МИНОБРНАУКИ РОССИИ
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ
ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО
ПРОФЕССИОНАЛЬНОГО ОБРАЗОВАНИЯ
«САНКТ-ПЕТЕРБУРГСКИЙ ПОЛИТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ ПЕТРА ВЕЛИКОГО»
Институт компьютерных наук и кибербезопасности
Высшая школа технологий искусственного интеллекта
Направление **02.03.01** : Математика и компьютерные науки

Лабораторная работа «Построение контексно-свободной грамматики подмножества естественного языка: Past Perfect английского языка»
по дисциплине «Теория алгоритмов»

Обучающийся: _____ Яшнова Дарья Михайловна
группа 5130201/20002

Руководитель: _____ Востров Алексей Владимирович

«___»_____ 2025г

Санкт-Петербург, 2025

Содержание

Введение

В данной лабораторной работе необходимо:

1. Реализовать распознавание вводимого предложения на принадлежность грамматике;
2. Реализовать генератор предложений в соответствии с выбранной грамматикой.

В качестве подмножества естественного языка был выбран past perfect английского языка.

1 Описание грамматики Past Perfect

Past Perfect (прошедшее совершенное время) используется для выражения действия, которое произошло до другого действия в прошлом или до определённого момента в прошлом. Оно подчёркивает завершённость действия и его связь с последующими событиями.

1.1 Формы предложений

1.1.1 Утвердительные предложения

Образуются по схеме: **Subject + had + V3 (Past Participle) + ...**

Subject	**had**	**V3**	**...**
She	had	finished	her work.
They	had	left	before we called.

1.1.2 Отрицательные предложения

Образуются по схеме: **Subject + had + not + V3 + ...**

Subject	**had**	**not**	**V3**	**...**
He	had	not	seen	that movie before.
We	had	not	visited	Paris until last year.

1.1.3 Вопросительные предложения

Образуются по схеме: **Had + Subject + V3 + ... ?**

Had	**Subject**	**V3**	**...**
Had	you	been	there before?
Had	she	finished	her report?

1.1.4 Отрицательные вопросительные предложения

Образуются по схеме: **Had + Subject + not + V3 + ... ?**

Had	**Subject**	**not**	**V3**	**...**
Had	they	not	received	the message?

2 Математическое описание

2.1 Грамматика и язык

В данной работе строится *контекстно-свободная грамматика*, задающая язык предложений в Past Perfect английского языка.

2.1.1 Формальное определение

Порождающая грамматика Хомского определяется как четверка $G = \langle T, N, S, R \rangle$, где:

- T - конечное множество терминалов (слова языка).
- N - конечное множество нетерминалов ($T \cap N = \emptyset$).
- $S \in N$ - начальный нетерминал.
- R - множество правил вывода вида $\alpha \to \beta$.

Для Past Perfect определим следующие компоненты:

2.1.2 Алфавиты

$$T = \{\text{had}, \text{not}, ?, ., \text{verbs\_V3}, \text{nouns}, \text{pronouns}, \text{adverbs}\}$$

$$N = \{\text{S}, \text{Affirm}, \text{Neg}, \text{Quest}, \text{Subj}, \text{VerbPhrase}, \text{TimeAdv}\}$$

2.1.3 Продукции

$$\begin{aligned} S &\to \text{Affirm} \mid \text{Neg} \mid \text{Quest} \\ \text{Affirm} &\to \text{Subj had V3 TimeAdv} \\ \text{Neg} &\to \text{Subj had not V3 TimeAdv} \\ \text{Quest} &\to \text{Had Subj V3 TimeAdv?} \\ \text{Subj} &\to \text{nouns} \mid \text{pronouns} \\ \text{V3} &\to \text{verbs\_V3} \\ \text{TimeAdv} &\to \text{adverbs} \mid \epsilon \end{aligned}$$

