華南師範大學

本科学生实验（实践）报告

院　　系：计 算 机 学 院

实验课程：编译原理

实验项目：LR(1)分析生成器

指导老师：黄煜廉

开课时间：2024 ～ 2025 年度第 1 学期

专　　业：网络工程

班　　级：网工二班

学　　生：肖翔

学　　号：20222132002

华南师范大学教务处

华 南 师 范 大 学 实 验 报 告

学生姓名 肖翔 **学　号** 20222132002

专　　业 网络工程 **年级、班级** 2022 级网工二班

课程名称 编译原理 **实验项目** LR(1)分析生成器

实验时间 2024 年 12 月 8 日

实验指导老师 黄煜廉 **实验评分**

一、实验内容

1.必做功能：

(1)要提供一个文法输入编辑界面，让用户输入文法规则（可保存、打开存有文法规则的文件）

(2)求出文法各非终结符号的 first 集合与 follow 集合，并提供窗口以便用户可以查看这些集合结果。【可以采用表格的形式呈现】

(3)需要提供窗口以便用户可以查看文法对应的 LR(0)DFA 图。(可以用画图的方式呈现，也可用表格方式呈现该图点与边数据）

(4)判断该文法是否为 SLR(1)文法。(应提供窗口呈现判断的结果，如果不是 SLR（1）文法，需要在窗口中显示其原因）

(5)需要提供窗口以便用户可以查看文法对应的 LR(1)DFA 图。(可以用画图的方式呈现，也可用表格方式呈现该图点与边数据）

(6)需要提供窗口以便用户可以查看文法对应的 LR(1)分析表。【LR(1)分析表采用表格的形式呈现】

(7)应该书写完善的软件文档

(8)应用程序应为 Windows 界面。

2.选做功能。

(1)需要提供窗口以便用户输入需要分析的句子。

(2)需要提供窗口以便用户查看使用 LR(1)分析该句子的过程。【可以使用表格的形式逐行显示分析过程】

二、实验目的

（1）要提供一个文法规则编辑的界面，以让用户输入文法规则（可输入，可保存、可打开源程序)。

（2)可由用户选择是否生成对应语法规则的 First 集、Follow 集、LR(0)DFA 图、LR(1)DFA 图、LR(1)分析表。

（3）实验 4 的实现只能选用的程序设计语言为：C++。

（4）要求应用程序的操作界面应为 Windows 界面。

（5）应该书写完善的软件文档。

三、实验文档：

（一）系统概述

1. 系统结构

系统分为 7 个模块：文法规则输入处理模块、求 First 集合模块、求 Follow 集合模块、生成 LR(0)DFA 图模块、生成 LR(1)DFA 图模块、生成 LR(1)分析表模块以及分析是否为

华 南 师 范 大 学 实 验 报 告

学生姓名 肖翔 学　号 20222132002

专　业 网络工程 年级、班级 2022 级网工二班

课程名称 编译原理 实验项目 LR(1)分析生成器

实验时间 2024 年 12 月 8 日

实验指导老师 黄煜廉 实验评分

SLR(1)文法模块。

2. 数据结构的选择

```
unordered_map<string, set<vector<string>>> grammarMap2;

// 文法unit
struct grammarUnit
{
    int gid;
    string left;
    vector<string> right;
    grammarUnit(string l, vector<string> r)
    {
        left = l;
        right = r;
    }
};

// 文法数组
deque<grammarUnit> grammarDeque;

// 文法查找下标
map<pair<string, string>, int> grammarToInt;

// 用于存储所有非终结符的vector
vector<string> nonTerminals;
// 用于存储所有终结符的vector
vector<string> terminals;
```

图 1 系统数据结构（1）

```
// First集合单元
struct firstUnit
{
    set<string> s;
    bool isEpsilon = false;
};

// 非终结符的First集合
map<string, firstUnit> firstSets;

struct followUnit
{
    set<string> s;
};

// 非终结符的Follow集合
map<string, followUnit> followSets;
```

图 2 系统数据结构（2）

华 南 师 范 大 学 实 验 报 告

学生姓名 肖翔 学　号 20222132002

专　　业 网络工程 年级、班级 2022 级网工二班

课程名称 编译原理 实验项目 LR(1)分析生成器

实验时间 2024 年 12 月 8 日

实验指导老师 黄煜廉 实验评分

```
// DFA表每一项项目的结构
struct dfaCell
{
    int cellid; // 这一项的编号，便于后续判断状态相同
    int gid; // 文法编号
    int index = 0; // .在第几位，如i=3, xxx.x，i=0, .xxxx, i=4, xxxx.
};
// 用于通过编号快速找到对应结构
vector<dfaCell> dfaCellVector;
struct nextStateUnit
{
    string c; // 通过什么字符进入这个状态
    int sid; // 下一个状态id是什么
};
// DFA表状态
struct dfaState
{
    int sid; // 状态id
    vector<int> originV;    // 未闭包前的cell
    vector<int> cellV;  // 存储这个状态的cellid
    bool isEnd = false; // 是否为规约状态
    vector<nextStateUnit> nextStateVector; // 下一个状态集合
    set<string> right_VNs; // 判断是否已经处理过这个非终结符
};
```

图 3 系统数据结构（3）

华 南 师 范 大 学 实 验 报 告

学生姓名 肖翔 学　号 20222132002

专　业 网络工程 年级、班级 2022 级网工二班

课程名称 编译原理 实验项目 LR(1)分析生成器

实验时间 2024 年 12 月 8 日

实验指导老师 黄煜廉 实验评分

```
// LR(1)项目结构
struct LR1Item
{
    int gid;  // 文法编号
    int index; //.在第几位，如i=3, xxx.x，i=0,.xxxx, i=4, xxxx.
    string lookahead; // 向前看符号（Lookahead符号）
};
// 状态编号
int scntLR1 = 0;
// 项目编号
int ccntLR1 = 0;
// DFA表每一项项目的结构
struct dfaCellLR1
{
    int cellid; // 这一项的编号，便于后续判断状态相同
    LR1Item item; // LR(1)项目信息
};
// 用于通过编号快速找到对应结构
vector<dfaCellLR1> dfaCellVectorLR1;
struct nextStateUnitLR1
{
    string c; // 通过什么字符进入这个状态
    int sid; // 下一个状态id是什么
};
// DFA表状态
struct dfaStateLR1
{
    int sid; // 状态id
    vector<int> originV;    // 未闭包前的cell
    vector<int> cellV;  // 存储这个状态的cellid
    bool isEnd = false; // 是否为规约状态
    vector<nextStateUnitLR1> nextStateVector; // 下一个状态集合
    set<string> right_VNs; // 判断是否已经处理过这个非终结符
};
```

图 4 系统数据结构（4）

```
// LR(1)分析表的单元结构，包含动作（action）和去向（goto）
struct LR1TableUnit
{
    map<string, string> action;
    map<string, string> goTo;
};

// 用于存储LR(1)分析表，以状态编号为索引
vector<LR1TableUnit> LR1Table;
```

图 5 系统数据结构（5）

本系统主要使用了结构体 struct、向量 vector、集合 set、映射 map、队列 queue 等数据

华 南 师 范 大 学 实 验 报 告

学生姓名 肖翔 **学　号** 20222132002

专　　业 网络工程 **年级、班级** 2022 级网工二班

课程名称 编译原理 **实验项目** LR(1)分析生成器

实验时间 2024 年 12 月 8 日

实验指导老师 黄煜廉 **实验评分**

结构。以下是对本系统部分数据结构的详细介绍。

（1）grammarMap2 用于表示不同终结符对应的文法规则。比如对于以下文法规则，grammarMap2[“Start”]这个集合将包含两个向量[“TESTA”]和[“TESTB”]，这意味着，非终结符 Start 对应的规则右部是 TESTA 和 TESTB。

Start -> TESTA

Start -> TESTB

TESTA -> a

TESTB -> b

（2）grammarUnit 用于表示单条文法规则，其中的 gid 表示当前文法规则的编号，left 表示文法规则的左部，right 表示文法规则的右部。

（3）grammarDequeue 是一个存储 grammarUnit 类型数据的队列，它为 grammarToInt 服务。

（4）grammartoInt 用于查找文法规则下标。通过遍历 grammarDequeue 建立文法单元到编号的映射，即为文法规则生成对应的下标。

（5）firstSets 和 followSets 分别表示非终结字符对应的 first 集和 follow 集，firstUnit 中的 isEpsilon 则表示非终结符对应的 first 集中是否含有空字符串（‘@’）。

（6）dfaCell 用于表示 LR(0)DFA 图状态中的项目，比如某状态中的 term->term .mulop factor 就表示一个项目，他可以用 dfaCell 表示。gid 是文法编号（grammar id）。它主要用于关联文法产生式和 DFA 中的项目。cellid 是 dfaCell 自身的编号。它用于在 dfaCellVector（存储所有 dfaCell 的向量）中唯一标识一个 dfaCell 结构。

