Аннотация

В работе рассмотрены основные проблемы, возникающие при согласовании единственного и множественного числа в русском языке.

Предложен алгоритм, решающий задачу для предложений, состоящих из существительных, местоимений, личных глаголов и (или) инфинитивов (с, возможно, единичным перечислением личных глаголов или инфинитивов с зависимыми словами указанных выше частей речи).

Алгоритм реализован на языке python. При этом, в случае возникновения ошибок предложен понятный механизм, позволяющий исключить эти ошибки без изменения программного кода.

Содержание

1. Введение

На стыке лингвистики и computer science в середине XX века возникла компьютерная лингвистика. Это научное направление развивается по мере развития электронно-вычислительных машин [5].

Стоит отметить, что попытки исследовать структуру естественного языка математическими методами проводились достаточно давно. Например, известно [3], что русский математик Марков Андрей Андреевич, рассматривал распределение доли гласных и согласных в тексте «Евгений Онегин».

Но, как уже было замечено выше, появление и быстрое развитие вычислительной техники повлекло за собой возникновение новой дисциплины — компьютерная лингвистика, рождение которой принято связывать с Джорджтаунским экспериментом, когда впервые были продемонстрированы возможности перевода с русского текста на английский и наоборот.

Следующим и очень важным этапом развития дисциплины явилось появление работы «Синтаксические структуры» Наома Хомского [7], которое послужило основой для создания многочисленных синтетических языков (в том числе языков программирования).

Однако, очень быстро выяснилось, что методы Хомского (прежде всего, рекурсивный грамматический разбор) плохо применимы к естественным языкам. Так, в отличие от искусственных языков, где практически отсутствует проблема многозначности, в любом естественном языке большая часть лексики многозначна: особенно это касается активной части лексики, активного словаря: того, что используется чаще [10].

Естественный язык — это живое и постоянно развивающиеся явление, тесно связанное с культурными и историческими особенностями своего носителя, которое очень сложно формализовать.

Например, сложно объяснить иностранцу почему у слова *«река»* множественное число *«реки»*, а у слова *«сестра»* множественное число — *«сёстры»*.

Итак, постепенно сформировалась новая научная дисциплина — Natural Language Processing (NLP).

NLP ставит перед собой весьма амбициозную цель: создание алгоритмов обработки естественного языка, которые понимают и реагируют на текстовые или го-

лосовые данные, отвечая собственным текстом или речью, причём во многом таким же образом, как это делают люди.

Однако, несмотря на очевидные достижения в этой области, остаётся очень много неразрешенных проблем, одной из которых является проблема согласования.

В наиболее общем виде проблема согласования изложена в работе Якова Георгиевича Тестелец «Введение в общий синтаксис» [4].

Согласно его исследованию, в русском языке проблему согласования исходя из синтаксических правил можно разделить на следующие основные части: согласование по времени, роду, падежу и числу.

Например, рассмотрим предложение: «Я принял решение покусать собаку первым».

Можно проиллюстрировать проблему согласования следующими несогласованным предложениями (в скобках приведены примеры низменной синтаксической структуры без потери семантики):

1. Согласование по времени: «Я принял решение покусал собаку первым». («Я принял решение и покусал собаку первым»).

2. Согласование по роду: «Она принял решение покусать собаку первым». («Она приняла решение покусать собаку первой»).

3. Согласование по числу: «Мы принял решение покусать собаку первым». («Мы приняли решение покусать собаку первыми»).

4. Согласование по падежу: «Я принял решение покусать собака первым». («Я принял решение покусать собаку первым»).

Стоит отметить, что несмотря на кажущуюся очевидность и простоту проблемы с точки зрения общегуманитарных представлений, алгоритмически она не решена ни для одной из перечисленных выше частей [13].

Видимо, в первую очередь это обусловлено тем, что русский язык — явление живое, постоянно меняющиеся, где большинство синтаксических правил изобилуют многочисленными исключениями.

