语法分析实验报告

09021220 周天逸

实验目的

构造LL(1)文法自顶向下语法分析程序

1. 输入字符串
2. 输出语法树(推导序列)

实验内容

1. 方法
    i. 定义LL(1)文法
    ii. 求出First， Follow 集
    iii. 构造LL(1)分析表
    iv. 根据分析表编程
2. LL(1)文法

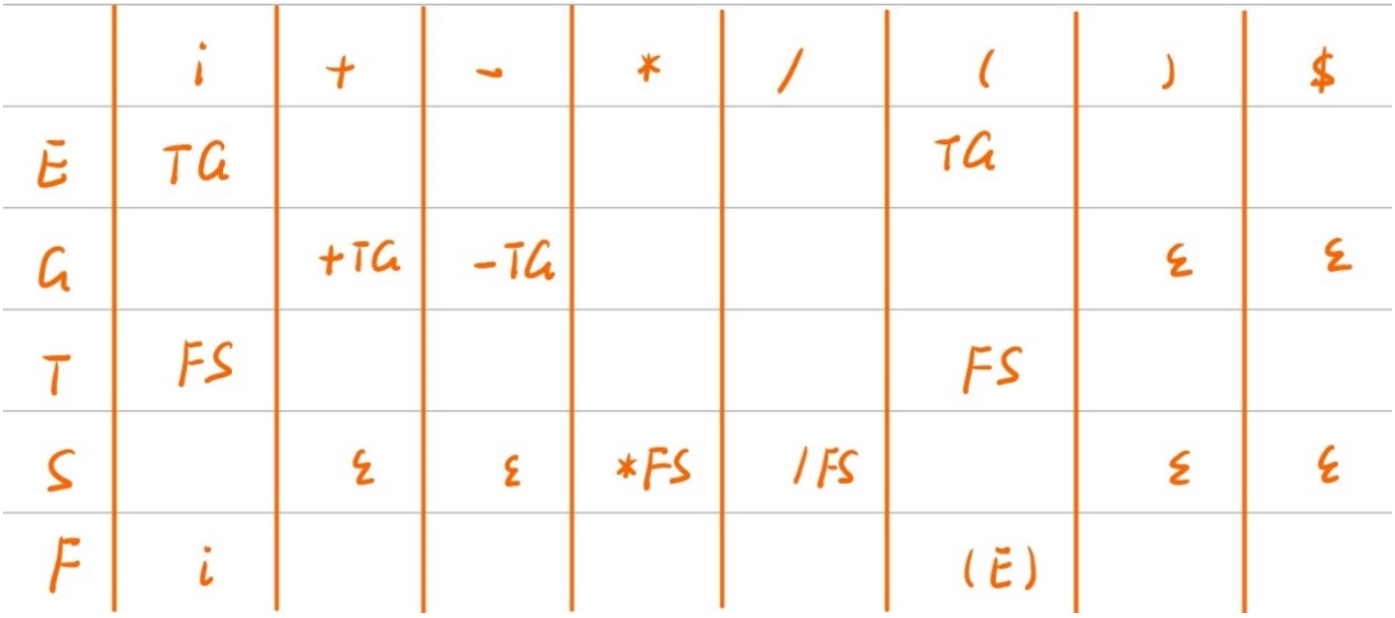
```
LL(1)文法
 (1) E->TG
 (2) G->+TG
 (3) G->-TG
 (4) G->ϵ
 (5) T->FS
 (6) S->*FS
 (7) S->/FS
 (8) S->ϵ
 (9) F->(E)
 (10) F->i
```

3. 构造First,Follow

	First	Follow
E->TG	(, i	$,)
G->+TG	+	$,)
G->-TG	-	$,)
G->ε	ε	$,)

T->FS	(,i	+,-,$,)
S->*FS	*	+,-,$,)
S->/FS	/	+,-,$,)
S->ε	ε	+,-,$,)
F->(E)	(	+,/,*,-,$,)
F->i	i	+,/,*,-,$,)

4. 构造分析表



5. 核心算法描述

   i. 先将‘$’与文法开始符E压入分析栈，并使输入字符指向input的第一位；
   ii. 弹出栈顶元素v，判断v是否与输入字符相同，是转至3，否转至4；
   iii. 输入字符移向input下一位，转至2；若已为最后一位转至8；
   iv. 判断v是否为非终结符，是转至5；否转至9；
   v. 判断v与输入字符是否有产生式，是转至6，否转至9；
   vi. 判断产生式是否为ε；是转至2，否转至7；
   vii. 将产生式逆序压入分析栈中，转至2；
   viii. 输出过程步骤，以及为合法字符串；
   ix. 输出错误位置并结束；

