

评阅项1计分＿＿＿＿＿＿

评阅项2计分＿＿＿＿＿＿

评阅项3计分＿＿＿＿＿＿

评阅项4计分＿＿＿＿＿＿

实验得分＿＿＿＿＿＿

评阅日期＿＿＿＿＿＿

**编译原理实验**

实验一：由底向上语法分析及中间代码生成

实验选题：由底向上语法分析

学生姓名：

学 号：

专业班级：

任课教师： 曹江莲

提交日期：

1. **实验目的**

通过本次实验，加深对移进规约分析法LR(0)的理解，学习程序设计语言的语法分析器的编程方法，理解中间代码生成的过程。

1. **实验内容**

输入一个CFG，及一个句子，判断文法是否是哪一类LR文法（限LR(0)和SLR(1)），要求输出文法的识别活前缀的DFA及分析表，并实现给定语句的分析过程。

1. **算法思想及问题对策**

1. 算法描述

将文法拓广，然后列出所有的活前缀，用闭包的方法构造LR0项目集规范族，搜索所有的项目集，看是否存在有移进\规约冲突或者一个项目集中有多个规约项，存在则SLR1方法构造分析表，否则LR0方法构造分析表。然后用遍历的方法得到DFA的状态转化图，并且用Go函数存储，利用Go函数创建LR0（或者SLR1）分析表。最后当然就是根据分析表，利用LR分析器来进行字符串的分析了。（注意，代码中用@代替活前缀中的点）

2．设计过程中的问题与对策

对于DFA的求解，由于是从一个状态出发，然后遍历字母表，查找有没有从当前状态，接受到字母表中的元素到达下一个状态，又就创建一个新的状态，由于在DFA中可能会出现循环的情况，所以我们这一过程会无休止地运行下去，所以我们需要判断下一状态是否是已经在之前创建的状态中存在的了，存在则不作处理，不存在则新建一个状态。

对于构造好分析表之后的分析，由于对于分析过程的了解不够深刻，导致对于Goto的处理不够严谨，每次Goto应该将状态向前移动之前规约的右部的个数再进行Goto的处理。

对于输入的产生式不好存储，所以采用字符串的方式输入，字符串的第一个字符产生式的左部，之后的字符串为产生式的右部。

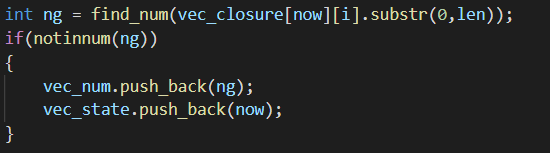
对于最后分析表的输出，由于存储的字母表没有按顺序，所以构造Action和Goto表的时候需要判断字母表的字符为哪一个，A->Z则记录到Goto表中，其他则记录到Action表中。

3. 本实验的特色

由于本实验的代码长度比较长，所以将功能函数全部都封装了起来，使代码看起来更加的整洁清晰，代码结构逻辑明了。

使用了c++的string和STL的vector和map，代码的实现更加的简单，程序的兼容性和健壮性更加的强。

对于Action，Goto和Go表的存储都是用的map，节约存储空间，提高搜索的效率，first和follow集合的定义也是用的map。

1. **具体实现**
2. 全局变量的说明（用代码中的注释说明）
3. 
4. Vec\_num和vec\_state的配合使用代码。

|  |  |
| --- | --- |
|  | **char AA = '@';//代替区分项目集的点**  **string inputstring;//输入的字符串**  **string instr;//产生式的输入存储**  **string str[20];//str[0]用来拓广，下标为0用来表示产生式的左部。**  **vector<string> vec\_str;//存储所有的活前缀。**  **vector<string> vec\_closure[500];//用来存储某个状态的lr0项目族，closure**  **vector<char> vec\_elem;//用来记录所有的终结符和非终结符**  **vector<int> vec\_num;//中间有下划线，显示不出来。**  **vector<int> vec\_state;//num和state是配合使用的相同的下标对应相同的元素，state是存储当前状态的前一个状态，num存储当前状态**  **map<int,map<char,int> >Action;//用来存储nfa状态转换图**  **map<int,map<char,int> >Goto;//用来记录goto**  **map<int,map<char,int> >Go;//用来记录DFA的状态转换映射函数**  **map<char,string> first;//first集合**  **map<char,string> follow;//follow集合** |