假设存在两个不同的 dfaCell 结构，它们可能对应相同的文法产生式（即 gid 相同），但它们在 DFA 中的位置或者其他属性可能不同，此时它们的 cellid 是不同的。

比如，对于文法产生式 A -> aB，在 DFA 构建过程中，可能在不同的状态中有两个项目都和这个产生式相关，一个是 A ->.aB，另一个是 A -> a.B。这两个项目的 gid 相同（因为都对应文法产生式 A -> aB），但它们的 cellid 不同，用于在 dfaCellVector 中区分这两个不同的项目。

（7）dfaState 表示 LR(0)DFA 图的状态。sid 用于唯一标识一个 DFA 的状态。在整个 LR(0)分析过程中，每个状态都有一个不同的编号，方便在构建和遍历 DFA 图时进行区分和引用。例如，在后续生成状态转移关系以及判断状态是否重复等操作中，通过这个 id 来准确地操作对应的状态。

originV 这个向量存储的是在求闭包操作之前，该状态所包含的项目（用 dfaCell 表示）的编号信息。在构建 DFA 状态的过程中，最初放入的项目编号会先记录在这里， originV 保留了这个初始的 “底子”，便于一些对比和处理操作，比如判断状态是否重复时会用到其初始状态下项目编号的情况。

cellV 存储当前状态所包含的所有项目编号，但它是在整个状态构建完成（包括经过闭包操作等一系列处理后）最终的项目编号集合。可以理解为是这个状态所涵盖的所有 LR(0)

华 南 师 范 大 学 实 验 报 告

学生姓名 肖翔 学 号 20222132002

专 业 网络工程 年级、班级 2022 级网工二班

课程名称 编译原理 实验项目 LR(1)分析生成器

实验时间 2024 年 12 月 8 日

实验指导老师 黄煜廉 实验评分

项目的一个汇总表示，通过这些项目编号能够在 dfaCellVector（存储所有 dfaCell 的向量）中找到对应的具体项目内容。

isEnd 用于指示这个状态是否为规约状态，即当前状态是否含有规约项。

nextStateVector 用于存储从当前状态出发，通过不同字符可以转移到的下一个状态的相关信息。每个 nextStateUnit 包含了转移所使用的字符以及目标状态的 id，这样就完整地描述了当前状态的状态转移关系，从而构建起整个 DFA 的状态转移图结构。

right_VNs 用于存储某一状态内已经生成过闭包的非终结符。避免对同一个非终结符重复生成闭包。

（8）生成 LR(1)DFA 图所用到的结构和 LR(0)的类似，不同的是 LR(1)的项目结构多了一个属性，即 lookahead，用于存储当前项目的向前搜索符。

（9）LR1TableUnit 是 LR(1)分析表的单元结构，包含动作（action）和去向（goto）。动作，以终结符为键，对应动作字符串（如移进 "s<状态编号>"、规约 "r<文法编号>"、接受 "ACCEPT" 等）为值。去向，以非终结符为键，对应状态编号为值（对于非终结符的转移情况）。

（二）实验过程

1. 求 First 集

求 First 集的算法采用讲义中提到的算法（图 6）。

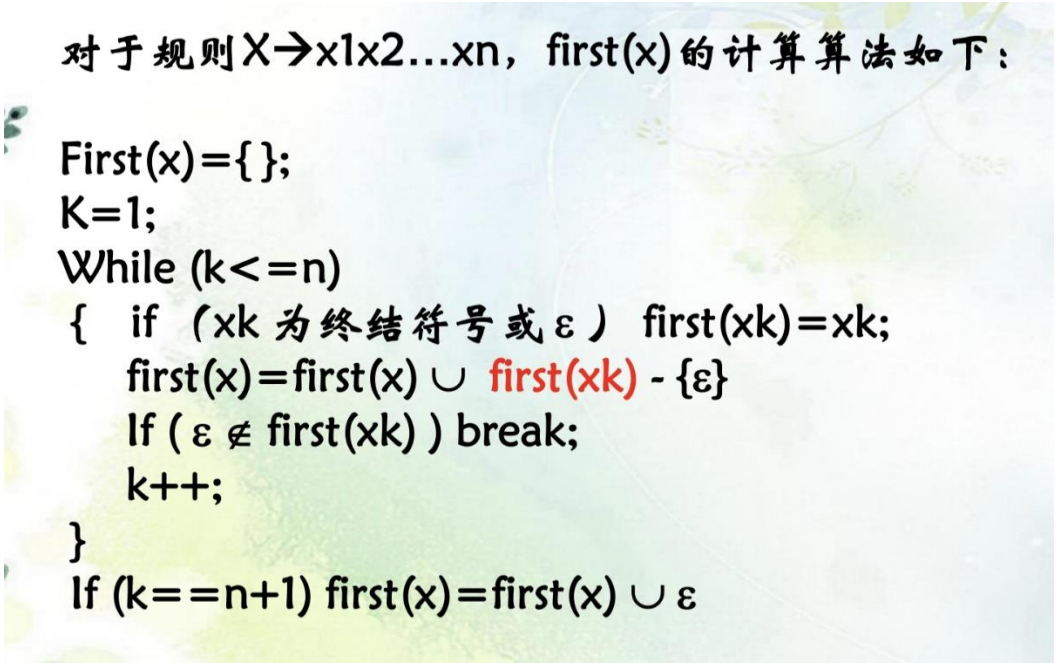

图 6 求 first 集伪代码

具体代码：

华 南 师 范 大 学 实 验 报 告

学生姓名 肖翔 学 号 20222132002

专 业 网络工程 年级、班级 2022 级网工二班

课程名称 编译原理 实验项目 LR(1)分析生成器

实验时间 2024 年 12 月 8 日

实验指导老师 黄煜廉 实验评分

```
bool calculateFirstSets()
{

    bool flag = false;
    for (auto& grammar : grammarMap2)
    {
        string nonTerminal = grammar.first;
        // 保存当前First集合的大小，用于检查是否有变化
        size_t originalSize = firstSets[nonTerminal].s.size();
        bool originalE = firstSets[nonTerminal].isEpsilon;
        for (auto& g : grammar.second)
        {
            int k = 0;
            // 求first集时有针对遍历文法规则循环的break，
            // 只要找到终结符或非终结符中的first集没有epsilon（此非终结符不可能成为epsilon，也就没有后面字符什么事了）就直接break
            // 说明求first集看左边
            while (k <= g.size() - 1)          ⚠comparison of integers of different signs: 'int' and 'unsigned long long
            {
                set<string> first_k;
                if (g[k] == "@")
                {
                    k++;
                    continue;
                }
                else if (isTerminal(g[k]))
                {
                    first_k.insert(g[k]);
                }
                else
                {
                    first_k = firstSets[g[k]].s;
                }
                firstSets[nonTerminal].s.insert(first_k.begin(), first_k.end());
                if (isTerminal(g[k]) || !firstSets[g[k]].isEpsilon)
                {
                    break;
                }
                k++;
            }
```

图 7 求 first 集代码（1）

```
                k++;
            }
            if (k == g.size())    ⚠comparison of integers of different signs: 'int' and 'std::vector::size_t
            {
                firstSets[nonTerminal].isEpsilon = true;
            }
        }
        // 看原始大小和是否变化epsilon，如果变化说明还得重新再来一次
        if (originalSize != firstSets[nonTerminal].s.size() || originalE != firstSets[nonTerminal].isEpsilon)
        {
            flag = true;
        }
    }
    return flag;
}

void getFirstSets()
{
    // 不停迭代，直到First集合不再变化
    bool flag = false;
    do
    {
        flag = calculateFirstSets();
    } while (flag);
}
```

图 8 求 first 集代码（2）

此代码的核心部分是遍历产生式右部以计算 First 集合，即 for (auto& g : grammar.second)的复合语句。对于当前非终结符的每个产生式右部 g 进行遍历，然后通

华 南 师 范 大 学 实 验 报 告

学生姓名 肖翔 学 号 20222132002

专 业 网络工程 年级、班级 2022 级网工二班

课程名称 编译原理 实验项目 LR(1)分析生成器

实验时间 2024 年 12 月 8 日

实验指导老师 黄煜廉 实验评分

过一个循环遍历产生式右部 g 的每个字符或字符串（因为非终结符有可能是以单词的形式出现），索引为 k。对于每个字符或字符串 g[k]，分情况处理：

①如果字符是'@'（表示空字符串），则跳过该字符，继续处理下一个字符；

②如果是终结符（通过 isTerminal 函数判断），则将该终结符插入到一个临时的 First 集合 first_k 中；

③如果是非终结符，则获取该非终结符已有的 First 集合（first_k = firstSets[g[k]].s;）。如果当前字符是终结符或者当前字符所对应的非终结符的 First 集合中不包含空字符串，那么就可以跳出当前对产生式右部的遍历循环（因为后面的字符对当前非终结符的 First 集合计算已经没有影响了）。假设有文法规则 A -> BC，非终结符 A 的文法规则中含非终结符 B 打头，如果 B 含有空串，则不能跳过，需要扫描 B 后面的字符 C，将字符 C 的 first 集合加入到 A 的 first 集合当中。

getFirstSets 函数的目的是通过不断调用 calculateFirstSets 函数来迭代计算每个非终结符的 First 集合，直到所有非终结符的 First 集合都不再发生变化为止。

