# 天津理工大学 实验报告

**学院名称：**计算机科学与工程学院

| 姓名 | 王帆 | 学号 | 20152180 | 专业 | 计算机科学与技术 |
|---|---|---|---|---|---|
| 班级 | 2015 级 1 班 | 实验项目 | 实验三：语义分析与中间代码生成 | | |
| 课程名称 | | 编译原理 | | 课程代码 | 0668056 |
| 实验时间 | | 2018 年 6 月 6 日 第 1、2 节<br>2018 年 6 月 11 日 第 5、6 节 | | 实验地点 | 软件实验室 7-219<br>软件实验室 7-219 |

**实验成绩考核评定分析**

| 实验过程综合评价 30 分 | 实验目标结果评价 20 分 | 程序设计规范性评价 20 分 | 实验报告完整性评价 30 分 | 实验报告雷同分析分类标注 | 实验成绩 |
|---|---|---|---|---|---|
| ■实验过程认真专注，能独立完成设计与调试任务 30 分<br>■实验过程认真，能较好完成设计与编成调试任务 25 分<br>■实验过程较认真，能完成设计与编成调试任务 20 分<br>■实验过程态度较好，基本完成设计与编成调试任务 15 分<br>■实验过程态度欠端正，未完成设计与编成调试任务 10 分 | ■ 功能完善，且人机交互界面友好 20 分<br>■ 满足功能要求，但人机交互界面一般 15 分<br>■基本满足功能需求，人机交互界面欠缺 10 分<br>■功能缺失 5 分 | ■程序易读性好 20 分<br>■程序易读性较好 15 分<br>■程序易读性欠缺 10 分<br>■程序易读性较差 5 分<br><br>**注：易读性要求标识符命名见名知意，程序编制采用嵌套方式，层次结构清晰可读，关键部分具有简明注释。 | ■报告完整 30 分<br>■报告较完整 25 分<br>■报告内容一般 20 分<br>■报告内容极少 10 分 | 凡雷同报告将不再重复评价前四项考核内容，实验成绩将按低学号雷同学生成绩除雷同人数计算而定。<br>标记为：<br>**S 组号-人数(组分)** | 前四项评价分数之总和<br><br>(** 雷同报告按第五项标准核算**) |
| | | | | | |

## 实验内容：

已知 G[E]： E→E+T | E-T | T

T→T*F | T/F | F

F→P^F | P

P→(E) | i

要求构造出符合语义分析要求的属性文法描述，并在完成实验二（语法分析）的基础上，进行语义分析程序设计，最终输出与测试用例等价的四元式中间代码序列。

## 实验目的：

1. 掌握语法制导翻译的基本功能，巩固对语义分析的基本功能和原理的认识；
3. 能够基于语法制导翻译的知识进行语义分析，掌握文法规则相应语义动作的设计方法；
5. 理解并处理语义分析中的异常和错误。

## 实验要求：

1. 在实验二的基础上，实现语法制导翻译功能，输出翻译后所得四元式序列；
2. 要求详细描述所选分析方法进行制导翻译的设计过程；
3. 完成对所设计分析器的功能测试，并给出测试数据和实验结果；
4. 为增加程序可读性，请在程序中进行适当注释说明；
5. 整理上机步骤，总结经验和体会，认真完成并按时提交实验报告。

****** 实验报告内容详述 ******

# 一、 基本原理

## 语法制导翻译

语法制导翻译简称 SDT(Syntax-directed translation)。基于属性文法的处理过程，对单词符号串进行语法分析，构造语法分析树，然后根据需要构造属性依赖图，遍历语法树并在语法树的各结点处按语义规则进行计算。

非终结符号可以有两种属性：

1. 综合属性：如果语法分析树上的结点 N 的某个属性 a 只能通过 N 的子结点和 N 本身的属性值来定义，那么属性 a 是结点 N 的一个综合属性；
2. 继承属性：如果语法分析树上的结点 N 的某个属性 b 只能通过 N 的父结点、N 本身和 N 的兄弟结点的属性值来定义，那么属性 a 是结点 N 的一个继承属性。

终结符号可以有综合属性，但不能有继承属性，终结符号的属性是由词法分析器提供的词法值。

## LR 分析法

LR 分析法是一种自顶向下的上下文无关语法分析器。LR 意指由左至右处理输入字符串，并以最右边优先派生的推导顺序（相对于 LL）建构语法树。能以此方式分析的语法称为 LR 语法。而在 LR(k)这样的名称中，k 代表的是分析时所需前瞻符号的数量，也就是除了目前处理到的输入符号之外，还得再向右引用几个符号之意

