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

Отчет по дисциплине «Математическая логика и теория автоматов»

Лабораторная работа №3

«Построение контексно-свободной грамматики подмножества естественного языка»

Обучающийся: _____ Шклярова Ксения Алексеевна

Группа: 5130201/20102

Руководитель: _____ Востров Алексей Владимирович

«_____»_____20___г.

Санкт-Петербург, 2025

Содержание

Введение

Целью данной лабораторной работы является разработка контекстно-свободной грамматики для подмножества естественного языка. В соответствии с выбранным вариантом, необходимо построить грамматику для английского языка, позволяющую генерировать утвердительные, отрицательные и вопросительные предложения во времени Present Perfect Continuous. Так же требуется реализовать программу, которая не только генерирует предложения, но и проверяет их на соответствие следующим критериям: использование только слов из заданного словаря и соблюдение грамматических правил, характерных для выбранного времени.

1 Описание грамматики Present Perfect Continuous

Present Perfect Continuous выражает длительное действие, которое началось в прошлом, происходило до настоящего момента или все еще происходит и имеет результат в настоящее время.

1.1 Примеры использования

- Действие началось в прошлом и продолжается в настоящем. Построить такое предложение помогут слова-спутники или вспомогательные слова в Present Perfect Continuous: for, since, all morning, lately, recently, for quite a while.

  — Since early morning I have been writing this essay.

- Present Perfect Continuous обозначает действие, которое началось когда-то в прошлом, длилось какое-то время и только что или совсем недавно завершилось.

  — My headache is killing me because I have been taking this new medicine.

- Present Perfect Continuous просто создано для повседневных разговоров, когда хочется поделиться новостями. Все дело в том, что именно с помощью этого времени можно спросить, чем недавно занимался человек.

  — Haruto: What have you all been doing recently?

  — Liam: I've been playing football a lot. I'm trying to become better at it so I've been spending a lot of time on the field!

1.2 Формы предложений

1.2.1 Утвердительные предложения

Использование Present Perfect Continuous в утвердительных предложениях следует формуле:

I/You/We/They/ + have been ('ve been) + V-ing

He/She/It + has been ('s been) + V-ing

Напрмер:

— She has been living in New York since 1993.

1.2.2 Отрицательные предложения

Предложения Present Perfect Continuous с отрицанием строятся по следующей формуле:

I/You/We/They/ + have not been (haven't been) + V-ing

He/She/It + has not been (hasn't been) + V-ing

Например:

— You have not been attending your lectures since last month.

1.2.3 Вопросительные предложения

В английском языке при образовании вопросов вспомогательный глагол переходит на первое место в предложении. В «презент перфект прогрессив» на первом месте стоит to have:

— It has been raining => Has it been raining?

— You have been trying to call her => Have you been trying to call her?

Вопрос. слово	Вспом. глагол	Подлежащее	Вспом. глагол + смысловой глагол
What	have	I/you/we/they	been + основная форма глагола + окончание ing?
Who	has	he/she/it	
Where			
When			
Why			
How long			

Рис. 1. Таблица формирования вопросительных предложений в Present Perfect Continuous

2 Математическое описание

2.1 Модель Хомского: порождающие грамматики

Определение: порождающая грамматика Хомского: $G = (T, N, S, R)$, где:

- T — конечное множество символов (терминальный словарь)

- N — конечное множество символов (НЕ терминальный словарь)

- $S \in N$ — начальный нетерминал

- R — конечное множество правил вида $\alpha \to \beta$, где α и β — цепочки над словарем $T \cup N$

Определение: языком, порождаемым грамматикой G, называется множество терминальных цепочек, выводимых из начального символа грамматики:

$$L(G) = \{\alpha \in T^* \mid S \Rightarrow_G^* \alpha\}$$

Вывод цепочек языка начинается с начального нетерминального символа. Цепочка языка — это последовательность терминальных символов, полученная из начального нетерминала путем применения правил грамматики. Нетерминальные символы являются вспомогательными и используются только для порождения терминальных цепочек; они не входят в состав цепочек языка.

2.2 Формальное определение грамматики Present Perfect Continuous

Грамматика Past Continuous может быть формально определена как контекстно-свободная грамматика $G = (V, \Sigma, R, S)$, где:

- V - множество нетерминальных символов:

  V = {sentence, declarative_pres_perf_cont, question_pres_perf_cont, subject, singular_subject, plural_subject, singular_proper_noun, plural_proper_noun, singular_proper_phrase, singular_person, plural_proper_phrase, plural_person, aux_verb, duration_adjunct, predefined_duration, time_point, year, weekday, time_of_day, time_period, time_unit, v_ing, school_subject, noun_phrase, adjective, activity, object, place}

- Σ - множество терминальных символов:

  Σ = {'has', 'have', 'been', '.', '?', 'he', 'she', 'it', 'I', 'you', 'we', 'they', 'David', 'John', 'Mary', 'Emma', 'Berlin', 'Paris', 'London', 'Tokyo', 'New York', 'France', 'Germany', 'the Smiths', 'girl', 'boy', 'man', 'woman', 'teacher', 'student', 'dog', 'cat', 'baby', 'friend', 'colleague', 'owner of', 'friends', 'colleagues', 'children', 'people', 'students', 'since', 'for', 'all', 'recently', 'lately', 'for quite a while', 'since morning', 'all day', 'all night', 'for ages', 'for hours', 'last', 'yesterday', 'january', 'february', 'christmas', '2020', '2021', '2022', '2023', '2024', '1999', '2010', '2015', 'monday', 'tuesday', 'wednesday', 'thursday', 'friday', 'saturday', 'sunday', 'morning', 'afternoon', 'evening', 'night',

