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 №3

По дисциплине:
«Математическая логика и теория автоматов»

Тема работы:

«Построение контекстно-свободной грамматики подмножества естественного языка»
«Выбранное подмножество: Past perfect passive voice английского языка»

Обучающийся: _____ Черепанов Михаил Дмитриевич

Руководитель: _____ Востров Алексей Владимирович

«____» _____ 20__ г.

Санкт-Петербург 2024

Содержание

Введение **3**

1 Математическое описание **4**
1.1 Описание выбранной грамматики 4
1.2 Математическое описание порождающей грамматики 4

2 Особенности реализации **7**
2.1 Проверка введенного предложения на соответсвие грамматики . . . 7
2.2 Генерация случайного предложения, соответствующего грамматике 8
2.3 Класс Main 11
2.3.1 Функция preprocessStr(String str) 11
2.3.2 Функция main 12

3 Результаты работы **14**

Заключение **17**

Список использованных источников **18**

Введение

В данной лабораторной работе необходимо:

1. Выбрать подмножество естественного языка, в качестве грамматики для построения;

2. Согласовать выбранную грамматику с преподавателем;

3. Реализовать распознавание вводимого предложения на принадлежность грамматике;

4. Реализовать генератор предложений в соответствии с выбранной грамматикой.

1 Математическое описание

1.1 Описание выбранной грамматики

Была выбранна грамматика:

Past perfect passive voice английского языка.

Past perfect passive voice используется в случаях:

1. Когда говорится о действии или событии, которое произошло раньше остальных упомянутых;

2. Когда события или действия - результат еще более ранних событий;

3. Когда говориться о событиях, которые произошли в прошлом и продолжались до определенного момента.

В данной работе реализованы действия с утвердительными, отрицательными и вопросительными предложениями.

Правила построения утвердительного предложения:

Subject + «had been» + V3 (Past Participle) + Придаточная часть (Опционально) + '.'

Правила построения отрицательного предложения:

Subject + «had not been» + V3 (Past Participle) + Придаточная часть (Опционально) + '.'

Правила построения отрицательного предложения:

«Had» + Subject + «been» + V3 (Past Participle) + Придаточная часть (Опционально) + '?'

1.2 Математическое описание порождающей грамматики

Порождающей грамматикой Хомского 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 $A \in T$, $\lambda \in N$

Для построения грамматики Past perfect passive voice было выбрано:

T = { stood, watched, spoke, guessed, showed, chewed, ate, laughed, cried, danced, sang arrived, came, swam, ran, rode, drove, travelled knew, believed, hated, felt, respected, understood, feared, listened, found, imagined, accepted, admired, suggested, tasted, regretted, appreciated, expected apple, orange, banana, pear, book, table, pen, doctor, teacher, engineer, artist, pen, notebook, pencil, paperclip, Paper, table, chair, computer, human, dog, cat, giraffe, hippopotamus, moonwalker handsome, terrible, lovely, mysterious, kind, honest, bad, great, cool, yellow, red, black, white, international, curly, active,

sharp, stupid, Smart, heavy confidently, badly, well, confidently, wisely, equally, magnificently, beautifully, funny, thoughtfully, surprised stood, watched, spoken, guessed, shown, chewed, eaten, laughed, cried, danced, sung London, Moscow, New York, Finland, Russia, USA he, she, it, we, they known, believed, hated, felt, respected, understood, feared, listened, found, imagined, accepted, admired, suggested, tasted, regretted, appreciated, expected, had, not, been, '?', '.'}

N = {Sentence , Sentence+ ,Sentence- , Sentence? , Noun_group, Noun_pronoun, V3_group, Add_state, Add_move, V3_move, Add_sent, V3_state, Pronoun, Place, V3_action, Abjective, Noun_object, link, Preposition, V2_state, V2_move, V2_action }

S = Sentence

Конечное множество продукций в БНФ нотации: R = {

<Sentence> ::=<Sentence+>|<Sentence->|<Sentence?>

<Sentence+> ::= <Noun_group> had been <V3 group> <Add sent> .

<Sentence-> ::= <Noun_group> had not been <V3 group> <Add sent> .

<Sentence?> ::= Had <Noun group> been <V3 group> <Add sent> ?