由于 LR 分析器尝试由分析树的叶节点开始，向上一层层通过文法规则的化简，最后规约回到树的根部（起始符号），所以它是一种由下而上的分析方法。许多编程语言使用 LR(1)描述文法，因此许多编译器都使用 LR 分析器分析源代码的文法结构。

LR 分析的优点如下：

1. 众多的编程语言都可以用某种 LR 或其变形分析法分析文法；
2. LR 分析器可以很有效率的建置；
3. 对所有“由左而右”扫描源代码的分析器而言，LR 分析器可以在最短的时间内侦测到文法错误。

# 二、 功能概述

1. 在实验二的基础上，实现对输入字符串的语法分析，判断其是否符合给定文法
2. 若符合给定文法，则对其进行语法制导翻译，形成四元式序列，提示“分析成功”
3. 若不符合给定文法，则提示“分析失败”

# 三、 设计过程

## 原始文法

1. E→E+T | E-T | T
2. T→T*F | T/F | F
3. F→P^F | P
4. P→(E) | i

## 构造拓广文法

1. E’ -> E
2. E -> E+T
3. E -> E-T
4. E -> T
5. T -> T*F
6. T -> T/F

7. T -> F
8. F -> P^F
9. F -> P
10. P -> (E)
11. P -> i

**构造项目集规范族以及 GO 函数**

按照拓广文法构造项目集规范族，其对应的 GO 函数与 FOLLOW 集，如下：

**GO 函数**

I0:Go(I0,E)=I1,Go(I0,T)=I2,Go(I0,F)=I3,Go(I0,P)=I4,Go(I0,()=I5,Go(I0,i)=I6,
I1:Go(I1,+)=I7,Go(I1,-)=I8,
I2:Go(I2,*)=I9,Go(I2,/)=I10,
I4:Go(I4,^)=I11,
I5:Go(I5,E)=I12,Go(I5,T)=I2,Go(I5,F)=I3,Go(I5,P)=I3,Go(I5,()=I4,Go(I5,i)=I6,
I7:Go(I7,T)=I13,Go(I7,F)=I3,Go(I7,P)=I4,Go(I7,()=I5,Go(I7,i)=I6,
I8:Go(I8,T)=I14,Go(I8,F)=I3,Go(I8,P)=I4,Go(I8,))=I5,Go(I8,i)=I6,
I9:Go(I9,F)=I15,Go(I9,P)=I4,Go(I9,()=I5,Go(I9,i)=I6,
I10:Go(I10,F)=I16,Go(I10,P)=I4,Go(I10,()=I5,Go(I10,i)=I6,
I11:Go(I11,F)=I17,Go(I11,P)=I4,Go(I11,()=I5,Go(I11,i)=I6,
I12:Go(I12,))=I18,Go(I12,+)=I7,Go(I12,-)=I8,
I13:Go(I13,*)=I9,Go(I13,/)=I10,
I14:Go(I14,*)=I9,Go(I14,/)=I10,

**FOLLOW 集**

FOLLOW(E')={#}
FOLLOW(E)={+,-,),#}
FOLLOW(T)={+,-,),#,*,/}
FOLLOW(F)={+,-,),#,*,/}
FOLLOW(P)={+,-,),#,*,/,^}

**构造 SLR(1)文法分析表**

| 状态 | ACTION | | | | | | | | | GOTO | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | + | - | * | / | ^ | ( | ) | i | # | E | T | F | P |
| **0** | | | | | | S5 | | S6 | | 1 | 2 | 3 | 4 |
| **1** | S7 | S8 | | | | | | | acc | | | | |
| **2** | r3 | r3 | S9 | S10 | | | r3 | | r3 | | | | |
| **3** | r6 | r6 | r6 | r6 | | | r6 | | r6 | | | | |
| **4** | r8 | r8 | r8 | r8 | S11 | | r8 | | r8 | | | | |
| **5** | | | | | | S5 | | S6 | | 12 | 2 | 3 | 4 |
| **6** | r10 | r10 | r10 | r10 | r10 | | r10 | | r10 | | | | |
| **7** | | | | | | S5 | | S6 | | | 13 | 3 | 4 |
| **8** | | | | | | S5 | | S6 | | | 14 | 3 | 4 |
| **9** | | | | | | S5 | | S6 | | | | 15 | 4 |
| **10** | | | | | | S5 | | S6 | | | | 16 | 4 |
| **11** | | | | | | S5 | | S6 | | | | 17 | 4 |

| | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **12** | S7 | S8 | | | | | S18 | | | | | | |
| **13** | r1 | r1 | S9 | S10 | | | r1 | | r1 | | | | |
| **14** | r2 | r2 | S9 | S10 | | | r2 | | r2 | | | | |
| **15** | r4 | r4 | r4 | r4 | | | r4 | | r4 | | | | |
| **16** | r5 | r5 | r5 | r5 | | | r5 | | r5 | | | | |
| **17** | r7 | r7 | r7 | r7 | | | r7 | | r7 | | | | |
| **18** | r9 | r9 | r9 | r9 | r9 | | r9 | | r9 | | | | |

构造文法的属性文法以及属性动作

| 产生式 | 语义动作 |
|---|---|
| S→=E | S.code := E.code \|\| Encode( id.place':='E.place ) |
| E→E1+E2 | E.place := newtemp;<br>E.code := E1.code \|\| E2.code \|\| Encode(E.place':='E1.place'+'E2.place) |
| E→E1-E2 | E.place := newtemp;<br>E.code := E1.code \|\| E2.code \|\| Encode(E.place':='E1.place'-'E2.place) |
| E→E1*E2 | E.place := newtemp;<br>E.code := E1.code \|\| E2.code \|\|Encode(E.place':='E1.place'*'E2.place) |
| E→E1\E2 | E.place := newtemp;<br>E.code := E1.code \|\| E2.code \|\|Encode(E.place':='E1.place'\'E2.place) |
| E→E1^E2 | E.place := newtemp;<br>E.code := E1.code \|\| E2.code \|\|Encode(E.place':='E1.place'^'E2.place) |
| E→（E1） | E.place:= E1.place;<br>E.code:= E1.code |
| E→ id | E.place:= id.place;  E.code:= '' |

