专题6\_语法制导翻译程序设计原理与实现技术

13281132 李若森 计科1301

# 实验目的

语法制导的基本概念；目标代码结构分析的基本方法；赋值语句语法制导生成四元式的基本原理和方法；该过程包括语法分析和语义分析过程。

# 实验内容

## 2.1 实验项目

完成以下描述赋值语句和算术表达式文法的语法制导生成中间代码四元式的过程。

G[A]:

A→V=E

E→E+T|E-T|T

T→T\*F|T/F|F

F→(E)|i

V→i

## 设计说明

终结符号i为用户定义的简单变量，即标识符的定义。

## 2.3 设计要求

1. 设计语法制导翻译过程，给出每一产生式对应的语义动作；
2. 设计中间代码四元式的结构（暂不与符号表有关）；
3. 输入串应是词法分析的输出二元式序列，即某算术表达式“专题1”的输出结果。输出为输入串的四元式序列中间文件；
4. 设计两个测试用例（尽可能完备），并给出程序执行结果。

## 2.4 任务分析

重点解决赋值语句文法的改写和语义动作的添加。

# 实现过程

## 3.1 扩展文法

G[s]:

S→A

A→V=E

E→E+T|E-T|T

T→T\*F|T/F|F

F→(E)|i

V→i

## 3.2 非终结符FOLLOW集

FOLLOW(S) = { # }

FOLLOW(A) = { # }

FOLLOW(V) = { = }

FOLLOW(E) = { +, -, ), # }

FOLLOW(T) = { +, -, \*, /, ), # }

FOLLOW(F) = { +, -, \*, /, ), # }

## 3.3 LR(0)分析器的构造

设DFA M的一个状态为i，该状态识别出的所有活前缀的有效项目集为Ci。则DFA M的状态集Q={C0, C1, C2, … , Cn}=C。C称为文法的LR(0)有效项目集规范族。

对Ci有三种操作：

|  |
| --- |
|  |
|  |

求文法的LR(0)有效项目集规范族C的算法：

|  |
| --- |
|  |

由上述算法可求得有效项目集规范族C={C0, C1, C2, … , C19}。

### 3.3.1 有效项目规范簇：

C0 = { S→·A, A→·V=E, V→·i }

C1 = { S→A· }

C2 = { S→V·=E }

C3 = { V→i· }

C4 = { S→V=·E, E→·E+T, E→·E-T, E→·T,

T→·T\*F, T→·T/F, T→·F, F→·(E), F→·i }

C5 = { S→V=E·, E→E·+T, E→E·-T }

C6 = { E→T·, T→T·\*F, T→T·/F }

C7 = { T→F· }

C8 = { F→(·E), E→·E+T, E→·E-T, E→·T,

T→·T\*F, T→·T/F, T→·F,F→·(E), F→·i }

C9 = { F→i· }

C10 = { E→E+·T, T→·T\*F, T→·T/F, T→·F, F→·(E), F→·i }

C11 = { E→E-·T, T→·T\*F, T→·T/F, T→·F, F→·(E), F→·i }

C12 = { T→T\*·F, F→·(E), F→·i }

C13 = { T→T/·F, F→·(E), F→·i }

C14 = { F→(E·), E→E·+T, E→E·-T }

C15 = { E→E+T·, T→T·\*F, T→T·/F }

C16 = { E→E-T·, T→T·\*F, T→T·/F }

C17 = { T→T\*F· }

C18 = { T→T/F· }

C19 = { F→(E)· }

### 3.3.2 冲突项目：

定义：

1. 既含移进项目,又含归约项目
2. 含有多个归约项目

在项目集中存在冲突项目：

C5:

C6:

C15:

C16:

冲突可解决，所以语义分析采用SLR(1)分析法分析。

## 3.4 SLR(1)分析法

LR(0)有效项目集规范族C中，若一个项目集含有m个移进项目，同时含n个归约项目，即

|  |
| --- |
|  |

若, FOLLOW(V1), …, FOLLOW(Vn)两两互不相交，满足此条件的文法为SLR(1)文法，可用SLR(1)分析方法解决。

|  |
| --- |
|  |

## 3.5 SLR(1)分析器的构造

1. 拓广文法G’
2. 构造(对G’)LR(0)有效项目集族C和Go函数
3. 若GO(Ci, a)=Cj，，则置ACTION(i, a)=Sj
4. 若，且，，则置ACTION(i, a)=rj

(为第j个产生式)

1. 若，(S为拓广文法开始符号)，则置ACTION(k, #)=acc
2. 若GO(Ci, A)=Cj，，置GOTO(i, A)=j
3. 其它：空白(error)

## 3.6 SLR(1)分析表

|  |
| --- |
| 表3-1 SLR(1)分析表 |
|  |

## 3.7 语义动作添加

|  |  |
| --- | --- |
| S→A | {} |
| A→V=E | { GEN(=, E·PLACE, \_, V·PLACE); } |
| E(1)→E(2)+T | { E(1)·PLACE=NEWTEMP; GEN(+,E(2)·PLACE,T·PLACE,E(1)·PLACE); } |
| E(1)→E(2)-T | { E(1)·PLACE=NEWTEMP; GEN(-,E(2)·PLACE,T·PLACE,E(1)·PLACE); } |
| E→T | { E·PLACE=T·PLACE; } |
| T(1)→T(2)\*F | { T(1)·PLACE=NEWTEMP; GEN(\*,T(2)·PLACE,F·PLACE,T(1)·PLACE); } |
| T(1)→T(2)/F | { T(1)·PLACE=NEWTEMP; GEN(/,T(2)·PLACE,F·PLACE,T(1)·PLACE); } |
| T→F | { T·PLACE=F·PLACE; } |
| F→(E) | { F·PLACE=E·PLACE; } |
| F→i | { F·PLACE=ENTRY(i); } |
| V→i | { V·PLACE=ENTRY(i); } |

