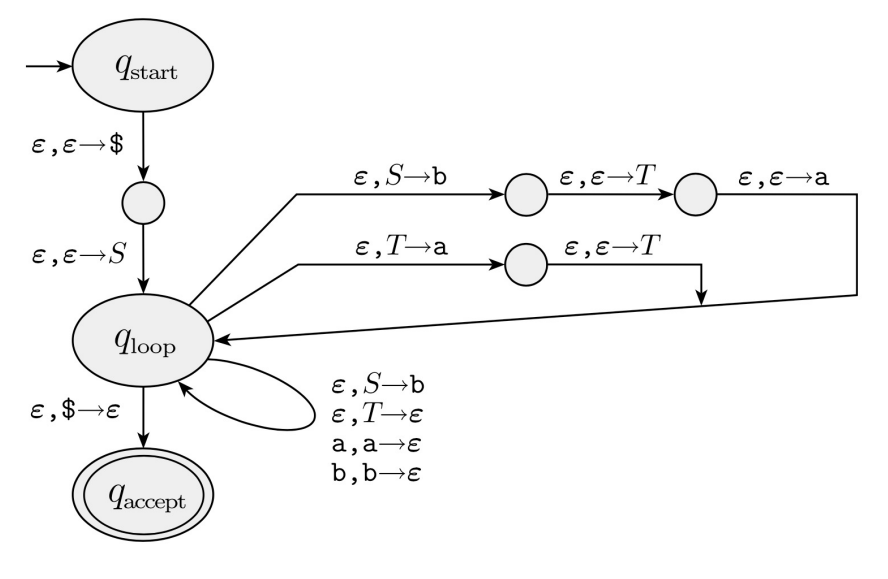
**编译原理第二章可选作业**

2154312 郑博远

答：

采用将CFG转为PDA的思想，例如将文法：

转换为PDA：

根据此思想，设计C++程序如下：

|  |
| --- |
| #include <iostream>  #include <vector>  #include <string>  #include <stack>  #include <algorithm>  #include <map>  using namespace std;  class PDA {  // 存储变元 约定第一个变元为起始符  vector<char> vars;  // 存储所有产生式  map<char, vector<string>> generations;  // 为了防止无限递归，设置最大递归深度  const int MAX\_DEPTH = 32;  bool submatch(stack<char> s, string sentence, int depth)  {  // 超过最大递归深度，则不再尝试  if (depth > MAX\_DEPTH)  return false;  // 已经匹配完全  if (s.empty())  if (sentence == "")  return true;  else  return false;  char top = s.top();  s.pop();  // 若此刻栈顶是变元（非终结符）  if (count(vars.begin(), vars.end(), top)) {  // 尝试各种产生式的可能性  for (auto str : generations[top]) {  stack<char> ss = s;  // 将产生式右端倒着压栈  for (int i = str.length() - 1; i >= 0; i--)  ss.push(str[i]);  if (submatch(ss, sentence, depth + 1))  return true;  }  return false;  }  // 若此刻栈顶是终结符  else {  // 若与字符串首个匹配  if (top == sentence[0])  return submatch(s, sentence.substr(1), depth + 1);  else  return false;  }  }  public:  PDA(vector<char> \_vars):vars(\_vars)  {  }  void addGeneration(char left, string right)  {  generations[left].push\_back(right);  }  bool match(string sentence)  {  stack<char> s;  s.push(vars[0]);  return submatch(s, sentence, 0);  }  };  int main()  {  PDA pda(vector<char>{'S', 'T'});  pda.addGeneration('S', "aTb");  pda.addGeneration('S', "b");  pda.addGeneration('T', "Ta");  pda.addGeneration('T', "");  cout << boolalpha << pda.match("aaab");  return 0;  } |