<Noun_group> ::= <Abjective> <Noun_object> | <Pronoun>

<Noun_pronoun> ::= <Noun_object> | <Pronoun>

<V3_group> ::= <V3_state> [<Add_state>] | <V3_action> [<Add_action>]

<Add_state>::= <Noun_pronoun>

<V3_move> ::= arrived | come | swum | run | ridden | drived | travelled

<Add_move> ::= <Preposition> <Place>

<Add_sent> ::= <Add sent> | <link> <Noun_group> <V2_group> | <Empty>

<V3_state> ::= known | believed | hated | felt | respected | understood | feared | listened | found | imagined | accepted | admired | suggested | tasted | regretted | appreciated | expected

<Pronoun> ::= he | she | it | we | they

<Place> ::= London | Moscow | New York | Finland | Russia | USA

<V3_action> ::= stood | watched | spoken | guessed | shown | chewed | eaten | laughed | cried | danced | sung

<Add action> ::=confidently | badly | well | confidently | wisely | equally | magnificently | beautifully | funny | thoughtfully | surprised

<Abjective>::= Handsome | Terrible | Lovely | Mysterious | Kind | Honest | Bad | Great | Cool | Yellow | Red | Black | White | International | Curly | active | sharp | stupid | Smart | heavy

<Noun_object> ::= apple | orange | banana | pear | book | table | pen | doctor | teacher | engineer | artist | pen | notebook | pencil | paperclip | Paper | table | chair | computer | human | dog | cat | giraffe | hippopotamus | moonwalker

<link>::= by | before

<Preposition> ::= from | to

<V2_group> ::= <V2_state> <Add_state> | <V2_move> <Add_move> | <V2_action> <Add_action>

<V2_state> ::= knew | believed | hated | felt | respected | understood | feared | listened | found | imagined | accepted | admired | suggested | tasted | regretted | appreciated | expected

<V2_move> ::= arrived | came | swam | ran | rode | drove | travelled

<V2_action> ::= stood | watched | spoke | guessed | showed | chewed | ate | laughed | cried | danced | sang

}

Для наглядности рассмотрим пример вывода утвердительного предложения в этой грамматике(рис.1).