2. 求 Follow 集

求 Follow 集的算法采用讲义中提到的算法（图 9）。

图 9 求 follow 集伪代码

具体代码：

华 南 师 范 大 学 实 验 报 告

学生姓名 肖翔 学 号 20222132002

专 业 网络工程 年级、班级 2022 级网工二班

课程名称 编译原理 实验项目 LR(1)分析生成器

实验时间 2024 年 12 月 8 日

实验指导老师 黄煜廉 实验评分

```
bool calculateFollowSets()
{
    bool flag = false;
    for (auto& grammar : grammarMap2)
    {
        string nonTerminal = grammar.first;

        for (auto& g : grammar.second)
        {
            // 在求follow集时，没有针对遍历文法规则循环的break，说明求follow集看右边
            for (int i = 0; i < g.size(); ++i)    ⚠comparison of integers of different signs: 'int' and 'std::vector::size…
            {
                if (isTerminal(g[i]) || g[i] == "@")
                {
                    continue;  // 跳过终结符
                }
                set<string> follow_k;
                size_t originalSize = followSets[g[i]].s.size();

                if (i == g.size() - 1)          ⚠comparison of integers of different signs: 'int' and 'unsigned long long'
                {
                    // Case A: A -> αB, add Follow(A) to Follow(B)
                    follow_k.insert(followSets[nonTerminal].s.begin(), followSets[nonTerminal].s.end());
                }
                else
                {
                    // Case B: A -> αBβ
                    int j = i + 1;
                    while (j < g.size())    ⚠comparison of integers of different signs: 'int' and 'std::vector::size_type'…
                    {
                        if (isTerminal(g[j]))
                        {   // 终结符直接加入并跳出
                            follow_k.insert(g[j]);
                            break;
                        }
                        else
                        {   // 非终结符加入first集合
                            set<string> first_beta = firstSets[g[j]].s;
                            // A的follow集包含B的first集
                            follow_k.insert(first_beta.begin(), first_beta.end());
                            if (!firstSets[g[j]].isEpsilon)
```

图 10 求 follow 集代码（1）

```
                            follow_k.insert(first_beta.begin(), first_beta.end());
                            if (!firstSets[g[j]].isEpsilon)
                            {
                                break;
                            }

                            ++j;
                        }
                    }
                    // If β is ε or β is all nullable, add Follow(A) to Follow(B)
                    if (j == g.size())    ⚠comparison of integers of different signs: 'int' and 'std::vector::size_type' (…
                    {
                        // A的follow集包含S的follow集
                        follow_k.insert(followSets[nonTerminal].s.begin(), followSets[nonTerminal].s.end());
                    }
                }
                addToFollow(g[i], follow_k);
                // 检查是否变化
                if (originalSize != followSets[g[i]].s.size())
                {
                    flag = true;
                }
            }
        }
    }
    return flag;
}

void getFollowSets()
{
    // 开始符号加入$
    addToFollow(trueStartSymbol, { "$" });
    addToFollow(startSymbol, { "$" });

    // 不停迭代，直到Follow集合不再变化
    bool flag = false;
    do
    {
        flag = calculateFollowSets();
    } while (flag);
}/**/
```

华 南 师 范 大 学 实 验 报 告

学生姓名 肖翔 **学　号** 20222132002

专　业 网络工程 **年级、班级** 2022 级网工二班

课程名称 编译原理 **实验项目** LR(1)分析生成器

实验时间 2024 年 12 月 8 日

实验指导老师 黄煜廉 **实验评分**

图 11 求 follow 集代码（2）

此代码的核心部分是遍历产生式右部以计算 Follow 集合。

for (auto& g : grammar.second)：对于当前非终结符的每个产生式右部 g 进行遍历。

for (int i = 0; i < g.size(); ++i)：通过一个内层循环遍历产生式右部 g 的每个字符，索引为 i。对于每个字符 g[i]，分情况处理：

① 如果字符是终结符（通过 isTerminal 函数判断）或者是空字符串（'@'），则跳过该字符，继续处理下一个字符。

② 如果当前字符是产生式右部的最后一个字符（即 i == g.size() - 1），则进入 Case A 情况：将当前非终结符 nonTerminal 的 Follow 集合中的所有字符插入到一个临时的 Follow 集合 follow_k 中。这是因为在产生式 A -> αB（这里 B 对应 g[i]）的情况下，B 的 Follow 集合应该包含 A 的 Follow 集合。

③ 如果当前字符不是产生式右部的最后一个字符，则进入 Case B 情况：首先定义 int j = i + 1;，用于从当前字符的下一个字符开始遍历。通过一个内层循环遍历当前字符后面的字符序列。对于每个后续字符 g[j]，分情况处理：

（1）如果是终结符，直接将该终结符插入到 follow_k 中，并跳出内层循环（因为后面的字符对当前处理已经没有影响了）。

（2）如果是非终结符，首先获取该非终结符的 First 集合（set<char> first_beta = firstSets[g[j]].s;），然后将其插入到 follow_k 中。接着，如果该非终结符的 First 集合中不包含空字符串（if (!firstSets[g[j]].isEpsilon)），则跳出内层循环(因为后面的字符对当前处理已经没有影响了)。否则,继续遍历下一个字符(++j;)。换言之，如果有空串在 g[j]的 first 集合中，g[j]就有成为空串的可能，此时相当于 g[i]后面跟着的是 g[j]后面的字符，当然 g[j]后面也可能没有字符，此时 g[i]相当于最后一个字符。无论那种情况都要++j，读取 g[j]后一位。如果 g[j]后面有字符就会继续循环（符合 j < g.size()的循环条件），将 g[j]后一个字符（即++j 后得到的 g[j]字符）的 first 集合加进 g[i]的 follow。如果 g[j]后面没有字符，那么++j 后，j 等于文法规则的长度，不符合循环条件，退出循环。然后将非终结符的 follow 集加入到 g[i]的 follow 集中。

（3）如果内层循环遍历完整个后续字符序列（即 j == g.size()），说明后续字符序列可能产生空字符串或者全部可空，此时将当前非终结符 nonTerminal 的 Follow 集合中的所有字符插入到 follow_k 中。这是因为在产生式 A -> αBβ（这里 B 对应 g[i]）的情况下，如果 β 可空，那么 B 的 Follow 集合应该包含 A 的 Follow 集合。

3. 生成 LR(0)DFA 图

华 南 师 范 大 学 实 验 报 告

学生姓名 肖翔 学　号 20222132002

专　业 网络工程 年级、班级 2022 级网工二班

课程名称 编译原理 实验项目 LR(1)分析生成器

实验时间 2024 年 12 月 8 日

实验指导老师 黄煜廉 实验评分

```
// 判断是不是新结构
int isNewCell(int gid, int index)
{
    for (const dfaCell& cell : dfaCellVector)
    {
        // 检查dfaCellVector中是否存在相同的gid和index的dfaCell
        if (cell.gid == gid && cell.index == index)
        {
            return cell.cellid; // 不是新结构
        }
    }
    return -1; // 是新结构
}

// 判断是不是新状态
int isNewState(const vector<int>& cellIds)
{
    for (const dfaState& state : dfaStateVector)
    {
        // 检查状态中的originV是否相同
        if (state.originV.size() == cellIds.size() &&
            equal(state.originV.begin(), state.originV.end(), cellIds.begin()))
        {
            return state.sid; // 不是新状态
        }
    }

    return -1; // 是新状态
}
```

图 11 求 LR(0)DFA 图的辅助函数

isNewCell 函数的目的是判断给定的文法编号 gid 和点（.）的位置索引 index 所表示的 dfaCell 项目是否是新出现的。通过遍历已有的 dfaCellVector，如果找到了相同 gid 和 index 的项目，就返回其已有的编号，说明不是新结构；如果遍历完都没找到，则返回-1，表示是新结构，后续需要创建新的 dfaCell 并添加到 dfaCellVector 中。

isNewState 函数用于判断给定的项目编号集合（通过 cellIds 表示）所代表的状态是否是新的。通过遍历已有的 dfaStateVector，对比每个状态的 originV（未闭包前的项目编号集合）与传入的 cellIds 是否完全相同，如果相同则返回该已有状态的 sid（编号），说明不是新状态；若遍历完都没有匹配的，则返回-1，意味着是新状态，后续要创建新的 dfaState 并添加到 dfaStateVector 中。