И то, что аналоговому прибору под названием человеческий мозг, особенно в период его бурного развития, представляется сравнительно несложной задачей, то

«объяснить» это машине Тьюринга, пусть и с очень быстрой считывающей головкой и очень длинной лентой, очень даже не просто.

Другими словами, построение алгоритма полностью решающую проблему согласования представляется чрезвычайно сложной задачей.

В данной работе была поставлена ***цель***: решить проблему согласования по числу; при этом предполагалось, что предложение состоит из существительных, местоимений, личных глаголов и (или) инфинитивов (с, возможно, единичным перечислением личных глаголов или инфинитивов с зависимыми словами указанных выше частей речи). Это ограничение было введено авторами сознательно, чтобы не утонуть в разборе всех возможных вариантов и решить задачу с максимальной полнотой и точностью.

2. Анализ проблемы

Задача согласования единственного и множественного числа не решена; причём, не только в русском языке.

Английский язык «проще» тем, что в нём строгий порядок слов: SVO («субъект – глагол – объект») [7]. Однако, даже для английского языка сформулированная проблема не решена.

Одной из наиболее успешных работ в этой области стала публикация Дэмиана Конвея (Damian Conway) «An algorithmic approach to English pluralization» [8]. В ней автор разрабатывает алгоритмы, преобразующие существительные, прилагательные и глаголы в единственном числе в соответствующие формы множественного числа. Также Конвей приводит алгоритм, позволяющий идентифицировать слова, отличающиеся только числом. Полная реализация данных алгоритмов была сделана автором публикации на языке Perl.

Русский язык, с одной стороны, относится к языкам с фиксированным порядком слов «SVO» (как и английский). Однако, с другой стороны, гибкий: SV / VS [11]. За счёт этого задача становится на порядок сложнее.

Согласование единственного и множественного числа в русском предложении было исследовано в бакалаврской диссертации Дзюбенко Василия Александровича в 2020 году [1]. Автором была разработана и реализована на языке python модель со стеком, совершающую свёртку и сдвиг, аналогичную GLR-анализатору; данная модель позволяла распознавать ошибки согласования в некоторых предложениях.

Тем не менее, В.А. Дзюбенко создал базу данных (выгрузив словарь библиотеки pymorphy2 [12]), в которой содержится информация о подавляющем большинстве слов русского языка. Данная база стала одним из базовых инструментов для решения нами поставленной проблемы.

3. Описание подхода

3.1. Идея

Как было сказано в анализе проблемы, именно база данных слов русского языка, разработанная В.А. Дзюбенко, стала базовым инструментом для решения задачи при помощи предложенного нами подхода.

Мы дополнили существующую базу данных несколькими таблицами; её схема представлена на рис. 1, а структура таблиц расписана в приложении А.

