专题3 LL(1)分析法

13281193 田容先

# 程序功能

此程序为语法分析程序，可以根据词法分析程序所得结果，对字符串用文法规则判断，看是否满足当前文法的要求。使用的是ll(1)分析表。

# 程序结构

## 主要数据结构

char input[30];//输入文法

char first[30][30];//first集

char follow[30][30];//follow集

char lowercase[30];//终结符号

char uppercase[30];//非终结符号

typedef struct{//文法，->左边是head，右边是tail

char head;

char tail[10];

bool flag;

}grammar;

char tm1[15], tm2[15];//tm1是table的横坐标，tm2是纵坐标

grammar M[10][10];//the parse table

grammar sentence[12] = { { 'E', "TD" }, { 'D', "ATD" }, { 'D', NONE }, { 'T', "FR" }, { 'R', "MFR" }, { 'R', NONE }, { 'F', "(E)" }, { 'F', "i" },

{ 'A', "+" }, { 'A', "-" }, { 'M', "\*" }, { 'M', "/" } };//用D代替E' R代替T'

/////////////栈///////////////////

#define SElemType char

typedef struct {//栈结构

SElemType \*base;

SElemType \*top;

int stacksize;

} SqStack;

## 函数定义

void findlowercase();//提取这个文法中的终结元素

void finduppercase();//提取这个文法中的非终结元素

void FIRST();//计算first集

void FOLLOW();//计算follow集

void read();//读入专题一中的输入字符串

void initsen();//初始化文法中的每个句子

int mapfirst(char head);//给出a,求head在first(a)中的位置

int mapfollow(char head);//给出a,求head在follow(a)中的位置

int maptable(char head,char tm[]);//求head在tm[]中的位置

bool Add(char dest[], char source[]);//将source中的元素插入到dest中，重复元素不增加

bool isuppercase(char c);//判断c是否为非终结符号（大写字母）

bool islowercase(char c);//判断c是否为终结符号（小写字母）

bool hasNone(char c);//first(c)中是否有ε

bool issymbol(char c);//判断c是否为终结符号（运算符）

bool isterminal(char c);//判断c是否为终结符号（全部）

bool exist(char[], char c);//判断c是否存在于char[]中

void inittable();//初始化分析表

void maketable();//生成分析表

bool control(SqStack &S);//总控程序

## 设计方法

此实验比较复杂，可能是我的代码不够简洁，使用了大量的代码。这个分析法一共分为几部分：产生first和follow集，生成LL(1)分析表，根据分析表进行判断。这三部都不是很容易，其中最复杂的是第一步。

许多地方原来我都考虑了数组末尾置为’\0’。后来发现我使用的vs2013版本，默认将数组元素全部置为’\0’，这样就不必再多此一举。

## 设计难点

**1**.**各个数据的使用**

在first和follow集中，我将first(a)的a放在数组首地址处。而在table中，我用了两个数组来存table表的表头，这也是我对这种映射的一种测试（事实证明比较麻烦）tm1和tm2两个数组分别从横坐标和纵坐标。他们的数组标号就是table的标号，他们的内容就是table那个位置存的内容。

我在使用数据时，有些麻烦，总会出现类似的结构：first[mapfirst(g.tail[0])][n]其中调用关系比较复杂，调试起来比较麻烦。这个部分以后我会尽量简洁。

ε如何表示我也修改过。一开始我想用空字符串来表示，但后来发现这样在识别时有问题，最后决定用ascii=1的元素来宏定义ε。 #define NONE 1//ε

**2.first集的建立**

first集虽然比较好理解，但是在ppt中的描述比较复杂，实现起来要考虑各种各样的特殊情况，比较复杂。而且ppt中有的地方描述不是很清楚，我在没动脑子的情况下，误解了意思，后来代码还要重新写。

做first集首先要计算终结元素和非终结元素，这两个部分是搜索整个文法的字符之后得出的，文法全部写在sentence数组中。我默认文法是固定的，所以将文法预先写入了数组中。就算要改变文法，也不会很困难。

first集是一个二维数组，我按照写入顺序来排列，首先是终结符的first集，全写完之后再接着写非终结符。first集的第一个元素是first(a)中的a。follow集也是一样的结构。