华 南 师 范 大 学 实 验 报 告

学生姓名 肖翔 学　号 20222132002

专　　业 网络工程 年级、班级 2022 级网工二班

课程名称 编译原理 实验项目 LR(1)分析生成器

实验时间 2024 年 12 月 8 日

实验指导老师 黄煜廉 实验评分

```
// DFS深度优先标记数组
set<int> visitedStates;

// 创建LR0的初始状态
void createFirstState()
{
    // 由于增广，一定会只有一个入口
    dfaState zero = dfaState();
    zero.sid = scnt++; // 给他一个id
    dfaStateVector.push_back(zero); // 放入数组中

    // 添加初始的LR0项，即E' -> .S
    dfaCell startCell;
    startCell.gid = 0; // 这里假设增广文法的编号为0
    startCell.index = 0;
    startCell.cellid = ccnt++;

    dfaCellVector.push_back(startCell);

    // 把初始LR0项放入初始状态
    dfaStateVector[0].cellV.push_back(startCell.cellid);
    dfaStateVector[0].originV.push_back(startCell.cellid);
}
```

图 12 求 LR(0)DFA 图的第一个状态

这个函数用于创建 LR(0)的初始状态。首先创建一个空的 dfaState 结构体实例 zero，并给它分配一个唯一的状态编号（通过 scnt++实现），然后将其添加到 dfaStateVector 中。接着创建一个代表初始 LR(0)项目（通常是增广后的开始符号产生式，这里假设为 E’ ->.S，对应的文法编号设为 0，点在最开始位置，索引为 0）的 dfaCell 结构体实例 startCell，给它分配一个项目编号（通过 ccnt++），并添加到 dfaCellVector 中。最后将这个初始项目的编号添加到刚创建的初始状态 dfaStateVector[0]的 cellV 和 originV 向量中，完成初始状态的构建，它是整个 DFA 构建的起点。

华 南 师 范 大 学 实 验 报 告

学生姓名 肖翔 学 号 20222132002

专 业 网络工程 年级、班级 2022 级网工二班

课程名称 编译原理 实验项目 LR(1)分析生成器

实验时间 2024 年 12 月 8 日

实验指导老师 黄煜廉 实验评分

```
void generateLR0State(int stateId)
{
    // DFS,如果走过就不走了
    if (visitedStates.count(stateId) > 0) {
        return;
    }

    // 标记走过了
    visitedStates.insert(stateId);

    qDebug() << stateId << endl;

    // 求闭包
    for (int i = 0; i < dfaStateVector[stateId].cellV.size(); ++i)    comparison of integers of differ…
    {
        dfaCell& currentCell = dfaCellVector[dfaStateVector[stateId].cellV[i]];
        // 如果点号在产生式末尾或者空串，则跳过（LR0不需要结束）
        if (currentCell.index == grammarDeque[currentCell.gid].right.size() ||    comparison of intege…
                vectorToString(grammarDeque[currentCell.gid].right) == "@")
        {
            dfaStateVector[stateId].isEnd = true;
            continue;
        }
        string nextSymbol = grammarDeque[currentCell.gid].right[currentCell.index];
        // 如果nextSymbol是非终结符，则将新项添加到状态中
        if (isNonTerminal(nextSymbol) && dfaStateVector[stateId].right_VNs.find(nextSymbol) ==
                dfaStateVector[stateId].right_VNs.end())
        {
            dfaStateVector[stateId].right_VNs.insert(nextSymbol);
            for (auto& grammar : grammarMap2[nextSymbol])
            {
                // 获取通过nextSymbol转移的新LR0项
                dfaCell nextCell = dfaCell();
                nextCell.gid = grammarToInt[make_pair(nextSymbol,vectorToString(grammar))];
                nextCell.index = 0;
                int nextcellid = isNewCell(nextCell.gid, nextCell.index);
```

图 13 求 LR(0)DFA 图核心代码（1）

```
                int nextcellid = isNewCell(nextCell.gid, nextCell.index);
                if (nextcellid == -1)
                {
                    nextCell.cellid = ccnt++;
                    dfaCellVector.push_back(nextCell);
                    dfaStateVector[stateId].cellV.push_back(nextCell.cellid);
                }
                else dfaStateVector[stateId].cellV.push_back(nextcellid);
            }
        }
    }
    map<string,dfaState> tempSave;// 暂存新状态
    // 生成新状态，但还不能直接存到dfaStateVector中，我们要校验他是否和之前的状态一样
    for (int i = 0; i < dfaStateVector[stateId].cellV.size(); ++i)    comparison of integers of differ…
    {
        dfaCell& currentCell = dfaCellVector[dfaStateVector[stateId].cellV[i]];
        // 如果点号在产生式末尾，则跳过（LR0不需要结束）
        if (currentCell.index == grammarDeque[currentCell.gid].right.size() ||    comparison of integ…
                vectorToString(grammarDeque[currentCell.gid].right) == "@")
        {
            continue;
        }
        // 下一个字符
        string nextSymbol = grammarDeque[currentCell.gid].right[currentCell.index];
        // 创建下一个状态（临时的）
        dfaState& nextState = tempSave[nextSymbol];
        dfaCell nextStateCell = dfaCell();
        nextStateCell.gid = currentCell.gid;
        nextStateCell.index = currentCell.index + 1;
        // 看看里面的项目是否有重复的，如果重复拿之前的就好，不重复生成
        int nextStateCellid = isNewCell(nextStateCell.gid, nextStateCell.index);
        if (nextStateCellid == -1)
        {
            nextStateCell.cellid = ccnt++;
            dfaCellVector.push_back(nextStateCell);
        }
```

图 14 求 LR(0)DFA 图核心代码（2）

华 南 师 范 大 学 实 验 报 告

学生姓名 肖翔 学　号 20222132002

专　业 网络工程 年级、班级 2022 级网工二班

课程名称 编译原理 实验项目 LR(1)分析生成器

实验时间 2024 年 12 月 8 日

实验指导老师 黄煜廉 实验评分

```
        else nextStateCell.cellid = nextStateCellid;
        nextState.cellV.push_back(nextStateCell.cellid);
        nextState.originV.push_back(nextStateCell.cellid);
        // 收集一下，方便后面画表
        if (isNonTerminal(nextSymbol))
        {
            VN.insert(nextSymbol);
        }
        else if (isTerminal(nextSymbol))
        {
            VT.insert(nextSymbol);
        }
    }
    // 校验状态是否有重复的
    for (auto& t : tempSave)
    {
        dfaState nextState = dfaState();
        int newStateId = isNewState(t.second.originV);
        // 不重复就新开一个状态
        if (newStateId == -1)
        {
            nextState.sid = scnt++;
            nextState.cellV = t.second.cellV;
            nextState.originV = t.second.originV;
            dfaStateVector.push_back(nextState);
        }
        // 如果重复那么nextState的sid是已经存在的状态的id
        else nextState.sid = newStateId;
        // 存入现在这个状态的nextStateVector
        // 无论是否重复，当前状态都需要指向 nextState
        // 也就是说重复则指向已有的状态，不重复就指向新建的状态
        nextStateUnit n = nextStateUnit();
        n.sid = nextState.sid;
        n.c = t.first;
        dfaStateVector[stateId].nextStateVector.push_back(n);
    }
```

图 15 求 LR(0)DFA 图核心代码（3）

```
    // 对每个下一个状态进行递归
    int nsize = dfaStateVector[stateId].nextStateVector.size();
    for (int i = 0; i < nsize; i++)
    {
        auto& nextunit = dfaStateVector[stateId].nextStateVector[i];
        generateLR0State(nextunit.sid);
    }
}

// 生成LR0入口
void getLR0()
{
    visitedStates.clear();

    // 首先生成第一个状态
    createFirstState();

    // 递归生成其他状态
    generateLR0State(0);
}
```

图 16 求 LR(0)DFA 图核心代码（4）

华 南 师 范 大 学 实 验 报 告

学生姓名 肖翔 **学　号** 20222132002

专　业 网络工程 **年级、班级** 2022 级网工二班

课程名称 编译原理 **实验项目** LR(1)分析生成器

实验时间 2024 年 12 月 8 日

实验指导老师 黄煜廉 **实验评分**

generateLR0State 函数是整个 LR(0)状态生成的核心逻辑，通过深度优先搜索（DFS）的方式递归地构建 DFA 的各个状态以及状态之间的转移关系。

此函数的代码逻辑可分为以下几个部分：

①避免重复访问。首先通过 visitedStates 集合来记录已经访问过的状态编号，如果当前传入的 stateId 已经在该集合中，说明这个状态已经处理过了，直接返回，避免重复构建和陷入死循环。否则，将当前状态编号插入 visitedStates 集合，表示已访问。

②求闭包操作。遍历当前状态（由 stateId 指定）的 cellV 向量中的每个项目（dfaCell）。对于每个项目，如果点（.）在产生式末尾或者产生式右部为空串，就将当前状态的 isEnd 标志设为 true，表示当前状态涉及规约操作，然后跳过该项目。如果点后面的下一个符号（通过 grammarDeque[currentCell.gid].right[currentCell.index]获取）是一个非终结符，并且这个非终结符还没有在当前状态的 right_VNs（前面提到过，right_VNs 用于存储某一状态内已经生成过闭包的非终结符。避免对同一个非终结符重复生成闭包。）集合中处理过，那么就需要进行闭包操作。具体做法是从 grammarMap 中获取该非终结符对应的所有产生式，为每个产生式创建一个新的 dfaCell 项目（点在最开始位置，索引为 0），然后通过 isNewCell 函数判断这个项目是否是新的，如果是新的就分配一个新的项目编号（通过 ccnt++），添加到 dfaCellVector 中，并将项目编号添加到当前状态的 cellV 向量中，这样就把相关项目添加到当前状态中，完成闭包扩展。

