编译原理课程实验报告

实验 2：语法分析

姓名	席凯迪	院系	软件学院	学号	161110507
任课教师	韩希先		指导教师	韩希先	
实验地点	研究院中 507		实验时间	2018.10.27	
实验课表现	出勤、表现得分		实验报告得分		实验总分	
	操作结果得分					

一、实验目的

1. 巩固对语法分析的基本功能和原理的认识
2. 通过对语法分析表的自动生成加深语法分析表的认识
3. 理解并处理语法分析中的异常和错误

通过老师上课的讲解和对语法分析知识的理解，对语法分析程序进行实现。在编程构造的语法分析表的基础之上实现 SLR(1)分析，并针对一类语句的文法给出其语法分析表的生成程序。

二、实验内容

1. 本次实验中，规定以下文法(参照课本 P180)：

S->E

E->E+T

E->T

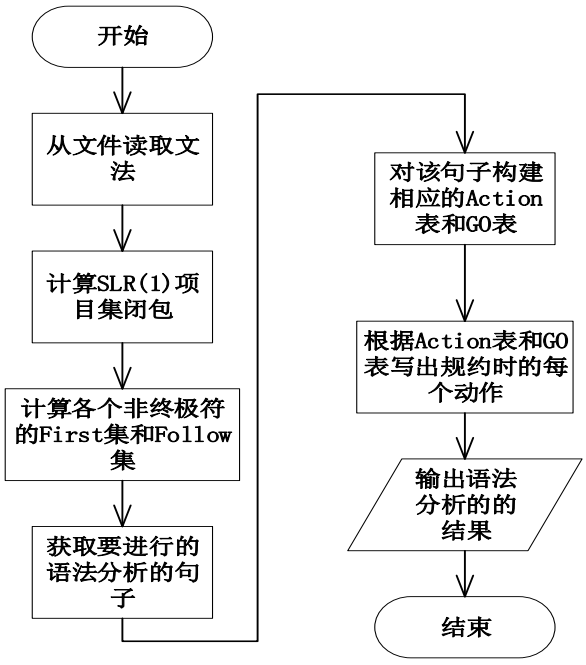
T->T*F

T->F

F->(E)

F->d

2.语法分析程序的总体结构及物理实现



计算项目集闭包：

1. 计算 I(i)：

```
public void CLOSURE(int state) {       //项目集存在closure.get(state)中，求I0用。
        int i,j,k;
        String item;
        for(i=0;i<closure.get(state).getItems().size();i++) {       //项目集中的每个项[A->A.Bb]
                j=3;
                while(closure.get(state).getItems().get(i).charAt(j)!='.'&&j<closure.get(state).ge
                    j++;
                }//找到.的位置
                j++;
                item = closure.get(state).getItems().get(i);
                if(j<item.length()&&isTerminal(item.charAt(j))==false) {        //如果是非终结符
                    for(k=0;k<wenfa.size();k++) {   //遍历文法
                        if(wenfa.get(k).charAt(0)==item.charAt(j)) {        //如果文法左部变量中有与 刚刚的
                                if(IsExist(GetItem(wenfa.get(k)),closure.get(state))==false) {
                                    closure.get(state).getItems().add(GetItem(wenfa.get(k)));   //
                                }
                        }
                    }
                }
            }
        }
```

```
public void CLOSURE(Closure C) {  //求除I0外的其他项目集闭包。
    int i,j,k;
    String item;
    for(i=0;i<C.getItems().size();i++) {
            j=3;
            while(C.getItems().get(i).charAt(j)!='.'&&j<C.getItems().get(i).length()) {
                j++;
            }  //找到.的位置
            j++;
            item =C.getItems().get(i);
            if(j<item.length()&&isTerminal(item.charAt(j))==false) {        //如果是非终结符
                for(k=0;k<wenfa.size();k++) {   //遍历文法
                    if(wenfa.get(k).charAt(0)==item.charAt(j)) {        //如果文法左部变量中有与 刚刚的非终结符
                            if(IsExist(GetItem(wenfa.get(k)),C)==false) {       //如果 循环 M[I][]发现M[I]
                                C.getItems().add(GetItem(wenfa.get(k)));    //将该项存入项目集中
                            }

                    }
                }
            }
        }
    }
```

```
public Closure Go(Closure C,char x) { //GO函数求除I0外所有项目集闭包
    int i,j;
    Closure clo = new Closure();
    clo.setId(closure.size());
    for(i=0;i<C.getItems().size();i++) {  //C中的每个项[A->A.Bb]
        j=3;
        while(C.getItems().get(i).charAt(j)!='.'&&j<C.getItems().get(i).length()) {      //循环，找到".
            j++;
        }
```

```
        j++;
        if(j<C.getItems().get(i).length()&&C.getItems().get(i).charAt(j)==x) {
            clo.getItems().add(getJ(C.getItems().get(i),j-1));//把原来项目集中的后继添加到该项目集中
        }
    }
    CLOSURE(clo);
    return clo;
}
```