## 四、 实验代码

```
using CompilerLab.COMUtil;
using System;
using System.Collections.Generic;
using System.Windows.Forms;


namespace CompilerLab.ChildForm.SemanticAnalysis
{
    public partial class SemanticAnalysis : Form
    {
```

```
public List<Inter> Intertable = new List<Inter>();//四元式表
bool con = false;//分析状态标识
int Index = 0;
int temp_num = 0;

public class Inter//四元式结构
{
    public int num;
    public int nxq;
    public string op;
    public string arg1;
    public string arg2;
    public string result;

    public Inter(int nxq, string op, string arg1, string arg2, string result)
    {
        this.nxq = nxq;
        this.op = op;
        this.arg1 = arg1;
        this.arg2 = arg2;
        this.result = result;
    }
}

public class TokenTable //token 表结构
{
    public int local;
    public string Words;
    public int token;
}

public class SymbolInfo//Symbo 表结构
{
    public string Words { get; internal set; }
    public int Token { get; internal set; }
    public string Type { get; internal set; }
}

List<TokenTable> tokenList = new List<TokenTable>();
List<SymbolInfo> symbolList = new List<SymbolInfo>();

public SemanticAnalysis()
{
    InitializeComponent();
}
```

```
        private void 消除文法左递归ToolStripMenuItem_Click(object sender, EventArgs e)
        {
            richTextBox.Text =
"E->TA\nT->FB\nF->PC\nC->^F|ε\nP->(E)|i\nA->+TA|-TA|ε\nB->/FB|*FB|ε";
        }

        private void 输入待分析字符串ToolStripMenuItem_Click(object sender, EventArgs e)
        {
            try
            {
                Dialog dialog = new Dialog();
                dialog.ShowDialog();
                if (dialog.flag)
                {
                    analysestr.Text = dialog.str;
                }
            }
            catch (Exception ex)
            {
                //错误提示
                MessageBox.Show(ex.Message, "错误提示", MessageBoxButtons.OK,
MessageBoxIcon.Stop);
            }
        }
        //总控函数
        private bool Parser()
        {
            string token = GetNextToken();
            if (token != "program")
            {
                MessageBox.Show("缺少关键字 program");
                return false;
            }
            token = GetNextToken();
            if (token == "")
            {
                MessageBox.Show("缺少程序名");
                return false;
            }
            token = GetNextToken();
            if (token != ";")
            {
                MessageBox.Show("缺少';'");
                return false;
```

```
    }
    token = GetNextToken();
    if (token == "const")
    {
        token = GetNextToken();
        //调用分析常量说明函数
    }
    if (token == "var")
    {
        token = GetNextToken();
        varst(token);
        //调用分析变量说明函数
    }
    token = GetNextToken();
    if (token == "begin")
    {
        while (Index != (tokenList.Count - 1))
        {
            token = GetNextToken();
            //分类各种可执行语句
            int s_chain = ST_SORT(token);
            if (s_chain != 0)
                BackPatch(ref s_chain, Intertable.Count + 1);
        }
    }
    token = GetNextToken();
    Gnecode("", "", "", "");
    if (token == ".")
    {
        return true;
    }
    else
    {
        MessageBox.Show("缺少'.'");
        return false;
    }
}

private void varst(string token)
{
    throw new NotImplementedException();
}

//分类语句翻译函数
private int ST_SORT(string token)
```

```
{
    int s_chain = 0;
    if (token == "if")
    {
        s_chain = ifs();
    }
    else if (token == "while")
    {
        s_chain = whiles();
    }
    else if (token == "repeat")
    {
        s_chain = repeats();
    }
    else if (token == "for")
    {
        s_chain = fors();
    }
    else if (token == "begin" || token == "end" || token == ";")
    {

    }
    else
    {
        string term = GetNextToken();
        if (term == ":=")
        {
            term = aexpr(token);
            Gnecode(":=", term, "", token);
        }
        term = GetNextToken();
        if (term == "until")
        {
            Index--;
        }
    }
    return s_chain;
}

//获取下一个 Token
private string GetNextToken()
{
    return tokenList[Index++].Words;
}
```

```
//加减运算
private string aexpr(string s_ret)
{
    string token = GetNextToken();
    string term1 = "";
    term1 = term(token, term1);
    while (true)
    {
        token = GetNextToken();
        if (token == "+" || token == "-")//当前符号为+或者-
        {
            string sym = token;
            token = GetNextToken();
            string term2 = "";
            term2 = term(token, term2);
            s_ret = NEWT();
            Gnecode(sym, term1, term2, s_ret);//编码四元式
            term1 = s_ret;
        }
        else
        {
            Index--;
            s_ret = term1;
            break;
        }
    }
    return s_ret;
}

//乘除运算
private string term(string token, string term)
{
    string fac1 = "";
    fac1 = factor(token, fac1);
    while (true)
    {
        token = GetNextToken();
        if (token == "*" || token == "/")//当前符号为*或者/
