编译原理第二次实验报告

1. 概述

这是编译原理的第二次实验报告

这次实验采用自下而上的语法分析技术。对于给定的若干个上下文无关语法规则，构造其**LR(1)**语法分析表。再对于某个特定的表达式，确定其是否是该语法的某个句子。

2. 实验环境

实验环境是**Windows10**系统，使用的语言是**Java**语言

3.数据结构描述

```
    private char     []VN         =new char[50];       //非终结符集
    private char     []VT         =new char[50];       //终结符集
    private String []F            =new String[50];     //产生式集
    private StringBuffer []FirstVN;                    //非终结符的First集
    private int []staStack   =new int[50];            //状态分析栈
    private char     []symStack   =new char[50];       //符号分析栈
    private boolean[]VNE;                               //非终结符与空串的关系表
    private int      F_index      =0;                   //产生式数组指针
    private int staStack_index=0;                      //状态栈指针
    private int      symStack_index=0;                 //符号栈指针
    private int      ERROR        =Integer.MAX_VALUE; //出错标志
    private char     emp          ='ε';                 //空串
    private String error         ="x";                 //分析表显示的出错标志
    private String acc           ="acc";               //分析表显示的分析成功标志
    private Vector<Vector<String>>  State        =new Vector<Vector<String>>();  //项集  //
    private int [][]Action;                            //Action动作数组
    private int [][]Goto;                              //Goto动作数组
    private StringBuffer  []bridge1;                   //描述项集之间的转换关系，在createLR1()中初始化
```

4. 思路与方法

按照书上的算法

1. 先构造终结符和非终结符列表

```
private void init(String s);
```

2. 先扩展语法产生式

```
private void addf0();
```

3. 构造每个非终结符的first集合

```
private void createAction();
```

4. 构造LR项集族

```
private void createLR1();
```

5. 接着构造action表和goto表

```
private void createAction();

private void createGoto();
```

5. 分析输入串

```
private void analysisInPutString(String s);
```

5. 核心算法描述

本此实验的核心算法有3个，分别是`构造LR(1)项集族`，`构造action表`(构造goto表类似)，`分析输入串`。下面就这三个核心算法进行描述

5.1 构造LR1项集族

```
private void createLR1(){                                    //生成LR1项集
   getI0();                                                 //求出第一个项集I0
   for(int j=0;j<State.size();j++){
      Vector vtemp   =(Vector)State.get(j);                  //取得上一个状态集合
      String s1      =getAfterPoint(vtemp);                 //取得上一个状态集合中的'.'后的所有符号
      for(int m=0;m<s1.length();m++){
         Vector v    =new Vector();                         //中间变量
         char c1=s1.charAt(m);                              //从s1中取一个字符
         for(int k=0;k<vtemp.size();k++){
            String s2  =vtemp.get(k).toString();
            if(!pointIsTheLastOne(s1)&&
                  s2.charAt(s2.indexOf('.')+1)==c1){          //此项不是规约项，可以尝试加入
               String s3=movePoint(s2);                     //中间变量
               if(!isInState(s3)&&!isInCurrentState(s3,v)){
                  v.addElement(s3);
                  for(int g=0;g<v.size();g++){
                     s3=v.get(g).toString();
                     if(!pointIsTheLastOne(s3)){            //此部分计算与求State0中的新增项集相同
                        char c2=s3.charAt(s3.indexOf('.')+1);
                        if(isInVN(c2)){                     //'.'后面的符号是非终结符吗?
                           for(int n=0;n<F_index;n++){
                              if(c2==F[n].charAt(0)){
                                 String s4=addPoint(F[n],s3);
                                 if(!isInState(s3)&&!isInCurrentState(s4,v))
                                    v.addElement(s4);
                              }
                           }
                        }
                     }
                  }
               }
            }
         }
         if(v.size()>0)             //只有v里面加入了项才把整个项集加入到项集数组
            State.add(arrange(v));
      }
   }
```