'dawn', 'dusk', 'midnight', 'noon', 'an hour', 'two hours', 'a day', 'a week', 'a month', 'years', 'decades', 'minutes', 'seconds', 'week', 'month', 'year', 'summer', 'winter', 'autumn', 'spring', 'season', 'waiting', 'working', 'studying', 'raining', 'sleeping', 'playing', 'reading', 'watching', 'living', 'teaching', 'learning', 'driving', 'math', 'history', 'english', 'science', 'pe', 'programming', 'art', 'music', 'physics', 'angry', 'happy', 'sad', 'excited', 'tired', 'busy', 'young', 'old', 'kind', 'homework', 'project', 'work', 'training', 'meal', 'assignment', 'research', 'experiment', 'movie', 'book', 'picture', 'performance', 'tv', 'documentary', 'game', 'play', 'concert'}

- R - множество правил вывода (см. БНФ ниже)
- S - начальный символ грамматики (*Sentence*)

2.3 Контекстно свободная грамматика

Грамматика является контекстно-свободной (относится к типу 2 по иерархии Хомского), что подтверждается следующими характеристиками:

- Все правила имеют имеют вид:

$$A \to \alpha$$

где $A \in V$ (одиночный нетерминал), а $\alpha \in (V \cup \Sigma)^*$.

- Правые части могут содержать любые комбинации терминалов и нетерминалов, но не зависят от контекста.

2.4 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 }, необязатель-ность [a], группировку (a | b).

Грамматика подмножества языка в Present Perfect Continuous в расширенной форме Бэкуса-Наура:

```
1 <sentence> ::= <declarative_pres_perf_cont> "."
2              | <question_pres_perf_cont> "?"
3
4 <declarative_pres_perf_cont> ::= <singular_subject> "has" ["not"] "been" <v_ing> <
      duration_adjunct>
5                                | <plural_subject> "have" ["not"] "been" <v_ing> <
                                       duration_adjunct>
6
```

```
7  <question_pres_perf_cont> ::= "has" <singular_subject> ["not"] "been" <v_ing> <
       duration_adjunct>
8                              | "have" <plural_subject> ["not"] "been" <v_ing> <
                                   duration_adjunct>
9
10 <subject> ::= <singular_subject> | <plural_subject>
11
12 <singular_subject> ::= "he" | "she" | "it" | <singular_proper_noun> | <
       singular_noun_phrase>
13 <plural_subject> ::= "i" | "you" | "we" | "they" | <plural_proper_noun> | <
       plural_noun_phrase>
14
15 <singular_proper_noun> ::= "david" | "john" | "mary" | "emma" | "berlin" | "paris" | "
       london" | "tokyo"
16                           | "new york" | "france" | "germany"
17 <plural_proper_noun> ::= "the smiths"
18
19 <singular_noun_phrase> ::= ["the"] <adjective_list> <singular_person>
20 <plural_noun_phrase>   ::= ["the"] <adjective_list> <plural_person>
21
22 <adjective_list> ::= <adjective> <adjective_list> |  <adjective_list> <adjective>|
23
24 <singular_person> ::= "girl" | "boy" | "man" | "woman" | "teacher" | "student" | "dog"
       | "cat" | "baby" | "friend" | "colleague" | "owner of" <subject>
25 <plural_person> ::= "friends" | "colleagues" | "children" | "people" | "students"
26
27 <duration_adjunct> ::= ["since" <time_point>]
28                       | ["for" <time_period>]
29                       | ["all" <time_of_day>]
30                       | <predefined_duration>
31
32 <predefined_duration> ::= "recently" | "lately" | "for quite a while" | "since morning"
33                           | "all day" | "all night" | "for ages" | "for hours"
34
35 <time_point> ::= <year> | <weekday> | "the" <time_of_day> | "last" <time_unit>
36                 | "yesterday" | "january" | "february" | "christmas"
37
38 <year> ::= "2020" | "2021" | "2022" | "2023" | "2024" | "1999" | "2010" | "2015"
39 <weekday> ::= "monday" | "tuesday" | "wednesday" | "thursday" | "friday" | "saturday" |
       "sunday"
40 <time_of_day> ::= "morning" | "afternoon" | "evening" | "night" | "dawn" | "dusk" | "
       midnight" | "noon"
41 <time_period> ::= "an hour" | "two hours" | "a day" | "a week" | "a month" | "years" |
       "decades" | "minutes" | "seconds"
```

```
<time_unit> ::= "week" | "month" | "year" | "summer" | "winter" | "autumn" | "spring" |
    "season"

<v_ing> ::= "waiting" ["for" <subject>]
          | "working" ["on" <activity>]
          | "studying" [<school_subject>]
          | "raining"
          | "sleeping"
          | "playing" ["with" <subject>]
          | "reading" [<object>]
          | "watching" [<object>]
          | "living" ["in" <place>]
          | "teaching" [<school_subject>]
          | "learning" [<school_subject>]
          | "driving" ["to" <place>]

<school_subject> ::= "math" | "history" | "english" | "science" | "pe" | "programming"
    | "art" | "music" | "physics"

<adjective> ::= "angry" | "happy" | "sad" | "excited" | "tired" | "busy" | "young" | "
    old" | "kind"

<activity> ::= "homework" | "project" | "work" | "training" | "meal" | "assignment" | "
    research" | "experiment"

<object> ::= "movie" | "book" | "picture" | "performance" | "tv" | "documentary" | "
    game" | "play" | "concert"

<place> ::= "london" | "new york" | "paris" | "berlin" | "tokyo" | "france" | "germany"
```

Примеры предложений по этой грамматике:

Утвердительные:

— She has been working on her project for a week.