2.1.4 Примеры выводов

1. Утвердительное предложение:

$$S \Rightarrow \text{Affirm} \Rightarrow \text{Subj had V3 TimeAdv} \Rightarrow \text{She had finished already.}$$

2. Отрицательное предложение:

$$S \Rightarrow \text{Neg} \Rightarrow \text{Subj had not V3 TimeAdv} \Rightarrow \text{They had not seen the film before.}$$

3. Вопросительное предложение:

$$S \Rightarrow \text{Quest} \Rightarrow \text{Had Subj V3 TimeAdv?} \Rightarrow \text{Had you visited London previously?}$$

2.2 БНФ задаваемого подмножества языка

Форма Бэкуса-Наура (БНФ) — форма представления контекстно-свободных формальных грамматик, в которой правые части продукций с одинаковой левой частью объединены. Расширенная БНФ включает «регулярные» конструкции: итерацию { a }, необязательность [a], группировку (a | b).

Грамматика подмножества языка в Past Perfect в расширенной форме Бэкуса-Наура:

```
<sentence>:= (affirmative | negative | question) ["." | "?" | "!"]

<affirmative>:= <subject> "had" <past_participle> <object> <adverbial>*
<negative>:= <subject> "had" "not" <past_participle> <object> <adverbial>*
<question>:= "had" <subject past_participle> <object>? <adverbial>*

<subject>:= <pronoun> | <noun_phrase>
<pronoun>:= "i" | "you" | "he" | "she" | "it" | "we" | "they"

<noun_phrase>:= (<determiner>)? <adjective>* (<noun> | <proper_noun> | <date>)
<determiner>:= "the" | "a" | "an" | "my" | "your" | "his" | "her" | "its"
| "our" | "their"

<adjective>:= "angry" | "happy" | "sad" | "excited" | "tired"
<noun>:= "movie" | "book" | "picture" | "performance" | "letter" |
"report" | "cat" | "dogs" | "bird" | "home" | "school" |"answer"

<proper_noun>:= "xuxa" | "bobik" | "tuzik" | "margo" | "pushok" |
"funtik" | "teacher" | "student" | "woman" | "girl" | "man"

<date>:= "2021", "2022", "2020" | "2023" | "2024"

<past_participle>:= <regular_past> | <irregular_past>
<regular_past>:= "killed" | "finished" | "saved" | "counted" | "painted" |
"visited" | "read" | "written" | "started" |
"returned" | "arrived" | "come"|"explained"

irregular_past: "eaten" | "seen" | "gone" | "taken" | "made" | "known"

<object>:= <noun_phrase>
<adverbial>:= (<adverb> | <prepositional_phrase> | <clause> | <date_phrase>)+
<adverb>:= "quickly" | "happily" | "yesterday"
<prepositional_phrase>:= <preposition> <noun_phrase>
<preposition>:= "before" | "after" | "in" | "on" | "at" | "by" | "from" | "to"
<clause>:= <sub_conj> <simple_sentence>
sub_conj: "before" | "after" | "when" | "while" | "because"

<date_phrase>:= ("by" | "after") <date>

<simple_sentence>:= (<simple_affirmative> | <simple_negative> | <simple_question>)
<simple_affirmative>:= <subject> <simple_verb object>?
<simple_negative>:= <subject> "did" "not" <simple_verb> <object>?
<simple_question>:= "did" <subject> <simple_verb> <object>?
```

```
<simple_verb>:= "explained" | "left" | "eaten" | "gone" | "visited" | "started"
| "returned" | "arrived" | "finished" | "read"
```

Примеры предложений

1. Утвердительное:

```
He had returned an angry teacher.
```

2. Отрицательное:

```
She had not started his sad performance.
```

3. Вопросительное:

```
Had they not written movie after boy went to Paris?
```

2.2.1 Особенности грамматики

- Грамматика не может быть упрощена до автоматной из-за рекурсии в правилах для `<DeclarativePast>` и `<PlaceAdjunct>`.
- Поддерживает сложные временные конструкции с `before/after/by`.
- Позволяет выражать завершенность действия относительно другого прошлого события.
- Включает варианты с наречиями `already/never/yet`.