        {
            string sym = token;
            token = GetNextToken();
            string fac2 = "";
            fac2 = factor(token, fac2);
            term = NEWT();
            Gnecode(sym, fac1, fac2, term);
```

```
            fac1 = term;
        }
        else
        {
            Index--;
            term = fac1;
            break;
        }
    }
    return term;
}

//四元式整理
private string factor(string token, string fac)
{
    int sym_i;
    if (token == "(")
    {
        fac = aexpr(fac);
        token = GetNextToken();
        if (token != ")")
        {
            //缺少")"
        }
    }
    else if (token == "-") fac = aexpr(fac);
    else if (isidentifier(token) != -1 || isconst(token) != -1)
    {
        if ((sym_i = isidentifier(token)) == -1)
        {
            sym_i = isconst(token);
        }
        fac = symbolList[sym_i].Words;
    }
    else
    {
        //错误
    }
    return fac;
}

//判别标识符
private int isidentifier(string token)
{
    int i;
```

```
    for (i = 0; i < symbolList.Count; i++)
    {
        if (symbolList[i].Words == token)
            break;
    }
    if (i < symbolList.Count || symbolList[i].Token == 36)
    {
        return i;
    }
    return -1;
}

//判别常量
private int isconst(string token)
{
    int i;
    for (i = 0; i < symbolList.Count; i++)
    {
        if (symbolList[i].Words == token)
            break;
    }
    if (i < symbolList.Count || symbolList[i].Type == "const")
    {
        return i;
    }
    return -1;
}

//四元式编码
private void Gnecode(string op, string arg1, string arg2, string result)
{
    Intertable.Add(new Inter(Intertable.Count + 1, op, arg1, arg2, result));
}

private string NEWT()
{
    string s_ret = "T";
    s_ret += temp_num.ToString();
    temp_num++;
    return s_ret;
}

/// <summary>
/// if语句翻译
/// </summary>
```

```
private int ifs()
{
    int e_tc = 0, e_fc = 0;
    int s1_chain = 0, s2_chain = 0, t_chain = 0, s_chain = 0;
    string token = GetNextToken();
    bexp(ref e_tc, ref e_fc, ref token);
    token = GetNextToken();
    if (token != "then")
    {
        //错误
    }
    BackPatch(ref e_tc, Intertable.Count);  //已知真出口 e_tc,回填
    token = GetNextToken();
    do   //处理嵌套语句
    {
        s1_chain = ST_SORT(token);
        token = GetNextToken();
        if (s1_chain != 0 && token != "else")
            BackPatch(ref s1_chain, Intertable.Count);
    } while (token != "end" && token != "else");
    if (token != "else")
        token = GetNextToken();
    if (token == "else")
    {
        int q = Intertable.Count + 1;
        Gnecode("j", "_", "_", "0");
        BackPatch(ref e_fc, Intertable.Count);   //回填假出口 e_fc
        t_chain = merg(ref q, ref s1_chain);
        GetNextToken();
        s2_chain = ST_SORT(token);
        if (s2_chain != 0)
            s_chain = merg(ref s2_chain, ref t_chain);
        else
            s_chain = t_chain;
        token = GetNextToken();
        if (token != "end")
        {
            do
            {
                if (s1_chain != 0 && token != "else")
                    BackPatch(ref s1_chain, Intertable.Count);
            } while (token != "end" && token != "else");
        }
        return s_chain;
    }
```

```
            else
            {
                Index--;
                if (s1_chain != 0)
                    return (merg(ref s1_chain, ref e_fc));
                else
                    return e_fc;
            }
        }

        private void bexp(ref int be_tc, ref int be_fc, ref string token)
        {
            int be1_tc = 0, be1_fc = 0;
            if (token == "not" || token == "(" || isidentifier(token) != -1 ||
isconst(token) != -1)
            {
                bt(ref be_tc, ref be_fc, ref token);
                token = GetNextToken();
                if (token == "or")
                {
                    BackPatch(ref be_fc, Intertable.Count + 1);     //回填假出口
                    token = GetNextToken();
                    bexp(ref be1_tc, ref be1_fc, ref token);  //处理 or 后的布尔表达式
                    merg(ref be1_tc, ref be_tc);  //合并真出口
                    be_tc = be1_tc;
                }
                else
                {
                    Index--;
                }
            }
        }

        private void bt(ref int bt_tc, ref int bt_fc, ref string token)
        {
            int bt1_tc = 0, bt1_fc = 0;
            if (token == "not" || token == "(" || isidentifier(token) != -1 ||
isconst(token) != -1)
            {
                bf(ref bt_tc, ref bt_fc, ref token);//处理单个因子
                token = GetNextToken();
                if (token == "and")
                {
                    BackPatch(ref bt_tc, Intertable.Count + 1);
                    token = GetNextToken();
```

```
                bt(ref bt1_tc, ref bt1_fc, ref token);
                merg(ref bt1_fc, ref bt_fc);
                bt_tc = bt1_tc;
                bt_fc = bt1_fc;
            }
            else
            {
                Index--;
            }
