МИНОБРНАУКИ РОССИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ
ВЫСШЕГО ПРОФЕССИОНАЛЬНОГО ОБРАЗОВАНИЯ

«САНКТ-ПЕТЕРБУРГСКИЙ ПОЛИТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ ПЕТРА ВЕЛИКОГО»

Институт Компьютерных Наук и Кибербезопасности

Высшая школа технологий искуственного интелекта

Направление 02.03.01 Математика и Компьютерные Науки

Лабораторная работа №5

По дисциплине:
«Математическая логика и теория автоматов»

Тема работы:

«Построение левого анализатора заданной грамматики»
«Вариант №14»

Обучающийся: _____ Черепанов Михаил Дмитриевич

Руководитель: _____ Востров Алексей Владимирович

«____» _____ 20__ г.

Санкт-Петербург 2024

Содержание

Введение

В рамках данной работы необходимо:

Построить множества FIRST и FOLLOW для каждого нетерминала грамматики и таблицу выбора (lookup table). На их основе реализовать детерминированный левый анализатор (проверка принадлежности цепочки грамматике). Назначить семантические действия части заданных продукций.

Грамматика моего варианта:

$S' \to S\$,\ S \to AaS \mid b,\ A \to CAb \mid B,\ B \to cSa \mid \varepsilon,\ C \to c \mid ab$

1 Математическое описание

1.1 Контекстно-свободные грамматики

Грамматикой G называется множество: {T,N,S,R}, где:

• T - конечное множество символов (терминальный словарь);

• N - конечное множество символов (нетерминальный словарь);

• S ∈ N - начальный нетерминал;

• R - конечное множество правил (продукций) вида $\alpha \to \beta$, где α и β цепочки над словарём $T \cup N$

Контекстно-свободной грамматикой называется грамматика, у которой левые части всех продукций являются одиночными нетерминалами. То есть все правила грамматики имеют вид:

$A \to \lambda$, где $A \in T$, $\lambda \in N$

1.2 LL(k) грамматики

LL(k) грамматики - подкласс контекстно-свободных грамматик, в котором можно выполнить нисходящий синтаксический анализ, просматривая входную цепочку слева при восстановлении левого вывода данной терминальной цепочки, заглядывая вперед на каждом шаге не более чем на k символов.

Для обозначения LL(K) грамматик введем функции FIRST и FOLLOW:

FIRST(A) - множество терминальных символов, с которых могут начинаться цепочки, выводимые из A.

FOLLOW(A) - множество терминальных символов, с которых могут начинаться цепочки, следующие за A в любых сентенциальных формах.

Формально грамматика явялется LL(k) если:

$S \to *wAx \to w\alpha x \to *wu$

$S \to *wAx \to w\beta x \to *wv$, где:

S,A - нетерминалы грамматики;

w - цепочка из терминалов;

α, β, x - цепочки из терминалов и нетерминалов.

A — нетерминал грамматики, для которого есть правила A $\to \alpha$ и A $\to \beta$;

Из условия $FIRST_k(u)$ следует условие $FIRST_k(v)$.

1.3 LL(1) грамматики

LL(1) - подмножество LL(K) грамматик, в которых можно выполнить низходящий синтаксический анализ, просматривая не более 1 символа вперед на каждом шаге.

Грамматика является LL(1), если:

1. $A \to \alpha, A \to \beta, FIRST(\alpha) \cap FIRST(\beta) = \oslash$

2. $A \to \alpha, A \to \beta, \epsilon \in FIRST(\alpha), FIRST(\alpha) \cap FOLLOW(A) \cap FIRST(\beta) = \oslash$,где:

A - нетерминал;

α, β - цепочки из терминалов и нетерминалов.

1.4 Анализ заданной грамматики

Грамматика заданного варианта в более удобном для анализа виде представлена на рис.1.

S'-> S$
S -> AaS
S-> b
A -> CAb
A -> B
B -> cSA
B -> empty
C -> c
C -> ab

Рис. 1: Грамматика заданного варианта.

На рис.2 приведены множества FIRST и FOLLOW и множество выбора на основе них для каждой продукции.

№	Продукция	FIRST	FOLLOW	Выбор
1	S'→S$	{a,b,c}		{a,b,c}
2	S→AaS	{a,c}		{a,c}
3	S→ b	{b}		{b}
4	A→ CAb	{a,c}		{a,c}
5	A→ B	{c,e}	{a,b}	{a,b,c}
6	B→ cSA	{c}		{c}
7	B→ e	{e}	{a,b}	{a,b}
8	C→ c	{c}		{c}
9	C→ ab	{a}		{a}

Рис. 2: Множества FIRST и FOLLOW.