## 3.8 主要数据结构

**pair<int, string>:**

用pair<int, string>来存储单个二元组。

对照表由专题1定义。其中，第一个元素为类型号，第二个为元素的值。当类型号小于40时，代表程序分界符，第二个元素不存储有效信息，用“-”代替；当类型号为40时，代表标识符，第二个元素存储标识符字符串；当类型号为41时，代表实数，第二个元素存储的是该实数的二进制值。

typedef pair<int, string> PIS;

**pair<string, int>:**

用pair<string, int>来存储终结符与非终结符的离散化值。

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 符号 | 离散值 | 符号 | 离散值 | 符号 | 离散值 |
| i | 1 | / | 6 | A | 11 |
| = | 2 | ( | 7 | V | 12 |
| + | 3 | ) | 8 | E | 13 |
| - | 4 | # | 9 | T | 14 |
| \* | 5 | S | 10 | F | 15 |

**struct Node:**

用struct Node来存储四元组，表示(op, arg1, arg2, res)

其中op为操作符，arg1和arg2为变量，res为运算结果。一目运算用arg1。

**vector<>:**

vector是C++中的动态数组，用来存储每一行的二元组。

## 3.9 函数定义

|  |
| --- |
| **init:** |
| void init(); |
| **功能：** |
| 初始化SLR(1)分析表，非终结符值栈（顶），关键字及识别码对照表，离散化（非）终结符 |
| **传入参数：**（无） |
| **传出参数：**（无） |
| **返回值：**（无） |
|  |
| **newTemp:** |
| string newTemp(); |
| **功能：** |
| 生成新临时变量 |
| **传入参数：**（无） |
| **传出参数：**（无） |
| **返回值：** |
| 新临时变量名 |
|  |
| **generate:** |
| void generate( int no, vector<Node> &vecGen ); |
| **功能：** |
| 根据规约式生成相应四元式 |
| **传入参数：** |
| no: 规约项 |
| **传出参数：** |
| vecGen: 四元式序列 |
| **返回值：**（无） |
|  |
| **sentenceAnalysis:** |
| bool sentenceAnalysis( vector<PIS> &vec, int &ecol,  vector<Node> &vecGen ); |
| **功能：** |
| 生成新临时变量 |
| **传入参数：** |
| vec: 输入二元式 |
| **传出参数：** |
| ecol: 出错标识符位置 |
| vecGen: 四元式序列 |
| **返回值：** |
| 是否成功解析。是则返回true，否则返回false。 |
|  |
| **getOutputName:** |
| void getOutputName( char \*inputName, char \*outputName ); |
| **功能：** |
| 得到输出文件名 |
| **传入参数：** |
| inputname:需处理的文件的文件名 |
| **传出参数：** |
| outputname:输出文件的文件名 |
| **返回值：**（无） |
|  |
| **errMsg:** |
| void errMsg( string filename, int rowNo, int colNo ); |
| **功能：** |
| 向屏幕输出错误信息 |
| **传入参数：** |
| filename: 正在处理的文件的文件名称 |
| rowNo: 出错行 |
| colNo: 出错列 |
| **传出参数：**（无） |
| **返回值：**（无） |
|  |
| **print:** |
| void print( FILE \*fp, vector<Node> &vecGen ); |
| **功能：** |
| 将四元式结果输出到文件中 |
| **传入参数：** |
| fp: 输出文件指针 |
| vTable: 二元式序列 |
| **传出参数：**（无） |
| **返回值：**（无） |

# 实验结果

|  |
| --- |
|  |
| 图4-1 正确测试项的返回结果 |
|  |
| 图4-2 错误测试项的返回结果 |

\* 实验测试数据和输出结果详见附录中测试相关文件。

# 实验心得

通过此次专题实验，我对SLR(1),LR(0)和四元式生成这些知识点更为熟练。

在推导SLR(1)分析表过程中，我复习了LR(0)部分知识点、SLR(1)部分知识点、FOLLOW部分知识点、四元式部分知识点。在熟练掌握了这些知识点后写程序也不再困难。

在做整个实验过程中我遇上了些问题。在推导有效项目规范簇时我重复推导了三遍又特别验证了一遍，一遍遍的推导让我对这部分知识点掌握的更为熟练而且后期推到速度明显快于第一次推导过程且结果确保了绝对的正确。在填写SLR(1)分析表时因为不熟悉填写规约项部分的要求使得我在这个上面花费了一些时间。在将分析表输入程序过程中因为对csv格式的文件有些误解导致处理输问题时出了些错误。

在整个实验过程中我遇上的最大的问题是在规约过程中会出现非终结符被覆盖的问题。例如i=i+(i+i+i)。在这个例子中等于号后的i会被规约为E，在后续规约过程中左括号后的i也会被规约为E，此时两个非终结符相同会导致前面规约出的值被覆盖导致程序运算结果出错。在老师指导下我为每个非终结符开一个栈，栈顶存储最新规约出的值，这样就不用担心出现覆盖问题，解决了这一问题。

目前该工程已上传至我的Github仓库中。仓库地址为https://github.com/lrscy/Principle-of-Compiler-Homework。

# 附录

1. parse.cpp和parse.h为语法制导翻译程序文件
2. main.cpp负责处理输入输出
3. parse.exe为生成的可执行文件
4. test1.txt和test2.txt为测试输入文件
5. test1.gen和test2.gen为测试文件对应的输出文件
6. ntable.txt为关键字及标识符对照表配置文件
7. table.txt为程序读取的SLR(1)分析表配置文件
8. grammar.txt为规约式集配置文件
9. SLR(1)分析表.xlsx为SLR(1)分析表