— They have been waiting for you since morning.

Отрицательные:

— He has not been studying math recently.

— We have not been playing football all evening.

Вопросительные:

— Has the teacher been explaining the lesson for an hour?

— Have you been sleeping all afternoon?

В данной грамматике наблюдается двусторонняя рекурсия в правилах:

<adjective_list> ::= <adjective> <adjective_list> | <adjective_list> <adjective>| ϵ

Так же в данной грамматике присутствует косвенная рекурсия в правилах вида:

<subject> ::= <singular_noun_phrase> | <plural_noun_phrase>

<singular_noun_phrase> ::= [the] <adjective>* <singular_person>

<singular_person> ::= girl | ... | owner of <subject>

Пример дерева построения предложения представлен на рис. 2:

Рис. 2. Пример дерева построения предложения

3 Особенности реализации

3.1 Генерация предложений

Словарь vocab хранит все варианты слов для генерации предложений, сгруппированные по грамматическим категориям.

Реализацию см. листинг 1.

Листинг 1. Реализация хранения словаря

```
vocab = {
    "Pronoun": ['I', 'you', 'he', 'she', 'it', 'they', 'we'],
    "Person": ['girl', 'boy', 'man', 'woman', 'teacher', 'student', 'dog', 'cat', 'baby', 'friend', 'colleague'],
    "ProperNoun": ['John', 'Mary', 'London', 'Paris', 'Emma', 'David', 'Tokyo', 'Berlin'],
    "Adjective": ['angry', 'happy', 'sad', 'excited', 'tired', 'busy', 'young', 'old', 'kind'],
    "Activity": ['homework', 'project', 'work', 'training', 'meal', 'assignment', 'research', 'experiment'],
    "Subject": ['math', 'history', 'english', 'science', 'PE', 'programming', 'art', 'music', 'physics'],
    "Object": ['movie', 'book', 'picture', 'performance', 'TV', 'documentary', 'game', 'play', 'concert'],
    "Place": ['London', 'New York', 'the city', 'a village', 'the country', 'France', 'Germany', 'home'],
    "Year": ['2020', '2021', '2022', '2023', '2024', '1999', '2010', '2015'],
    "Weekday": ['Monday', 'Tuesday', 'Wednesday', 'Thursday', 'Friday', 'Saturday', 'Sunday'],
    "TimeOfDay": ['morning', 'afternoon', 'evening', 'dawn', 'dusk', 'midnight', 'noon'],
    "TimePeriod": ['an hour', 'two hours', 'a day', 'a week', 'a month', 'years', 'decades', 'minutes', 'seconds'],
    "TimeUnit": ['week', 'month', 'year', 'summer', 'winter', 'autumn', 'spring', 'season'],
    "MiscDurations": [
        'recently', 'lately', 'for quite a while', 'since morning',
        'all day', 'all night', 'for ages', 'for hours'
    ]
}
```

3.1.1 Генерация именных групп

Функция noun_phrase() генерирует именную группу в единственном числе с возможным рекурсивным вложением через конструкцию "owner of". Она принимает один параметр — глубину

рекурсии depth, которая по умолчанию равна 0. Это значение используется для ограничения уровня вложенности и предотвращения бесконечной рекурсии. Возвращает данная функция строку с именной группой. Если глубина превышает 1, функция сразу возвращает фиксированное значение "the student". Далее с определёнными вероятностями выбирается один из четырёх вариантов: местоимение (40%), имя собственное (20%), именная группа с прилагательными (20%) или рекурсивная структура (owner of X) с вероятностью 20%. В своей работе функция использует глобальный словарь vocab, в частности его ключи "Pronoun "ProperNoun "Adjective"и "Person".

Функция plural_noun_phrase() возвращает строку, представляющую именную группу во множественном числе. Она не принимает аргументов, но использует глобальные данные — список прилагательных из vocab["Adjective"]. Внутри функции случайным образом выбирается от нуля до двух прилагательных и одно существительное из фиксированного списка множественных форм (friends, colleagues, students и т. д.). Полученные слова соединяются в строку, к которой с вероятностью 50% добавляется артикль "the".

Функция subject() выбирает подлежащее для предложения. Она не принимает аргументов, но использует глобальные данные: словари vocab["Pronoun"] и vocab["ProperNoun"], а также вызывает другие функции (noun_phrase и plural_noun_phrase). На выходе возвращает строку с подлежащим. Случайным образом (по заданным вероятностям) подлежащее может быть: местоимением (25%), именем собственным (15%), фразой "the smiths"(15%), именной группой в единственном числе через noun_phrase() (20%) или множественным подлежащим через plural_noun_phrase() (25%).

Реализацию данных функций см. в листинге 2.