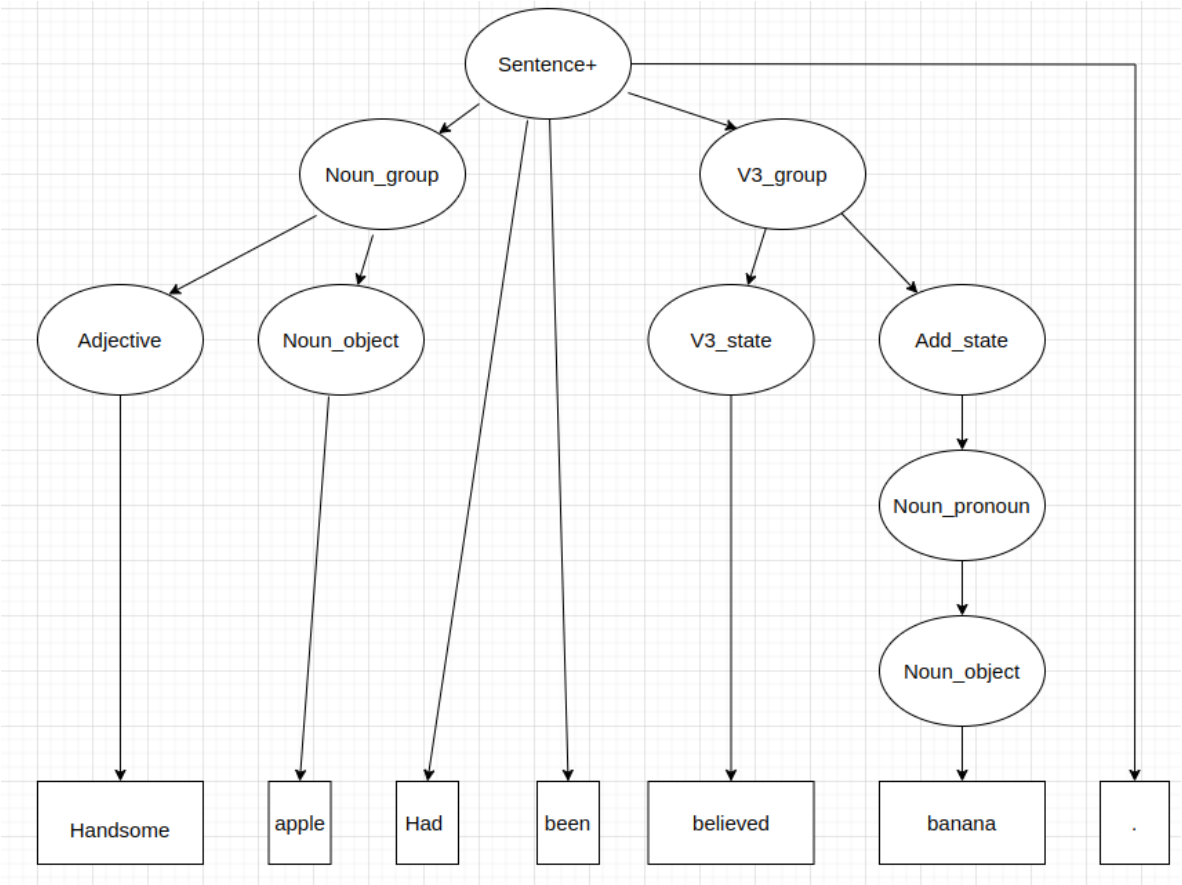

Рис. 1: Пример вывода утвердительного предложения.

2 Особенности реализации

Лабораторная работа реализована на языке Java.

Структура проекта включает в себя 4 класса:

1. Класс Dict - в нем происходит хранение слов из множества терминальных символов. Хранение реализованно с помощью коллекции ArrayList в виде строковых констант.

2. Класс GrammarChecker - предназначен для проверки введенного предложения на соответсвие грамматики.

3. Класс GrammarGenerator - предназначен для генерации случайных предложений, соответсвующих грамматике.

4. Класс Main - в нем происходит общение с пользователем.

2.1 Проверка введенного предложения на соответсвие грамматики

Проверка введенного предложения на соответсвие грамматики реализована в классе GrammarChecker при помощи метода рекурсивного спуска:

Для каждого из нетерминальных символов написана функция, название которой совпадает с названием нетерминала.

Внутри каждой из таких функций последовательно происходит проверка токенов(слов) на соответсвие грамматики при помощи вызовов следующих функций. В случае несоответствия, выбрасывается исключение, которое затем обрабатывается в main.

Пример функции для терминала SentenceP:

```
1   void SentenceP()throws RuntimeException{
2         curr_index=0;
3         currWord=tokens.get(0);
4         NounGroup();
5         if(currWord.equals("had")) NextWord(); else throw new
           ↪  RuntimeException("0");
6         if(currWord.equals("been")) NextWord(); else throw new
           ↪  RuntimeException("0");
7         V3Group();
8         AddSent();
9     }
```

Текущее слово в последовательности токенов определяется с помощью поля:

int curr_index;

Последовательность токенов хранится в контейненере:

ArrayList<String> tokens;

С помощью функции NextWord() реализован переход к следующему слову в последовательности.

Исходный код функции NextWord():

```
void NextWord() throws RuntimeException{
        curr_index++;
        if(curr_index<tokens.size())
            currWord=tokens.get(curr_index);
        else throw new RuntimeException("0");
    };
```

2.2 Генерация случайного предложения, соответствующего грамматике

Генерация случайного предложения, соответствующего грамматике реализована в классе GrammarGenerator и происходит на основе синтаксических диаграмм.

Внутри класса GrammarGenerator создан внутренний класс Node, код котрого представлен ниже:

```
class Node {
        String name;
        Vector<Vector<Node>> matrix_roles;
        ArrayList<String> tokens;

        public void setTokens(ArrayList<String> tokens) {
            this.tokens = tokens;
        }

        public void setMatrix_roles(Vector<Vector<Node>> matrix_roles)
        ↪ {
            this.matrix_roles = matrix_roles;
        }

        String getRandomToken() {
            return tokens.get(new Random().nextInt(tokens.size()));
        }

        Vector<Node> getRandomRules() {
            return matrix_roles.get(new
            ↪ Random().nextInt(matrix_roles.size()));
        }

        Node(String name) {
            this.name = name;
            matrix_roles = new Vector<>();
            tokens = new ArrayList<>();
        }
    }

```

Каждый объект класса Node представляет собой один из элементов декомпозиции диаграммы. В поле matrix_roles храняться другие объеты класса Node, внутренние элементы декомпозиции.

matrix_roles представляет из себя матрицу, где в соответсвующих строках хранятся узлы, расположенные на диаграмме последовательно.