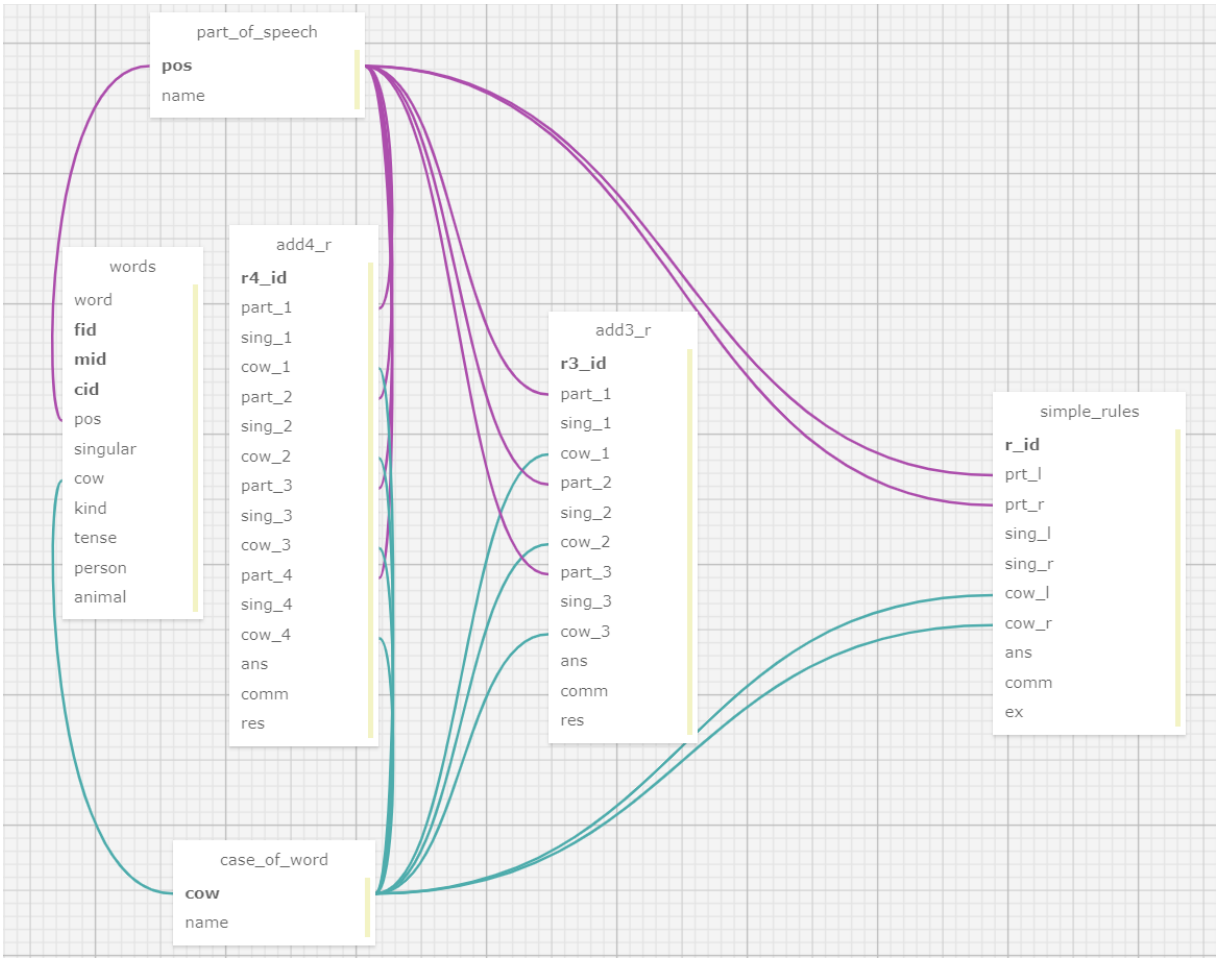

Рис. 1 – Концептуальная схема усовершенствованной базы данных

В рамках предложенного нами подхода мы описываем слово при помощи трёх параметров: часть речи, число и падеж. Нами было выявлено более сотни правил для словосочетаний длины 2, 3 или 4, по которым можно определить наличие или отсутствие ошибки в согласовании единственного и множественного числа. Полученные результаты представлены в приложении В.

В основе предложенного нами подхода лежит гипотеза, согласно которой од-

но и то же предложение являться и не являться ошибочным одновременно с точки зрения согласования единственного и множественного числа не может.

Также считаем, что в предложении нет орфографических, пунктуационных и др. ошибок, поскольку данная задача была успешно решена, например, компанией LanguageTooler GmbH [13].

3.2. Подготовительный этап

На вход программе подаётся предложение, состоящее из существительных, местоимений, личных глаголов или (и) инфинитивов (с, возможно, перечислением инфинитивов или личных глаголов с зависимыми словами, принадлежащим указанным частям речи).

Полученное предложение передаётся функции space(), которая преобразует считанную строку в список. Механизм её работы описывает алгоритм 1.