Листинг 2. Реализация генерации именных групп

```
def noun_phrase(depth=0):
    # Защита от бесконечной рекурсии в "owner of owner of ..."
    if depth > 1:
        return "the student"

    r = random.random()
    singular_persons = ["girl", "boy", "man", "woman", "teacher", "student",
                        "dog", "cat", "baby", "friend", "colleague"]

    if r < 0.4:
        return random.choice(vocab["Pronoun"]).lower()
    elif r < 0.6:
        return random.choice(vocab["ProperNoun"]).lower()
    elif r < 0.8:
        adjective_count = random.randint(0, 2)
        adjectives = [random.choice(vocab["Adjective"]).lower() for _ in range(
            adjective_count)]
```

```
        noun = random.choice(singular_persons)
        phrase = adjectives + [noun]
        return "the " + ' '.join(phrase) if random.random() < 0.5 else ' '.join(phrase)
    else:
        # "owner of" конструкция     рекурсивный вызов
        inner_subject = noun_phrase(depth + 1)
        return "owner of " + inner_subject

def plural_noun_phrase():
    plural_persons = ["friends", "colleagues", "children", "people", "students"]
    adjective_count = random.randint(0, 2)
    adjectives = [random.choice(vocab["Adjective"]).lower() for _ in range(
        adjective_count)]
    noun = random.choice(plural_persons)
    phrase = adjectives + [noun]
    return "the " + ' '.join(phrase) if random.random() < 0.5 else ' '.join(phrase)

def subject():
    r = random.random()
    if r < 0.25:
        return random.choice(vocab["Pronoun"]).lower()
    elif r < 0.4:
        return random.choice(vocab["ProperNoun"]).lower()
    elif r < 0.55:
        return "the smiths"  # plural_proper_noun по грамматике
    elif r < 0.75:
        return noun_phrase()
    else:
        return plural_noun_phrase()
```

3.1.2 Генерация глагольных групп

Функция v_ing генерирует глагол в форме Present Continuous с возможным дополнением. Она не принимает аргументов, но использует глобальные словари vocab["Activity"], vocab["Subject"], vocab["Object"], vocab["Place"], результатом выполнения является глагол с возможным дополнением. Внутри функции из списка выбирается одна пара: глагол и связанная с ним функция, которая возвращает дополнение (например, "for the teacher "on homework"). Для большинства глаголов существует 70% вероятность добавления такого дополнения.

Реализацию данной функции см. в листинге 3.

Листинг 3. Реализация генерации глагольных групп

```
def v_ing():
    choices = [
```

```
3         ("waiting", lambda: "for " + noun_phrase() if random.random() < 0.7 else ""),
4         ("working", lambda: "on " + random.choice(vocab["Activity"]) if random.random()
              < 0.7 else ""),
5         ("studying", lambda: random.choice(vocab["Subject"]) if random.random() < 0.7
              else ""),
6         ("raining", lambda: ""),
7         ("sleeping", lambda: ""),
8         ("playing", lambda: "with " + noun_phrase() if random.random() < 0.7 else ""),
9         ("reading", lambda: random.choice(vocab["Object"]) if random.random() < 0.7
              else ""),
10        ("watching", lambda: random.choice(vocab["Object"]) if random.random() < 0.7
              else ""),
11        ("living", lambda: "in " + random.choice(vocab["Place"]) if random.random() <
              0.7 else ""),
12        ("teaching", lambda: random.choice(vocab["Subject"]) if random.random() < 0.7
              else ""),
13        ("learning", lambda: random.choice(vocab["Subject"]) if random.random() < 0.7
              else ""),
14        ("cooking", lambda: random.choice(['dinner', 'lunch']) if random.random() < 0.7
               else ""),
15        ("driving", lambda: "to " + random.choice(vocab["Place"]) if random.random() <
              0.7 else "")
16    ]
17    verb, tail_func = random.choice(choices)
18    tail = tail_func()
19    return verb + (' ' + tail if tail else '')
```

3.1.3 Генерация обстоятельств

Функция time_point() генерирует выражение, обозначающее момент времени. Она не принимает аргументов, но использует глобальные списки годов, дней недели, времени суток и других временных меток. С вероятностью 20% она возвращает одну из следующих категорий: год, день недели, выражение вида "the"+ время суток, фразу "last"+ единица времени (например, "last week") или фиксированное слово.

Функция duration_adjunct создаёт обстоятельство длительности действия. Она не принимает аргументов, но использует функцию time_point() и глобальные списки временных периодов, частей суток и выражений длительности. По 25% вероятности функция возвращает один из следующих вариантов: конструкция "since"+ момент времени, "for"+ длительность, "all"+ часть суток, либо одно из предопределённых выражений длительности, таких как "for ages "recently "since morning".

Реализацию данных функций см. в листинге 4.