③生成新状态（临时处理）。接下来，遍历当前状态的 cellV 向量中的项目（再次遍历是为了生成状态转移相关信息），对于点不在产生式末尾的项目，获取其点后面的下一个字符 nextSymbol，以这个字符为键，在 tempSave 映射中获取或创建一个临时的 dfaState（用于暂存新状态信息）。然后创建一个新的 dfaCell 项目，表示点往后移动一位后的情况（通过 nextStateCell 表示，其 gid 不变，index 加 1），同样通过 isNewCell 函数判断是否重复，若不重复则分配新编号并添加到 dfaCellVector 中，然后将这个项目编号添加到临时状态的 cellV 和 originV 向量中。同时，根据 nextSymbol 是终结符还是非终结符，分别将其添加到 VT 或 VN 集合中，方便后续用表格生成 LR(0)的 DFA 图。

④校验并添加新状态。遍历 tempSave 中的每个临时状态，通过 isNewState 函数判断这个临时状态是否与已有的状态重复。如果是新状态，就给它分配一个新的状态编号（通过 scnt++），创建一个新的 dfaState 结构体实例，将临时状态的相关信息复制过来，添加到 dfaStateVector 中；如果是已有的状态，就获取其已有的状态编号。然后创建一个 nextStateUnit 结构体实例，记录从当前状态通过 nextSymbol 字符转移到目标状态（新状态或已有状态）的信息，将其添加到当前状态的 nextStateVector 中，这样就构建好了当前状态到其他状态的转移关系。

⑤递归处理下一个状态。最后，遍历当前状态的 nextStateVector，获取每个下一个状态的相关信息，对每个下一个状态递归调用 generateLR0State 函数，继续构建整个 DFA 的状态图结构，直到所有状态及其转移关系都构建完成。

4. 生成 LR(1)DFA 图

华 南 师 范 大 学 实 验 报 告

学生姓名 肖翔 学　号 20222132002

专　　业 网络工程 年级、班级 2022 级网工二班

课程名称 编译原理 实验项目 LR(1)分析生成器

实验时间 2024 年 12 月 8 日

实验指导老师 黄煜廉 实验评分

```
int isNewCellLR1(int gid, int index, string lookahead)
{
    qDebug() << "generateLR1State: 20";
    for (const dfaCellLR1& cell : dfaCellVectorLR1)
    {

        if (cell.item.gid == gid && cell.item.index == index && cell.item.lookahead == lookahead)
        {
            return cell.cellid; // 不是新结构
        }
    }
    return -1; // 是新结构
}
// 判断是不是新状态（LR(1)，考虑同心项和Lookahead符号）
int isNewStateLR1(const vector<int>& cellIds)
{
    for (const dfaStateLR1& state : dfaStateVectorLR1)
    {
        if (state.originV.size() == cellIds.size() &&
            equal(state.originV.begin(), state.originV.end(), cellIds.begin()))
        {// 进一步检查Lookahead符号是否都相同
            bool sameLookahead = true;
            for (size_t i = 0; i < state.originV.size(); ++i)
            {
                int cellId = state.originV[i];
                int otherCellId = cellIds[i];
                if (dfaCellVectorLR1[cellId].item.lookahead !=
                    dfaCellVectorLR1[otherCellId].item.lookahead)
                {
                    sameLookahead = false;
                    break;
                }
            }
            if (sameLookahead)
            {
                return state.sid; // 不是新状态
            }
        }
    }
    return -1; // 是新状态
}
```

图 17　求 LR(1)DFA 图的辅助函数

在判断 LR(1)的项目和状态是否相同时还要考虑项目和状态中的向前搜索符是否相同。只有两条项目的文法规则编号、点的索引以及向前搜索符完全一致时，才能说明此两条项目在 LR(1)中是相同的。只有两个状态的同心项以及向前搜索符完全一致时，才能说明此两个状态在 LR(1)中是相同的。

华 南 师 范 大 学 实 验 报 告

学生姓名 肖翔 学 号 20222132002

专 业 网络工程 年级、班级 2022 级网工二班

课程名称 编译原理 实验项目 LR(1)分析生成器

实验时间 2024 年 12 月 8 日

实验指导老师 黄煜廉 实验评分

```
// 创建LR(1)的初始状态
void createFirstStateLR1()
{
    qDebug() << "createFirstStateLR1 start";
    // 由于增广，一定会只有一个入口
    dfaStateLR1 zero = dfaStateLR1();
    zero.sid = scntLR1++; // 给他一个id
    dfaStateVectorLR1.push_back(zero); // 放入数组中

    // 获取开始符号对应的文法编号（假设增广后的开始符号相关文法在grammarDeque中第一个位置）
    int startGid = grammarDeque[0].gid;

    // 添加初始的LR(1)项，即E' ->.S,$
    dfaCellLR1 startCell;
    startCell.item.gid = startGid; // 这里假设增广文法的编号为0
    startCell.item.index = 0;
    startCell.item.lookahead = "$";
    startCell.cellid = ccntLR1++;

    dfaCellVectorLR1.push_back(startCell);

    // 把初始LR(1)项放入初始状态
    dfaStateVectorLR1[0].cellV.push_back(startCell.cellid);
    dfaStateVectorLR1[0].originV.push_back(startCell.cellid);
}
```

图 18 求 LR(1)DFA 图核心代码（1）

```
void generateLR1State(int stateId)
{
    // DFS,如果走过就不走了
    if (visitedStatesLR1.count(stateId) > 0) {
        return;
    }

    // 标记走过了
    visitedStatesLR1.insert(stateId);
    // 求闭包
    for (int i = 0; i < dfaStateVectorLR1[stateId].cellV.size(); ++i)    △comparison of integers of different signs: 'int'…
    {
        getFirstSets();
        getFollowSets();
        dfaCellLR1& currentCell = dfaCellVectorLR1[dfaStateVectorLR1[stateId].cellV[i]];
        // 如果点号在产生式末尾或者空串，则跳过（LR1不需要结束）
        vector<string> grammarRight = grammarDeque[currentCell.item.gid].right;
        if (currentCell.item.index == grammarRight.size() ||     △comparison of integers of different signs: 'int' and 'st…
                find(grammarRight.begin(), grammarRight.end(), "@") != grammarRight.end())
        {
            dfaStateVectorLR1[stateId].isEnd = true;
            continue;
        }
        string nextSymbol = grammarRight[currentCell.item.index];
        string lookahead = "";
        // 如果点后的字符（串）是非终结符且是文法规则的最后一个字符（串），那么下一条项目的lookahead就是当前项目左部的follow集
        if(isNonTerminal(nextSymbol) && currentCell.item.index + 1 >= grammarRight.size()){    △comparison of integers of …
            for(string follow: followSets[grammarDeque[currentCell.item.gid].left].s){    △Missing reference in range-for …
                lookahead += follow;
                lookahead += "/";
            }
            if(lookahead.length()){
                lookahead.pop_back();
            }
        }
        else if(isTerminal(nextSymbol)){
            continue;
        }
        else{
            // 查看点后面字符（串）的后一个字符（串）
            string nextNextSymbol = grammarRight[currentCell.item.index + 1];
```

图 19 求 LR(1)DFA 图核心代码（2）

华 南 师 范 大 学 实 验 报 告

学生姓名 肖翔 学 号 20222132002

专 业 网络工程 年级、班级 2022 级网工二班

课程名称 编译原理 实验项目 LR(1)分析生成器

实验时间 2024 年 12 月 8 日

实验指导老师 黄煜廉 实验评分

```
            if(nextNextSymbol == "@"){
                // 合法的文法规则右部若有多个终结符或非终结符，那么右部最后一个字符不可能是空字符串，因此可以把空串的后一个字符（串）作为点后面字
                nextNextSymbol = grammarDeque[currentCell.item.gid].right[currentCell.item.index + 2];
            }
            else if(isTerminal(nextNextSymbol)){
                // 如果点后面字符（串）的后一个字符（串）是终结符，则将其作为下一条项目的lookahead
                lookahead = nextNextSymbol;
            }
            else if(isNonTerminal(nextNextSymbol)){
                // 如果点后面字符（串）的后一个字符（串）是非终结符，则将此非终结符的first集作为下一条项目的lookahead
                for(string first: firstSets[nextNextSymbol].s){    ⚠Missing reference in range-for with non trivial type (…
                    lookahead += first;
                    lookahead += "/";
                }
                if(lookahead.length()){
                    lookahead.pop_back();
                }
            }
        }

        // 如果nextSymbol是非终结符，则将新项添加到状态中
        if (isNonTerminal(nextSymbol) && dfaStateVectorLR1[stateId].right_VNs.find(nextSymbol) ==
                dfaStateVectorLR1[stateId].right_VNs.end())
        {
            dfaStateVectorLR1[stateId].right_VNs.insert(nextSymbol);
            for (auto& grammar : grammarMap2[nextSymbol])
            {
                // 获取通过nextSymbol转移的新LR0项
                dfaCellLR1 nextCell = dfaCellLR1();
                nextCell.item.gid = grammarToInt[make_pair(nextSymbol,vectorToString(grammar))];
                nextCell.item.index = 0;
                int nextcellid = isNewCellLR1(nextCell.item.gid, nextCell.item.index, lookahead);
                if (nextcellid == -1)
                {
                    nextCell.cellid = ccntLR1++;
                    nextCell.item.lookahead = lookahead;
                    dfaCellVectorLR1.push_back(nextCell);
                    dfaStateVectorLR1[stateId].cellV.push_back(nextCell.cellid);
                }
                else dfaStateVectorLR1[stateId].cellV.push_back(nextcellid);
            }
```