Алгоритм 1 – Предварительная обработка входных данных

```
1:  function SPACE(str1)
2:      str1 ← str1.lower()        ▷ Приводим полученную строку к нижнему регистру
3:      str2 ← «»        ▷ В этой переменной будет храниться преобразованная строка
4:      l ← len(str1)
5:      for from i = 0 to l − 1 do
6:          if str1[i] =«,» or str1[i] =«,»  then
7:              str2 ← str2 + « »
8:          end if
9:          str2 ← str2 + str1[i]
10:     end for
11:     if str2[len(str2) −1] = « » then
12:         str2 ← str2[ : len(str2) −1]                    ▷ Отбрасываем этот пробел
13:     end if
14:     return str2.split()          ▷ Возвращаем список, полученный из строки str2
    разбиением её по пробелам
15: end function
```

Таким образом, функция space() возвращает список, состоящий из слов и знаков препинания исходной строки. Так, приняв на вход строку *«Он ел, пил и спал»*, функция вернёт список [*«он»*, *«ел»*, *«,»*, *«пил»*, *«и»*, *«спал»*].

3.3. Обработка предложения

Итак, как было сказано выше, проверка согласования единственного и множественного числа в русском языке — процесс сложный: нужно учесть много критериев.

Нами было принято решение декомпозировать задачу.

Для начала (при наличии перечислений инфинитивов или личных личных глаголов) предложение упрощается: перечисление мы заменяем на инфинитив или личный глагол соответственно (параллельно проверяя, что внутри заменяемой части нет ошибок в согласовании единственного и множественного числа). Если же перечисления не обнаружено, сразу переходим к следующему этапу.

Затем проверяем предложение без перечислений при помощи разработанной нами системы правил.

Так, получив на вход список [*«он»*, *«ел»*, *«,»*, *«пил»*, *«и»*, *«спал»*], программа преобразует его в [*«он»*, *«учил»*] (заменив перечисление личных глаголов в единственном числе *«ел»*, *«,»*, *«пил»*, *«и»*, *«спал»* на личный глагол в единственном числе *«учил»*). После этого будет принято решение о том, что *ошибок в согласовании единственного и множественного числа не обнаружено* (в соответствии с правилом 4 таблицы 7 приложения В).

Согласно теореме Гёделя о неполноте, формальная арифметика либо противоречива, либо неполна [2]. Чтобы избежать противоречивости разработанной системы, мы включили лишь те правила, которые встречаются на практике, а не перебрали все возможные комбинации используемых нами параметров.

3.3.1. Упрощение предложения

Под упрощением мы будем понимать замену перечисления инфинитивов или личных глаголов одиночным инфинитивом или личным глаголом.

За упрощение предложения отвечает функция comma(), которая принимает на вход список, полученный из исходного предложения при помощи функции space(), описанной выше; а возвращает список, в виде которого представлено упрощённое предложение. Функция comma() вызывается только в том случае, если в предложении есть «и» или «или». Механизм её работы описывает алгоритм 2.

Алгоритм 2 – Обработка перечислений

```
1: function COMMA(l)                    ▷ l — подготовленная строка в виде списка
2:     part ← [ ]      ▷ Список значений параметра «часть речи» для данного слова (изначально пустой)
3:     end← (−1)                        ▷ Индикатор нахождения начала перечисления
4:     left ← (−1)      ▷ Левая и правая границы заменяемого участка изначально
5:     right ← (−1)                     ▷ инициализируем невозможными значениями: (−1)
```

Алгоритм 3 – Продолжение алгоритма 2

```
    llen ←len(l)                                  ▷ Длина исходного списка
    if «и» ∈ l then
        w← «и»                  ▷ w хранит союз, использующийся в перечислении
    else if «или» ∈ l then
        w← «или»
    end if
    id1 ← l.index(w)                              ▷ Записываем индекс w в списке
```

В рамках установленных нами ограничений «и» или «или» могут быть использованы только для перечислений личных глаголов или инфинитивов.