На рис.4 приведена таблица выбора.

	a	b	c	$
S’	1	1	1	ош
S	2	3	2	ош
A	4\5	5	4\5	ош
B	7	7	6	ош
C	9	ош	8	ош

Рис. 3: Lookup table.

Как видно по рис.2 - lookup table - неоднозначна, поэтому данная грамматика не является LL(1), а значит детерминированный LL(1) анализатор по ней построить нельзя.

Известными мне методами привести грамматику к LL(1) не удалось, и было принято решение - строить недетерминированный анализатор.

1.5 Пример вывода цепочки

На рис.4 приведен пример вывода цепочки в данной грамматике.

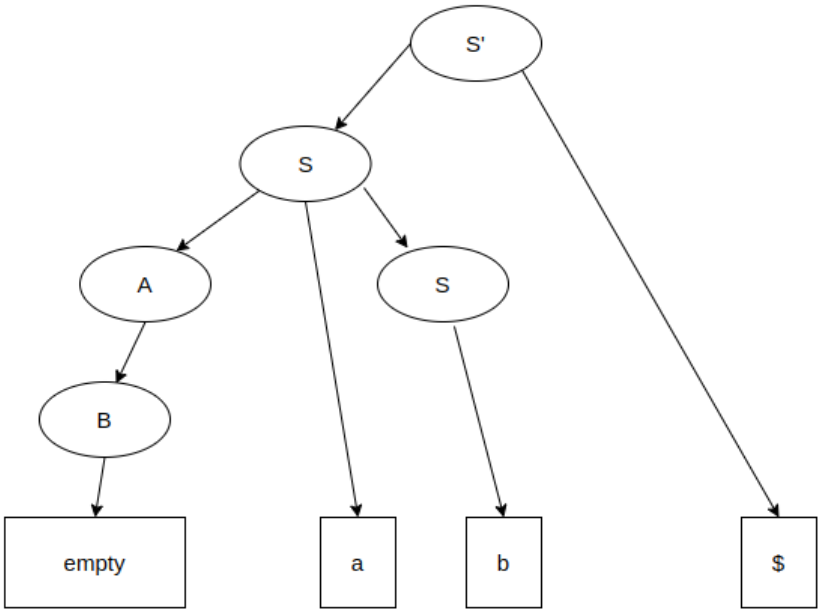

Рис. 4: Пример вывода цепочки.

1.6 Назначение семантических действий

Каждой из продукций грамматики были назначены семантические действия $y_1 ... y_9$ в порядке, представленном на рис. 5:

S'-> y1 S$
S -> y2 AaS
S-> y3 b
A -> y4 CAb
A -> y5 B
B -> y6 cSA
B -> y7 empty
C -> y8 c
C -> y9 ab

Рис. 5: Пример вывода цепочки.

Каждое из $y_1 ... y_9$ - обозначает некоторые арифметические действия над исходным числом - аналог азартной игры.

Изначально у пользователя 0 у. е.

Семантические действия соответсвуют:

y_1 - (+1)
y_2 - (+2)
y_3 - (-1)
y_4 - (-2)
y_5 - (*2)
y_6 - (/2)
y_7 - (+3)
y_8 - (*(-1)) - Глобальный переворот.
y_9 - (*5) - Опасный джекпот.

После завершения анализа введенной строки пользователь выйграет или проиграет N у.е.

2 Особенности реализации

Программа реализована на языке c++ и состоит из функции main и класса Grammar.

В функции main происходит взаимодействие с пользователем;

В классе Grammar реализована генерация случайной строки, в соответсвии с грамматикой и проверка строки на соответсвие грамматике.

2.1 Класс Grammar

Создание lookup table

Таблица хранится в виде map<pair<char,char>,vector<int».

Ключем являются значения строки и столбца таблицы. Значение - вектор номеров продукций, которые нужно проверить.

Хранение продукций

Хранение продукций так же реализовано с помощью хэш-таблицы:

map<int, pair<pair<char,string>,pair<int,string»>. Ключ - номер продукции, значение - кортеж из правил продукции(первая пара) и правил вывода семантических действий(вторая пара).

2.1.1 Генерация строки

Генерация строки реализована в методе generateRandom.

Вход функции: Правила грамматики.

Выход функции: Случайно сгенерированная строка, соответсвующая грамматике.

В теле функции используется стек символов. Изначально в стек записывается начальный нетерминал S'.