Пример дерева разбора представлен на рис.1.

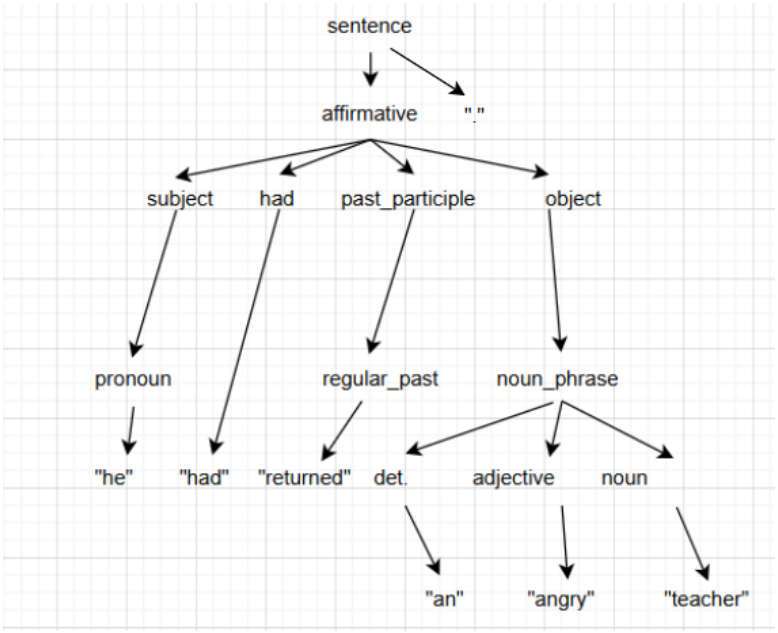

Рис. 1: Дерево разбора

3 Особенности реализации

3.1 Структуры данных

Следующие структуры данных представляют терминальные символы грамматики:

3.1.1 Местоимения (Pronouns)

```
pronoun = ["I", "you", "he", "she", "it", "we", "they"]
```

3.1.2 Определители (Determiners)

```
determiner = ["the", "a", "an", "my", "your", "his",
"her", "its", "our", "their"]
```

3.1.3 Прилагательные (Adjectives)

```
adjective = ["angry", "happy", "sad", "excited", "tired"]
```

3.1.4 Существительные (Nouns)

```
noun = ["movie", "book", "picture", "performance", "letter",
"report", "cat", "dogs", "bird", "home", "school", "answer"]
```

3.1.5 Имена собственные (Proper Nouns)

```
proper_noun = ["Xuxa", "Bobik", "Tuzik", "Margo", "Pushok",
"Funtik", "teacher", "student", "woman", "girl", "man"]
```

3.1.6 Даты (Dates)

```
date = ["2020", "2021", "2022", "2023", "2024"]
```

3.1.7 Глаголы (Verbs)

- Правильные формы Past Participle:

```
regular_past = ["killed", "finished", "saved", "counted",
"painted", "visited", "read", "written", "started", "returned",
"arrived", "come", "explained"]
```

- Неправильные формы Past Participle:

```
irregular_past = ["eaten", "seen", "gone",
     "taken", "made", "known"]
```

- Простые формы глаголов:

```
simple_verb = ["explained", "left", "eaten",
    "gone", "visited", "started", "returned", "arrived",
    "finished", "read"]
```

3.1.8 Наречия (Adverbs)

```
adverb = ["quickly", "happily", "yesterday", "suddenly", "quietly"]
```

3.1.9 Предлоги и союзы

- Предлоги:

```
preposition = ["before", "after", "in", "on", "at", "by", "from", "to"]
```

- Подчинительные союзы:

```
sub_conj = ["before", "after", "when", "while", "because"]
```

3.2 Основные функции

3.2.1 validate_past_perfect(sentence)

Назначение	Проверяет, соответствует ли предложение грамматике Past Perfect
Входные данные	• `sentence: str` - предложение для проверки
Выходные данные	• `bool` - `True` если предложение валидно, `False` если нет
Описание	Использует парсер Lark для проверки соответствия грамматике. При ошибке парсинга возвращает `False`.