Алгоритм 4 – Продолжение алгоритма 3

```
    if llen > id1 +1 then
        resp1 ← l[id1 +1].[pos, singular, cow]
    end if
    if llen > id1 +2 then
        resp2 ← l[id1 +2].[pos, singular, cow]
    end if
    if llen > id1 +3 then
        resp3 ← l[id1 +3].[pos, singular, cow]
    end if
    if llen > id1 +4 then
        resp4 ← l[id1 +4].[pos, singular, cow]
    end if
    if id1 −1 ≥ 0 then
        resl1 ← l[id1 −1].[pos, singular, cow]
    end if
    if id1 −2 ≥ 0 then
        resl2 ← l[id1 −2].[pos, singular, cow]
    end if
    if id1 −3 ≥ 0 then
        resl3 ← l[id1 −3].[pos, singular, cow]
    end if
    if id1 −4 ≥ 0 then
        resl1 ← l[id1 −4].[pos, singular, cow]
    end if
    if id1 −5 ≥ 0 then
        resl5 ← l[id1 −5].[pos, singular, cow]
    end if
    if id1 −6 ≥ 0 then
        resl6 ← l[id1 −6].[pos, singular, cow]
    end if
    if id1 −7 ≥ 0 then
        resl7 ← l[id1 −7].[pos, singular, cow]
    end if
    if id1 +1 < llen then
```

Алгоритм 5 – Продолжение алгоритма 4

```
47:         for from i = 0 to len(resp1)−1 do
48:             if resp1[i][0] =«6» then                    ▷ Слово оказалось инфинитивом
49:                 part← «6» and right ← id1 +1
50:             end if
51:         end for
52:         if right= (−1) then                             ▷ Если же это не инфинитив
53:             for from i = 0 to len(resp1)−1 do
54:                 if resp1[i][0] =«5» then          ▷ Слово оказалось личным глаголом
55:                     right ←id1+1 and part ← «5»
56:                     sng ←l[i][1]                  ▷ Для личных глаголов важно число
57:                     break
58:                 end if
59:             end for
60:         end if
61:     end if
```

Таким образом, в результате исполнения блока 5 будет определено, слова (словосочетания) какой части речи перечисляются (если в предложении присутствует перечисление с союзом).

Заметим, что в случае перечисления с союзом за союзом идёт слово той же части речи, что и остальные перечисляемые слова. Например: *«Он хотел* ***читать*** *книги,* ***рисовать*** *картины и* ***познавать*** *тайны мироздания»*. Легко видеть, что в предложении перечисляются инфинитивы, и в то же время после союза «и» идёт инфинитив «познавать».

Если перечисляются инфинитивы, то упрощение предложения идёт согласно алгоритму 6.

Прежде всего, нужно определить начало левого операнда союза. В зависимости от длины буквосочетания, возможны различные варианты:

1. Словосочетание длины 6. Например, инф. + сущ. + сущ. + сущ. + сущ. + сущ.: *«Организовать проверку знаний требований охраны труда»*.

2. Словосочетание длины 5. Например, инф. + сущ. + сущ. + сущ. + сущ.: *«Организовать проверку знаний основ программирования»*.

3. Словосочетание длины 4. Например, инф. + инф. + сущ. + сущ.: *«Пойти спать сном младенца»*.

4. Словосочетание длины 3. Например, инф. + сущ. + сущ.: *«Оценить игру слов»*.

5. Словосочетание длины 2. Например, инф. + инф.: *«Пойти позавтракать»*.

6. Одиночный инфинитив. Например: *«Быть»*.

Итак, первым делом инициализируем левую границу заменяемого «куска» списка.