5.2 构造action表

```
Action=new int[State.size()][VT.length];//行数:state的长度 列数:终结符的长度
for(int i=0;i<State.size();i++){
   for(int j=0;j<VT.length;j++){
      Action[i][j]=getAction(i,j);
   }
}

private int getAction(int i,int j){
        Vector v=(Vector)State.get(i);                    //第i个项集合
        char  c=VT[j];                                     //第j个终结符，此时VT已添加了句子括
号'#'，所以c可以等于'#'
        //以下部分是移进动作
        if(isInAfterPoint(c,v)){                           //如果此终结符是这个项集的'.'后面的符
号，则检查其移进后的状态
            for(int k=0;k<v.size();k++){                   //查找c在这个项集的位置
                String s=v.get(k).toString();
                if((!pointIsTheLastOne(s))&&c==s.charAt(s.indexOf('.')+1)){      //如果'.'不是最
后一个符号，移进
                    for(int m=0;m<State.size();m++){
                        Vector vtemp=(Vector)State.get(m);
                        for(int n=0;n<vtemp.size();n++){
                            if(movePoint(s).equals(vtemp.get(n).toString())){
                                bridge1[i].append(c);
                                bridge2[i][bridge1[i].length()-1]=m;
                                return m;                  //进入m项集,即当前状态经c字符后变成下一
个m状态
                            }
                        }
                    }
                }
            }
        }
        for(int p=0;p<v.size();p++){
            if(pointIsTheLastOne(v.get(p).toString())){
                String s1=delete_a_char('.',v.get(p).toString());    //去掉'.'
                String s2=s1.substring(0, s1.indexOf(','));
                for(int q=0;q<F_index;q++){
                    if(s2.equals(F[q])){
                        if(s1.substring(s1.indexOf(',')).contains(String.valueOf(c)))//规约项逗
号后面是否含有c
                            return -q;
                    }
                }
            }
        }
        return ERROR;                          //出错
    }
```

5.3 基于LR(1)分析表的语法分析

```
private void analysisInPutString(String s){
        s=s.concat(String.valueOf('#'));                    //添加句子括号'#'，以便分析
        int s_index=0;                                       //次索引指向被分析符号串的头
        int i=0;
        int j=0;
        int k=0;                                             //中间变量
        int step=0;
        staStack[0]=0;
        symStack[0]='#';                                     //初始化状态栈与符号栈
        while(true){
            step++;                                          //步骤数加1
            i=staStack[staStack_index];                     //栈顶状态
            j=toColumn(s.charAt(s_index));                  //被分析符号头字符在vt数组中的位置
            if(isInVT(s.charAt(s_index))&&j!=ERROR){        //如果是非终结符
                if(Action[i][j]>0){                          //如果是移进
                    k=Action[i][j];
                    if(k==ERROR){
                        System.out.println( "分析失败！\n"+"出错原因：在第"+step+
                                "步分析到第"+i+"个状态时，"+"当前面临的符号
是"+s.charAt(s_index)+
                                "，\n而在分析表的Action表中("+i+","+s.charAt(s_index)+")这个位
置"+
                                "是出错标志！");
                        break;
                    }
                    staStack[++staStack_index]=k;
                    symStack[++symStack_index]=s.charAt(s_index++);
                    System.out.print("移入\t");
                }
                else if(Action[i][j]<0){                     //如果是归约
                    k=-Action[i][j];                         //按照第k个产生式归约
                    staStack_index  -=F[k].length()-1;      //栈指针减去第k个产生式的右部的长度
                    symStack_index  -=F[k].length()-1;
                    symStack[symStack_index++]=F[k].charAt(0);//把句柄归约
                    int temp=Goto[staStack[staStack_index]][toColumn(F[k].charAt(0))];//取得
Goto[栈顶状态][刚规约完的VN]
                    if(temp==ERROR){
                        System.out.println( "分析失败！\n"+"出错原因：在第"+step+
                                "步分析到第"+staStack[staStack_index]+"个状态时，"+"当前面临的符
号是"+
                                F[k].charAt(0)+"，\n而在分析表的Goto表中
("+staStack[staStack_index]+","+
                                F[k].charAt(0)+")这个位置是出错标志！");
                        break;
                    }
                    staStack[++staStack_index]=temp;
                    System.out.print("归约\t");
                }else{
                    System.out.println( "分析成功！此符号串是这个文法的句子！");
                    break;
                }
                //输出状态

                System.out.print("状态表：");
```

```
                for (int l = 0; l < staStack_index; l++) {
                    System.out.print(staStack[l]);
                }
                System.out.print("\t\t符号表: ");
                for (int l = 0; l < symStack_index; l++) {
                    System.out.print(symStack[l]);
                }
                System.out.println();
            }else{
                System.out.println( "分析失败！\n"+"出错原因：在第"+step+
                        "步当前面临符号"+s.charAt(s_index)+"不属于该文法的终结符！");
                break;
            }
        }
    }
```

6 测试用例

6.1成功用例

输入**1:**上下文无关文法

```
F[0]= "LE";// 表示L->E
F[1]= "EE+T";
F[2]= "ET";
F[3]= "TT*F";
F[4]= "TF";
F[5]= "F(E)";
F[6]= "Fi";
```