求first集，首先将所有终结符放入first集，只需要照抄就可以。

然后将非终结符放入，找到每个文法head的first集加入。

第三步中循环增加元素，直到元素不再改变为止。判断是否改变，我使用了flag标志位。刚进入循环时，置flag为false，如果变化了，则置flag为true。

这里我用了这种形式，

flag = flag | Add(first[mapfirst(sentence[i].head)], first[mapfirst(sentence[I].head)]);

add函数会返回是否成功进行了增加，只要有一处改变，flag就要变为true，所以使用或运算。

void FIRST(){//first集0号位是该符号本身，1号位开始是它的集合

findlowercase();

finduppercase();

int upperp = 0;

int firstp = 0;//当前first集的行指针

bool flag = true;//判断文法是否还在变化

while (flag){

flag = false;

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

for (firstp = 0; lowercase[firstp] != '\0'; firstp++){//为终结符号时

first[firstp][0] = first[firstp][1] = lowercase[firstp];

//first[firstp][2] = '\0';

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

for (upperp = 0; uppercase[upperp] != '\0'; upperp++, firstp++){//为非终结符号时

int i = 0;

int firstq = 1;//当前行的指针

first[firstp][0] = uppercase[upperp];

while (i<SenLength){//在文法中搜索一轮，找到当前所求的first集

if (sentence[i].head == uppercase[upperp] && (isterminal(sentence[i].tail[0]) || sentence[i].tail[0] == 1)){

if (first[firstp][firstq] != sentence[i].tail[0]){ flag = true; }

first[firstp][firstq] = sentence[i].tail[0];

firstq++;

}

i++;

}

}

first[firstp][0] = '\0';

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

int i;

for (i = 0; i<SenLength; i++){//每个文法都要进行第三步i，目前判断的是sentence[i]

//i

if (isuppercase(sentence[i].tail[0])){

flag = flag | Add(first[mapfirst(sentence[i].head)], first[mapfirst(sentence[i].tail[0])]);

}

}

for (i = 0; i < SenLength; i++){//每个文法都要进行第三步ii，目前判断的是sentence[i]

//ii

int I = 1;//课件中的i

int j = I - 1;//课件中的j

bool fl = false;

while (isuppercase(sentence[i].tail[I]) && sentence[i].tail[I] != '\0'){

if (!hasNone(sentence[i].tail[j])){

break;

}

fl = true;

I++;

j = I - 1;

}

I--; //此时j所指元素为第一个first集无ε的，I指向'\0'，所以应该往前移一位，之后I指向为第一个first集无ε的

if (fl){ flag = flag | Add(first[mapfirst(sentence[i].head)], first[mapfirst(sentence[I].head)]); }

//iii

if (isuppercase(sentence[i].tail[I]) && sentence[i].tail[I + 1] == '\0' && I != 0){//说明I是最后一个，所有符号都是非终结符号

char ch[1];

ch[0] = NONE;

flag = flag | Add(first[mapfirst(sentence[i].head)], ch);

}

}

}

}

**3.分析表建立**

follow集算法相对简单，在此不再赘述。

在这里，我也设立了一个flag，作用是判断first集中是否有ε，如果有的话就要进行下面的，查找follow集的操作。这个部分的循环嵌套比较多，我在如何嵌套的部分花了一些时间。

void maketable(){

FIRST();

FOLLOW();

inittable();

int i, j;

grammar g;

char a;

bool flag = false;//判断是否ε属于first()

for (i = 0; tm2[i] != '\0'; i++){

//从sentence中找当前第i行非终结符的文法

for (j = 0; sentence[j].head != '\0'; j++){

if (sentence[j].head == tm2[i] && sentence[j].tail[0] != NONE){

g = sentence[j];

int n;

for (n = 1; (a = first[mapfirst(g.tail[0])][n]) != '\0'; n++){//循环sentnece[i].tail[0]的first集

//a是first(D)的元素(E->DT)

//这段是找follow集求E->ε的部分

if (first[mapfirst(g.head)][n] == NONE){

flag = true;

}//说明需要输入E->ε,查看follow集

//////////////////////////////////

int k;

for (k = 0; tm1[k] != '\0'; k++){//tm1[k]是table中k号位的表头

if (a == tm1[k]){

M[i][k] = g;

break;