2. 计算 Closure 的控制结构

```
public void setClosure() {
    Closure clo = new Closure();
    clo.getItems().add(items.get(0));
    clo.setId(0);
    closure.add(clo);
    CLOSURE(0);

    String element ="";//保存状态Ix可接收的字符。
    for(int i=0;i<closure.size();i++) {
        element =  getWords(closure.get(i).getItems());
        if(element.length()!=0) {
            for(int j=0;j<element.length();j++) {
                clo = Go(closure.get(i),element.charAt(j));
                if(ifRepeat(clo)==-1) {
                    closure.add(clo);
                }

            }
        }
    }
}
```

计算 FIRST 集和 FOLLOW 集

```
public void First() {//求解first集
    int index,after,before;
    after = 0;
    before = 0;

    first = new String[nonterminal.length()];
    for(int j=0;j<nonterminal.length();j++) {
        first[j] = ""+nonterminal.charAt(j);
    }
    do{
        //System.out.println("asd");
        before=after;
        for(int i=0;i<nonterminal.length();i++) {
            for(int j=0;j<wenfa.size();j++) {
                if(first[i].charAt(0)==wenfa.get(j).charAt(0)) {
                    if(isTerminal(wenfa.get(j).charAt(3))) { //产生式第一个是终结符
                        first[i]=removeSameString(first[i]+""+wenfa.get(j).charAt(3));
                    }
                    else{//产生式第一个不是终结符，E,T等
                        index =getNonterminal(wenfa.get(j).charAt(3));
                        if(first[index].length()>1) {
                            for(int k=1;k<first[index].length();k++) {
                                first[i]=removeSameString(first[i]+""+first[index].charAt(k));
                            }
                        }
                    }
                }
            }
        }
        after =0;
        for(int a=0;a<nonterminal.length();a++) {
            after += first[a].length();
        }
```

```
            after += first[a].length();
        }
    }while(after!=before);
}
public int nonterminalPositon(String wf,char x) {//非终结符是否在文法的候选式中，是则返回非终结符的位置，不是则返回-1.
public void Follow() {//计算follow集
    int index,before,after,position;
    before = 0;
    after = 0;
    follow = new String[nonterminal.length()];
    for(int j=0;j<nonterminal.length();j++) {
        follow[j] = ""+nonterminal.charAt(j);
    }
    follow[0]=follow[0]+'#';
    do {
        before = after;
        for(int i=0;i<nonterminal.length();i++) {
            for(int j=0;j<wenfa.size();j++) {
                position=nonterminalPositon(wenfa.get(j),nonterminal.charAt(i));
                if(position!=-1) {
                    if(position!=wenfa.get(j).length()-1) { //该语法变量不在产生式末尾
                        if(isTerminal(wenfa.get(j).charAt(position+1))){  //该语法变量后是终结符
                            follow[i]=removeSameString(follow[i]+""+wenfa.get(j).charAt(position+1));
                        }
                        else {//该语法变量后不是终结符，E,T等
                            index =getNonterminal(wenfa.get(j).charAt(position+1));
                            if(follow[index].length()>1) { //
                                for(int k=1;k<first[index].length();k++) {
                                    follow[i]=removeSameString(follow[i]+""+first[index].charAt(k));
                                }


                                    follow[i]=removeSameString(follow[i]+""+first[index].charAt(k));
                                }
                            }
                        }
                    }
                    else {//该语法变量在产生式末尾
                        index =getNonterminal(wenfa.get(j).charAt(0));
                        if(follow[index].length()>1) {

                            for(int k=1;k<follow[index].length();k++) {
                                follow[i]=removeSameString(follow[i]+""+follow[index].charAt(k));

                            }
                        }
                    }
                }
            }
        }
        after =0;
        for(int a=0;a<nonterminal.length();a++) {
            after += follow[a].length();

        }
    }while(after!=before);
}
```

