int maxn = 1e2 + 5;
const int inf = 0x3f3f3f3f;


int main(int argc, char *argv[]){
    // freopen("in.in", "r", stdin);
    // freopen("out.out", "w", stdout);

    if(SLRAnalysis()) cout << "Yes,it is correct." << endl;
    else cout << "No,it is wrong." << endl;

    return 0;
}
```

调试数据

给出两组测试数据：

```
// in.in
(lparen,()
(ident,a)
(plus,+)
(number,15)
(rparen,))
(times,*)
```

```
// out.out
No,it is wrong.
```

```
// in.in
(ident,akldjsfkl)
(plus,+)
(lparen,()
(ident,alk)
(times,*)
(ident,askd)
(minus,-)
(ident,jfdkj)
(times,*)
(ident,ksfj)
(slash,/)
(ident,jsadlk)
(plus,+)
(lparen,()
(ident,a)
(slash,/)
(ident,v)
(minus,-)
(ident,d)
(plus,+)
(ident,b)
(rparen,))
(rparen,))
(slash,/)
(ident,jfj)
```

```
// out.out
Yes,it is correct.
```

实验体会

本次实验是完成 **LR分析法** 的自下而上分析，相比较上一次实验的预测分析法，LR分析法主要的差别是分析表的构造上，通过活前缀DFA的构造，以及由此分析得到SLR分析表，将图的转换变为表内各状态的转换。实验中，自我认为最难的并不是程序的编写，而是构造活前缀DFA以及得到分析表的步骤，这一步中画图花费了很多的时间，同时也再一次的复习了相关的理论知识内容，当这一切准备工作完成后，只需根据此编写相关的代码即可，以为上个实验已经完成了最基础的诸如符号、产生式等类的编写，所以这次只需考虑分析表以及分析程序的编写，这次实验的简化也一定程度上达到了上一次实验中 **代码重复使用** 的目的。实验整体难度不大，是进一步的对理论知识的复习过程。

HTML