3.2.2 interactive_validation()

Назначение	Интерактивный режим проверки предложений
Входные данные	Предложения введенные пользователем
Выходные данные	Результат валидации - строка
Описание	Бесконечный цикл, запрашивающий предложения для проверки до ввода 'q'. Для каждого предложения вызывает `validate_past_perfect()` и выводит результат.

3.2.3 generate_sentence()

Назначение	Генерирует случайное предложение в Past Perfect
Входные данные	Псевдослучайное число
Выходные данные	• `str` - сгенерированное предложение с правильной пунктуацией
Описание	Случайно выбирает между утвердительным, отрицательным и вопросительным типом, вызывает соответствующую функцию генерации и добавляет правильную пунктуацию.

3.3 Функции генерации частей предложения

3.3.1 generate_subject()

Назначение	Генерирует подлежащее
Входные данные	Псевдослучайное число
Выходные данные	• `str` - подлежащее (местоимение или именная группа)
Описание	С вероятностью 70% выбирает местоимение, иначе генерирует именную группу (без дат).

3.3.2 generate_noun_phrase(exclude_date=False)

Назначение	Генерирует именную группу
Входные параметры	• `exclude_date: bool` - исключать ли даты из генерации
Выходные данные	• `str` - именная группа
Описание	Генерирует группу с возможными: определителем (с правильным артиклем), 0-2 прилагательными и существительным/именем собственным/датой.

3.3.3 generate_past_participle()

Назначение	Выбирает форму Past Participle
Входные данные	Псевдослучайное число
Выходные данные	• `str` - глагол в форме Past Participle
Описание	Случайно выбирает из списков правильных и неправильных глаголов.

3.3.4 generate_object()

Назначение	Генерирует дополнение
Входные данные	Псевдослучайное число
Выходные данные	• `str` - дополнение (именная группа)
Описание	Вызывает `generate_noun_phrase()` для генерации дополнения.

3.3.5 generate_adverbial()

Назначение	Генерирует обстоятельство
Входные данные	Псевдослучайное число
Выходные данные	• `str` - обстоятельство
Описание	Случайно выбирает между: наречием, предложной группой, придаточным предложением или указанием даты.

3.3.6 generate_adverbials()

Назначение	Генерирует несколько обстоятельств
Входные данные	Псевдослучайное число
Выходные данные	• `str` - последовательность обстоятельств
Описание	С 50% вероятностью добавляет хотя бы одно обстоятельство, затем с 30% вероятностью добавляет дополнительные.

3.4 Функции генерации типов предложений

3.4.1 generate_affirmative()

Назначение	Генерирует утвердительное предложение
Входные данные	Псевдослучайное число
Выходные данные	• `str` - предложение в форме Past Perfect (без пунктуации)
Описание	Генерирует структуру: Подлежащее + "had"+ Past Participle + Дополнение + Обстоятельства.

3.4.2 generate_negative()

Назначение	Генерирует отрицательное предложение
Входные данные	Псевдослучайное число
Выходные данные	• `str` - отрицательное предложение (без пунктуации)
Описание	Генерирует структуру: Подлежащее + "had not"+ Past Participle + Дополнение + Обстоятельства.

3.4.3 generate_question()

Назначение	Генерирует вопросительное предложение
Входные данные	Псевдослучайное число
Выходные данные	• `str` - вопрос (без вопросительного знака)
Описание	Генерирует структуру: "had"+ Подлежащее + Past Participle + (Дополнение с 70% вероятностью) + Обстоятельства.