Затем, пока стек не пуст, выбираются случайно продукции, ничинающиеся с верхнего нетерминала в стеке. Правая часть продукции записывается в стек. Если на вершине стека лежит терминал - записываем его в результирующую строку.

Исходный код функции:

```
1   string  generateRandom(){
2         stack<char> st;
3         resultStr.clear();
4         while (!st.empty()) {
5             st.pop();
6         }
7         st.push('Q');
8         while(!st.empty()){
9             char simbol = st.top();
10            st.pop();
11
12            if(nonterminals.find(simbol) != string::npos){
13                int numberProd;
```

```
                do{
                    numberProd = rand()%9+1;
                }
                while (productions[numberProd].first.first!=simbol);
                string forAddToStack =
                ↪  productions[numberProd].first.second;
                reverse(forAddToStack.begin(),forAddToStack.end());
                for(auto e: forAddToStack){
                    st.push(e);
                }
            }
            else if(terminals.find(simbol) != string::npos){
                resultStr.push_back(simbol);
            }
            else{
                cout<<"  Generation error"<<endl;
            }
        }
        if(resultStr == "b$" && rand()%2==0){
            resultStr = generateRandom();
        }
        if(resultStr.size()>100){
            resultStr = generateRandom();
        }
        return resultStr;
    }

```

2.1.2 Проверка строки

Проверка строки реализована функциями GlobalCheck и RecCheck.

GlobalCheck

Вход функции: строка для проверки.

Выход функции: true или false в зависимости от результата проверки. Печать вывода строки в данной грамматике или сообщения об ошибке.

В теле функции создается стек, в который кладется начальный нетерминал, вызывается рекурсивная функция recCheck.

Исходный код функции GlobalCheck:

```
 bool GlobalCheck(string strIn){
        ampunt_money=0;
        long localAmountMoney=0;
        stack<char>st;
        st.push('Q');
        string strOut="";
```

```
        vector<string> logOutput;
        helpLogOut.clear();
        bool isCorrect = RecCheck(st,strIn,logOutput,localAmountMoney);
        if(isCorrect){
            cout<<"Строка соответсвует грамматике. Вывод
            ↪  строки:"<<endl;
            for(auto e: helpLogOut){
                cout<< e<<endl;
            }
            cout<<"Ваш выйгрыш: "<<ampunt_money;
            cout<< endl<<endl;

            return true;
        }
        else{
            cout<<"Строка не соотвествует
            ↪  грамматике."<<endl<<endl<<endl;
            return false;
        }
    }
```

RecCheck

Вход функции: Стек с текущей сентенциальной формой, правила грамматики, текущее кол-во у. е. для реализации семантических действий, строка для проверки;

Выход функции: Последовательность продукций для вывода строки.

В теле функции RecCheck реализована проверка на соответвие строки грамматике, генерация последовательности вывода и семантических действий на основе алгоритма поиска в глубину.

Условием нахождения корректного пути вывода является то, что стек пуст и были обработаны все символы строки;

Если выполнилось одно их этих условий, значит текущая ветка обхода не дает верного решения - откатываемся назад.

Исходный код функции:

```
 bool RecCheck(stack<char>st,string strIn,vector<string> logOut, long
 ↪  localAmountMoney, string pushBackingString=""){
        if(pushBackingString !="")
            logOut.push_back(pushBackingString);

        if(st.empty() && strIn.empty()){
            helpLogOut = logOut;
            ampunt_money=localAmountMoney;
            return true;
        }
        if(st.empty() || strIn.empty()){
```

```
11              return false;
12          }
13
14          char topSt=st.top();
15          st.pop();
16          if(terminals.find(topSt) != string::npos) {                    //
            ↪  на вершине стека терминал
17              if (topSt == strIn[0]) {
18                  strIn.erase(0, 1);
19                  return RecCheck(st, strIn, logOut, localAmountMoney);
20              }
21
22              if (topSt != strIn[0]) {
23                  return false;
24              }
25          }
26          else if(nonterminals.find(topSt) != string::npos) {
            ↪  // на вершине стека нетерминал
27              vector<int> productionsVector =
                ↪  table[make_pair(topSt,strIn[0])];
28              for(int numberProduction:productionsVector){
29                  stack<char> helpSt = st;
30                  pair<char,string> productionPair;
31                  productionPair = productions[numberProduction].first;
32                  string action
                    ↪  =productions[numberProduction].second.second;
33                  int actionNumber =
                    ↪  productions[numberProduction].second.first;
34                  string nextPushBackingString = (string(1,
                    ↪  productionPair.first) == "Q"? "S'":string(1,
                    ↪  productionPair.first))
35                          +" -> "+
36                          (productionPair.second != "" ?
                            ↪  productionPair.second : "empty")
37                          + "      ( "+action +" )";
38                  localAmountMoney =
                    ↪  calculateMoney(localAmountMoney,actionNumber);
39
                    ↪  reverse(productionPair.second.begin(),productionPair.second.end())
40                  for(auto e: productionPair.second){
41                      helpSt.push(e);
42                  }
43
                    ↪  if(RecCheck(helpSt,strIn,logOut,localAmountMoney,nextPushBackingSt
                    ↪  == true){
```

```
44                      return true;
45                  }
46
47              }
48
49          }
50          return false;
51      }
```