        }

    }

    private void bf(ref int bf_tc, ref int bf_fc, ref string token)
    {
        string bterm1 = "", bterm2 = "", rop = "";
        if (token == "not")
        {
            token = GetNextToken();
            bexp(ref bf_tc, ref bf_fc, ref token);
        }
        else if (token == "(")
        {
            token = GetNextToken();
            bexp(ref bf_tc, ref bf_fc, ref token);
            token = GetNextToken();
            if (token != ")")
            {
                //右括号不匹配
            }
        }
        else //token 字不是括号,而是变量,则是关系运算
        {
            Index--;  //退回一个 token 字
            bterm1 = aexpr(bterm1);  //调用算术表达式的分析
            rop = GetNextToken();
            if (rop == ">" || rop == ">=" || rop == "<" || rop == "<=" || rop == "="
|| rop == "<>")
            {
                bterm2 = aexpr(bterm2);
                string str1 = "j" + rop;
                bf_tc = Intertable.Count + 1;
                Gnecode(str1, bterm1, bterm2, "0");
                bf_fc = Intertable.Count + 1;
                Gnecode("j", "_", "_", "0");
```

```
        }
        else  //常量
        {
            Gnecode("jnz", bterm1, "_", "0");
            Gnecode("j", "_", "_", "0");
        }
    }
}

private void BackPatch(ref int bf, int nxq)
{
    int index1 = bf, index2;
    do
    {
        index2 = Convert.ToInt32(Intertable[index1 - 1].result);
        Intertable[index1 - 1].result = nxq.ToString();
        index1 = index2;
    } while (index1 != 0);
}

private int merg(ref int bt1, ref int bt2)
{
    Intertable[bt1 - 1].result = bt2.ToString();
    return bt1;
}

/// <summary>
/// for 语句翻译
/// </summary>
private int fors()
{
    String token = GetNextToken();
    int q = 0, f_again = 0, f_chain = 0, s1_chain = 0;
    string f_place = "", e1_place = "", e2_place = "", str2;
    f_place = token;
    token = GetNextToken();
    if (token != ":=")
    {
        //错误
    }
    e1_place = aexpr(e1_place);  //处理第一个表达式
    Gnecode(":=", e1_place, "_", f_place);  //产生赋初值的四元式
    token = GetNextToken();
    if (token != "to")
    {
```

```
        //错误
    }
    e2_place = aexpr(e2_place);  //处理第二个表达式
    str2 = NEWT();   //产生一个临时变量放入 str2 中
    Gnecode(":=", e2_place, "_", str2);  //产生终值的四元式
    q = Intertable.Count + 1;
    Gnecode("j", "_", "_", (q + 2).ToString());  //产生 jmp 的四元式
    f_again = q + 1;
    Gnecode("+", f_place, "1", f_place);   //产生 i := i+1 的四元式
    Gnecode("j>", f_place, str2, "0");   //产生比较的四元式
    f_chain = Intertable.Count + 1;
    Gnecode("j", "_", "_", "0");
    token = GetNextToken();
    if (token != "do")
    {
        //错误
    }
    int f_compare = q + 2;
    BackPatch(ref f_compare, Intertable.Count + 1);  //回填真出口
    token = GetNextToken();    //处理 do 之后的语句
    do   //处理嵌套语句
    {
        s1_chain = ST_SORT(token);
        token = GetNextToken();
        if (s1_chain != 0 && token != "end")
            BackPatch(ref s1_chain, Intertable.Count + 1);
    } while (token != "end");
    if (s1_chain != 0)
        BackPatch(ref s1_chain, f_again);  //回填假出口
    Gnecode("j", "_", "_", f_again.ToString());
    return f_chain;
}

/// <summary>
/// while 语句翻译
/// </summary>
private int whiles()
{
    int r_head = Intertable.Count + 1;  //记录 repeat 语句开头的四元式编号
    int s1_chain = 0, be_tc = 0, be_fc = 0;
    string token = GetNextToken();
    bexp(ref be_tc, ref be_fc, ref token);   //处理布尔表达式
    token = GetNextToken();
    if (token != "do")
    {
```

```
            //错误
        }
        BackPatch(ref be_tc, Intertable.Count + 1);   //回填真出口
        token = GetNextToken();
        do    //处理嵌套语句
        {
            s1_chain = ST_SORT(token);   //处理循环体
            token = GetNextToken();
            if (s1_chain != 0 && token != "end")
                BackPatch(ref s1_chain, Intertable.Count + 1);
        } while (token != "end");
        int q = Intertable.Count + 1;
        Gnecode("j", "_", "_", "0");
        s1_chain = merg(ref q, ref s1_chain);
        if (s1_chain != 0)
            BackPatch(ref s1_chain, r_head);
        return be_fc;
    }
    //四元式显示
    private void ShowIntertable()
    {
        FourlistView.Visible = true;
        foreach (Inter intertableItem in Intertable)
        {
            FourlistView.Items.Add(intertableItem.num.ToString());
            FourlistView.Items[this.FourlistView.Items.Count -
1].SubItems.Add(intertableItem.op.ToString());
            FourlistView.Items[this.FourlistView.Items.Count -
1].SubItems.Add(intertableItem.arg1.ToString());
            FourlistView.Items[this.FourlistView.Items.Count -
1].SubItems.Add(intertableItem.arg2.ToString());
            FourlistView.Items[this.FourlistView.Items.Count -
1].SubItems.Add(intertableItem.result.ToString());
        }
    }
    private void 开始SToolStripMenuItem_Click(object sender, EventArgs e)
    {
        Parser();
        If(con){
         ShowIntertable();
         analyseResult.text=”分析成功”;
         }
         else analyseResult.text=”分析失败”;

    }
```

```
    }
}
```