3.4.4 generate_simple_sentence()

Назначение	Генерирует простое предложение для придаточных
Входные данные	Псевдослучайное число
Выходные данные	• `str` - простое предложение в Past Simple
Описание	Случайно выбирает между утвердительной и отрицательной формой, использует простые глаголы.

3.5 Код программы

```
from lark import Lark
import random

GRAMMAR = """
     start: sentence

    sentence: question_punct | affirmative_punct | negative_punct

    question_punct: question "?"
    affirmative_punct: affirmative "."
    negative_punct: negative "."

    question: "had" subject past_participle object? adverbial*
    affirmative: subject "had" past_participle object adverbial*
    negative: subject "had" "not" past_participle object adverbial*

    subject: pronoun | noun_phrase
    pronoun: "i" | "you" | "he" | "she" | "it" | "we" | "they"

    noun_phrase: (determiner)? adjective* (noun | proper_noun | date)
    determiner: "the" | "a" | "an" | "my" | "your" | "his" | "her" | "its
       " | "our" | "their"
    adjective: "angry" | "happy" | "sad" | "excited" | "tired"
    noun: "movie" | "book" | "picture" | "performance" | "letter" | "
       report" | "cat" | "dogs" | "bird" | "home" | "school" |"answer"
    proper_noun: "xuxa" | "bobik" | "tuzik" | "margo" | "pushok" | "
       funtik" | "teacher" | "student" | "woman" | "girl" | "man"

    date: "2021"| "2022" |"2020" | "2023" | "2024"
```

```
27
28      past_participle: regular_past | irregular_past
29      regular_past: "killed" | "finished" | "saved" | "counted" | "painted"
            | "visited" | "read" | "written" | "started" | "returned" | "
            arrived" | "come"|"explained"
30      irregular_past: "eaten" | "seen" | "gone" | "taken" | "made" | "known
            "
31
32      object: noun_phrase
33      adverbial: (adverb | prepositional_phrase | clause | date_phrase)+
34      adverb: "quickly" | "happily" | "yesterday" | /[a-z]+ly/
35      prepositional_phrase: preposition noun_phrase
36      preposition: "before" | "after" | "in" | "on" | "at" | "by" | "from"
            | "to"
37      clause: sub_conj simple_sentence
38      sub_conj: "before" | "after" | "when" | "while" | "because"
39
40      date_phrase: ("by" | "after") date
41
42      simple_sentence: (simple_affirmative | simple_negative | simple_
            question)
43      simple_affirmative: subject simple_verb object?
44      simple_negative: subject "did" "not" simple_verb object?
45      simple_question: "did" subject simple_verb object?
46      simple_verb: "explained" | "left" | "eaten" | "gone" | "visited" | "
            started" | "returned" | "arrived" | "finished" | "read"
47
48      %import common.WS
49      %ignore WS
50  """
51
52  parser = Lark(GRAMMAR, parser='earley')
53
54  def interactive_validation():
55      print("Past Perfect Tense Validator")
56      print("Enter sentences to validate (press 'q' to exit):")
57      print("=" * 50)
58
59      while True:
60          sentence = input("> ").strip()
61          if sentence.lower() == 'q':
62              print("Exiting...")
63              break
64
65          result = validate_past_perfect(sentence)
66          print(sentence)
67          print(f"Result: {'VALID' if result else 'INVALID'}")
68          print("=" * 50)
69
70  def validate_past_perfect(sentence):
71      try:
72          parser.parse(sentence.lower())
73          return True
74      except Exception as e:
75          print(f"Error parsing: {e}")
76          return False
77
78  pronoun = ["I", "you", "he", "she", "it", "we", "they"]
79  determiner = ["the", "a", "an", "my", "your", "his", "her", "its", "our",
        "their"]
80  adjective = ["angry", "happy", "sad", "excited", "tired"]
```

```
noun = ["movie", "book", "picture", "performance", "letter", "report", "cat", "dogs", "bird", "home", "school", "answer"]
proper_noun = ["Xuxa", "Bobik", "Tuzik", "Margo", "Pushok", "Funtik", "teacher", "student", "woman", "girl", "man"]
date = ["2020", "2021", "2022", "2023", "2024"]
regular_past = ["killed", "finished", "saved", "counted", "painted", "visited", "read", "written", "started", "returned", "arrived", "come", "explained"]
irregular_past = ["eaten", "seen", "gone", "taken", "made", "known"]
adverb = ["quickly", "happily", "yesterday", "suddenly", "quietly"]
preposition = ["before", "after", "in", "on", "at", "by", "from", "to"]
sub_conj = ["before", "after", "when", "while", "because"]
simple_verb = ["explained", "left", "eaten", "gone", "visited", "started", "returned", "arrived", "finished", "read"]