Параллельные ветки хранятся в разных строках матрицы.

В конструкторе класса GrammarGenerator для каждого терминального символа грамматики создаются объекты класса Node:

```
 Node SentP = new Node("SentP");
    Node SentN = new Node("SenN");
    Node SentQ = new Node("SentQ");
    Node NounGroup = new Node("NounGroup");
    Node Had = new Node("Had");
    Node Not = new Node("Not");
    Node Been = new Node("Been");
    Node V3Group = new Node("V3Group");
    Node AddSent = new Node("AddSent");
    Node Abjective = new Node("Abjective");
    Node NounPronoun = new Node("NounPronoun");
    Node NounObject = new Node("NounObject");
    Node Pronoun = new Node("Pronoun");
    Node V3State = new Node("V3State");
    Node AddState = new Node("AddState");
    Node V3Move = new Node("V3Move");
    Node AddMove = new Node("AddMove");
    Node V3Action = new Node("V3Action");
    Node AddAction = new Node("AddAction");
    Node Preposition = new Node("Preposition");
    Node Place = new Node("Place");
    Node Link = new Node("Link");

    Node V2Group = new Node("V2Group");
    Node V2Action = new Node("V2Action");
    Node V2Move = new Node("V2Move");
    Node V2State = new Node("V2State");


```

А затем для каждого узла происходит заполнение матрицы matrix_roles в соответсвии с правилами грамматики.

Пример заполнения matrix_roles для терминала SentP:

```
Vector<Vector<Node>> helpVec = new Vector<Vector<Node>>() {{
            add(new Vector<Node>() {{
```

```
                add(NounGroup);
                add(Had);
                add(Been);
                add(V3Group);
                add(AddSent);
            }});
        }};
        SentP.setMatrix_roles(new Vector<>(helpVec));

```

Генерация предложения по заданному нетерминалу происходит в методе GenerateGen(Node role).

Метод принимает на вход: один из узлов Node;

Возвращает: строку, сгенерированную в соответсвии с поступившим на вход узлом.

В теле метода реализован обход элемента синтаксической диаграммы. В случае встречи параллельных ветвей, путь выбирается случайно.

Исходный код метода GenerateGen(Node role):

```
String GenerateGen(Node role) {
        Vector<Node> vecNode = new Vector<>();
        vecNode.add(role);
        int amountApply = 0;
        for (int i = 0; i < vecNode.size(); i++) {
            if (!vecNode.get(i).matrix_roles.isEmpty()) {
                amountApply++;
                Vector<Node> helpVec = vecNode.get(i).getRandomRules();
                vecNode.remove(i);
                vecNode.addAll(i, helpVec);
                i--;
            }
        }
        Vector<String> tokens = new Vector<>();
        for (int i = 0; i < vecNode.size(); i++) {
            tokens.add(vecNode.get(i).getRandomToken());
        }
        String sentence = String.join(" ", tokens);

        if (!sentence.isEmpty()) {
            sentence = Character.toUpperCase(sentence.charAt(0)) +
             ↪  sentence.substring(1);
        }
        return sentence;
    }
```

2.3 Класс Main

В классе Main реализовано пользовательское меню, предварительная обработка введенных предложений, инициализация классов GrammarChecker и GrammarGenerator.

2.3.1 Функция preprocessStr(String str)

Вход: Строка, введенная пользователем;
Выход: Список токенов, в виде ArrayList<String>;
В теле функции происходит:
1. Удаление лишних пробелов пробелов;
2. Перевод всех символов в нижний регистр;
3. Преобразование сокращение hadn't в had not, если оно встретилось;
4. Разделение строки по прообелам;
Исходный код функции preprocessStr(String str):

```
public static ArrayList<String> preprocessStr(String str) {

        str=str.trim();
        String lastChar = str.substring(str.length() - 1);

        str = str.substring(0, str.length() - 1);
        str = str.replaceAll("[^a-zA-Z0-9 -]", "");

        ArrayList<String> result = new ArrayList<>();
        // Разделяем строку по пробелам
        String[] words = str.split("\\s+");

        for (String word : words) {
            word=word.toLowerCase();
            result.add(word);
        }

        result.add(lastChar);
        for (int i = 0; i < result.size(); i++) {
            String word = result.get(i);
            if (word.equalsIgnoreCase("hadn't") ||
            ↪  word.equalsIgnoreCase("hadnt")) {
                // Заменяем "hadn't" на "had not"
                result.set(i, "had");
                // Вставляем "not" после "had"
                result.add(i + 1, "not");
            }
        }
        return result;
    }
```

2.3.2 Функция main

Вход: Запрос пользователя на генерацию случайного предложения или проверку введенного предложения;

Выход: Случайное предложение или сообщения о результатах проверки.