}

}

}

if (flag == true){

for (n = 1; (a = follow[mapfollow(tm2[i])][n]) != '\0'; n++){

int k;

for (k = 0; tm1[k] != '\0'; k++){

if (tm1[k] == a ){

M[i][k].head = tm2[i];

M[i][k].tail[0] = NONE;

M[i][k].flag = true;

}

}

flag = false;

}

}

}

}

}

}

**4.主程序**

总控程序也不是很难，只要将ppt中的伪代码实现就好了。但ppt中if else的对应关系让人误解，所以还有需要改一下代码。

bool control(SqStack &S){

int i = 0;//the pointer to point the word being input

char a = input[0];

char x;

bool flag = true;

Push(S, '#');

Push(S, 'E');

while (flag == true){

Pop(S, x);

if (isterminal(x)){//lower case

if (x == a){

a = input[++i];//NextSym

}

else{

return ERROR;

}

}

else if (x == '#'){

if (x == a){

i++;//NextSym

flag = false;

}

else return ERROR;

}

else if (M[maptable(x,tm2)][maptable(a,tm1)].flag == true){//倒序入栈

grammar t = M[maptable(x, tm2)][maptable(a, tm1)];

if (t.tail[0] != 1){//说明不推ε

int i;

for (i = 0; t.tail[i] != '\0'; i++){}

i--;

for (; i >= 0; i--){

Push(S, t.tail[i]);

}

}

}

else return ERROR;

}

return true;

}

# 程序测试

## 测试用例

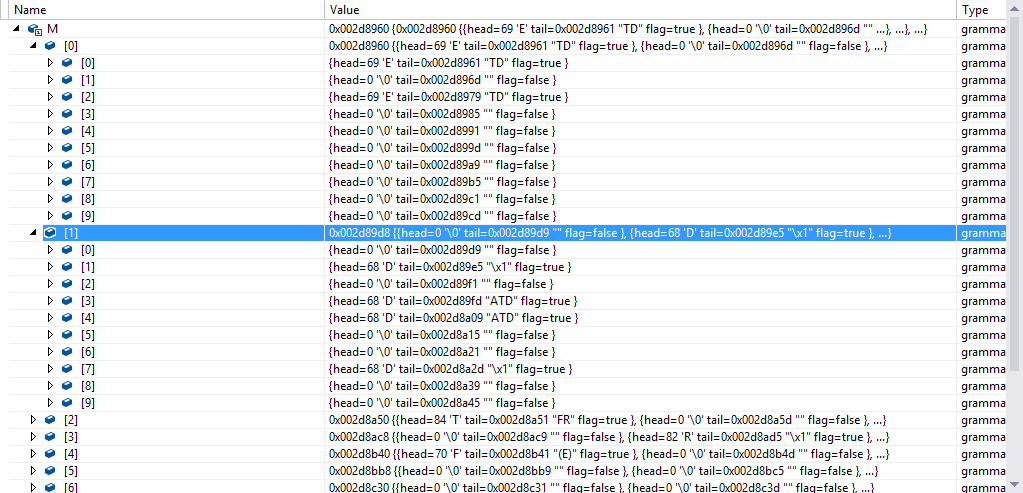
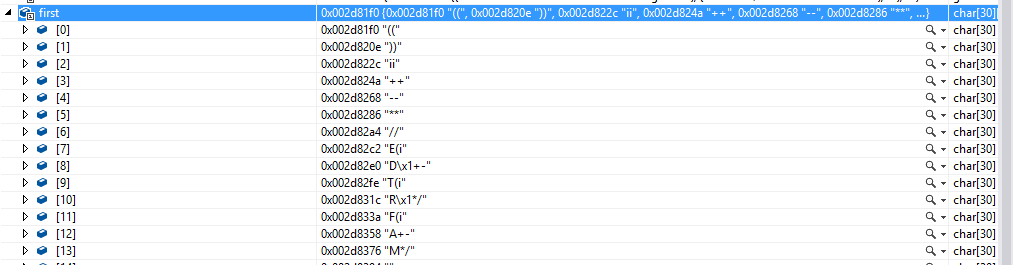
out.txt