3 Результаты работы

На рис. 6 показан пример генерации трех случайных цепочек.

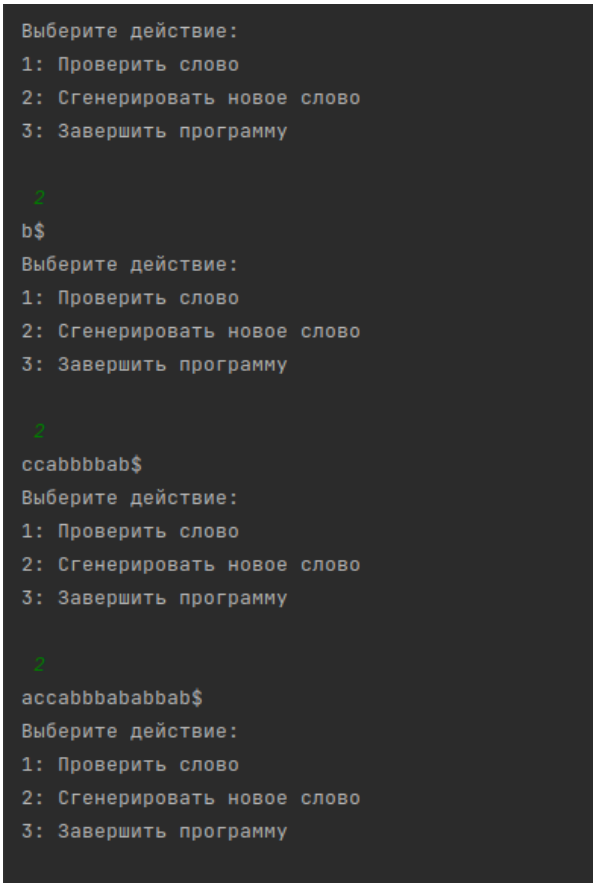

Рис. 6: Пример генерации случайных цепочек.

На рис. 7,8 показаны примеры проверки корректных цепочек.

