# 实验2 LR(0)语法分析

一、实验内容

（1）LR分析器能够构造来识别所有能用上下文无关文法写的程序设计语言的结构。

（2）LR分析方法是已知的最一般的无回溯，移进-归约方法，它能够和其他移进-归约方法一样有效地实现。

（3）LR方法能分析的文法类是预测分析法能分析的文法类的真超集。

用LR分析法编制语法分析程序，实现可以采用任何一种编程工具。

|  |  |  |
| --- | --- | --- |
| **编号** | **实验题目** | **内容介绍** |
| **A004** | **构造识别活前缀的DFA** | **内存中可以用表矩阵形式，也可分一个同学做成图形化界面。** |
| **A005** | **LR(0)分析表构造** | **编写一个程序，输入一个LR（0）方法，输出LR(0)分析表** |
| **A006** | **构造LR(0)语法分析器** | **编写一个程序，构造LR(0)语法分析器，根据测试文法进行语句分析。** |

我做的内容是构造LR(0)语法分析器

二、设计思路

（1）定义并初始化LR(0)分析表和文法产生式（G）。

（2）定义分析函数，用于执行LR(0)分析过程。

（3）定义打印分析过程函数，将分析过程的表格打印在屏幕上。

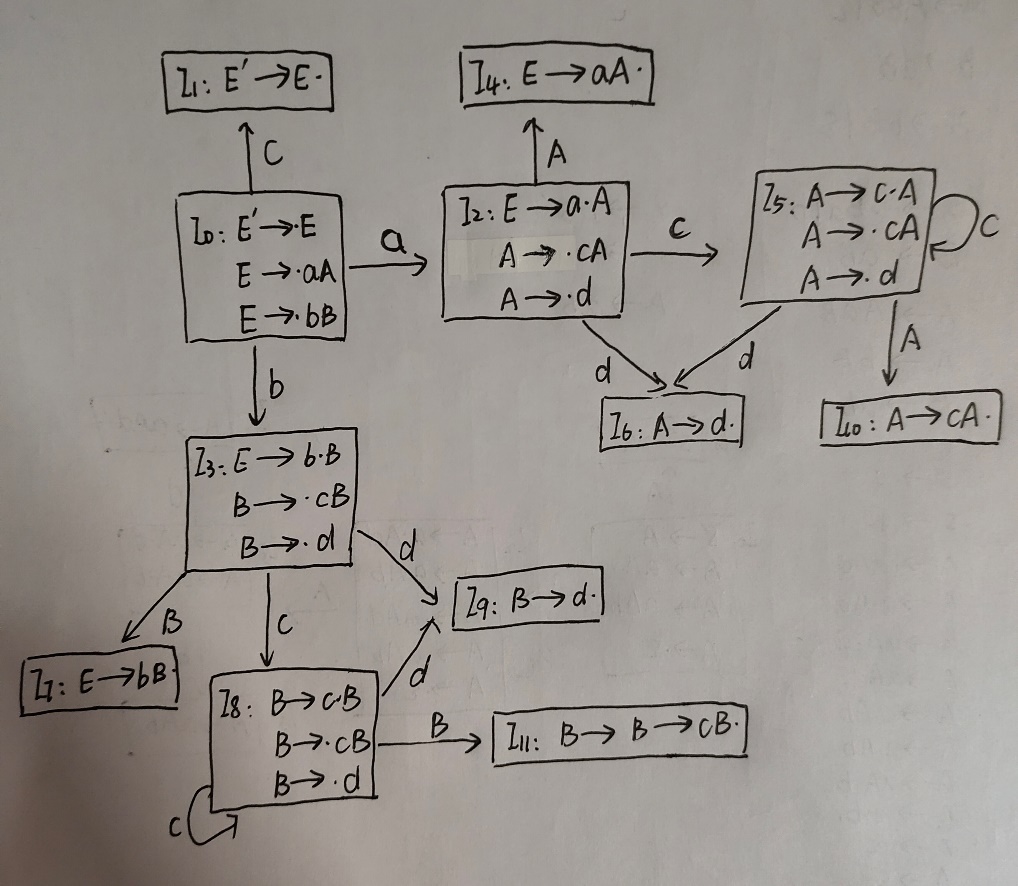
三、算法实现

所用文法：

G(E):  
E->aA|bB  
A->cA|d  
B–>cB|d

文法仅为测试所用，可以按实际需要修改文法和分析表。（因为在写时小组成员还没把他们写的文法和分析表发给我，就在网上找了一个）

LR(0)项目规范族：



1、数据结构

（1）LRTable是一个长度为12的哈希表数组，每个表项代表一个状态，键是输入符号，值是状态转移或规约动作。

unordered\_map<char, std::string> LRTable[12];

（2）G是一个二维数组，表示文法产生式。每个产生式包含两部分，左部和右部。

string G[7][2];

（3）使用两个栈，状态栈（statusStack）和符号栈（symbolStack），来辅助进行LR(0)分析。

stack<int> statusStack; stack<char> symbolStack;

2、初始化LRTable和G

void initLRTable() {

LRTable[0]['a'] = "s2";LRTable[0]['b'] = "s3";LRTable[0]['E'] = "1";

LRTable[1]['#'] = "acc";

LRTable[2]['c'] = "s5";LRTable[2]['d'] = "s6";LRTable[2]['A'] = "4";

LRTable[3]['c'] = "s8";LRTable[3]['d'] = "s9";LRTable[3]['B'] = "7";

LRTable[4]['a'] = "r1";LRTable[4]['b'] = "r1";LRTable[4]['c'] = "r1";LRTable[4]['d'] = "r1";LRTable[4]['#'] = "r1";

LRTable[5]['c'] = "s5";LRTable[5]['d'] = "s6";LRTable[5]['A'] = "10";