Листинг 4. Реализация генерации обстоятельств

```
def time_point():
    r = random.random()
    if r < 0.2:
        return random.choice(vocab["Year"])
    elif r < 0.4:
        return random.choice(vocab["Weekday"])
    elif r < 0.6:
        return "the " + random.choice(vocab["TimeOfDay"])
    elif r < 0.8:
        return "last " + random.choice(vocab["TimeUnit"])
    else:
        return random.choice(["yesterday", "january", "february", "christmas"])

def duration_adjunct():
    r = random.random()
    if r < 0.25:
         return "since " + time_point()
    elif r < 0.5:
        return "for " + random.choice(vocab["TimePeriod"])
    elif r < 0.75:
        return "all " + random.choice(vocab["TimeOfDay"])
    else:
        return random.choice(vocab["MiscDurations"])
```

3.1.4 Сборка предложения

Функция generate_sentence() генерации предложения. Она не принимает аргументов, но вызывает множество других функций: subject(), is_singular(), v_ing(), duration_adjunct(). Возвращает данная функция сформированное предложение. Сначала определяется тип предложения — вопросительное или утвердительное, а также наличие отрицания. Затем выбирается подлежащее, определяется его число, подбирается вспомогательный глагол ("has"или "have"), генерируется смысловой глагол с возможным дополнением и добавляется обстоятельство времени с вероятностью 50%. После этого собирается полное предложение в правильном порядке с соответствующим знаком препинания — точкой или вопросительным знаком.

Реализацию данной функций см. в листинге 5.

Листинг 5. Реализация функции для генерации предложений

```
def generate_sentence():
    question = random.choice([True, False])
    negative = random.choice([True, False])
    subj = subject()
```

```
    singular = is_singular(subj)
    aux = "has" if singular else "have"
    v = v_ing()
    duration = duration_adjunct()
    parts = []
    if question:
        parts.append(aux.capitalize())
        parts.append(subj)
        if negative:
            parts.append("not")
        parts.append("been")
        parts.append(v)
        if random.choice([True, False]) == True:
            parts.append(duration)
        return " ".join(parts) + "?"
    else:
        parts.append(subj.capitalize())
        parts.append(aux)
        if negative:
            parts.append("not")
        parts.append("been")
        parts.append(v)
        if random.choice([True, False]) == True:
            parts.append(duration)
        return " ".join(parts) + "."
```

3.2 Валидация предложений

3.2.1 Правила валидации

Класс SubjectVerbAgreement проверяет согласование подлежащего и сказуемого в предложениях Present Perfect Continuous.

Метод declarative_pres_perf_cont(self, tree) применяется к дереву разбора утвердительного предложения. Он принимает дерево tree, результатом выполнения является извлечение из него подлежащего и вспомогательного глагола, и их передача в метод _check_agreement() для проверки согласования. Если согласование нарушено — выбрасывается исключение.

Метод question_pres_perf_cont(self, tree) аналогичен предыдущему, но работает с деревом вопросительного предложения. Он принимает дерево, результатом выполнения является извлечение из него подлежащего и вспомогательного глагола, и их передача в метод _check_agreement() для проверки согласования.

Метод _check_agreement(self, subject, aux) проверяет согласование между подлежащим и вспомогательным глаголом. Он принимает subject (в виде дерева) и строку aux. Результатом

является ValueError, если было найдено несоответствие между подлежащим и вспомогательным глаголом. Сначала подлежащее приводится к строке с помощью _flatten(), затем нормализуется (удаляется "the переводится в нижний регистр). После этого подлежащее сравнивается с заранее заданными множествами единственных и множественных форм.

Метод _flatten(self, tree) используется для получения строкового представления поддерева. Он принимает дерево tree и рекурсивно объединяет все его терминалы в строку с пробелами между словами. Возвращает строковое представление всех терминалов в узле.

Реализацию данного класса см. в листинге 6.

Листинг 6. Реализация правил валидации

```
class SubjectVerbAgreement(Interpreter):
    def declarative_pres_perf_cont(self, tree):
        subject = tree.children[0]
        aux = tree.children[1]
        self._check_agreement(subject, str(aux))
    def question_pres_perf_cont(self, tree):
        aux = str(tree.children[0])
        subject = tree.children[1]
        self._check_agreement(subject, aux)
    def _check_agreement(self, subject, aux):
        subject_text = self._flatten(subject).lower().strip()

        # Убираем артикль 'the' перед проверкой
        if subject_text.startswith("the "):
            subject_text = subject_text[4:]

        singular_subjects = {"he", "she", "it", "david", "john", "mary", "emma", "
            student"}
        plural_subjects = {"i", "you", "we", "they", "students"}

        if subject_text in singular_subjects and aux != "has":
            raise ValueError(f"Subject-verb agreement error: '{subject_text}' with '{
                aux}'")
        elif subject_text in plural_subjects and aux != "have":
            raise ValueError(f"Subject-verb agreement error: '{subject_text}' with '{
                aux}'")

    def _flatten(self, tree):
        if isinstance(tree, Tree):
            return ' '.join(self._flatten(c) for c in tree.children)
        return str(tree)
```

3.2.2 Инициализация парсера

Для парсинга предложений была использована библиотека Lark. Lark - это современная библиотека для парсинга в Python, которая предоставляет мощные инструменты для работы с контекстно-свободными грамматиками.

Так же грамматика была описана в строковой переменной grammar в формате BNF (Backus-Naur Form).

Инициализация парсера parser = Lark(grammar, start='start', lexer='basic') : grammar - BNF-грамматика в строковом формате, start- Начальное правило грамматики ('sentence'), lexer - Использует базовый лексер Lark.

Реализацию данных функций см. в листинге 7.