6. 数据结构

   i. 使用一维数组存放非终结符和终结符
   ii. 使用二位数组存放LL(1)分析表

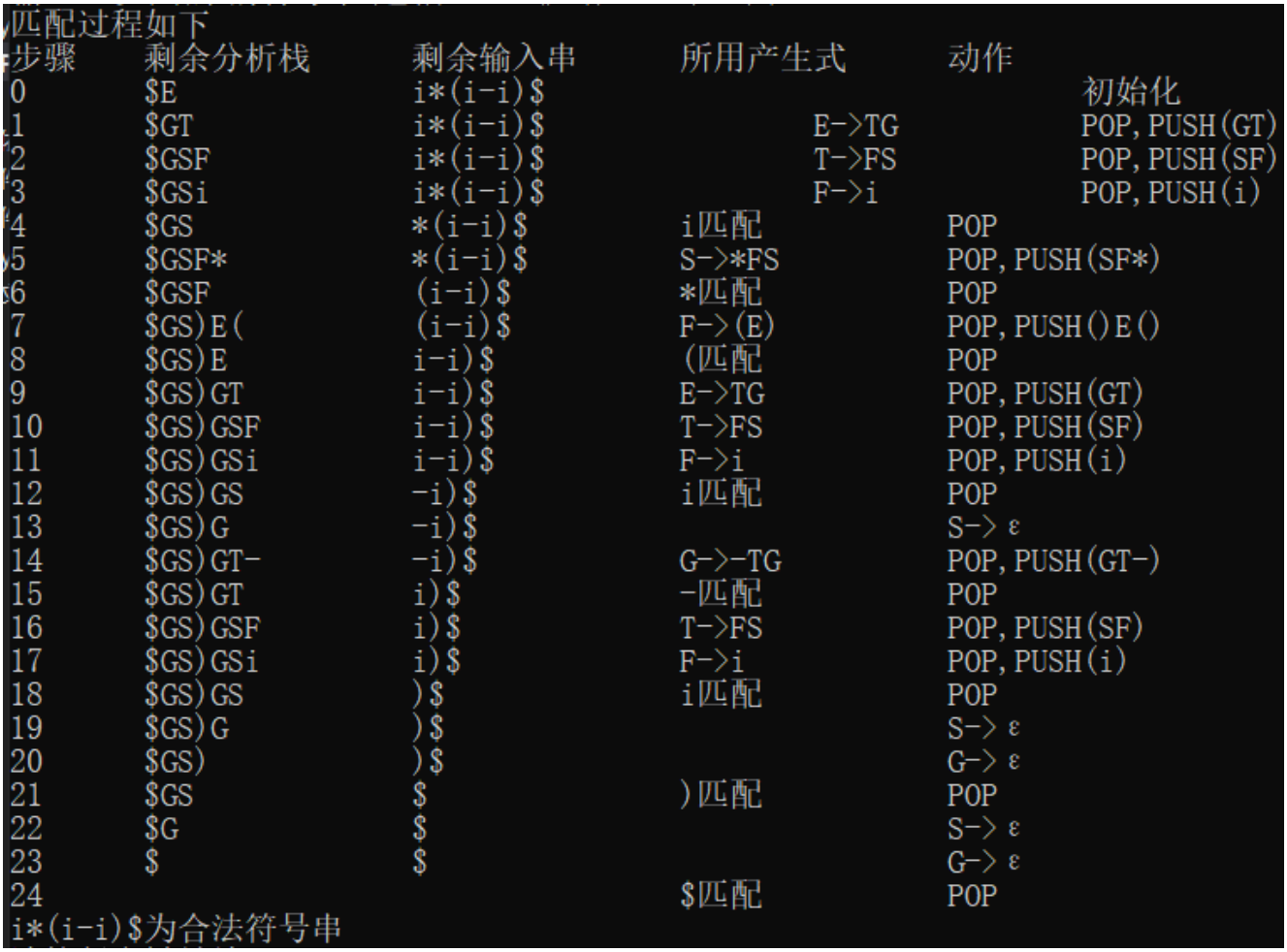
```
char Vn[M] = { 'E', 'G', 'T', 'S', 'F' };
char Vt[N] = { 'i', '+', '-', '*', '/', '(', ')', '$' };
string L[M][N] = {
    { "TG", "ERROR", "ERROR", "ERROR", "ERROR", "TG", "ERROR", "ERROR" },
    { "ERROR", "+TG", "-TG", "ERROR", "ERROR", "ERROR", "NULL", "NULL" },
    { "FS", "ERROR", "ERROR", "ERROR", "ERROR", "FS", "ERROR", "ERROR" },
    { "ERROR", "NULL", "NULL", "*FS", "/FS", "ERROR", "NULL", "NULL" },
    { "i", "ERROR", "ERROR", "ERROR", "ERROR", "(E)", "ERROR", "ERROR" }
};
```