def generate_sentence():
    sentence_type = random.choice(['affirmative', 'negative', 'question'])

    if sentence_type == 'affirmative':
        sentence = generate_affirmative()
        return sentence[0].upper() + sentence[1:] + '.'
    elif sentence_type == 'negative':
        sentence = generate_negative()
        return sentence[0].upper() + sentence[1:] + '.'
    else:
        sentence = generate_question()
        return sentence[0].upper() + sentence[1:] + '?'

def generate_subject():
    if random.random() < 0.7:
        return random.choice(pronoun)
    else:
        return generate_noun_phrase(exclude_date=True)

def generate_noun_phrase(exclude_date=False):
    parts = []
    if random.random() < 0.6:
        next_word = ""
        if random.random() < 0.5:
            next_word = random.choice(adjective + noun + proper_noun)
        else:
            next_word = random.choice(noun + proper_noun)

        if next_word[0].lower() in ['a', 'e', 'i', 'o', 'u'] and random.choice(determiner) in ['a', 'an']:
            parts.append('an' if next_word[0].lower() in ['a', 'e', 'i', 'o', 'u'] else 'a')
        else:
            parts.append(random.choice([d for d in determiner if d not in ['a', 'an']]))

    parts += random.sample(adjective, k=random.randint(0, 2))

    choices = ['noun', 'proper_noun']
    if not exclude_date and random.random() < 0.2:
        choices.append('date')

    noun_type = random.choice(choices)

    if noun_type == 'noun':
```

```
133             parts.append(random.choice(noun))
134         elif noun_type == 'proper_noun':
135             parts.append(random.choice(proper_noun))
136         else:
137             parts.append(random.choice(date))
138 
139         return ' '.join(parts)
140 
141     def generate_past_participle():
142         return random.choice(regular_past + irregular_past)
143 
144     def generate_object():
145         return generate_noun_phrase()
146 
147     def generate_adverbial():
148         adverbial_type = random.choice(['adverb', 'prepositional_phrase', '
                clause', 'date_phrase'])
149 
150         if adverbial_type == 'adverb':
151             return random.choice(adverb)
152         elif adverbial_type == 'prepositional_phrase':
153             return f"{random.choice(preposition)} {generate_noun_phrase()}"
154         elif adverbial_type == 'clause':
155             return f"{random.choice(sub_conj)} {generate_simple_sentence().
                    lower()}"
156         else:
157             return f"{random.choice(['by', 'after'])} {random.choice(date)}"
158 
159     def generate_adverbials():
160         adverbials = []
161         if random.random() < 0.5:
162             adverbials.append(generate_adverbial())
163             while random.random() < 0.3:
164                 adverbials.append(generate_adverbial())
165         return ' '.join(adverbials)
166 
167     def generate_affirmative():
168         parts = [
169             generate_subject().lower(),
170             "had",
171             generate_past_participle(),
172             generate_object(),
173             generate_adverbials()
174         ]
175         return ' '.join([p for p in parts if p]).strip()
176 
177     def generate_negative():
178         parts = [
179             generate_subject().lower(),
180             "had not",
181             generate_past_participle(),
182             generate_object(),
183             generate_adverbials()
184         ]
185         return ' '.join([p for p in parts if p]).strip()
186 
187     def generate_question():
188         parts = [
189             "had",
190             generate_subject().lower(),
191             generate_past_participle(),
```

```
            generate_object() if random.random() < 0.7 else "",
            generate_adverbials()
        ]
        return ' '.join([p for p in parts if p]).strip()

    def generate_simple_sentence():
        sentence_type = random.choice(['simple_affirmative', 'simple_negative'])

        if sentence_type == 'simple_affirmative':
            return f"{random.choice(pronoun).lower()} {random.choice(simple_verb)} {generate_object()}"
        else:
            return f"{random.choice(pronoun).lower()} did not {random.choice([v for v in simple_verb if v not in irregular_past])} {generate_object()}"

    for _ in range(10):
        sentence = generate_sentence()
        is_valid = validate_past_perfect(sentence)
        print(f"Generated: {sentence}")
        print(f"Validation: {'VALID' if is_valid else 'INVALID'}")
        print("=" * 60)
```