Листинг 7. Реализация парсинга предложений

```
grammar = """
start: sentence

sentence: declarative_pres_perf_cont "."
        | question_pres_perf_cont "?"

declarative_pres_perf_cont: singular_subject "has" ["not"] "been" v_ing
    duration_adjunct | plural_subject "have" ["not"] "been" v_ing duration_adjunct
question_pres_perf_cont: "has" singular_subject ["not"] "been" v_ing duration_adjunct |
     "have" plural_subject ["not"] "been" v_ing duration_adjunct

subject: singular_subject | plural_subject

singular_subject: "he" | "she" | "it" | singular_proper_noun | singular_noun_phrase
plural_subject: "i" | "you" | "we" | "they" | plural_proper_noun | plural_noun_phrase

singular_proper_noun: "david" | "john" | "mary" | "emma" | "berlin" | "paris" | "london
    " | "tokyo"
                    | "new york" | "france" | "germany"
plural_proper_noun: "the smiths"

singular_noun_phrase: ["the"] adjective* singular_person
plural_noun_phrase: ["the"] adjective* plural_person

singular_person: "girl" | "boy" | "man" | "woman" | "teacher" | "student" | "dog" | "
    cat" | "baby" | "friend" | "colleague" | "owner of" subject
plural_person: "friends" | "colleagues" | "children" | "people" | "students"


duration_adjunct: ["since" time_point]
```

```
                | ["for" time_period]
                | ["all" time_of_day]
                | predefined_duration

predefined_duration: "recently" | "lately" | "for quite a while" | "since morning"
                   | "all day" | "all night" | "for ages" | "for hours"

time_point: year | weekday | "the" time_of_day | "last" time_unit
          | "yesterday" | "january" | "february" | "christmas"

year: "2020" | "2021" | "2022" | "2023" | "2024" | "1999" | "2010" | "2015"
weekday: "monday" | "tuesday" | "wednesday" | "thursday" | "friday" | "saturday" | "
    sunday"
time_of_day: "morning" | "afternoon" | "evening" | "night" | "dawn" | "dusk" | "
    midnight" | "noon"
time_period: "an hour" | "two hours" | "a day" | "a week" | "a month" | "years" | "
    decades" | "minutes" | "seconds"
time_unit: "week" | "month" | "year" | "summer" | "winter" | "autumn" | "spring" | "
    season"

v_ing: "waiting" ["for" noun_phrase]
     | "working" ["on" activity]
     | "studying" [school_subject]
     | "raining"
     | "sleeping"
     | "playing" ["with" noun_phrase]
     | "reading" [object]
     | "watching" [object]
     | "living" ["in" place]
     | "teaching" [school_subject]
     | "learning" [school_subject]
     | "cooking" ["dinner" | "lunch"]
     | "driving" ["to" place]

school_subject: "math" | "history" | "english" | "science" | "pe" | "programming" | "
    art" | "music" | "physics"

noun_phrase: subject

adjective: "angry" | "happy" | "sad" | "excited" | "tired" | "busy" | "young" | "old" |
     "kind"

activity: "homework" | "project" | "work" | "training" | "meal" | "assignment" | "
    research" | "experiment"
```

```
64
65 object: "movie" | "book" | "picture" | "performance" | "tv" | "documentary" | "game" |
       "play" | "concert"
66
67 place: "london" | "new york" | "the city" | "a village" | "the country" | "france" | "
       germany" | "home"
68
69 %import common.WS
70 %ignore WS
71 """
72
73 parser = Lark(grammar, start='start', lexer='basic')
```

3.2.3 Валидация предложений

Функция validate(sentence) проверяет грамматическую корректность переданного предложения. Она принимает строку sentence, если проверка прошла успешно, возвращается True. Если возникли ошибки при разборе или проверке согласования — возвращается False. Реализация функции заключается в следующем: сначала предложение приводится к нижнему регистру (для унификации), затем функция парсит его через parser.parse() => строит дерево разбора. Потом запускает SubjectVerbAgreement().visit(tree).

Реализацию данной функции см. в листинге 8.

Листинг 8. Реализация функции для валидации предложений

```
1 def validate(sentence):
2     try:
3         tree = parser.parse(sentence.lower())  # Приводим к нижнему регистру
4         SubjectVerbAgreement().visit(tree)
5         return True
6     except Exception as e:
7         return False
```

3.3 Вспомогательные функции

Функция is_singular(subj: str) определяет, является ли переданное подлежащее единственным числом. Она принимает строку subj и использует глобальные списки vocab["Pronoun"], vocab["ProperNoun"] и vocab["Person"], а также фиксированный набор множественных форм. В случае успеха возвращает True, иначе False. Внутри функция сначала удаляет артикль "the"(если он есть), приводит строку к нижнему регистру и проверяет, входит ли подлежащее в известные множественные формы. Если нет — проверяет, состоит ли оно только из прилагательных и единственного существительного.

Реализацию данной функции см. в листинге 9.

Листинг 9. Реализация вспомогательных функций

```
def is_singular(subj: str) -> bool:
    subj = subj.lower().replace("the ", "")
    singular = set(s.lower() for s in vocab["Pronoun"] + vocab["ProperNoun"] + vocab["Person"])
    plural = {"i", "you", "we", "they", "students", "friends", "children", "people", "colleagues"}

    if subj in plural:
        return False
    return subj in singular or all(word in vocab["Adjective"] + vocab["Person"] for word in subj.split())
```

3.4 Пользовательское меню

Функция get_user_input() -> str запрашивает у пользователя ввод предложения вручную, проверяет его на корректность (не пустой ввод) и возвращает строку или None (если пользователь решил отменить ввод). На вход принимает введенную пользователем строку. Возвращаемое значение: str — введённая пользователем строка с предложением, None — если пользователь ввёл q для отмены.