3.Closure 的结构：

```
public class Closure {
 private int id;
 private ArrayList<String> items = new ArrayList<String>();
public int getId() {
    return id;
}
public void setId(int id) {
    this.id = id;
}
public ArrayList<String> getItems() {
    return items;
}
public void setItems(String item) {
    items.add(item);
}

}
```

4.构建 ACTION 表和 GOTO 表（语法分析表）的核心代码：

```
    public void setActionandGo(){
        char element,r,s;
        r='r';
        s='s';
        int c,t,n,index,position;
        c=closure.size();
        t=total.length();
        n=nonterminal.length();
        String acc = closure.get(1).getItems().get(0);
        String item;
        SLR = new String[c][t];
        ACTION = new String[c][t-n];
        GOTO = new int[c][n];
        for(int i=0;i<c;i++) {
            for(int j=0;j<t;j++) {
                SLR[i][j]="";
            }
        }
        for(int i=0;i<c;i++) {
            for(int j=0;j<t-n;j++) {
                ACTION[i][j]="error";
            }
        }
        for(int i=0;i<c;i++) {
            for(int j=0;j<n;j++) {
                GOTO[i][j]=-1;
            }
        }
```

```
        SLR[1][terminal.length()-1] ="acc";
        ACTION[1][terminal.length()-1] ="acc";
        for(int i=0;i<closure.size();i++) {
            for(int j=0;j<closure.get(i).getItems().size();j++) {
                item = closure.get(i).getItems().get(j);
                element=getWord(item);
                index = getElementPositon(element);//得到字符在所有字符中的位置
                if(element=='$'&&!(item.equals(acc))) {  //得到r*
                        for(int k=0;k<terminal.length();k++) {
                            position = findWenfa(item);
                            if(k<terminal.length()&&isInFollow(position,terminal.charAt(k))) {
                                SLR[i][k]=""+r+position;
                                ACTION[i][k]=""+r+position;
                            }
                        }
                }
                else if(isTerminal(element)&&element!='$') {
                    position = ifRepeat(Go(closure.get(i),element));
                    SLR[i][index] = ""+s+position;
                    ACTION[i][index]=""+s+position;
                }
                else if(!isTerminal(element)) {
                    position = ifRepeat(Go(closure.get(i),element));
                    SLR[i][index] = ""+position;
                    index = index-terminal.length();
                    GOTO[i][index] =position;
                }
            }
        }
    }
```

5. 语法分析时可能的错误有当扫描下一个输入时出现一个不正确的后继，此时的处理即不会把出错点的输入符号移进入栈，当语法分析程序发现错误时将会从栈顶开始退栈，直到发现在特定语法变量 A 上具有转移的状态 S 为止，然后丢弃 0 个或多个输入符号，直到找到符号 a∈FOLLOW（(A)为止。接着分析程序把状态 GOTO(S,A)压入栈，并恢复正常分析。