1. 普通函数的说明

|  |  |
| --- | --- |
| 1 | **int notelem(char a) 输入字符a,判断字符是否在输入的文法中存在，存在即返回0否则返回1.** |
|  |  |
| 2 | **void reads() 输入文法，然后对输入的文法进行了一系列的处理，首先就是拓广将拓广后的文法拆分，按照不同的左部非终结符存储到不同的instr[X]中，然后就是将文法中出现的终结符和非终结符压入到vec\_elem中** |
|  |  |
| 3 | **int canbedo() 对所有的项目集进行判断，若某个项目集存在移进\规约冲突或者存在多个规约项目，那么返回0，如果全部遍历完毕没有发现上述情况则返回1.** |
| 4 | **int inThisfirst(char a,char b)判断字符b是否在first[a]里面，如果在就返回1，否则返回0**  **void get\_first() 需要用到上面的inThisfirst函数。遍历所有的拓广后的文法str，分2中情况进行判断：**  **A.若第一个为终结符，则判断这个终结符是否已经在str[0]的first集合中，不在则加入该first集合中，否则跳过。**  **B.若第一个为非终结符，那么遍历该非终结符的first集合，对每一个first集合中的元素执行A。**  **循环执行上述步骤，直到所有的first集合都没有变化，用一个flag变量来标识。** |
|  |  |
| 5 | **int inThisfollow(char a,char b) 该函数和inThisfirst函数一样，都是判断b字符是否在follow[a]集合之中**  **void get\_follow() 该函数用来求follow集合，遍历所有的文法项目，找非终结符，然后分情况处理：**  **如果该非终结符是产生式的最后一个字符，那么将左部非终结符的follow集加入到改非终结符的follow集合之中，当然对于每个元素都需要进行判断是否已经存在于该follow集合之中了，存在则跳过，不存在则加入。**  **如果该非终结符不是最后一个，那么又可以分2中情况：**   1. **若该非终结符的后一个元素为终结符，则将该终结符加入到这个非终结符的follow集合之中（唯一性判断不在继续说明了）。** 2. **若后一个元素为非终结符记为s，那么将first[s]除去空集的值加入到该非终结符的follow集合之中（唯一性判断）。** |
|  |  |
| 6 | **int findx(char a)， 在str中查找左部非终结符为a的产生式，返回该产生式在vec\_str中的下标，用于之后规约时便于找到规约的产生式。** |
|  |  |
| 7 | **int in\_follow(char fo,char c)查找字符c是否在fo的follow集合之中** |

1. 关键函数的说明

|  |  |
| --- | --- |
| 1 | **void closure()首先获取所有的活前缀，然后将活前缀压入vec\_str数组之中，然后调用getclosure()获取所有的项目集和构造DFA。然后就是打印项目族。** |
|  |  |
| 2 | **void toadd(int now,int i)这个函数是对一个已经创建的项目进行构造，因为活前缀的.之后的一个元素如果是非终结符的话，那么就要将左部为该终结符的并且.在第一位的项目加入到该项目集中。当然不能有重复的加入，所以要判断每次的加入是否已经存在。**  **if(vec\_str[flag][1]=='@'&&!inthis(now,vec\_str[flag]))这段代码就是用来判断重复。**    **int inthis(int cll,string a)输入项目集的下标cll和字符串a，判断该字符串是否出现在下标为cll的项目集中** |
|  |  |
| 3 | **int inclosure(string a) 判断某个项目是否已经在已经产生的状态之中存在了。**  **int notinnum(int nu) 寻找vec\_num中是否有nu，存在返回0，否则返回1.**  **int find\_num(string a) 在str中查找字符串a，找到返回其下标，否则返回-1，好像和前面一个函数比较像。** |