7. 案例分析

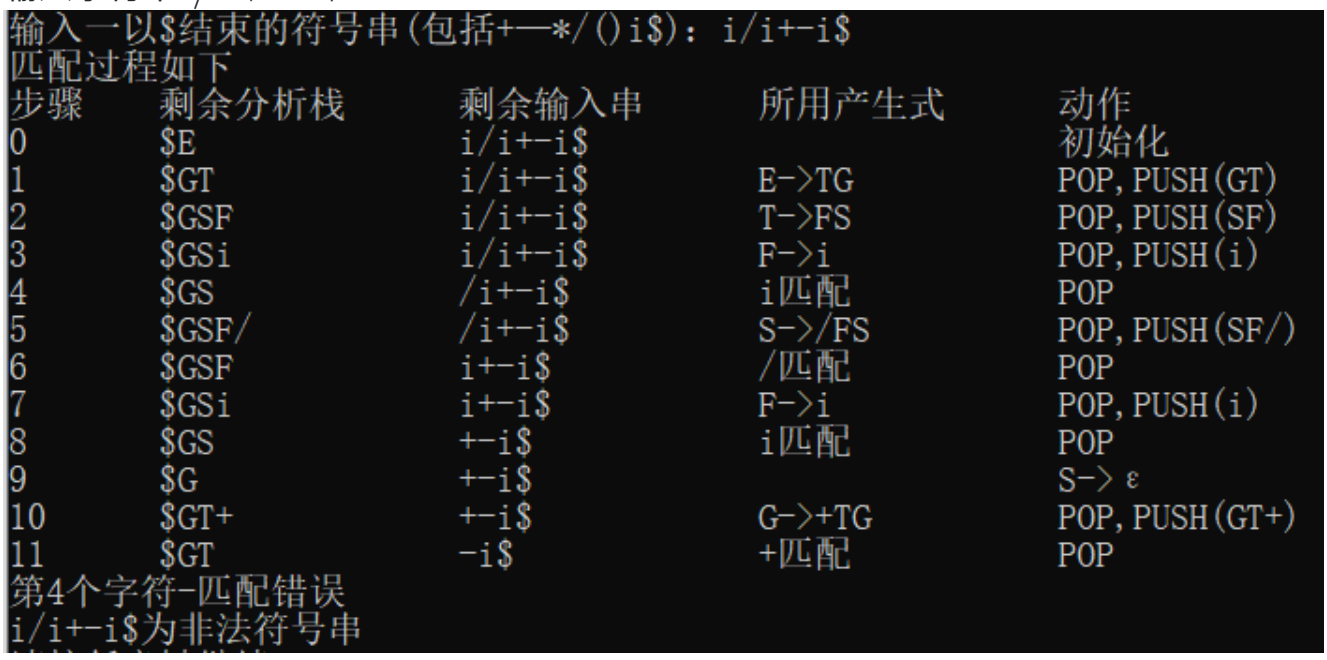
i. 输入字符串$i+i*i$

```
输入一以$结束的符号串(包括+—*/()i$)：i+i*i$
匹配过程如下
步骤    剩余分析栈      剩余输入串      所用产生式      动作
0       $E              i+i*i$                          初始化
1       $GT             i+i*i$          E->TG           POP,PUSH(GT)
2       $GSF            i+i*i$          T->FS           POP,PUSH(SF)
3       $GSi            i+i*i$          F->i            POP,PUSH(i)
4       $GS             +i*i$           i匹配           POP
5       $G              +i*i$                           S-> ε
6       $GT+            +i*i$           G->+TG          POP,PUSH(GT+)
7       $GT             i*i$            +匹配           POP
8       $GSF            i*i$            T->FS           POP,PUSH(SF)
9       $GSi            i*i$            F->i            POP,PUSH(i)
10      $GS             *i$             i匹配           POP
11      $GSF*           *i$             S->*FS          POP,PUSH(SF*)
12      $GSF            i$              *匹配           POP
13      $GSi            i$              F->i            POP,PUSH(i)
14      $GS             $               i匹配           POP
15      $G              $                               S-> ε
16      $               $                               G-> ε
17                                      $匹配           POP
i+i*i$为合法符号串
```