10 i 12 10 i 14 10 i

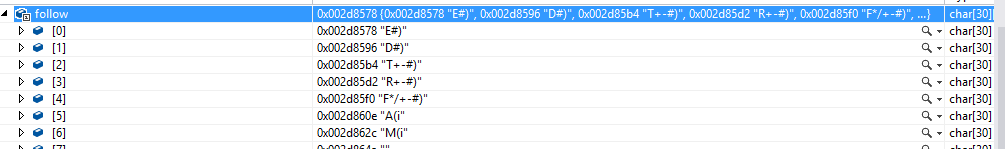
//前面有个空格，最后有空格

其中10表示标识符，12表示+ 14表示\*。 “i+i\*i”

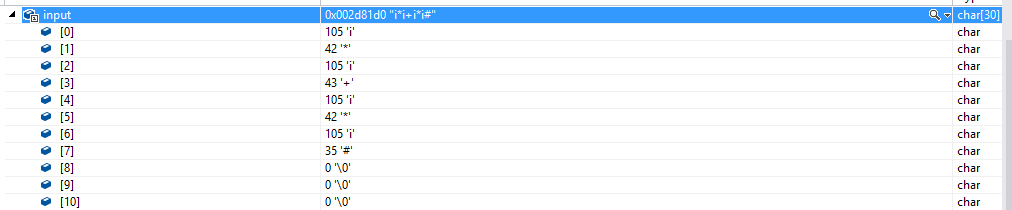
## 测试结果

 分析表列出两个项，其他类似。

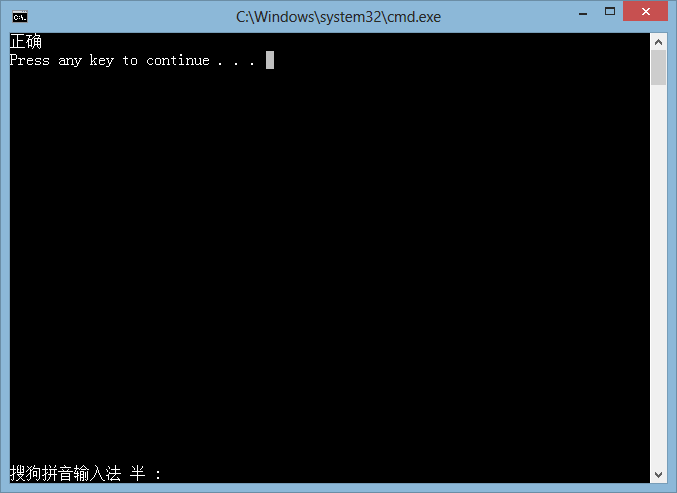
first集



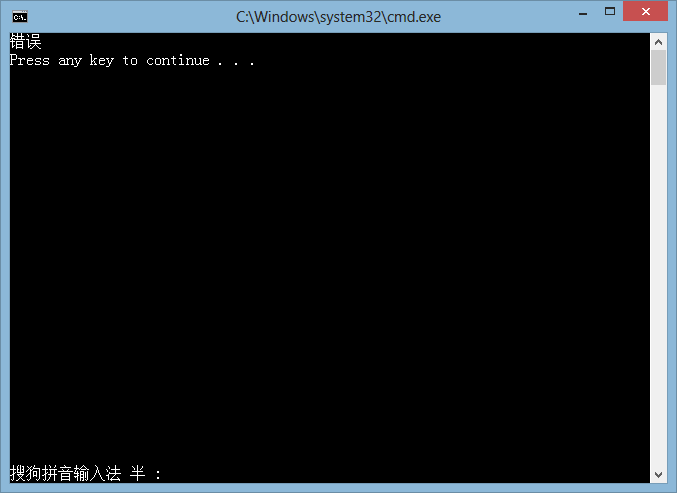
follow集



输入串



错误用例：

10 i 14 10 i 12 10 i 14 10 i 14  “i+i\*i\*”

# 实验总结

本次实验学习了LL(1)分析法，这个是这些实验中比较难的一个了，完成的也比较困难。这个程序的bug比较多，不容易调试成功。但同时也从这里面学到了不少的知识，有所收获。