三、实验结果

项目集闭包：

```
closure:0       S->.E
closure:0       E->.E+T
closure:0       E->.T
closure:0       T->.T*F
closure:0       T->.F
closure:0       F->.(E)
closure:0       F->.d
------------------------------
closure:1       S->E.
closure:1       E->E.+T
------------------------------
closure:2       E->T.
closure:2       T->T.*F
------------------------------
closure:3       T->F.
------------------------------
closure:4       F->(.E)
closure:4       E->.E+T
closure:4       E->.T
closure:4       T->.T*F
closure:4       T->.F
closure:4       F->.(E)
closure:4       F->.d
------------------------------
closure:5       F->d.
------------------------------
--------------------------------
closure:6       E->E+.T
closure:6       T->.T*F
closure:6       T->.F
closure:6       F->.(E)
closure:6       F->.d
--------------------------------
closure:7       T->T*.F
closure:7       F->.(E)
closure:7       F->.d
--------------------------------
closure:8       F->(E.)
closure:8       E->E.+T
--------------------------------
closure:9       E->E+T.
closure:9       T->T.*F
--------------------------------
closure:10      T->T*F.
--------------------------------
closure:11      F->(E).
--------------------------------
```

语法分析表：

SLR(1)分析表:

	+	*	(	)	d	#	S	E	T	F
0			s4		s5			a1	a2	a3
1	s6					acc				
2	r2	s7		r2		r2				
3	r4	r4		r4		r4				
4			s4		s5			a8	a2	a3
5	r6	r6		r6		r6				
6			s4		s5				a9	a3
7			s4		s5					a10
8	s6			s11						
9	r1	s7		r1		r1				
10	r3	r3		r3		r3				
11	r5	r5		r5		r5				

对应语句的语法分析结果：

状态栈	符号栈	剩余字符串	动作
[0]	#	d*(d+d)+(d+d)+d*d#	移入
[0, 5]	#d	*(d+d)+(d+d)+d*d#	根据F->d归约
[0, 3]	#F	*(d+d)+(d+d)+d*d#	根据T->F归约
[0, 2]	#T	*(d+d)+(d+d)+d*d#	移入
[0, 2, 7]	#T*	(d+d)+(d+d)+d*d#	移入
[0, 2, 7, 4]	#T*(	d+d)+(d+d)+d*d#	移入
[0, 2, 7, 4, 5]	#T*(d	+d)+(d+d)+d*d#	根据F->d归约
[0, 2, 7, 4, 3]	#T*(F	+d)+(d+d)+d*d#	根据T->F归约
[0, 2, 7, 4, 2]	#T*(T	+d)+(d+d)+d*d#	根据E->T归约
[0, 2, 7, 4, 8]	#T*(E	+d)+(d+d)+d*d#	移入
[0, 2, 7, 4, 8, 6]	#T*(E+	d)+(d+d)+d*d#	移入
[0, 2, 7, 4, 8, 6, 5]	#T*(E+d	)+(d+d)+d*d#	根据F->d归约
[0, 2, 7, 4, 8, 6, 3]	#T*(E+F	)+(d+d)+d*d#	根据T->F归约
[0, 2, 7, 4, 8, 6, 9]	#T*(E+T	)+(d+d)+d*d#	根据E->E+T归约
[0, 2, 7, 4, 8]	#T*(E	)+(d+d)+d*d#	移入
[0, 2, 7, 4, 8, 11]	#T*(E)	+(d+d)+d*d#	根据F->(E)归约
[0, 2, 7, 10]	#T*F	+(d+d)+d*d#	根据T->T*F归约
[0, 2]	#T	+(d+d)+d*d#	根据E->T归约
[0, 1]	#E	+(d+d)+d*d#	移入
[0, 1, 6]	#E+	(d+d)+d*d#	移入
[0, 1, 6, 4]	#E+(	d+d)+d*d#	移入
[0, 1, 6, 4, 5]	#E+(d	+d)+d*d#	根据F->d归约
[0, 1, 6, 4, 3]	#E+(F	+d)+d*d#	根据T->F归约
[0, 1, 6, 4, 2]	#E+(T	+d)+d*d#	根据E->T归约
[0, 1, 6, 4, 8]	#E+(E	+d)+d*d#	移入
[0, 1, 6, 4, 8, 6]	#E+(E+	d)+d*d#	移入
[0, 1, 6, 4, 8, 6, 5]	#E+(E+d	)+d*d#	根据F->d归约
[0, 1, 6, 4, 8, 6, 3]	#E+(E+F	)+d*d#	根据T->F归约
[0, 1, 6, 4, 8, 6, 9]	#E+(E+T	)+d*d#	根据E->E+T归约
[0, 1, 6, 4, 8]	#E+(E	)+d*d#	移入
[0, 1, 6, 4, 8, 11]	#E+(E)	+d*d#	根据F->(E)归约
[0, 1, 6, 3]	#E+F	+d*d#	根据T->F归约
[0, 1, 6, 9]	#E+T	+d*d#	根据E->E+T归约
[0, 1]	#E	+d*d#	移入
[0, 1, 6]	#E+	d*d#	移入
[0, 1, 6, 5]	#E+d	*d#	根据F->d归约
[0, 1, 6, 3]	#E+F	*d#	根据T->F归约
[0, 1, 6, 9]	#E+T	*d#	移入
[0, 1, 6, 9, 7]	#E+T*	d#	移入
[0, 1, 6, 9, 7, 5]	#E+T*d	#	根据F->d归约
[0, 1, 6, 9, 7, 10]	#E+T*F	#	根据T->T*F归约
[0, 1, 6, 9]	#E+T	#	根据E->E+T归约
[0, 1]	#E	#	接受