LRTable[6]['a'] = "r4";LRTable[6]['b'] = "r4";LRTable[6]['c'] = "r4";LRTable[6]['d'] = "r4";LRTable[6]['#'] = "r4";

LRTable[7]['a'] = "r2";LRTable[7]['b'] = "r2";LRTable[7]['c'] = "r2";LRTable[7]['d'] = "r2";LRTable[7]['#'] = "r2";

LRTable[8]['c'] = "s8";LRTable[8]['d'] = "s9";LRTable[8]['B'] = "11";

LRTable[9]['a'] = "r6";LRTable[9]['b'] = "r6";LRTable[9]['c'] = "r6";LRTable[9]['d'] = "r6";LRTable[9]['#'] = "r6";

LRTable[10]['a'] = "r3";LRTable[10]['b'] = "r3";LRTable[10]['c'] = "r3";LRTable[10]['d'] = "r3";LRTable[10]['#'] = "r3";

LRTable[11]['a'] = "r5";LRTable[11]['b'] = "r5";LRTable[11]['c'] = "r5";LRTable[11]['d'] = "r5";LRTable[11]['#'] = "r5";

G[0][0] = "S"; G[0][1] = "E";

G[1][0] = "E"; G[1][1] = "aA";

G[2][0] = "E"; G[2][1] = "bB";

G[3][0] = "A"; G[3][1] = "cA";

G[4][0] = "A"; G[4][1] = "d";

G[5][0] = "B"; G[5][1] = "cB";

G[6][0] = "B"; G[6][1] = "d";

}

3、构造LR(0)分析器

（1）初始化状态栈和符号栈，并将输入字符串的第一个字符作为输入符号。

（2）在循环中，从状态栈的顶部获取当前状态和输入符号，查找LR(0)分析表中对应的动作。

如果动作是"acc"（接受状态），则分析成功，结束循环。

如果动作以"s"开头，表示移进状态，将下一个状态入栈，并将下一个输入符号加入符号栈，同时移动输入指针。

如果动作以"r"开头，表示规约状态，根据产生式的右部长度，从状态栈和符号栈中弹出相应数量的元素，然后将产生式的左部作为新的符号入栈，并更新状态栈。

如果动作不是"s"或"r"，表示出现了错误，分析失败，结束循环。

（3）每次迭代结束后，调用printTable()函数打印分析过程的表格。

bool LR0Analysis(std::string input) {

std::cout << input << std::endl;

printf("%8s\t %8s\t %8s\t\n", "状态栈", "符号栈", "输入串");

statusStack.push(0);

symbolStack.push('#');

int p = 0;

while (true) {

int status = statusStack.top(); // 状态

char nextChar = input.at(p); // 输入字符

int nextStatus = -1;

if (LRTable[status].find(nextChar) != LRTable[status].end()) {

std::string info = LRTable[status][nextChar];

printTable(p, input, info);

if (info == "acc") {

break;

}

if (info.at(0) == 's') { // 移进状态

nextStatus = std::stoi(info.substr(1));

statusStack.push(nextStatus);

symbolStack.push(nextChar);

p++;

}

else if (info.at(0) == 'r') { // 规约状态

nextStatus = std::stoi(info.substr(1));

for (int i = 0; i < G[nextStatus][1].length(); i++) {

symbolStack.pop();

statusStack.pop();

}

char temp = G[nextStatus][0].at(0);

symbolStack.push(temp);

int s = statusStack.top();

statusStack.push(std::stoi(LRTable[s][temp]));

}

else { // GOTO 转移

break;

}

}

else {

return false;

}

}

return true;

}

4、打印分析过程

void printTable(int p, std::string input, std::string action) {

std::string status = "";

std::string symbol = "";

std::string in = "";

std::stack<int> tmp1 = statusStack;

while (!tmp1.empty()) {

status += std::to\_string(tmp1.top());

tmp1.pop();

}

std::stack<char> tmp2 = symbolStack;

while (!tmp2.empty()) {

symbol += tmp2.top();

tmp2.pop();

}

for (int i = p; i < input.length(); i++) {

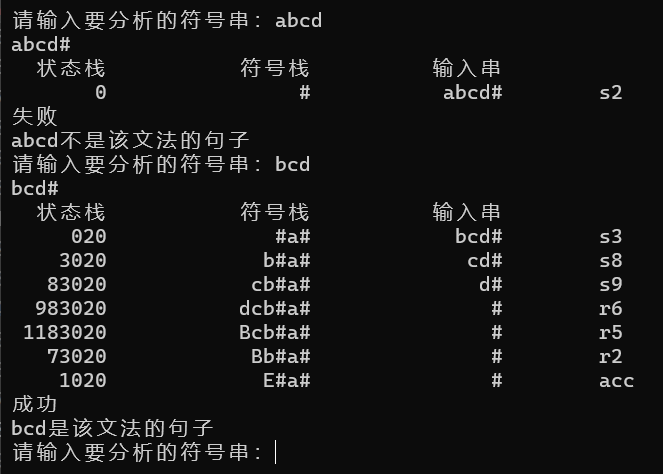
in += input.at(i);

}

printf("%8s\t %8s\t %8s\t %s\n", status.c\_str(), symbol.c\_str(), in.c\_str(), action.c\_str());

}

四、测试结果



五、心得体会

通过这次实验，我不仅对LR(0)分析方法有了更深入的理解，还提升了自己在语法分析领域的算法设计和调试能力。我学会了构建项目集规范族和构建分析表的方法，并通过验证和测试，保证了分析器的正确性。这次实验不仅加深了我对语法分析理论的认识，也对我在编译原理和相关领域的学习和实践提供了宝贵的经验和基础。