图 20 求 LR(1)DFA 图核心代码（3）

```
    }
}

// 暂存新状态（通过哪一个字符（串）进入暂存的新状态）
map<string, dfaStateLR1> tempSave;
// 生成新状态，但还不能直接存到dfaStateVector中，我们要校验他是否和之前的状态一样
qDebug() << "cellV size: " << dfaStateVectorLR1[stateId].cellV.size();
for (int i = 0; i < dfaStateVectorLR1[stateId].cellV.size(); ++i)    ⚠comparison of integers of different signs: 'int'…
{
    dfaCellLR1& currentCell = dfaCellVectorLR1[dfaStateVectorLR1[stateId].cellV[i]];
    vector<string> grammarRight = grammarDeque[currentCell.item.gid].right;
    // 如果点号在产生式末尾，则跳过（LR1不需要结束）
    if (currentCell.item.index == grammarRight.size() || find(grammarRight.begin(), grammarRight.end(), "@") != grammarR
    {
        continue;
    }
    // 下一个字符
    string nextSymbol = grammarDeque[currentCell.item.gid].right[currentCell.item.index];
    // 创建下一个状态（临时的）
    dfaStateLR1& nextState = tempSave[nextSymbol];
    dfaCellLR1 nextStateCell = dfaCellLR1();
    nextStateCell.item.gid = currentCell.item.gid;
    nextStateCell.item.index = currentCell.item.index + 1;
    nextStateCell.item.lookahead = currentCell.item.lookahead;

    // 看看里面的项目是否有重复的，如果重复拿之前的就好，不重复生成
    int nextStateCellid = isNewCellLR1(nextStateCell.item.gid, nextStateCell.item.index, nextStateCell.item.lookahead);
    if (nextStateCellid == -1)
    {
        nextStateCell.cellid = ccntLR1++;
        dfaCellVectorLR1.push_back(nextStateCell);
    }
    else nextStateCell.cellid = nextStateCellid;
    nextState.cellV.push_back(nextStateCell.cellid);
    nextState.originV.push_back(nextStateCell.cellid);

    // 收集一下，方便后面画表
    if (isNonTerminal(nextSymbol))
    {
        VNLR1.insert(nextSymbol);
    }
```

图 21 求 LR(1)DFA 图核心代码（4）

华 南 师 范 大 学 实 验 报 告

学生姓名 肖翔 学 号 20222132002

专 业 网络工程 年级、班级 2022 级网工二班

课程名称 编译原理 实验项目 LR(1)分析生成器

实验时间 2024 年 12 月 8 日

实验指导老师 黄煜廉 实验评分

```
        else if (isTerminal(nextSymbol))
        {
            VTLR1.insert(nextSymbol);
        }
    }
    // 校验状态是否有重复的
    for (auto& t : tempSave)
    {
        dfaStateLR1 nextState = dfaStateLR1();
        int newStateId = isNewStateLR1(t.second.originV);
        // 不重复就新开一个状态
        if (newStateId == -1)
        {
            nextState.sid = scntLR1++;
            nextState.cellV = t.second.cellV;
            nextState.originV = t.second.originV;
            dfaStateVectorLR1.push_back(nextState);
        }
        // 如果重复那么nextState的sid是已经存在的状态的id
        else nextState.sid = newStateId;
        nextStateUnitLR1 n = nextStateUnitLR1();
        n.sid = nextState.sid;
        n.c = t.first;
        dfaStateVectorLR1[stateId].nextStateVector.push_back(n);
    }
    // 对每个下一个状态进行递归
    int nsize = dfaStateVectorLR1[stateId].nextStateVector.size();
    for (int i = 0; i < nsize; i++)
    {
        auto& nextunit = dfaStateVectorLR1[stateId].nextStateVector[i];
        generateLR1State(nextunit.sid);
    }
}

// 生成LR(1)入口
void getLR1()
{
    visitedStatesLR1.clear();
    createFirstStateLR1();
    generateLR1State(0);
}
```

图 22 求 LR(1)DFA 图核心代码（5）

可以看到，求 LR(1)的 DFA 图和求 LR(0)的 DFA 图的过程十分相似（相同部分不再详细展开），不同在于求 LR(1)DFA 的时候需考虑项目的向前搜索符号。当前项目的点所对应的非终结符的 follow 集恰恰是由此非终结符所得项目的向前搜索符，也就是说我们可以在求闭包的过程中求项目的向前搜索符。此处将详细展开如何求项目的向前搜索符（lookahead）。

整体思路：

求 lookahead 的代码是在构建 LR(1) 项目的闭包以及状态转移过程中，用于确定新生成的 LR(1) 项目的向前搜索符（lookahead）的逻辑。其核心依据是根据当前正在处理的文法规则中，点号（.）所处位置以及其后面符号的类型（终结符、非终结符等情况），结合已计算出的 First 集合和 Follow 集合来确定合适的 lookahead 值。

具体情况解释：

①点后是非终结符且是文法规则右部最后一个元素的情况。当点号（.）后面紧跟着的符号（通过 nextSymbol 获取）是非终结符，并且这个非终结符已经处于文法规则右部的最后位置（即 currentCell.item.index + 1 >= grammarRight.size()）时，新生成项目的 lookahead 应该是当前项目左部非终结符的 Follow 集合。

例如，对于文法规则 A -> .B，如果要生成新的 LR(1) 项目基于 B 的后续项目，此时

华 南 师 范 大 学 实 验 报 告

学生姓名 肖翔 **学　号** 20222132002

专　　业 网络工程 **年级、班级** 2022 级网工二班

课程名称 编译原理 **实验项目** LR(1)分析生成器

实验时间 2024 年 12 月 8 日

实验指导老师 黄煜廉 **实验评分**

B 后面没有其他符号了，那么新生成项目的 lookahead 就取 A 的 Follow 集合中的元素，通过遍历 followSets[grammarDeque[currentCell.item.gid].left].s（也就是 A 的 Follow 集合），将其中元素添加到 lookahead 字符串中。

②如果点号后面紧跟着的符号（nextSymbol）是终结符，按照 LR(1) 项目闭包及状态转移的逻辑，此时不需要基于这个终结符去生成新的项目或者做其他特殊处理，所以直接跳过后续关于生成新 LR(1) 项目的相关逻辑，继续处理当前状态下其他的 LR(1) 项目。

③点后是非终结符且不是文法规则右部最后一个元素的情况。首先获取点号后面字符的下一个符号（通过 nextNextSymbol 获取），这是为了进一步判断后续如何确定 lookahead：

（1）如果 nextNextSymbol 是空字符串（@），由于合法的文法规则右部若有多个终结符或非终结符，最后一个字符不可能是空字符串，所以尝试获取再下一个字符（串）（通过 grammarDeque[currentCell.item.gid].right[currentCell.item.index + 2]）来作为真正要判断的后续符号，以便确定 lookahead。

（2）如果 nextNextSymbol 是终结符，那么很直接地就将这个终结符作为新生成项目的 lookahead，即将 lookahead 赋值为 nextNextSymbol，因为终结符本身就是明确的向前搜索符号了。

（3）如果 nextNextSymbol 是非终结符，此时新生成项目的 lookahead 应该是这个非终结符的 First 集合。所以通过遍历 firstSets[nextNextSymbol].s（也就是 nextNextSymbol 这个非终结符的 First 集合），将其中元素添加到 lookahead 字符串中，以此来确定新生成 LR(1) 项目的向前搜索符。