Функция show_menu() отображает консольное меню и вызывает соответствующую функцию. На вход принимает введенное пользователем значение, результатом выполнения является вызов соответствующей функции.

Реализацию данных функций см. в листинге 10.

Листинг 10. Реализация пользовательского меню

```
def get_user_input() -> str:
    print("\nВведите предложение на английском языке во времени Present Perfect Continuous:")

    while True:
        user_input = input("\nВведите предложение или 'q' для отмены: ").strip()

        # Проверка на команду выхода
        if user_input.lower() == 'q':
            return None

        # Проверка пустого ввода
        if not user_input:
            print("Ошибка: Пустой ввод. Пожалуйста, введите предложение.")
```

```
14              continue
15
16          return user_input
17
18  def show_menu():
19      while True:
20          print("\nМеню:")
21          print("1. Сгенерировать случайно предложение")
22          print("2. Ввести предложение вручную")
23          print("3. Выход")
24
25          choice = input("Выберите действие (1-3): ")
26
27          if choice == "1":
28              date = generate_sentence()
29              print(f"\nСгенерированное предложение: {date}")
30              print("Проверка корректности:", "Корректно" if validate(date) else "Некорре
                    ктно")
31          elif choice == "2":
32              date = get_user_input()
33              if validate(date):
34                  print(f"\nВведенное предложение: {date}")
35                  print("Проверка корректности: Корректно")
36              else:
37                  print(f"\nВведенное предложение: {date}")
38                  print("Проверка корректности: Некорректно")
39          elif choice == "3":
40              print("Выход из программы.")
41              break
42          else:
43              print("Некорректный ввод.")
```

4 Результаты работы программы

На рис. 3-7 показаны результаты работы программы.

```
Введите предложение на английском языке во времени Present Perfect Continuous:

Введите предложение или 'q' для отмены:
Ошибка: Пустой ввод. Пожалуйста, введите предложение.

Введите предложение или 'q' для отмены: She has been reading book Monday.

Введенное предложение: She has been reading book Monday.
Проверка корректности: Некорректно
```

Рис. 3. Ввод пустой строки в качестве предложения, ввод неправильного предложения

```
Меню:
1. Сгенерировать случайно предложение
2. Ввести предложение вручную
3. Выход
Выберите действие (1-3): 0
Некорректный ввод.

Меню:
1. Сгенерировать случайно предложение
2. Ввести предложение вручную
3. Выход
Выберите действие (1-3): 1

Сгенерированное предложение: Have the smiths not been waiting for owner of london?
Проверка корректности: Корректно
```

Рис. 4. Неверный ввод при выборе пункта в меню,генерация вопросительного предложения

```
Меню:
1. Сгенерировать случайно предложение
2. Ввести предложение вручную
3. Выход
Выберите действие (1-3): 1

Сгенерированное предложение: It has not been playing with owner of david since the afternoon.
Проверка корректности: Корректно
```