输入**2:**待分析字符串

```
String s="i*(i+i)";
```

输出**:**

产生式：

```
(0)S'==>L
(1)L==>E
(2)E==>E+T
(3)E==>T
(4)T==>T*F
(5)T==>F
(6)F==>(E)
(7)F==>i
```

LR1项目集：

```
I0      S'==>.L,#
        L==>.E,#
        E==>.E+T,#+
        E==>.T,#+
        T==>.T*F,#+*
        T==>.F,#+*
        F==>.(E),#+*
        F->.i,#+*
I1      S'->L.,#
I2      L==>E.,#
        E->E.+T,#+
I3      E==>T.,#+
        T->T.*F,#+*
I4      T->F.,#+*
I5      F==>(.E),#+*
        E==>.E+T,)+
        E==>.T,)+
        T==>.T*F,)+*
        T==>.F,)+*
        F==>.(E),)+*
        F->.i,)+*
I6      F->i.,#+*
I7      E==>E+.T,#+
        T==>.T*F,#+*
        T==>.F,#+*
        F==>.(E),#+*
        F->.i,#+*
I8      T==>T*.F,#+*
        F==>.(E),#+*
        F->.i,#+*
I9      F==>(E.),#+*
        E->E.+T,)+
I10     E==>T.,)+
        T->T.*F,)+*
I11     T->F.,)+*
I12     F==>(.E),)+*
        E==>.E+T,)+
        E==>.T,)+
        T==>.T*F,)+*
        T==>.F,)+*
        F==>.(E),)+*
        F->.i,)+*
```

```
I13     F->i.,)+*
I14     E->E+T.,#+
I15     T->T*F.,#+*
I16     F->(E).,#+*
I17     E==>E+.T,)+
        T==>.T*F,)+*
        T==>.F,)+*
        F==>.(E),)+*
        F->.i,)+*
I18     T==>T*.F,)+*
        F==>.(E),)+*
        F->.i,)+*
I19     F->(E.),)+*
I20     E->E+T.,)+
I21     T->T*F.,)+*
I22     F->(E).,)+*
```

LR1分析表：

状 态	+	*	(	)	i	#	L	E	T	F
0	x	x	S5	x	S6	x	1	2	3	4
1	x	x	x	x	x	acc	x	x	x	x
2	S7	x	x	x	x	r1	x	x	x	x
3	r3	S8	x	x	x	r3	x	x	x	x
4	r5	r5	x	x	x	r5	x	x	x	x
5	x	x	S12	x	S13	x	x	9	10	11
6	r7	r7	x	x	x	r7	x	x	x	x
7	x	x	S5	x	S6	x	x	x	14	4
8	x	x	S5	x	S6	x	x	x	x	15
9	S17	x	x	S16	x	x	x	x	x	x
10	r3	S18	x	r3	x	x	x	x	x	x
11	r5	r5	x	r5	x	x	x	x	x	x
12	x	x	S12	x	S13	x	x	19	10	11
13	r7	r7	x	r7	x	x	x	x	x	x
14	r2	x	x	x	x	r2	x	x	x	x
15	r4	r4	x	x	x	r4	x	x	x	x
16	r6	r6	x	x	x	r6	x	x	x	x
17	x	x	S12	x	S13	x	x	x	20	11
18	x	x	S12	x	S13	x	x	x	x	21
19	x	x	x	S22	x	x	x	x	x	x
20	r2	x	x	r2	x	x	x	x	x	x
21	r4	r4	x	r4	x	x	x	x	x	x
22	r6	r6	x	r6	x	x	x	x	x	x

```
移入    状态表：0               符号表：#
归约    状态表：0               符号表：F
归约    状态表：0               符号表：T
移入    状态表：03              符号表：Ti
移入    状态表：038             符号表：Ti*
移入    状态表：0385            符号表：Ti*(
归约    状态表：0385            符号表：Ti*F
归约    状态表：0385            符号表：Ti*T
归约    状态表：0385            符号表：Ti*E
移入    状态表：03859           符号表：Ti*Ei
```

```
移入     状态表：0385917          符号表：Ti*Ei+
归约     状态表：0385917          符号表：Ti*EiF
归约     状态表：0385917          符号表：Ti*EiT
归约     状态表：0385             符号表：Ti*E
移入     状态表：03859            符号表：Ti*Ei
归约     状态表：038              符号表：TiF
归约     状态表：0                符号表：T
归约     状态表：0                符号表：E
归约     状态表：0                符号表：L
分析成功！此符号串是这个文法的句子！
```

