形 式 语 言 与 自 动 机

实验报告

实验名称：上下文无关文法的变换

课 程 名 称 ： 形式语言与自动机
姓名（学号）： 叶淳瑜 2019212686
涂欣宇 2019212693
蒲昊龙 2019212968
李孟非 2019210933
学 院 ： 计算机学院
专 业 ： 计算机类
班 级 ： 2019211318
指 导 教 师 ： 左兴权老师

1 实验概述

上下文无关文法对产生式的右部没有限制，这种完全自由的形式有时会对文法分析带来不良影响，而对文法的某些限制形式在应用中更方便。通过上下文无关文法的变换，在不改变文法的语言生成能力的前提下，可以消除文法中的 ε 产生式、单产生式、以及无用符号。

要求编程实现消除上下文无关文法中的 ε 产生式、单产生式、以及无用符号的算法。输入是一个上下文无关文法，输出是与该文法等价的没有 ε 产生式、单产生式、无用符号的上下文无关文法。

2 成员分工

叶淳瑜	项目规划、部分文档编写以及核心代码编写工作
涂欣宇	部分代码编写与调试工作
蒲昊龙	部分代码编写与调试工作
李孟非	部分文档及报告编写、部分代码调试工作

3 环境描述

开发平台：Visual Studio Code

编程语言及标准：C++11

编译器：MinGW

4 设计思路及核心算法

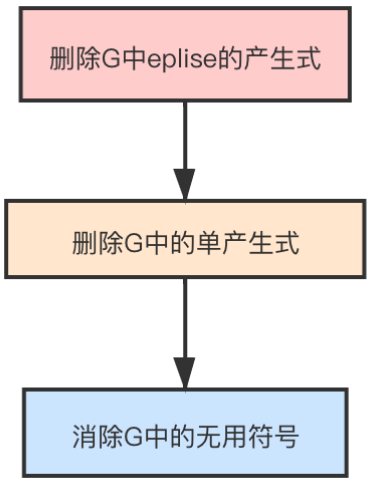
数据结构定义：

```
//生成式
struct PRO
{
    string key;                 //生成符左边
    deque<string> dervation; //生成符右边
};
```

生成式定义如图，key 代表->左边的字符串，dervation 代表->右边的字符串。其中 dervation 由于会频繁在两端插入元素，因此使用双端队列存储。

整体框架简介：

为了达到实验目的，应按照顺序依次完成消除 ε 产生式、消除单产生式、消除无用符号的功能。



于是代码主要分为四个模块，除了主函数有三个最重要的功能

函数，分别完成消除 ε 产生式、单产生式、以及无用符号的功能如下：

- void emptyPro(vector<PRO> &pros);
- void unitPro(vector<PRO> &pros);
- void usefulPro(vector<PRO> &pros);

此外，为了完成算法工作，程序还加入了其他的必要辅助函数，如下：

- void merge(vector<PRO> &pros);

消除变换过程中产生的重复符号

- void printPro(vector<PRO> &pros);

输出生成式

- bool inVector(vector<char> v, char element);

判断元素是否在 vector 中

- PRO makePro(deque<string> symbol);

格式化生成式

- bool readFile(vector<PRO> &pros);

读入文法文件

主要功能函数具体实现：

1. 主函数

在主函数中，先读入规定的输入格式文件，然后分别调用三个

主要的功能函数，完成完成消除 ε 产生式、单产生式、以及无用符号，最后输出变换的生成式。

2. bool readFile(vector<PRO> &pros);

首先读入 txt 文件中储存的 CFG 文法。利用 STL 库中 regex 实现以->或|为间隔读取字符串，然后分别每条生成式存入 vector<PRO> pros 中，其中 key 代表->左边的字符串，dervation 代表->右边的字符串。

```
//读入文法文件
bool readFile(vector<PRO> &pros)
{
    string line;
    ifstream file("CFG.txt", ios::in);
    deque<string> tmp;
    regex rgx("(\\->)|(\\|)"); //以->或|为间隔读取

    if (!file.good())
        return false;

    while (getline(file, line))
    {
        if (line == "")
            continue;

        sregex_token_iterator it(line.begin(), line.end(), rgx, -1);
        for (sregex_token_iterator end; it != end; it++)
            tmp.push_back(*it);

        pros.push_back(makePro(tmp));
        tmp.clear();
    }
    file.close();
    return true;
}
```

读取生成式

存入生成式

3. void emptyPro(vector<PRO> &pros);

首先找出可致空符号集合，如果起始符 S 属于可致空符号集，那么增加一个 S1 作为新的起始符。然后在各个生成式中找出可致空符号，每个可致空符号分别取自身或 ε 。

4. void unitPro(vector<PRO> &pros);

找出单生成式，然后用单生成式的右边替换掉单生成式的左边，直至没有单生成式。

5. void usefulPro(vector<PRO> &pros);