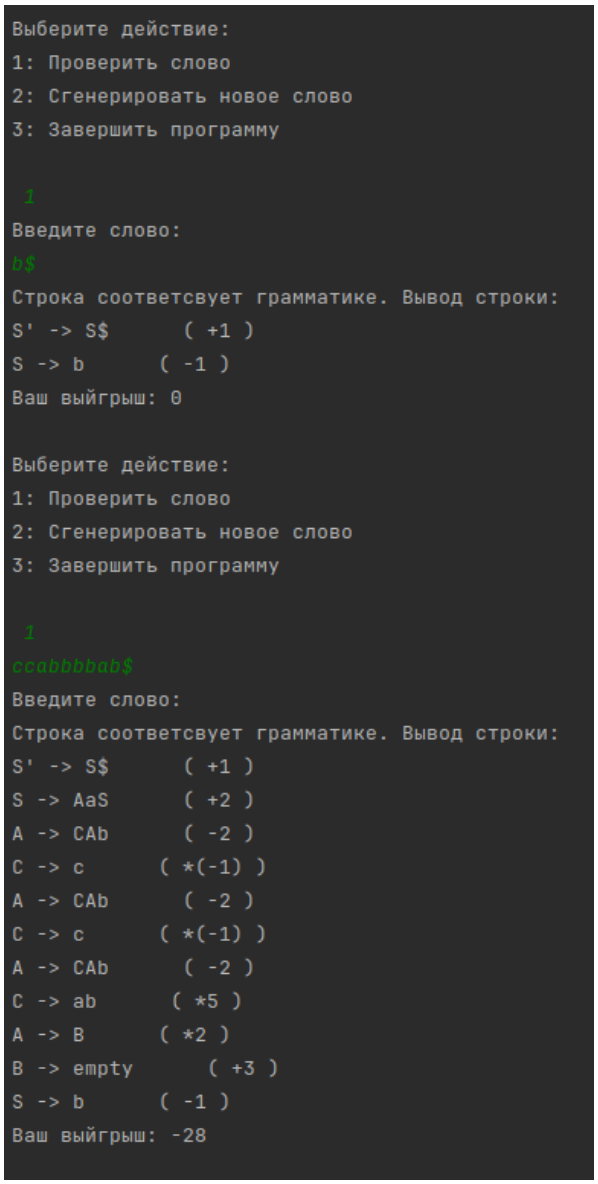

Рис. 7: Пример проверки корректных цепочек №1.

```
Выберите действие:
1: Проверить слово
2: Сгенерировать новое слово
3: Завершить программу

 1
Введите слово:
accabbbababbab$
Строка соответсвует грамматике. Вывод строки:
S' -> S$      ( +1 )
S -> AaS      ( +2 )
A -> B      ( *2 )
B -> empty      ( +3 )
S -> AaS      ( +2 )
A -> B      ( *2 )
B -> cSA      ( /2 )
S -> AaS      ( +2 )
A -> CAb      ( -2 )
C -> c      ( *(-1) )
A -> CAb      ( -2 )
C -> ab      ( *5 )
A -> B      ( *2 )
B -> empty      ( +3 )
S -> b      ( -1 )
A -> CAb      ( -2 )
C -> ab      ( *5 )
A -> B      ( *2 )
B -> empty      ( +3 )
S -> b      ( -1 )
Ваш выйгрыш: 302
```

Рис. 8: Пример проверки корректной цепочки №2.

На рис. 9. показан пример обработки пустой цепочки.

```
Выберите действие:
1: Проверить строку
2: Сгенерировать новую строку
3: Завершить программу

 1
Введите слово:

Строка не соотвествует грамматике.
```

Рис. 9: Пример обработки пустой цепочки.

На рис. 10 показан пример вывода сообщений об ошибках.

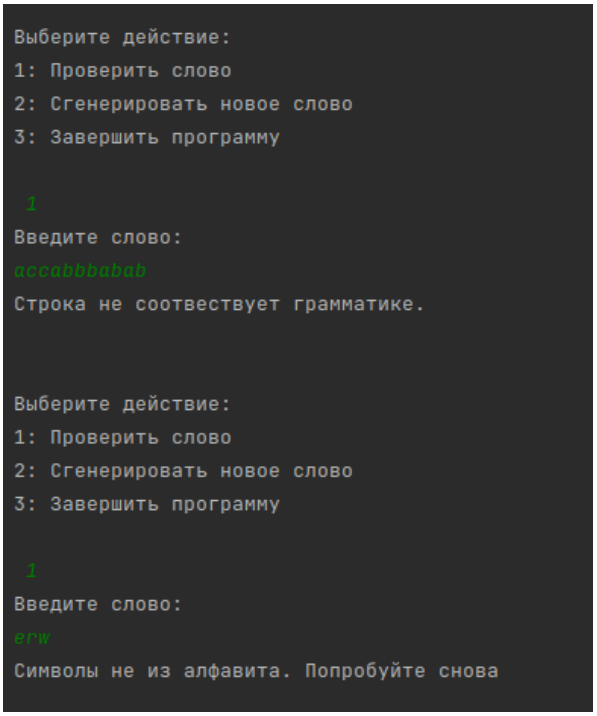

Рис. 10: Вывод сообщений об ошибках.

Заключение

Итак, в ходе работы была проанализирована контекстно-свободная грамматика заданного варианта, построены множества FIRST и FOLLOW для продукций грамматики.

Грамматика не является LL(1), поэтому для ее анализа был реализован недетерминированный анализатор.

Были реализованы методы генерации цепочки в соответсвии с заданой грамматикой и проверка цепочки на соответсвие с ней. Заданы семантические действия для каждой из цепочек.

Приемущества реализации:

1. Использование алгоритма на основе поиска в глубину.

2. Использование стека для хранения сентенциальных форм грамматики.

Недостатки реализации:

1. Нерациональное использование контейнеров для хранения продукций и семантических действий.

2. В худшем случае сложность реализованного алгоритма проверки цепочек на соответсвие грамматике является экспоненциальной.

Масштабирование:

Программу можно масштабировать путем реализации алгоритма генерации наиболее успешных цепочек в контексте максимизации выйгрыша в реализованной игре на основе зависимости семантических действия.

Список использованных источников

[1] Ф.А Новиков. Дискретная математика для программистов: Учебник для вузов. 3-е изд. — СПб: Питер, 2008. —384 с.

[2] А.В. Востров. Лекция №12 по дисциплине «Математическая логика и теория автоматов». 2024г. 55 сл.