4 Результаты работы программы

Ниже представлены результаты работы функции генерации и валидации.

```
Генерация предложений:
Generated: I had known tired report.
Validation: VALID
============================================================
Generated: They had finished our angry sad picture.
Validation: VALID
============================================================
Generated: Had they made excited sad teacher?
Validation: VALID
============================================================
Generated: We had not finished their tired excited performance.
Validation: VALID
============================================================
Generated: It had not explained excited movie.
Validation: VALID
============================================================
Generated: It had not killed Pushok.
Validation: VALID
============================================================
Generated: She had started sad angry movie because it finished tired letter.
Validation: VALID

Проверка предложений:
They had not killed the sad happy woman.
Result: VALID

They had not killed the sad happy woman?
Result: INVALID

They had not killed the sad happy woman
Result: INVALID

They had killed the sad happy woman.
Result: VALID

They had killed the sad happy woman?
Result: INVALID

They had killed the sad happy woman
Result: INVALID

They had not killed the sad happy.
Result: INVALID

They not killed the sad happy woman.
Result: INVALID
```

Заключение

В ходе лабораторной работы были разработаны анализатор и генератор предложений подмножества английского языка в прошедшем совершенном времени (Past Perfect). Для этого была создана контекстно-свободная грамматика, соответствующая второму типу в иерархии Хомского. На основе этой грамматики была записана БНФ-нотация. Программа написана на языке Python.

Анализатор способен обрабатывать подмножество естественного языка. Однако у него есть ограничения: жесткая структура и отсутствие смысловой связи между словами в сгенерированных предложениях.

Достоинства

- Модульность кода. Каждая функция выполняет одну задачу (например, generate_subject(), generate_object()).

- Код поддерживает случайную генерацию разных типов предложений (утвердительные, отрицательные, вопросы). Учитывается вероятность появления разных элементов (например, обстоятельств с вероятностью 50%).

Недостатки

- Ограниченный словарный запас. Списки слов (noun, adjective, adverb и т. д.) фиксированы и небольшие. Нет возможности легко расширять их без изменения кода.

- Из-за случайной генерации могут получаться грамматически правильные, но странные фразы (например, "Had they eaten the happy 2023?").

Для масштабирования программы можно расширить словари и добавить новые слова. Также стоит рассмотреть методы подбора связанных слов для более естественного звучания предложений. Одним из возможных решений может быть использование эмбеддингов, обученных на готовых текстах.

Список литературы

[1] Электронный ресурс ВШТИИ
URL:https://tema.spbstu.ru/compiler/
(Дата обращения: 13.04.2025).

[2] Карпов, Ю. Г. Теория автоматов, Санкт-Петербург : Питер, 2003.
URL:https://djvu.online/file/eeLVKnyRZPXfl
(Дата обращения: 17.04.2025).