函数分为两步，先后完成删去非产生，删去非可达。两步的完成方法相似，以下是删去非产生的思路。

用一个可变数组存储可生成的符号，首先将 a~z 加入此集合，然后不断遍历生成式集合，如果生成符右边存在可生成的符号，那么这个符号是可生成的，不断循环，直到可生成的符号集合大小不在改变。之后，利用可生成符号，遍历生成式，如果某个符号不是生成的，那么删去这个符号。删去非可达的符号流程同理。

6. void merge(vector<PRO> &pros);

在消去 ε 产生式与单产生式的同时，算法会产生冗余重复的符号。为了使用体验与严谨性，小组编写函数消除了变换过程中产生的重复符号。逻辑十分简单，函数实现如下。

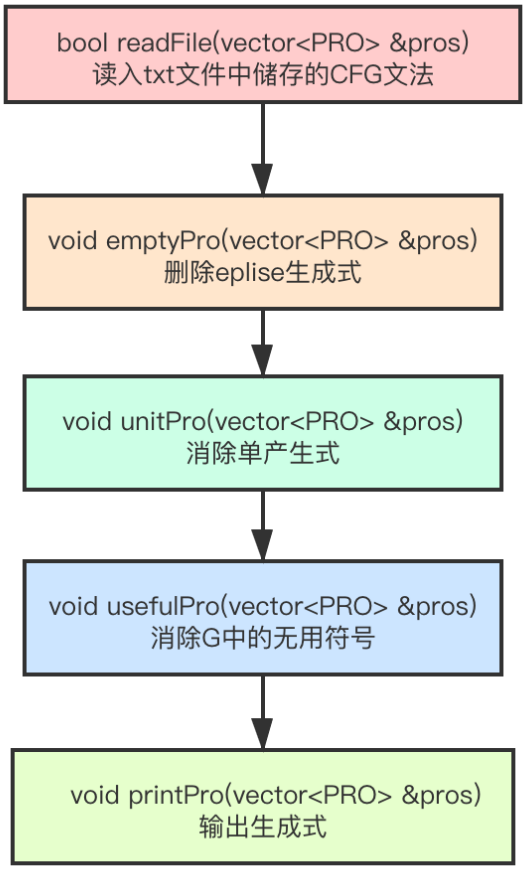
```
//消除重复符号
void merge(vector<PRO> &pros)
{
    for (auto i = pros.begin(); i < pros.end(); i++)
    {
        for (auto j = (*i).dervation.begin(); j < (*i).dervation.end(); j++)
        {
            for (auto k = j + 1; k < (*i).dervation.end();)
            {
                if (*k == *j)
                    k = (*i).dervation.erase(k);
                else
                    k++;
            }
        }
    }
}
```

遍历生成式

消除重复符号

5 代码流程

代码主要流程图如下：



6 输入输出及执行效果

- 输入：CFG.txt 文件，格式如下

```
S->a|bA|B|ccD
A->*
B->aA|E
C->ddC
D->ddd
E->aa
```

其中，上图为上下文无关文法 G=(N,T,P,S)的生成式 P，其中用‘*’代表空串，用‘->’表示定义为，‘|’分隔生成符号。

- 输出：在命令行中输出，格式如下：

```
Original productions:
S -> a|bA|B|ccD
A -> *
B -> aA|E
C -> ddC
D -> ddd
E -> aa
After removing empty productions:
S -> a|bA|B|ccD|b
B -> aA|E|a
C -> ddC
D -> ddd
E -> aa
After removing unit productions:
S -> a|bA|ccD|b|aA|aa
B -> aA|a|aa
C -> ddC
D -> ddd
E -> aa
After removing useless symbols:
S -> a|ccD|b|aa
D -> ddd
```

其中，先输出原上下文无关文法的生成式 P；

然后按照顺序，依次消去 P 中的 epsilon 产生式、单产生式、无用符号。最终的输出经检验即为实验要求。

7 改进思路和方法

- 鲁棒性

程序缺少对一些错误输入的检验，可以加入一些检验函数，来判断输入是否合法，增加程序的鲁棒性。

- 局限性

加入新符号只能是 S1，这虽然简化了程序设计，但限制了 CFG 的功能性与兼容性。例如有状态集含有 S1，就会出现重复的 S1。

- 效率问题、复杂度太高

在消除空转移中，遍历时间复杂度为 $o(n^2)$，消除重复符号时，遍历时间复杂度为 $o(n^3)$,去除单个空转移符号，时间复杂度为 $o(n^3)$,执行消除重复的问题效率太低，时间复杂度太高。可以采用 break 等语句有效降低程序循环次数，提高效率。

- 迭代器使用不规范

在删除重复符号、删除生成式，和增加符号的操作时，由于一些对容器的操作如删除元素或移动元素等会修改容器的内在状态，这会使得原本指向被移动元素的迭代器失效，也可能同时使其他迭代器失效。程序中出现一些迭代器使用不规范的问题，虽然没有 bug 产生，但是这样的使用方法还是不够规范。小组应再深入学习 STL 使用规范，多练习，多实践，提高编程水平。