最关键的就是void getclosure()，这个函数的功能是产生DFA项目集，我的实现就是首先创建一个0状态，也就是拓广产生的第一个状态，然后对该状态进行toadd函数的拓展，然后遍历所有的字母表元素，看项目中有没有结果该元素所能到达的字符串，存在即可认为有一个可以进行处理的新状态。当然该状态不一定是一个新的状态，他可能是之前存在的某一状态的值，也有可能是当前状态的一个值，所以要进行判断

|  |  |
| --- | --- |
|  | **if(inclosure(vec\_str[k])==-1)//第一个,在其他的状态中都没有找到。**  **{**  **if(flag)**  **{ vec\_closure[cllo].push\_back(vec\_str[k]);**  **flag = 0;**  **Go[now][elem] = cllo;**    **}**  **else{**  **vec\_closure[cllo].push\_back(vec\_str[k]);**  **Go[now][elem] = cllo;**  **}**  **}**  **else{//处理在其他项目集里有的情况。在其他项目集找到了**  **if(!flag) vec\_closure[cllo].push\_back(vec\_str[k]);**  **string a = vec\_str[k];**  **for(int i = 0;i<=cllo;i++)//在其他项目集找该字符串**  **{**  **for(int j = 0;j<vec\_closure[i].size();j++)**  **{**  **if(vec\_closure[i][j]==a)**  **{**  **Go[now][elem] = i;**  **}**  **}**  **}**  **}** |

上面代码的格式有点乱，说明一下，就是如果在已经存在的状态中找到该状态则不进行处理，如果没有找到，若已经生成了当前状态则把该值放到这个状态之中，否则不就行处理。在进行项目集生成的同时也可以创建好Go表（见上图中Go项目的赋值）

|  |  |
| --- | --- |
| 1 | **int findstr(string n) 和find\_num一样，找vec\_str中的字符串n的位置** |
| 2 | **void lr0()和void slr1()是分别用来创建LR0分析表和SRL1分析表，利用DFA的映射可以很容易生成，只不过需要对于Action和Goto进行特判，非终结符的跳转存于Goto中，终结符的跳转存于Action表中。**   1. **LR0的分析对于规约项目的Action要将该行全部置为规约项在vec\_num中的位置，为了和之前的状态转移区分开来，我们将其值设置为负数。**   **for(int i = 0;i<vec\_state.size();i++)**  **{**  **for(int j = 0;j<vec\_elem.size();j++)**  **{**  **if(vec\_elem[j]>='A'&&vec\_elem[j]<='Z') continue;**  **else{**  **Action[vec\_state[i]][vec\_elem[j]]=-vec\_num[i];**  **}**  **}**  **}** |
|  | 1. **SLR1的设置就是不是全行设置，而是要找到规约项目的左部非终结符的follow集合，将follow集合中所有终结符的位置置为规约项目位置的负数。**   **for(int i = 0;i<vec\_state.size();i++)**  **{**  **for(int j = 0;j<vec\_elem.size();j++)**  **{**  **int n = vec\_num[i];**  **if(!in\_follow(str[n][0],vec\_elem[j])) continue;**  **if(vec\_elem[j]>='A'&&vec\_elem[j]<='Z') continue;**  **else{**  **Action[vec\_state[i]][vec\_elem[j]]=-vec\_num[i];**  **}**  **}**  **}** |

最后就是分析函数了，这个函数很简单

|  |  |
| --- | --- |
|  | **vector<char> char\_stack;//符号栈**  **char\_stack.push\_back('#');**  **vector<int> state\_stack;//状态栈**  **state\_stack.push\_back(0);**  **int p = 0;//指示输入串的位置指针**  **int top\_state = 0;**  **char top\_char;** |

每次获得栈的栈顶，和输入串的当前输入符号，求他们的Action表，分2中情况：

1. 若得到的值是整正数，那么将该状态值压入栈顶，输入串的指针后移一位。
2. 若得到的是一个负数，将该负数置换为正数，找到规约的str项目获得该项目的左部S以及右部元素的个数len，将状态栈和符号栈弹出len个元素（若没有这么多则报错），然后将S压入符号栈的栈顶，压入当前状态。