## 五、 功能测试

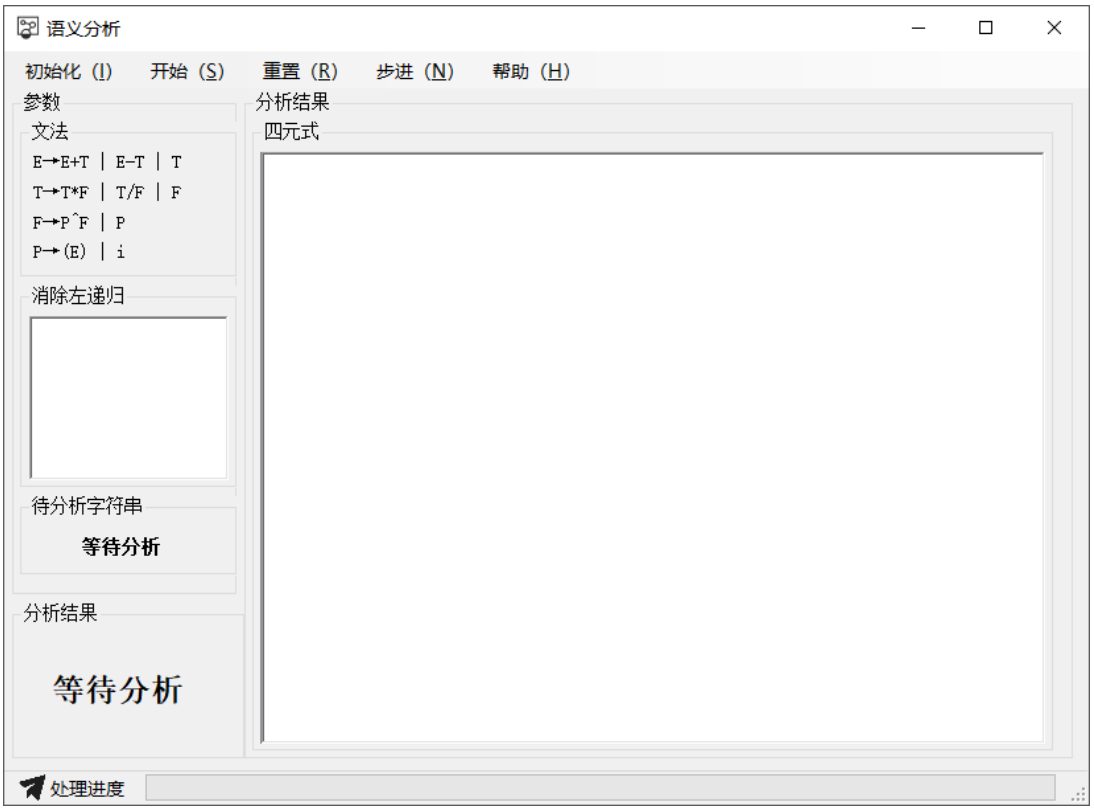
测试用例：

正确测试用例：i+i*i^i

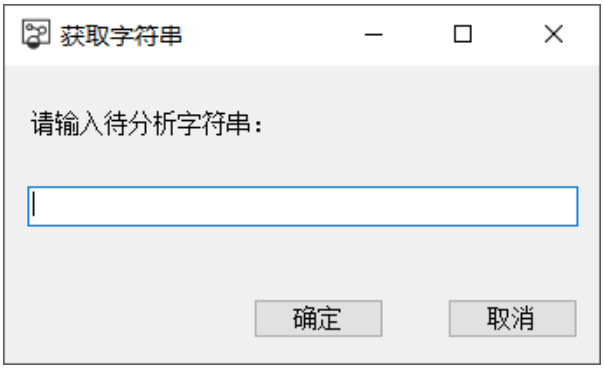
错误测试用例：)i+i

测试过程：

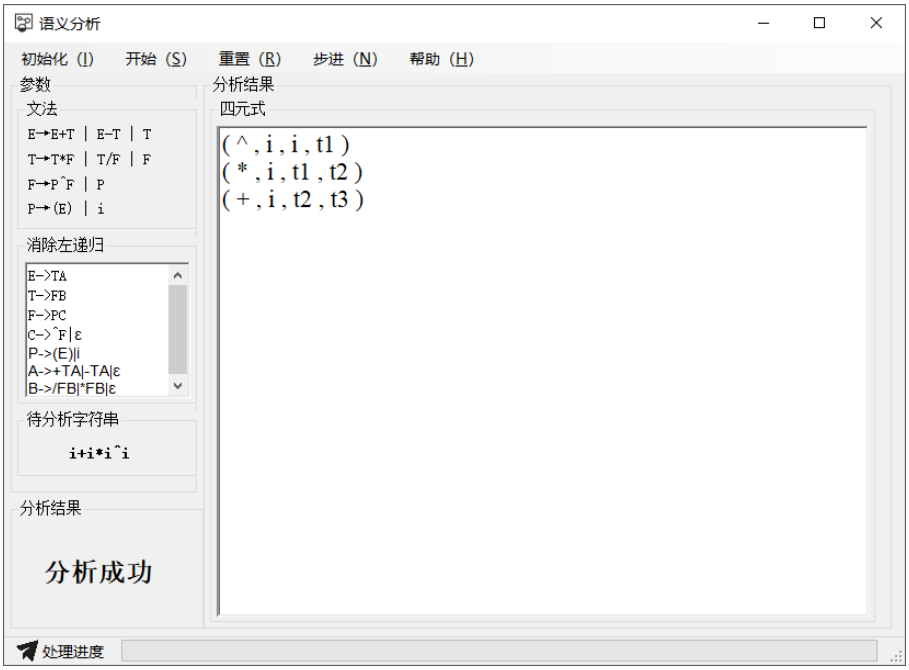
1.打开实验软件，选择“语义分析”，进入语义分析子程序；



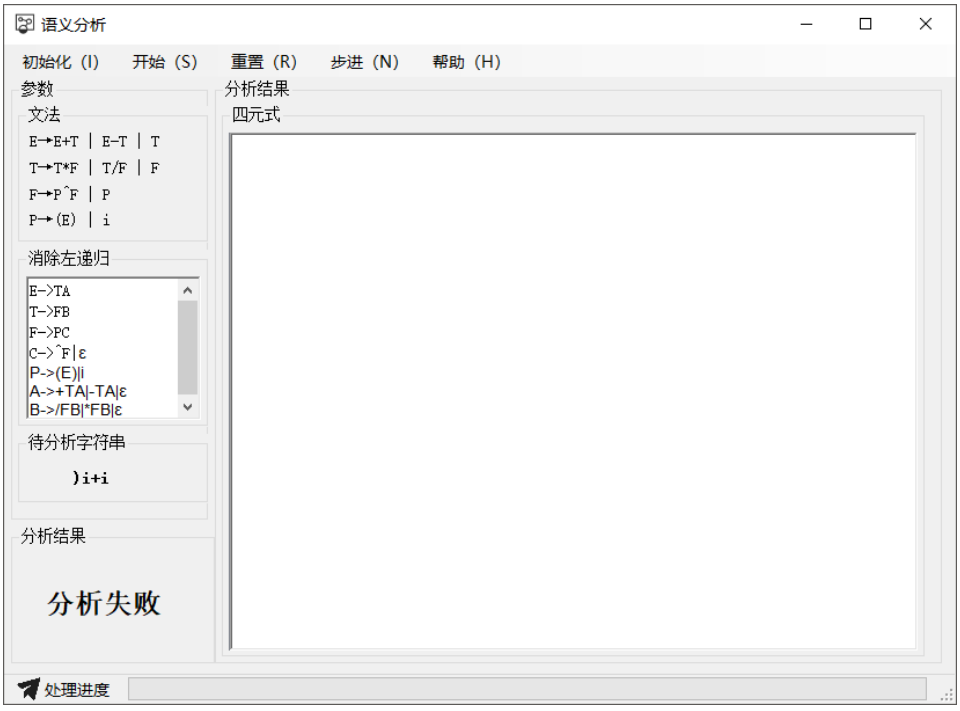

2.点击“初始化”，分别选择“消除左递归”与“字符串输入”，如下图所示




3.点击“开始”，对该字符串进行语法分析和语义分析，输入正确测试用例的结果如下：




输入错误测试用例的结果如下：




## 六、 心得体会

通过本次实验，我加深了对语法制导翻译的理解，同时使用 SLR(1)分析法对语法进行了分析。在实际问题中，为了处理不同的翻译问题，如类型检查、各种控制语句的翻译中，我参考了一部分网络上的资料与代码，并进行调试运行。对于自顶向下的分析方法，语法制导翻译可以继承语法分析过程中属性传递的信息，实现从产生式到语义动作的转换。