5. 生成LR(1)分析表

华 南 师 范 大 学 实 验 报 告

学生姓名 肖翔 学 号 20222132002

专 业 网络工程 年级、班级 2022 级网工二班

课程名称 编译原理 实验项目 LR(1)分析生成器

实验时间 2024 年 12 月 8 日

实验指导老师 黄煜廉 实验评分

```
// 根据当前状态和符号（终结符或非终结符），获取转移到的下一个状态编号，如果不存在则返回 -1
int getNextStateId(int stateId, string symbol)
{
    for (const nextStateUnitLR1& nextUnit : dfaStateVectorLR1[stateId].nextStateVector) {
        if (nextUnit.c == symbol) {
            return nextUnit.sid;
        }
    }
    return -1;
}

// 判断当前状态针对某个终结符是移进还是规约等动作，返回相应动作字符串
string getActionForTerminal(int stateId, string terminal)
{
    for (int itemId : dfaStateVectorLR1[stateId].cellV) {
        dfaCellLR1& itemCell = dfaCellVectorLR1[itemId];
        // 如果点未到文法规则末尾，且点后符号是遇到的终结符，那么移进
        if (itemCell.item.index < grammarDeque[itemCell.item.gid].right.size() &&    ⚠comparison of int…
            grammarDeque[itemCell.item.gid].right[itemCell.item.index] == terminal) {
            // 是移进动作，返回移进对应的状态编号
            return "s" + to_string(getNextStateId(stateId, terminal));
        }
        // 如果点已到文法规则末尾，且遇到的终结符在项目的向前搜索符中，那么规约
        else if (itemCell.item.index == grammarDeque[itemCell.item.gid].right.size() &&    ⚠comparison …
                 itemCell.item.lookahead.find(terminal)!= string::npos) {
            // 是规约动作，返回对应的文法编号
            return "r" + to_string(itemCell.item.gid);
        }
    }
    return "";  // 如果既不是移进也不是规约，返回空字符串
}
```

图 23 求 LR(1)分析表的辅助函数

getActionForTerminal 函数的作用是根据给定的状态编号 stateId 和终结符 terminal，判断在该状态下遇到这个终结符时应执行的动作（移进、规约或者无动作）并返回相应的动作字符串。它通过遍历当前状态（dfaStateVectorLR1[stateId]）中的所有项目（通过 cellV 向量中的项目编号获取对应的 dfaCellLR1 项目）来进行判断：

①如果某个项目中 “点” 的位置不在产生式末尾（即 itemCell.item.index < grammarDeque[itemCell.item.gid].right.length()），并且 “点” 后面的符号就是给定的终结符 terminal，那么说明在此状态下遇到该终结符应该执行移进操作，此时调用 getNextStateId 函数获取通过该终结符转移到的下一个状态编号，并返回形如 "s<状态编号>" 的移进动作字符串。

②如果某个项目中 “点” 的位置在产生式末尾（即 itemCell.item.index == grammarDeque[itemCell.item.gid].right.length()），并且该项目的向前看符号（lookahead）中包含给定的终结符 terminal（通过 find 函数判断），那么说明在此状态下遇到该终结符应该执行规约操作，返回形如 "r<文法编号>" 的规约动作字符串。

③如果遍历完所有项目都没有符合上述移进或规约的情况，说明在此状态下遇到该终结符没有对应的有效动作，返回空字符串。

getNextStateId 函数根据给定的状态编号 stateId 和一个符号（可以是终结符也可以是非终结符），在当前状态的状态转移信息中查找是否存在通过该符号转移到的下一个状态，如果找到则返回对应的状态编号，若没找到（即不存在这样的转移情况）则返

华 南 师 范 大 学 实 验 报 告

学生姓名 肖翔 学 号 20222132002

专 业 网络工程 年级、班级 2022 级网工二班

课程名称 编译原理 实验项目 LR(1)分析生成器

实验时间 2024 年 12 月 8 日

实验指导老师 黄煜廉 实验评分

回 -1。

```
// 生成LR(1)分析表的函数
void generateLR1AnalysisTable()
{
    set<string> VTLR1_copy;
    VTLR1_copy = VTLR1;
    VTLR1_copy.insert({"$"});
    LR1Table.resize(dfaStateVectorLR1.size());  // 根据LR(1) DFA状态数量初始化分析表大小
    for (size_t stateId = 0; stateId < dfaStateVectorLR1.size(); ++stateId) {// 遍历每个LR(1)状态来填充分析表
        LR1TableUnit& tableUnit = LR1Table[stateId];
        // 处理终结符对应的动作
        for (string terminal : VTLR1_copy) {    △Missing reference in range-for with non trivial type (…
            string action = getActionForTerminal(stateId, terminal);
            if (!action.empty()) {
                tableUnit.action[terminal] = action;
            }
        }
        // 处理非终结符对应的去向（goto）
        for (string nonTerminal : VNLR1) {    △Missing reference in range-for with non trivial type (st…
            int nextStateId = getNextStateId(stateId, nonTerminal);
            if (nextStateId!= -1) {
                tableUnit.goTo[nonTerminal] = to_string(nextStateId);
            }
        }
        dfaStateLR1 currentState = dfaStateVectorLR1[stateId];
        int fisrtCellId = currentState.cellV[0];
        if (currentState.isEnd &&
//            currentState.cellV.size() == 1 &&
            grammarDeque[dfaCellVectorLR1[fisrtCellId].item.gid].left == trueStartSymbol &&
            dfaCellVectorLR1[fisrtCellId].item.index ==     △comparison of integers of different signs:…
                grammarDeque[dfaCellVectorLR1[fisrtCellId].item.gid].right.size() &&
            dfaCellVectorLR1[fisrtCellId].item.lookahead == "$") {
            tableUnit.action["$"] = "ACCEPT";
            qDebug() << "ACCEPT?";
        }
    }
}
```

图 24 求 LR(1)分析表核心代码

generateAnalysisTable 函数是生成 LR(1) 分析表的核心函数，它基于已经生成的 LR(1) DFA 表信息构建 LR(1) 分析表。主要执行以下几个步骤：

①初始化分析表大小。根据已经生成的 LR(1) DFA 状态数量来初始化 LR1Table 的大小，确保每个 LR(1) 状态都有对应的分析表单元来存储信息。

②填充终结符对应的动作信息。外层循环遍历每个 LR(1) 状态（通过状态编号 stateId），获取当前状态对应的分析表单元引用 tableUnit（通过 LR1Table[stateId]）。

内层循环遍历终结符集合 VTLR1，对于每个终结符 terminal，调用 getActionForTerminal 函数获取在当前状态下遇到该终结符时应执行的动作字符串 action，如果动作字符串不为空（说明存在有效的移进或规约等动作），则将该动作字符串存入当前状态分析表单元的 action map 中，以终结符为键，对应的动作字符串为值（通过 tableUnit.action[terminal] = action;），这样就记录好了每个状态下针对不同终结符的动作信息。

③填充非终结符对应的去向（goto）信息。同样外层循环遍历每个 LR(1) 状态，内层循环遍历非终结符集合 VNLR1，对于每个非终结符 nonTerminal，调用 getNextStateId 函数获取从当前状态通过该非终结符转移到的下一个状态编号

华 南 师 范 大 学 实 验 报 告

学生姓名 肖翔 学 号 20222132002

专 业 网络工程 年级、班级 2022 级网工二班

课程名称 编译原理 实验项目 LR(1)分析生成器

实验时间 2024 年 12 月 8 日

实验指导老师 黄煜廉 实验评分

nextStateId，如果 nextStateId 不为 -1（说明存在这样的转移情况），则将对应的状态编号转换为字符串并存入当前状态分析表单元的 goTo map 中，以非终结符为键，状态编号字符串为值（通过 tableUnit.goTo[nonTerminal] = to_string(nextStateId);），完成非终结符去向信息的记录。

④处理接受状态。如果当前状态的规约项目文法左部是文法扩展后的开始符号、项目的向前搜索符号是'$'（在本系统中，能出现这种情况的都是状态第一条项目），则此项目需要 ACCEPT。因此，首先获取当前状态的首条项目。然后根据项目结构体中的 index 字段，用于判断 “点” 是否在产生式末尾，与对应的文法产生式的右部长度进行比较。同时还要判断此项目的文法左部是否为文法的真开始符号（即 trueStartSymbol，因为文法可能会经过增广处理）。最后需访问 LR1Item 结构体里的 lookahead 字段，用于判断向前看符号是否含有 '$'。如果上述条件均满足，则说明此项目需要被接受，将当前状态分析表单元中对应 '$' 这个终结符的动作设置为 "ACCEPT"（通过 tableUnit.action["$"] = "ACCEPT";），