В main реализовано пользовательское меню, общение с пользователем, вызов методов классов GrammarChecker и GrammarGenerator.

Исходный код функции main:

```
1  public static void main(String[] args) {
2          Scanner scanner = new Scanner(System.in);
3          checker = new GrammarChecker();
4          generator = new GrammarGenerator();
5          boolean flag;
6
7          while(true){
8              System.out.println("\nДля проверки предложения введите
               ↪  1;\n" +
9                      "Для генерации введите 2;\n" +
10                     "Для выхода введите Q");
11             String action = scanner.nextLine();
12             if(action.equals("1")) {// проверить
13                 System.out.println("Введите предложение:");
14                 String str = scanner.nextLine();
15                 CheckStr(str);
16             }
17             else if(action.equals("2")) {// сгенерировать
18                 System.out.println("Для генерации утвердительного
                   ↪  предложения введите 1:\n" +
19                         "Для генерации отрицательного предложения
                           ↪  введите 2:\n" +
20                         "Для генерации вопросительного предложения
                           ↪  введите 3:");
21
22                 String str = scanner.nextLine();
23                 String newSent;
24                 if(str.equals("1")) {
25                     newSent = generator.GenerateP();
26                 }
27                 else if(str.equals("2")) {
28                     newSent = generator.GenerateN();
29                 }
30                 else if(str.equals("3")) {
31                     newSent = generator.GenerateQ();
32                 }
33                 else{
```

```
34                      System.out.println("Некорректный ввод! Попробуйте
                        ↪  снова:");
35                      break;
36                  }
37
38                  System.out.println(newSent);
39              }
40              else if(action.equals("Q")) {// выход
41                  break;
42              }
43              else{
44                  System.out.println("Некорректный ввод! Попробуйте
                    ↪  снова:");
45              }
46          }
47      }
```

3 Результаты работы

Результат генерации случайных предложений (одно утвердительное, одно отрицательное, одно вопросительное) представлен на рис. 2

```
Для проверки предложения введите 1;
Для генерации введите 2;
Для выхода введите Q
2
Для генерации утвердительного предложения введите 1:
Для генерации отрицательного предложения введите 2:
Для генерации вопросительного предложения введите 3:
1
Moonwalker had been watched.

Для проверки предложения введите 1;
Для генерации введите 2;
Для выхода введите Q
2
Для генерации утвердительного предложения введите 1:
Для генерации отрицательного предложения введите 2:
Для генерации вопросительного предложения введите 3:
2
Table had not been chewed magnificently.

Для проверки предложения введите 1;
Для генерации введите 2;
Для выхода введите Q
2
Для генерации утвердительного предложения введите 1:
Для генерации отрицательного предложения введите 2:
Для генерации вопросительного предложения введите 3:
3
Had pen been appreciated life by way cried well?

Для проверки предложения введите 1;
Для генерации введите 2;
Для выхода введите Q
```

Рис. 2: Результат генерации случайных предложений.

Результат проверки введенных пользователем предложений (одно утвердительное, одно отрицательное, одно вопросительное) представлен на рис. 3