ii. 输入字符串$i*(i-i)$



iii. 输入字符串$i/i+-i\$$



8. 核心代码

i. 查找LL表对应的内容

```
if (L[m][n] == "ERROR")
        {
            cout << "第" << i << "个字符" << input[i] << "匹配错误" << endl;
```

```
            cout << input << "非法串" << endl;
            break;
        }
        if (L[m][n] == "NULL")
        {
            cout << j << "\t";
            outstack();
            cout << "\t\t";
            outinput(input, i);
            cout << "\t\t\t\t" << v << "->" << "ε" << endl;
            continue;
        }
        else
        {
            //
            string t = "";
            for (int j = L[m][n].length() - 1; j >= 0; j--)
            {
                t += L[m][n][j];
                analyse.push(L[m][n][j]);
            }
            cout << j << "\t";
            outstack();
            cout << "\t\t";
            outinput(input, i);
            cout << "\t\t" << v << "->" << L[m][n];
            cout << "\t\tPOP,PUSH(" << t << ")";
        }
```

9. 分析体会

   i. 本次实验前期从First、Follow集计算以及据此构造分析表过程比较容易。编程过程中，通过构造分析表以及利用stack结构对程序流程进行控制并实现分析流程输出。

   ii. 通过这次实验对之前学到的语法分析有了进一步的了解，加深了对于语法分析的步骤的理解与领悟，对于今后对编译原理的学习有很大的帮助。

实验二

构造LR(1)文法自底向上语法分析程序

1. 输入字符串
2. 输出语法树(推导序列)
3. 使用文法

```
(0) S'->E

(1) E->E+T

(2) E->T

(3) T->T*F

(4) T->F

(5) F->(E)

(6) F->i
```

4. 构造LR分析表

```
读取的文件名为：exp.txt
-----------------ACTION表-----------------
+   *   (   )   i   $
N   N   s4  N   s5  N
s6  N   N   N   N   acc
r2  s7  N   r2  N   r2
r4  r4  N   r4  N   r4
N   N   s4  N   s5  N
r6  r6  N   r6  N   r6
N   N   s4  N   s5  N
N   N   s4  N   s5  N
s6  N   N   s11 N   N
r1  s7  N   r1  N   r1
r3  r3  N   r3  N   r3
r5  r5  N   r5  N   r5
-----------------GOTO表-----------------
```

将LR文法所需要的分析表写在文件中，每次使用是直接打开即可。

Goto 表：为读入非终结符后状态的转移

```
------------------GOTO表------------------
E    T    F
1    2    3
-1   -1   -1
-1   -1   -1
-1   -1   -1
8    2    3
-1   -1   -1
-1   9    3
-1   -1   10
-1   -1   -1
-1   -1   -1
-1   -1   -1
-1   -1   -1
```

5. 本次实验使用的数据结构

   i. 还是与LL文法分析类似，通过二维数组表示状态转移，一维数组表示终结符，非终结符和产生式

```
// ACTION表
// + * ( ) i #
string action[12][6];
// goto表
// a b #
int _goto[12][3];
string vt = "+*()i#";       // 终结符表
string vn = "ETF";          // 非终结符表
string LR[6] = { "E->E+T", "E->T", "T->T*F", "T->F", "F->(E)", "F->i"
};   // 存放产生式
```

