消模块的代码代码如下：

void G::erase\_epsilon(){

    std::set<char> nullset = {'#'};

    std::set<std::string> N\_ = Algo\_1(nullset);//调用过算法1后已经找到的能够到达Epsilon的非终结符

    std::map<std::string, std::set<std::vector<std::string>>> P\_;//需要生成的P

    std::string left;

    std::vector<std::string> right;

    //对N中每个非终结符进行遍历，对其所有生成式进行遍历，若生成式中包含N\_中的非终结符，则将其轮流替换为Epsilon，直接消去，数目先0个，再1个，直到生成式中不再包含N\_中的非终结符，并把替换后的生成式加入P\_

    //注意，生成式右边不能完全为Epsilon，即不能出现A->#的情况

    for(auto it = N.begin(); it != N.end(); it++){

        for(auto itt = P[\*it].begin(); itt != P[\*it].end(); itt++){

            int count = 0;

            for(auto ittt = itt->begin(); ittt != itt->end(); ittt++){

                if(N\_.find(\*ittt) != N\_.end()){

                    count++;

                }

            }

            if(count == 0){

                if(itt->size() == 1 && (\*itt)[0] == "#"){

                    continue;

                }

                else{

                    P\_[\*it].insert(\*itt);

                }

            }

            else{

                if(itt->size() == 1)

                    P\_[\*it].insert(\*itt);//生成式右边只有一个非终结符，且为N\_中的非终结符，直接加入P\_

                else{

                    std::vector<std::string> temp;

                    std::string s;

                    for(auto ittt = itt->begin(); ittt != itt->end(); ittt++){

                        s += \*ittt;

                    }

                    generateReplacements(N\_, s, 0, temp);

                    for(auto ittt = temp.begin(); ittt != temp.end(); ittt++){

                        std::string replaced\_s;

                        right.clear();

                        for(auto itttt = ittt->begin(); itttt != ittt->end(); itttt++){

                            if(\*itttt != '#')

                                replaced\_s += \*itttt;

                        }

                        if(replaced\_s != "")

                            right.push\_back(replaced\_s);

                        P\_[\*it].insert(right);

                    }

                }

            }

        }

    }

    //若起始符能够推导出Epsilon，即N\_中包含S，则需要添加一个新的起始符S1，S1->S|Epsilon

    if(N\_.find(S) != N\_.end()){

        left = S + "1";

        N\_.insert(left);

        right.push\_back(S);

        right.push\_back("#");

        P\_[left].insert(right);

    }

    P = P\_;

}

具体思路为首先获得经过算法1后已经找到的能够到达状态的非终结符集合N\_，然后对原来语法中N非终结符集的每一个非终结符为索引对所有生成式进行遍历，然后对生成式进行匹配，若生成式中包含N\_中的元素，则说明需要替换，count计数加一，统计一个生成式中需要被替换的字符数目，若count数值为0，有两种情况，一是生成式的右侧就是，那么该语句不能被加入到新的P\_中，S的情况放到最后考虑，第二种情况就是生成式右侧无需替换，可直接把该生成式加入到新的P\_中，若conut不为0，说明生成式右侧有非终结符可以被空字符替换，若右侧只有一个字符，则替换后为空，不可加入，因此我们可以直接把该生成式加入P\_中，若有多个字符，由于替换成是轮流的，数目会成指数级增长，因此我们可以先利用递归函数来将匹配的非终结符轮流替换成#，递归代码如下

void G::generateReplacements(std::set<std::string> N\_, std::string s, int index, std::vector<std::string>& result) {

    // 如果当前索引已经到了字符串末尾，将当前字符串加入结果集中

    if (index == s.size()) {

        result.push\_back(s);

        return;

    }

    // 如果当前字符属于N\_集合，则生成两个分支，一个是替换成 '#'，一个是不替换

    if (N\_.find(s.substr(index, 1)) != N\_.end()) {

        // 替换成 '#'

        std::string replaced\_s = s;

        replaced\_s[index] = '#';

        generateReplacements(N\_, replaced\_s, index + 1, result);

        // 不替换

        generateReplacements(N\_, s, index + 1, result);

    }

    // 如果当前字符不属于，直接跳到下一个字符

    else {

        generateReplacements(N\_, s, index + 1, result);

    }

}

这样我们就实现了对生成式右侧的可达的非终结符的初步替换，之后再对返回的temp进行遍历，把#全部跳过，其余字符储存进生成式右侧，抛弃长度为0的生成式便完成了对的消除，接下来讨论对起始符S的消除，若N\_中存在S，则再额外加入一条S1->S|，并且在N\_中插入S1，最后我们再将P\_赋值给P，由此，对的消除便已彻底完成。