四、实验中遇到的问题总结

一、遇到的问题：

1.如何计算 Closure

根据课本上 Closure 的生成算法，首先生成文法的 I(0)闭包，然后根据读入的下一个字符和 I(0)闭包来计算以后的项目集闭包。

2.文法，项目集闭包等的存储结构的选择

由于文件中的文法条目未知，如果选用确定空间大小的数组来存储，有可能存在空间不足或者空间浪费的情况，因此 Arraylist 成了更好的选择，首先创建一个对象，让后用 add 方法直接往该对象中添加即可。

二、思考题

思考题 1.如何进行状态转移

解决办法：根据 GO 函数（状态转移函数）的返回值（一个项目集闭包）和判断重复的函数 ifRepreat 函数（返回值类型为 int）来进行状态转移。其中在 ifRepreat 函数中返回值表示的是如果重复则返回重复的项目集的下标，否则返回-1。

思考题 2. 思考还可以什么形式来给出语法分析的结果？

还可以通过自顶向下的分析方法来给出语法分析的结果。即从初始符号向句子推导，如果能够得到该句子，则表明该句子是由该文法生成的，否则，此句子不是由该文发生成的。

思考题 3. 如果在语法分析中遇到了语法错误，是应该中断语法分析呢，还是应该进行适当处理后继续语法分析，你是怎么处理的？

先进行适当的处理再继续进行语法分析。当出现输入符号错误时，不会把出错点的输入符号移进入栈，当语法分析程序发现错误时将会从栈顶开始退栈，直到发现在特定语法变量 A 上具有转移的状态 S 为止，然后丢弃 0 个或多个输入符号，直到找到符号 a∈FOLLOW（(A)为止。接着分析程序把状态 GOTO(S,A)压入栈，并恢复正常分析。

五、实验体会

通过此次实验，我对自底向上语法分析有了更深的了解，对求文法的 FIRST 集和 FOLLOW 集以及项目集闭包有了更深的认识。另外，提升了自己对算法的实现能力。

指导教师评语：

日期：