编译原理

实

验

报

告

**实验名称：****\_\_实验二：自底向上的语法分析LR(1)\_**

**姓名：\_\_\_\_\_\_\_\_\_\_\_ 李秋阳\_\_\_\_\_\_\_\_\_\_\_**

**学号：\_\_\_\_\_\_\_\_\_\_\_180110527\_\_\_\_\_\_\_\_\_\_\_**

**学院：\_\_\_\_\_\_\_计算机科学与技术学院\_\_\_\_**

**专业：\_\_\_\_\_\_\_\_\_\_\_计算机类\_\_\_\_\_\_\_\_\_****\_\_\_**

1. **实验目的与方法**

**实验目的：**

* + - 1. **深入了解语法分析程序实现原理及方法**
      2. **理解LR(1)分析法是严格的从左向右扫描和自底向上扫描的语法分析器**

**实验方法：**

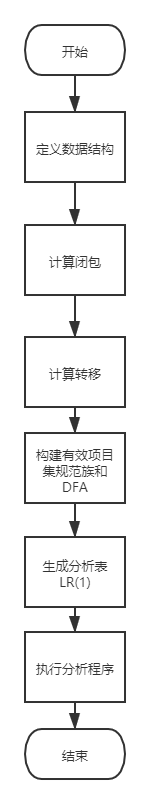
**语言：C++14**

**环境：Clion（C++编译器） 编译工作台（生成LR(1)分析表、状态机等）**

1. **实验总体流程与函数功能描述**

**在课程实验要求的基础上，完成了对输入的LR(1)文法自动构建FIRST集，自动构建有效项目集规范族和DFA，自动构建LR(1)分析表，执行分析程序输入串，输出所采用的产生式。**

**代码整体流程：**

****

**数据结构：**

产生式结构体，左部符号和右部符号串：

**struct** Production {  
 **char** left{};  
 vector<**char**> rights;  
 **bool operator**==(Production& rhs) **const** {  
 **if** (left != rhs.left)  
 **return false**;  
 **for** (**int** i = 0; i < rights.size(); i++) {  
 **if** (i >= rhs.rights.size())  
 **return false**;  
 **if** (rights[i] != rhs.rights[i])  
 **return false**;  
 }  
 **return true**;  
 }  
};

LR1项目：

**struct** LR1Item {  
 Production p;**int** location{};**char** next{};  
};

LR1项目集：

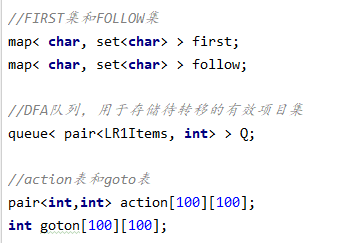
**struct** LR1Items {  
 vector<LR1Item> items;  
};

LR1项目集规范族：

**struct** CanonicalCollection {  
 vector<LR1Items> items;vector< pair<**int**,**char**> > g[100];  
}CC;

文法结构体：

**struct** Grammar {  
 **int** num{};  
 vector<**char**> T;  
 vector<**char**> N;  
 vector<Production> prods;  
}grammar;