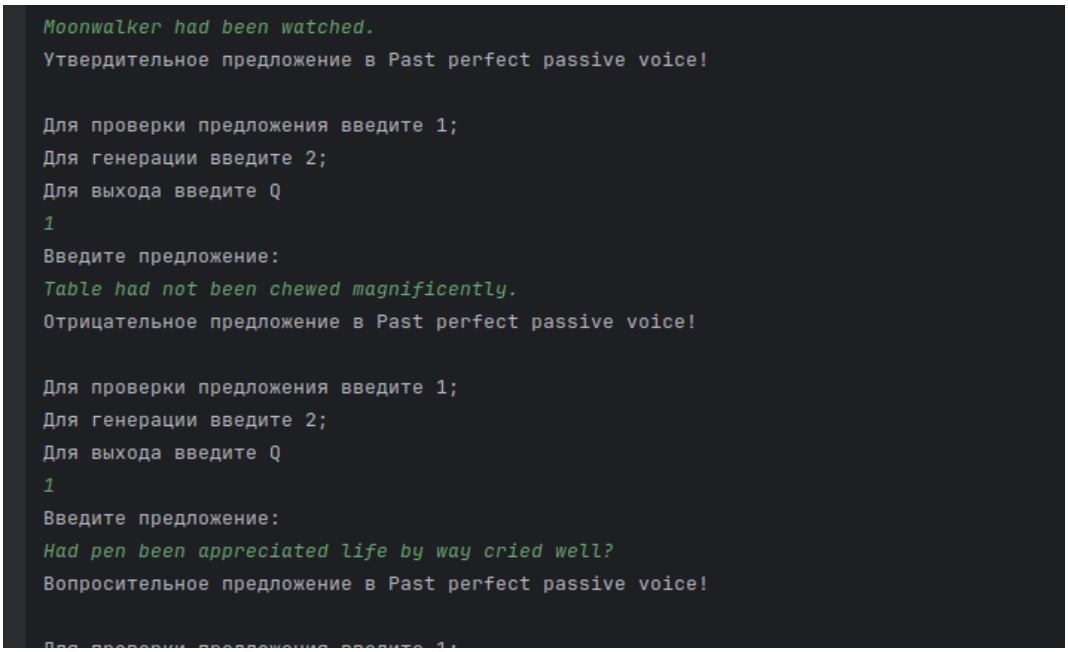

Рис. 3: Результат генерации случайных предложений.

Пример вывода сообщений об ошибках во введенных предложениях представлен на рис. 4

```
Для проверки предложения введите 1;
Для генерации введите 2;
Для выхода введите Q
1
Введите предложение:
Year had been shown confidently before table swam from.
Основная часть предложения распознана корректно, но в придаточной части допущены ошибки!
 Придаточная часть может находится только в Past simple!

Для проверки предложения введите 1;
Для генерации введите 2;
Для выхода введите Q
1
Введите предложение:
Year had been confidently before table swam from Russia.
Предложение не является предложением в Past perfect passive voice,
Или взяты слова не из выбранного подмножества языка.

Для проверки предложения введите 1;
Для генерации введите 2;
Для выхода введите Q
1
Введите предложение:
Year had been shown confidently before table swam from
Предложение в моем подмножестве языка всегда должно заканчиваться точкой или вопросительным знаком!
 Попробуйте еще раз.

Для проверки предложения введите 1;
```

Рис. 4: Вывод об ошибках.

Заключение

Итак, в результате работы была составлена контекстно-свободная грамматика подмножества past perfect passive voice английского языка.

Была реализована программа, позволяющая:

1. Генерировать случайные предложения в соответствии с этой грамматикой.
2. Проверять предложения на соответствие этой грамматике.

Были реализованы действия над утвердительными, отрицательными и вопросительными предложениями. Семантика слов не учитывается, времена и лица внутри предложений не согласуются.

Преимущества реализации:

1. Использование метода рекурсивного спуска для проверки предложения на соответствие грамматике;
2. При вводе предложений, состоящих из основной и придаточной части, если основная часть корректна, но в придаточной допущены ошибки, программа выведет соответствующее сообщение (рис. 4).

Недостатки реализации:

1. Использование разных контейнеров: ArrayList и Vector для одних и тех же целей (хранение строк). Это делает программу более сложной к восприятию и усложняет масштабирование.
2. Использование механизма выброса исключений в методах класса GrammarChecker ставит ограничение на количество распознаваемых ошибок и усложняет масштабирование.

Масштабирование:

Программу можно масштабировать путем добавления новых терминальных и нетерминальных символов. Для этого нужно будет только изменить содержимое контейнеров в классе Dict и добавить новые методы в классах GrammarChecker и GrammarGenerator.

Кроме этого, если изменить механизм определения ошибок в методе рекурсивного спуска (см. Недостатки реализации (П.2)), программу можно масштабировать путем определения точных местоположений несоответствий грамматике.

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 №8 по дисциплине «Математическая логика и теория автоматов». 2024г. 52 сл.