6.3 失败用例

输入**1:**上下文无关文法

```
F[0]= "LE";// 表示L->E
F[1]= "EE+T";
F[2]= "ET";
F[3]= "TT*F";
F[4]= "TF";
F[5]= "F(E)";
F[6]= "Fi";
```

输入**2:**待分析字符串

```
String s = "i(i+i)"
```

输出:

产生式：

```
(0)S'==>L
(1)L==>E
(2)E==>E+T
(3)E==>T
(4)T==>T*F
(5)T==>F
(6)F==>(E)
(7)F==>i
```

LR1项集：

```
I0  S'==>.L,#
    L==>.E,#
    E==>.E+T,#+
    E==>.T,#+
    T==>.T*F,#+*
    T==>.F,#+*
    F==>.(E),#+*
    F->.i,#+*
I1  S'->L.,#
I2  L==>E.,#
    E->E.+T,#+
I3  E==>T.,#+
    T->T.*F,#+*
I4  T->F.,#+*
I5  F==>(.E),#+*
    E==>.E+T,)+
    E==>.T,)+
    T==>.T*F,)+*
    T==>.F,)+*
    F==>.(E),)+*
    F->.i,)+*
I6  F->i.,#+*
I7  E==>E+.T,#+
    T==>.T*F,#+*
    T==>.F,#+*
    F==>.(E),#+*
    F->.i,#+*
I8  T==>T*.F,#+*
    F==>.(E),#+*
    F->.i,#+*
I9  F==>(E.),#+*
    E->E.+T,)+
I10 E==>T.,)+
    T->T.*F,)+*
I11 T->F.,)+*
I12 F==>(.E),)+*
    E==>.E+T,)+
    E==>.T,)+
    T==>.T*F,)+*
    T==>.F,)+*
    F==>.(E),)+*
    F->.i,)+*
```

```
I13 F->i.,)+*
I14 E->E+T.,#+
I15 T->T*F.,#+*
I16 F->(E).,#+*
I17 E==>E+.T,)+
    T==>.T*F,)+*
    T==>.F,)+*
    F==>.(E),)+*
    F->.i,)+*
I18 T==>T*.F,)+*
    F==>.(E),)+*
    F->.i,)+*
I19 F->(E.),)+*
I20 E->E+T.,)+
I21 T->T*F.,)+*
I22 F->(E).,)+*

LR1分析表:
状 态 +   *   (   )   i   #   L   E   T   F
0   x   x   S5  x   S6  x   1   2   3   4
1   x   x   x   x   x   acc x   x   x   x
2   S7  x   x   x   x   r1  x   x   x   x
3   r3  S8  x   x   x   r3  x   x   x   x
4   r5  r5  x   x   x   r5  x   x   x   x
5   x   x   S12 x   S13 x   x   9   10  11
6   r7  r7  x   x   x   r7  x   x   x   x
7   x   x   S5  x   S6  x   x   x   14  4
8   x   x   S5  x   S6  x   x   x   x   15
9   S17 x   x   S16 x   x   x   x   x   x
10  r3  S18 x   r3  x   x   x   x   x   x
11  r5  r5  x   r5  x   x   x   x   x   x
12  x   x   S12 x   S13 x   x   19  10  11
13  r7  r7  x   r7  x   x   x   x   x   x
14  r2  x   x   x   x   r2  x   x   x   x
15  r4  r4  x   x   x   r4  x   x   x   x
16  r6  r6  x   x   x   r6  x   x   x   x
17  x   x   S12 x   S13 x   x   x   20  11
18  x   x   S12 x   S13 x   x   x   x   21
19  x   x   x   S22 x   x   x   x   x   x
20  r2  x   x   r2  x   x   x   x   x   x
21  r4  r4  x   r4  x   x   x   x   x   x
22  r6  r6  x   r6  x   x   x   x   x   x

移入   状态表：0        符号表：#
分析失败!
出错原因:在第2步分析到第6个状态时，当前面临的符号是(，\
而在分析表的Action表中(6,()这个位置是出错标志!
```

遇到的困难

最大的困难就是数据结构了，算法都不难，都是书上提供的。但是究竟怎么把"项集族"这个概念转化成代码是最难的一步，最后还是选定了vector的vector这种比较蹩脚的办法，但是也解决了问题。

不足与感受

不足:生成LR(1)分析表的时候没怎么考虑出错的情况。也就是说，如果一个位置出现了rs冲突或者rr冲突，第一个的时候就已经返回了，可能存在的第二个就没考虑。

感受:感觉……比第一次简单一点，可能是把作业做了才做实验的缘故吧，还是温习了一遍LR(1)语法分析过程，总的来讲还不错。