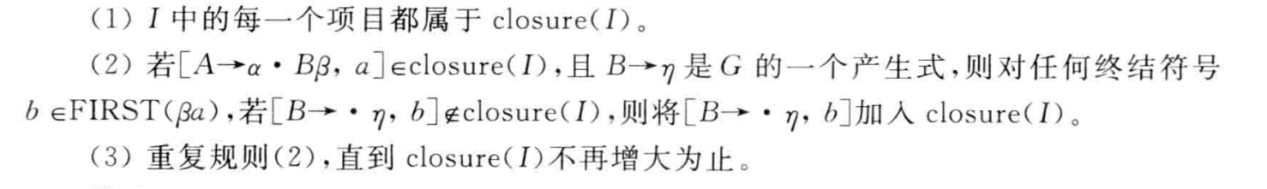


**辅助函数：**

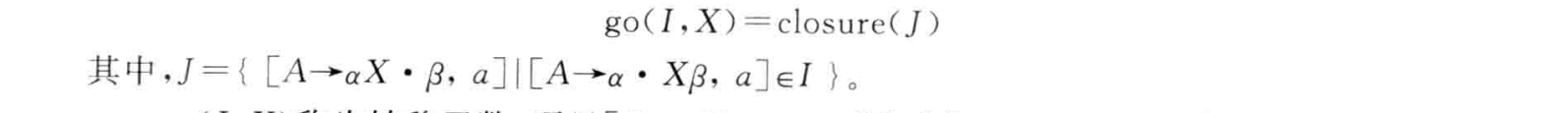
*/\* 判断ch是否是非终结符, 若是返回其序号 \*/***int** isInN(**char** ch);  
*/\* 判断ch是否是终结符, 若是返回其序号 \*/***int** isInT(**char** ch);  
*/\* 求(T U N)的FIRST集 \*/***void** getFirstSet();  
*/\* 产找alpha串的FIRST集， 保存到FS集合中 \*/***void** getFirstByAlphaSet(vector<**char**> &alpha, set<**char**> &FS);  
*/\* 求非终结符的FOLLOW集 \*/***void** getFollowSet();  
*/\* 判断是LR0项目t否在有效项目集I中 \*/***bool** isInLR0Items(LR0Items &I, LR0Item &t);  
*/\* 打印某个项目集 \*/***void** printLR0Items(LR0Items &I);  
*/\* 判断是否在项目集规范族中，若在返回序号 \*/***int** isInCanonicalCollection(LR0Items &I);  
*/\* 读入并初始化语法 \*/***void** initGrammar();

**主要函数：**

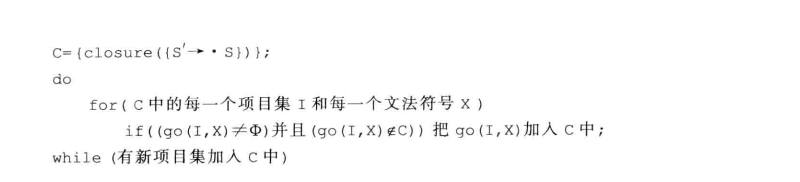
计算闭包实现原理：



计算转移实现原理：



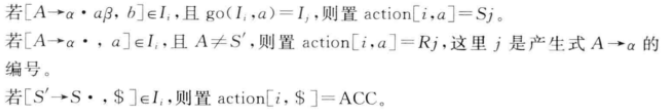
构建有效项目集规范族和DFA：

****

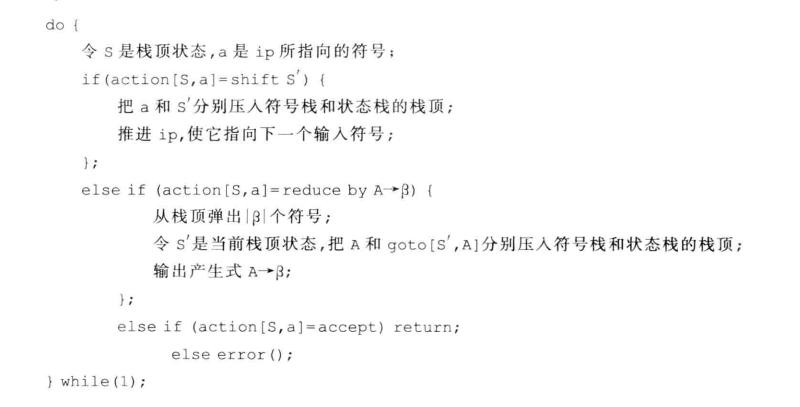
生成分析表：

构造文法G的拓广文法G’ -> 构造G’ 的LR(0)项目集规范族C={I0, I1, … , In} -> 对于状态i（代表项目集Ii）进行分析动作 -> 若go(Ii, A) = Ij, A为非终结符号，则置goto[i, A]=j -> 分析表中但凡不能用前两部规则填入信息的空白表项均置为出错标志error -> 分析程序的初态是包含项目S’ → ·S的有效项目集所对应的状态