6. 核心算法

   i. 使用两个栈，一个为状态栈一个是符号栈，最开始时，只有状态0和文法开始符号E

   ii. 使用一个指针依次遍历每一个输入符号。

   iii. 在action表中对应栈顶和输入符号，进行转移。如果发现表中为N，则说明匹配错误，是非法串。

   iv. 如果要进行规约时，遍历产生式数组，找到使用的产生式。之后弹出对应数目的符号

和状态，并将产生式头部放入栈顶。

v. 查看Goto表，如果发现在当前状态下栈顶的非终结符满足要求，则将要转移的状态加入状态栈中。

vi. 最后如果能到达acc状态则说明满足要求为合法串。

7. 结果展示

```
------------------待分析表达式1: i+(i*i)*(i+i)#--------------------
表达式1: i+(i*i)*(i+i)#分析开始
序号      状态栈         符号栈         输入串         描述
0   0   #   i+(i*i)*(i+i)#   初始状态
1   5 0     #i  +(i*i)*(i+i)#       s5 移进
2   3 0     #F            归约 F->i
3   2 0     #T            归约 T->F
4   1 0     #E            归约 E->T
5   6 1 0   #E+ (i*i)*(i+i)#        s6 移进
6   4 6 1 0     #E+(    i*i)*(i+i)#     s4 移进
7   5 4 6 1 0   #E+(i   *i)*(i+i)#      s5 移进
8   3 4 6 1 0   #E+(F             归约 F->i
9   2 4 6 1 0   #E+(T             归约 T->F
10  7 2 4 6 1 0     #E+(T*  i)*(i+i)#       s7 移进
11  5 7 2 4 6 1 0   #E+(T*i )*(i+i)#        s5 移进
12  10 7 2 4 6 1 0  #E+(T*F           归约 F->i
13  2 4 6 1 0   #E+(T             归约 T->T*F
14  8 4 6 1 0   #E+(E             归约 E->T
15  11 8 4 6 1 0    #E+(E)  *(i+i)#     s11 移进
16  3 6 1 0     #E+F              归约 F->(E)
17  9 6 1 0     #E+T              归约 T->F
18  7 9 6 1 0   #E+T*   (i+i)#      s7 移进
19  4 7 9 6 1 0     #E+T*(  i+i)#       s4 移进
20  5 4 7 9 6 1 0   #E+T*(i +i)#        s5 移进
21  3 4 7 9 6 1 0   #E+T*(F           归约 F->i
22  2 4 7 9 6 1 0   #E+T*(T           归约 T->F
23  8 4 7 9 6 1 0   #E+T*(E           归约 E->T
24  6 8 4 7 9 6 1 0     #E+T*(E+    i)#      s6 移进
25  5 6 8 4 7 9 6 1 0   #E+T*(E+i   )#      s5 移进
26  3 6 8 4 7 9 6 1 0   #E+T*(E+F           归约 F->i
27  9 6 8 4 7 9 6 1 0   #E+T*(E+T           归约 T->F
28  8 4 7 9 6 1 0   #E+T*(E           归约 E->E+T
29  11 8 4 7 9 6 1 0    #E+T*(E)    #       s11 移进
30  10 7 9 6 1 0    #E+T*F            归约 F->(E)
31  9 6 1 0     #E+T              归约 T->T*F
32  1 0     #E            归约 E->E+T
33  1 0     #E  i+(i*i)*(i+i)#   accept 接受!
```

注意：在exp文件中可以输入想要判断的字符串。

8. 核心代码展示

```
    else if(actionStr[0] == 'r') {   // 归约
        int num = stoi(actionStr.substr(1));    // 选用第几个产生式归约
        int len = LR[num-1].length()-3;
        while(len--) {
            chars.pop();         // 出栈，归约
            state.pop();
            charsStr = charsStr.substr(0, charsStr.length()-1);
            copyState.pop();   // 便于输出
        }
        chars.push(LR[num-1][0]);   // 产生式左部入符号栈
        charsStr += LR[num-1][0];

        int nowState = state.top();
        int gidx = findNonTerminator(LR[num-1][0]);
        int newState = _goto[nowState][gidx];
        state.push(newState);
        copyState.push(newState);

        cout << cnt++ << '\t' << s2string(copyState) << '\t' << charsStr
<< '\t';
        printRestInput(exp, idx, len);
        cout << '\t' << " 归约 " << LR[num-1] << endl;
    } else if(actionStr[0] == 's') {    // 移进
        int newState =  stoi(actionStr.substr(1));
        state.push(newState);
        copyState.push(newState);

        chars.push(nowChar);
        charsStr += nowChar;
        ++idx;  // 输入指针后移

        cout << cnt++ << '\t' << s2string(copyState) << '\t' << charsStr
<< '\t';
        printRestInput(exp, idx, len);
        cout << '\t' << actionStr << " 移进 " << endl;
```

展示的是判断当前的action表的状态是移入还是规约。如果移入，指针后移，状态栈和符号栈都加入新元素。如果规约，找到对应产生式，弹出相应的个数。

9. 思考与总结

i. LL(1) 文法： 采用自顶向下的解析方法。LL(1) 文法表示左到右扫描输入，左推导产生式表示在任何时刻我们只需要看向输入的一个符号来决定如何展开产生式。

ii. LR(1) 文法： 采用自底向上的解析方法。LR(1) 文法表示从右到左扫描输入，同时进行

右推导表示在任何时刻我们只需要看向输入的一个符号来决定如何规约。

iii. LL(1) 文法： 构建 LL(1) 预测分析表相对较简单，因为每个表项至多包含一个产生式。LL(1) 文法的分析表通常比较小，构建速度相对较快。

iv. LR(1) 文法： 构建 LR(1) 分析表可能更复杂，因为每个表项可能包含多个产生式，且可能涉及到移进-归约冲突和规约-规约冲突。因此，LR(1) 文法的分析表可能比较庞大，构建速度相对较慢。