6. 判断文法是否为 SLR(1)文法

```
// 检查移进-规约冲突
bool SLR1Fun1()
{
    for (const dfaStateLR1& state : dfaStateVectorLR1)
    {
        set<string> a; // 规约项目的左边集合
        set<string> rVT; // 终结符
        if (!state.isEnd) continue; // 不是规约状态不考虑
        for (int cellid : state.cellV) // 遍历规约状态内的项目
        {
            const dfaCellLR1& cell = dfaCellVectorLR1[cellid]; // 当前cell
            const grammarUnit gm = grammarDeque[cell.item.gid]; // 获取文法
            // 判断是不是规约项目
            if (cell.item.index == gm.right.size() || vectorToString(gm.right) == "@")    ⚠comparison of integers of diffe…
            {
                a.insert(gm.left);
            }
            else // 如果当前项目不是规约项目，则判断点后符号是不是终结符（涉及移进操作）
            {
                if (isTerminal(gm.right[cell.item.index]))
                {
                    rVT.insert(gm.right[cell.item.index]);
                }
            }
        }
        for (string c : a)💡          ⚠Missing reference in range-for with non trivial type (std::string) [clazy-range-loop]
        {
            for (string v : rVT)💡    ⚠Missing reference in range-for with non trivial type (std::string) [clazy-range-loop]
            {
                // 在面对输入符号v时，既可以按照非终结符c对应的产生式进行规约，又可以将v移进，这就产生了移进-规约冲突
                if (followSets[c].s.find(v) != followSets[c].s.end())
                {
                    return true;
                }

            }
        }
    }
    return false;
}
```

图 25 判断是否为 SLR(1)文法核心代码（1）

华 南 师 范 大 学 实 验 报 告

学生姓名 肖翔 学 号 20222132002

专 业 网络工程 年级、班级 2022 级网工二班

课程名称 编译原理 实验项目 LR(1)分析生成器

实验时间 2024 年 12 月 8 日

实验指导老师 黄煜廉 实验评分

```
bool SLR1Fun2()
{
    // 检查规约-规约冲突
    for (const auto& state : dfaStateVectorLR1)
    {
        set<string> a; // 规约项目的左边集合
        if (!state.isEnd) continue; // 不是规约状态不考虑
        for (int cellid : state.cellV) // 遍历规约状态内的项目
        {
            const dfaCellLR1& cell = dfaCellVectorLR1[cellid]; // 当前cell
            const grammarUnit gm = grammarDeque[cell.item.gid]; // 获取文法
            if (cell.item.index == gm.right.size() || vectorToString(gm.right) == "@")  // 判断是不是规约项目    △comparison…
            {
                a.insert(gm.left);
            }
        }
        for (string c1 : a)        △Missing reference in range-for with non trivial type (std::string) [clazy-range-loop]
        {
            for (string c2 : a)    △Missing reference in range-for with non trivial type (std::string) [clazy-range-loop]
            {
                if (c1 != c2)
                {   // 判断followSets[c1]和followSets[c2]是否有交集
                    set<string> followSetC1 = followSets[c1].s;
                    set<string> followSetC2 = followSets[c2].s;
                    set<string> intersection;
                    // 利用STL算法求交集
                    set_intersection(
                        followSetC1.begin(), followSetC1.end(),
                        followSetC2.begin(), followSetC2.end(),
                        inserter(intersection, intersection.begin())
                    );
                    // 如果交集非空，说明存在规约-规约冲突
                    if (!intersection.empty())   return true;
                }
            }
        }
    }
    return false;
}
```

图 26 判断是否为 SLR(1)文法核心代码（2）

（1）SLR1Fun1 函数的主要目的是检查文法的 DFA 状态集合中，是否存在移进-规约冲突情况。函数具体逻辑：

①循环遍历 dfaStateLR1Vector 中存储的所有 DFA 状态。如果当前状态不是规约状态（通过 state.isEnd 标志判断，true 表示是规约状态），则跳过当前循环，继续检查下一个状态。只有规约状态才有可能出现移进-规约冲突，所以只关注这类状态。

②对于是规约状态的情况，遍历当前规约状态的 cellV 向量，获取项目编号。根据项目编号 cellid 从 dfaCellVector 中获取当前的 dfaCell。再根据 dfaCell 中的 gid（文法编号）从 grammarDeque 中获取对应的文法单元，这样就能获取到具体的文法产生式信息。

③区分规约项目和移进相关情况。判断当前项目是否为规约项目。如果项目中 “点”（.）的位置在产生式右部的末尾，或者产生式右部为空串（gm.right == "@"），则认为这是一个规约项目，将该产生式左部的非终结符插入到集合 a 中，因为这个非终结符对应的产生式在当前状态需要进行规约操作。

④如果不是规约项目，说明可能涉及移进操作。进一步判断当前项目 “点” 后面的字符（gm.right[cell.index]）是否为终结符，如果是终结符，则将其插入到集合 rVT 中，表示在当前状态下遇到这个终结符时可能需要进行移进操作。

⑤检查移进-规约冲突。两层嵌套循环遍历集合 a 中的每个非终结符 c（代表规约操作相

华 南 师 范 大 学 实 验 报 告

学生姓名 肖翔 **学　号** 20222132002

专　　业 网络工程 **年级、班级** 2022 级网工二班

课程名称 编译原理 **实验项目** LR(1)分析生成器

实验时间 2024 年 12 月 8 日

实验指导老师 黄煜廉 **实验评分**

关符号）和集合 rVT 中的每个终结符 v（代表移进操作相关符号）。检查非终结符 c 的 Follow 集合中是否包含终结符 v。如果包含，意味着在面对输入符号 v 时，既可以按照非终结符 c 对应的产生式进行规约，又可以将 v 移进，这就产生了移进-规约冲突，此时直接返回 true，表示检测到冲突。

⑥如果遍历完所有状态及其项目后，都没有发现移进-规约冲突，那么函数最终返回 false。

（2）SLR1Fun2 函数的主要目的是检查文法的 DFA 状态集合中，是否存在规约-规约冲突情况。此函数前面部分与 SLR1Fun1 函数相似，在此不再赘述，不同的是检查规约-规约冲突情况的逻辑，在此详细展开：

①两层嵌套循环遍历集合 a 中的每一对不同的非终结符 c1 和 c2（因为要检查不同规约项目之间是否存在冲突。

②对于每一对非终结符 c1 和 c2，分别获取它们的 Follow 集合，然后定义一个新的集合 intersection 用于存储它们的交集。

通过 set_intersection 函数（这是 C++ STL 中用于求两个集合交集的函数，并非本系统函数）来计算 followSetC1 和 followSetC2 的交集，并将结果存储到 intersection 集合中。

③检查计算得到的交集集合是否为空，如果不为空，说明非终结符 c1 和 c2 对应的规约项目存在规约 - 规约冲突，因为它们的 Follow 集合有共同的符号，在语法分析遇到这些共同符号时，不知道该按照哪个产生式进行规约，此时函数直接返回 true，表示检测到冲突。

④如果遍历完所有状态及其规约项目后，都没有发现规约-规约冲突，那么函数最终返回 false，表示该文法不存在这种类型的冲突。

（三）测试

测试结果详见作业文件夹。

四、实验总结（心得体会）

通过此次实验，我对 LR(0) 和 LR (1) 分析法的原理、状态机的构建过程、项目集闭包的计算、分析表的生成以及如何判断文法是否为 SLR (1) 文法等核心概念有了透彻的理解。这使我明白了 LR 分析法如何通过对文法规则的分析，构建状态转移图和分析表，从而实现对输入句子的准确语法分析，以及不同变体之间的区别和联系。例如，在求 LR (1) 项目的向前搜索符（lookahead）时，我明白了其计算逻辑是根据点号（.）在文法规则中的位置以及其后符号的类型，结合 First 集和 Follow 集来确定的。

通过此次试验我还清楚地认识到 LR(0)文法、LR(1)文法以及 SLR(1)文法的区别。LR(0)文法分析能力最弱，当它遇到文法规则出现规约-移进冲突时无法分析，此时则需要在分析文法时引入 follow 集，这就是 SLR(1)文法。SLR(1)分析法能够通过 follow 集告诉 DFA 何时规约，从而能够在一定程度内避免移进-规约问题，但是 SLR(1)分析法不能完

华 南 师 范 大 学 实 验 报 告

学生姓名 肖翔 **学　号** 20222132002

专　业 网络工程 **年级、班级** 2022 级网工二班

课程名称 编译原理 **实验项目** LR(1)分析生成器

实验时间 2024 年 12 月 8 日

实验指导老师 黄煜廉 **实验评分**

全避免移进-规约问题，也无法避免规约-规约问题。此时则需要再构建 DFA 图中就考虑向前搜索符，这就是 LR(1)分析法。

在实验过程中，不可避免地遇到了各种各样的问题，如逻辑错误、运行时错误、数据结构使用不当等。通过仔细分析错误信息、调试代码（使用 qDebug()输出中间结果、逐步跟踪程序执行流程等方法），我逐渐学会了如何定位问题的根源，并找到有效的解决方案。例如，在生成 LR (1) DFA 图的过程中，出现了数组越界访问的错误，通过仔细检查代码中对向量索引的操作，发现是在处理点号后面字符的逻辑中，没有正确判断边界条件。经过修正后，成功解决了问题。

总的来说，本次实验是对自底向上分析方法有关知识的一次综合性实践应用，通过实现 LR (1) 分析生成器，我将理论知识与实际编程紧密结合，不仅加深了对 LR 分析法的理解，还在编程技能、问题解决能力等方面得到了提升。

五、参考文献：

《编译原理复习提纲》

《编译原理及实践》