分析动作如下：

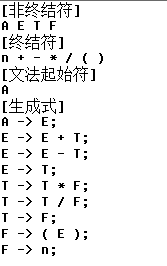


分析程序：

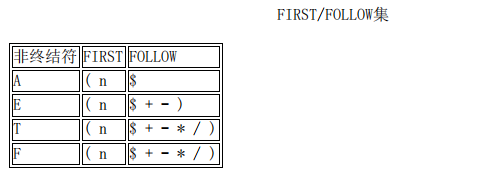


1. **实验内容**

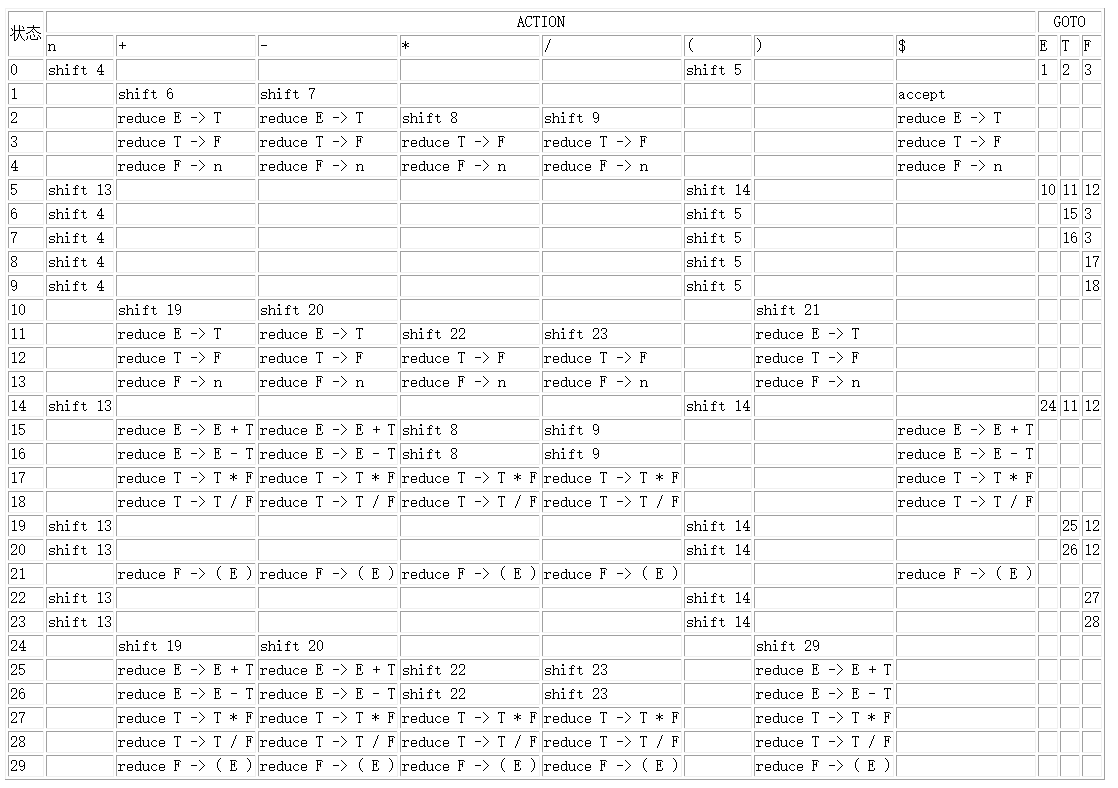
**语法：**

****

**FIRST/FOLLOW集：**



**LR(1)分析表：**

****

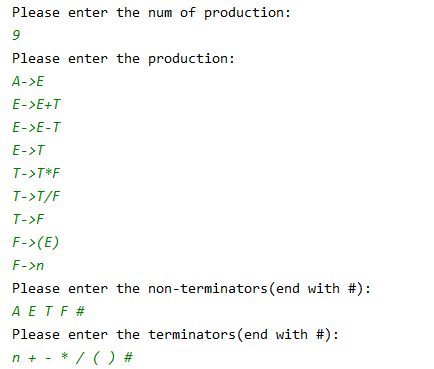
****

**LR(1)状态机：**

****

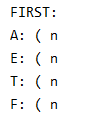
1. **实验结果与分析**

**输入：**

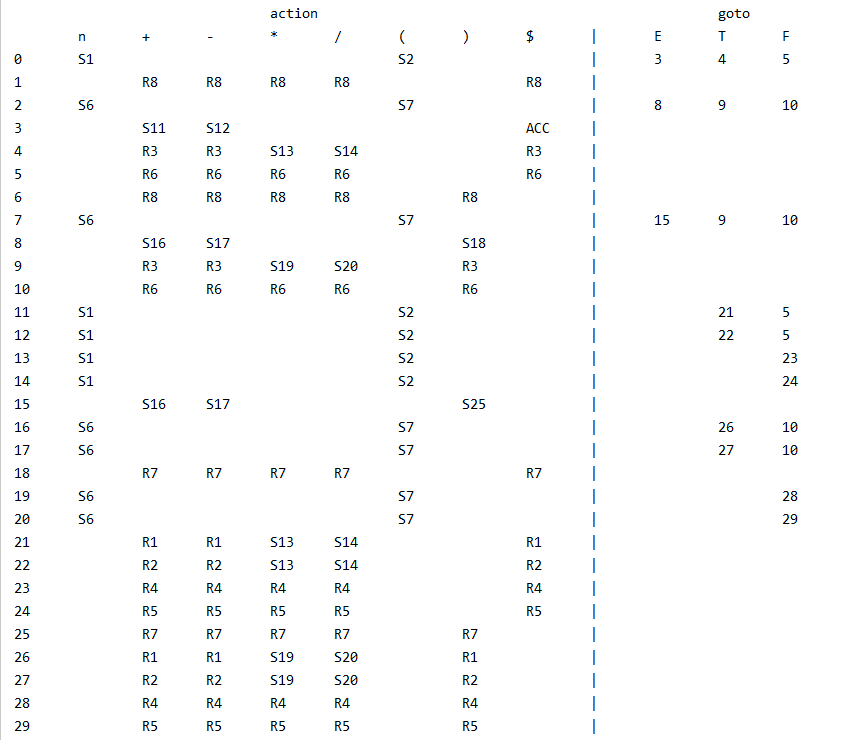
****

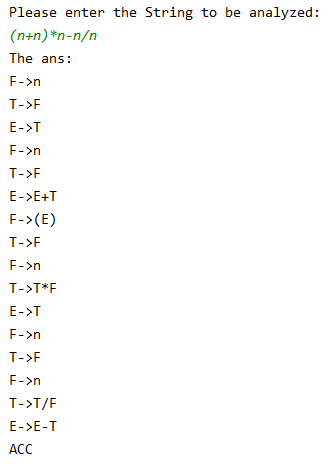
**输出：**

FIRST/FOLLOW集、项目集规范族、LR(1)分析表







****

LR输出的是最右推导的逆序列，经验证，程序自动构建的有效项目集规范族和DFA均正确，其分析表亦正确，对给定的输出串分析输出的产生式验证也正确。

1. **实验中遇到的困难与解决办法**

实验考察了我们对于语法分析方法的理解，实现的困难在于本身对于理论课程所讲内容理解不够到位，导致开始实验时对于整个语法分析程序的实现流程没有一个整体的印象。因此，在开始写代码之前花费了很多时间重新学习教材上响应内容。另外，通过编译工作台生成的分析表处理较为费力，为了加深对于语法分析的理解和方便程序实现，我选择了从头到尾实现对输入的LR(1)文法自动构建FIRST集，自动构建有效项目集规范族和DFA，自动构建LR(1)分析表，执行分析程序输入串，输出所采用的产生式整个过程，获得预期的结果，加深了对于语法分析的理解。