Рис. 5. Генерация отрицательного предложения

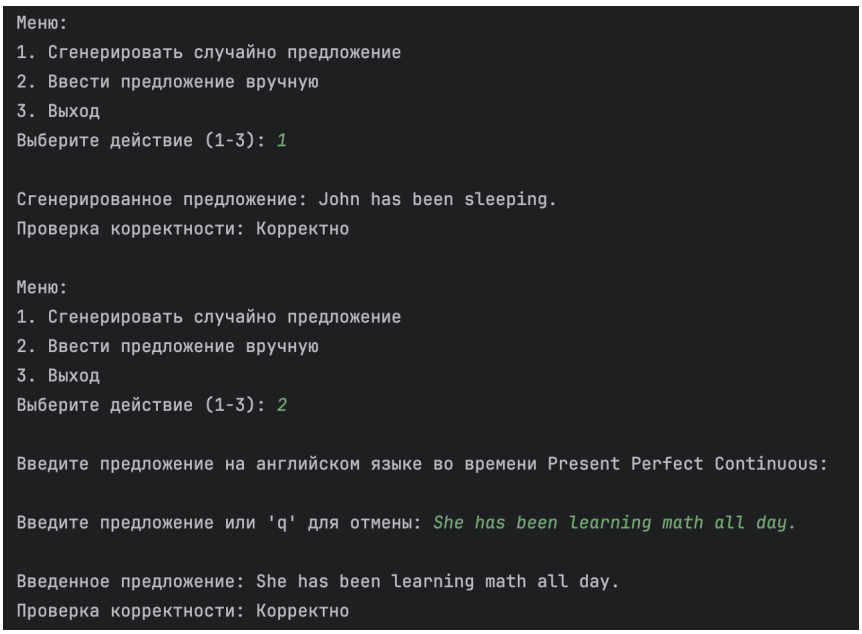

Рис. 6. Генерация утвердительного предложения, ввод утвердительного предложения

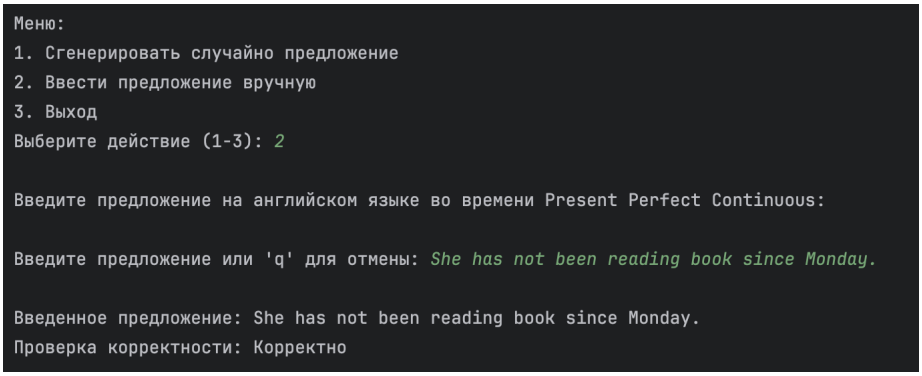

Рис. 7. Ввод отрицательного предложения

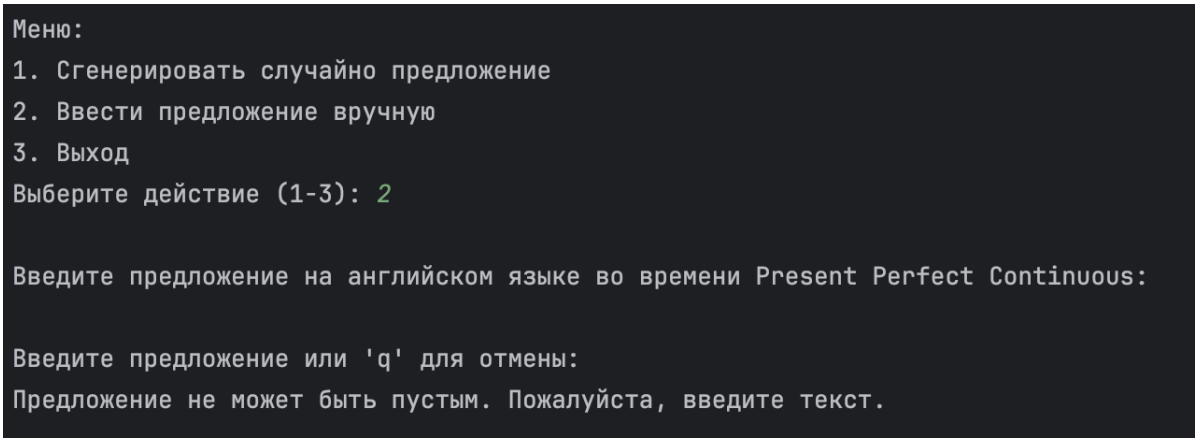

Рис. 8. Ввод пустого предложения

Заключение

В ходе выполнения лабораторной работы построена контексно-свободная грамматика подмножества Preset Perfect Continuous английского языка. Реализованна программа, которая предназначена для работы с предложениями во времени Present Perfect Continuous английского языка. Она включает две основные функции:

1. Генерация предложений — создаются утвердительные, отрицательные и вопросительные предложения, соответствующие заданной грамматике.

2. Проверка корректности предложений , введённых пользователем, на соответствие формальным правилам грамматики.

Для генерации предложений используются рекурсивные функции, которые строят подлежащее, глагольную группу и обстоятельства на основе заранее определённого словаря слов и вероятностного выбора конструкций. Это позволяет создавать разнообразные, но грамматически корректные предложения. При этом разнообразие генерируемых предложений может быть дополнительно увеличено за счёт расширения словаря vocab , что даст возможность использовать новые слова и выражения без изменения самой структуры грамматики.

Анализатор реализован с использованием библиотеки Lark и представляет собой синтаксический парсер, основанный на контекстно-свободной грамматике, описанной в формате БНФ.

Работа демонстрирует практическое применение теории формальных грамматик Хомского, в частности, контекстно-свободных грамматик. Реализованная грамматика порождает контекстно-свободный язык, заданный в нотации Бэкуса — Наура (БНФ). За счет рекурсивных правил (например, для определения subject) и комбинаторного подхода к синтезу предложений система генерирует бесконечное множество грамматически правильных высказываний.

Достоинства: генерация предложений осуществляется с применением вероятностного подхода (функция random.random()) для обеспечения разнообразия грамматических конструкций, так же поддерживаются рекурсивные структуры, таких как сложносочиненные именные группы, что позволяет создавать предложения различной сложности. Проверка грамматической корректности обеспечивает согласование подлежащего и сказуемого. Функция noun_phrase(depth) обеспечивает контроль глубины рекурсии при генерации именных групп, что позволяет регулировать сложность предложений.

Недостатки: все слова жестко закодированы в словаре vocab, что исключает возможность динамического расширения лексики. Рекурсивные вызовы и множественные проверки могут замедлять генерацию при больших объёмах.

Масштабирование: можно расширить набор поддерживаемых времён и грамматических конструкций. Кроме того, можно реализовать возможность динамической загрузки словарного запаса из внешних источников.

Работа выполнена на языке программирования Python в среде разработки PyCharm версии 2023.3.4.

Список литературы

[1] Вострoв, А. В. Математическая логика

URL:https://tema.spbstu.ru/compiler (Дата обращения: 23.03.2025).

[2] Сети, Р.; Ахо, А. Компиляторы: принципы, технологии и инструменты / Р. Сети, А. Ахо. - М.: Издательство «Наука», 2006. - С. 104.