重复执行上述步骤，直到程序报错或者遇到acc，程序退出。

1. **测试与运行情况**
2. 测试样例数据

测试用的文法

E-->(L)|a

L-->L,E|E

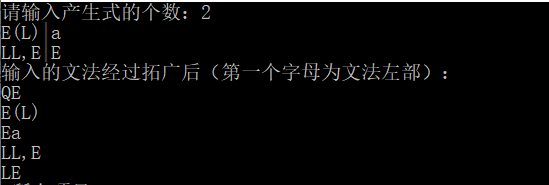
程序中输入数据如下：

|  |  |
| --- | --- |
|  | **2**  **E(L)|a**  **LL,E|E** |

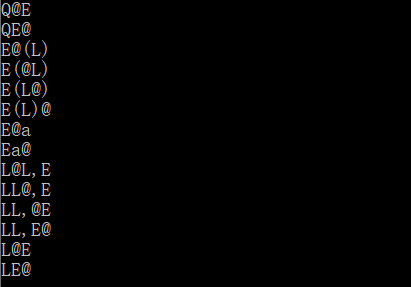
第一个字符是产生式的左部，之后为产生式的右部。

1. 程序执行结果

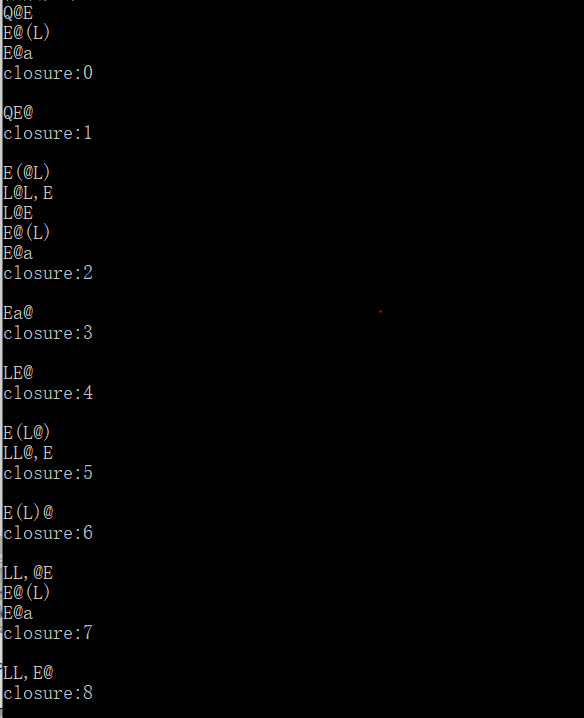
文法的输入和拓广



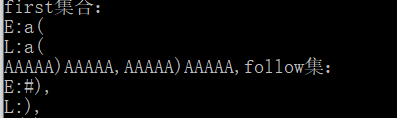
文法产生的所有项目（活前缀）



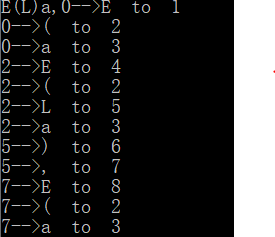
文法的所有项目集



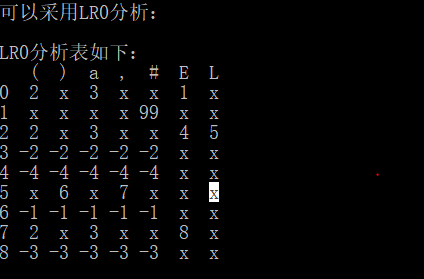
求first和follow集合



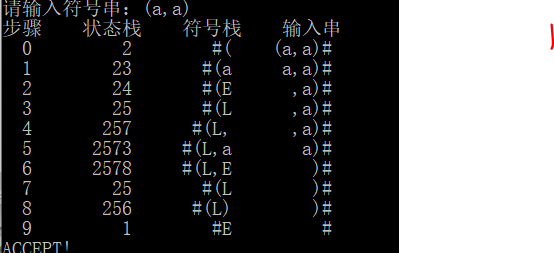
DFA的状态的映射输出



判断是否可以进行LR0分析或者SLR1分析，构造分析表



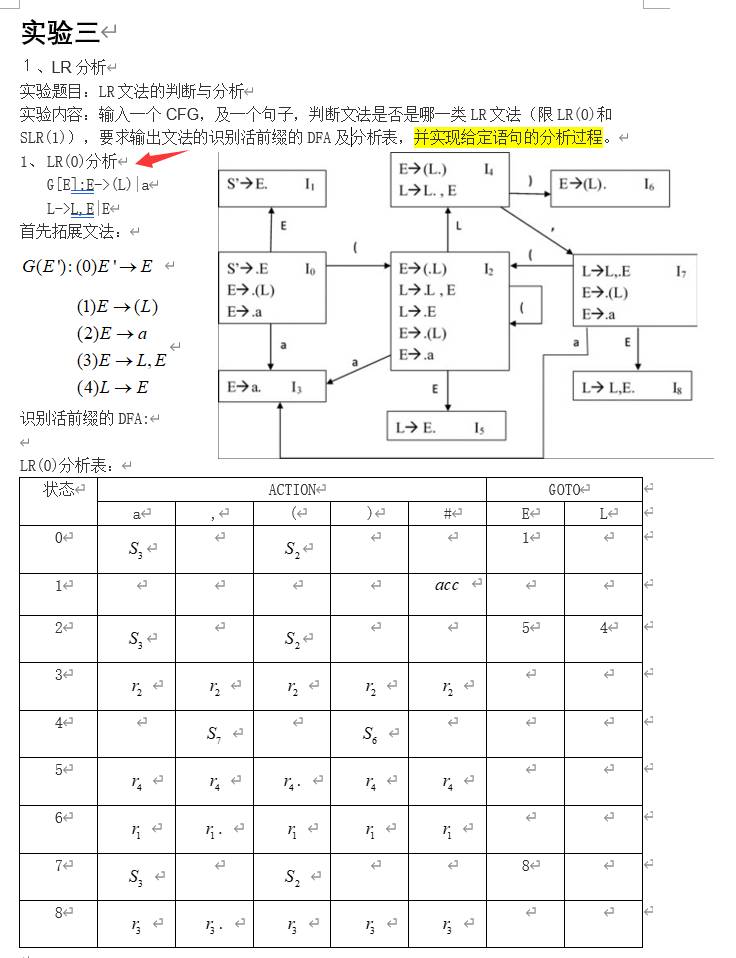
输入字符串，进行分析。



1. 程序执行结果分析

对照下图的DFA和分析表，和程序的输出结果一致，所以DFA和分析表的设置都是么有问题的。

对于字符串（a,a）的分析，通过手动分析，结果也与输出的结果一致。



1. **实验总结**
2. 程序存在的问题
3. 没有对产生式右部有空的情况进行处理。
4. 对于DFA的构造的算法还是有些粗糙，算法的思路较为简单，实现的代码写的过于的复杂。
5. 对于最后分析的时候状态栈的输出采用的是int的类型，这样就导致第一个状态为0会无法在cmd窗口显示。
6. 由于没有对空进行处理，所以follow集的求法也非常的简单，没有去考虑空的情况。

2. 收获

A) 通过这次实验对于从底向上的分析方法的理解更加的深刻了，对于分析表的构造也更加的得心应手了。

B) 了解了LR0分析方法和SLR1分析方法的区别，了解了他们所不能处理的冲突情况，并且能够对这些冲突情况进行简单的判别。

C) 熟练掌握了项目集规范族闭包的求法。

D) 完善了对于LR分析的知识体系，以前的分析是照着书本上的步骤一步步地分析下去，没有去了解为什么要这样分析，通过这次实验对此进行了深入的了解。