Алгоритм 6 – Продолжение алгоритма 5

```
62:     if part =«6» then                ▷ Если перечисляемая часть речи — инфинитив
63:         if id1−6 ≥ 0 and left= (−1) and «,»∉ l[id1−6 : id1] then      ▷ Проверяем буквосочетания длины 6
64:             for from i = 0 to len(resl6)−1 do
65:                 if resl6[i][0] =part then
66:                     r← check(l[id1−6 : id1])
67:                     if «N»∈ r then
68:                         return [«он», «писали» ]   ▷ Заведомо неверное предложение
69:                     else if «Y»∈ r then
70:                         left ← id1−6   ▷ Инициализировали границу левого операнда союза
71:                         break
72:                     end if
73:                 end if
74:             end for
75:         end if
76:         if id1−5 ≥ 0 and left= −1 and «,»∉ l[id1−5 : id1] then
77:             for from i = 0 to len(resl5)−1 do
78:                 if resl5[i][0] =part then
79:                     r← check(l[id1−5 : id1])
80:                     if «N»∈ r then
81:                         return [«он», «писали» ]
82:                     else if «Y»∈ r then
83:                         left ← id1−5
84:                         break
85:                     end if
86:                 end if
87:             end for
88:         end if
89:         if id1−4 ≥ 0 and left= −1 and «,»∉ l[id1−4 : id1] then
90:             for from i = 0 to len(resl4)−1 do
91:                 if resl4[i][0] =part then
92:                     r← check(l[id1−4 : id1])
93:                     if «N»∈ r then
94:                         return [«он», «писали» ]
95:                     else if «Y»∈ r then
96:                         left ← id1−4
97:                         break
98:                     end if
99:                 end if
100:            end for
```

Алгоритм 7 – Продолжение алгоритма 6

```
        end if
        if id1−3 ≥ 0 and left= −1 and «,»∉ l[id1−3 : id1] then
            for from i = 0 to len(resl3)−1 do
                if resl3[i][0] =part then
                    r← check(l[id1−3 : id1])
                    if «N»∈ r then
                        return [«он», «писали» ]
                    else if «Y»∈ r then
                        left ← id1−3
                        break
                    end if
                end if
            end for
        end if
```

Таким образом, в результате выполнения данного фрагмента кода будет определены границы левого и правого операндов союза.

Алгоритм 8 – Продолжение алгоритма 7

```
        if id1−2 ≥ 0 and left= −1 and «,»∉ l[id1−2 : id1] then
            for from i = 0 to len(resl2)−1 do
                if resl2[i][0] =part then
                    r← check(l[id1−2 : id1])
                    if «N»∈ r then
                        return [«он», «писали» ]
                    else if «Y»∈ r then
                        left ← id1−2
                        break
                    end if
                end if
            end for
        end if
        if id1−1 ≥ 0 and left= (−1) then
            for from i = 0 to len(resl1)−1 do
                if l[i][0] =part then
                    left ← id1−1
                    break
                end if
            end for
        end if
    end if
    Алгоритм 9
    Алгоритм 15
    Алгоритм 16
    return l
end function
```

В алгоритмах 6, 7 и 8 неоднократно фигурирует функция check(). В данном случае она используется для проверки предложения, не содержащего знаки пунктуации. Её описание будет в следующем параграфе (алгоритм 34).

Далее возможен один из двух вариантов:

- Союз связывает только два слова или словосочетания, или слово и словосочетание.
- Союз используется для перечисления 3 и более словосочетаний и (или) слов.

В первом случае предложение готово к упрощению: «кусок» от left до right заменяем единичным инфинитивом.

Во втором же случае необходимо продолжить анализ предложения, сдвигая левую границу заменяемого участка.

Для начала будем искать участки между двумя запятыми (при их наличии). Особенность данного этапа заключается в том, что между запятыми может оказаться ошибочное словосочетание, — потому фрагменты между запятыми нужно также проверять на согласованность.

Также важен порядок рассмотрения случаев: в первую очередь следует искать самые «короткие» словосочетания между запятыми (иначе можем «захватить» подстроку с запятыми). Этот и последующие этапы описаны в алгоритме 9.

Алгоритм 9 – Фрагмент алгоритма 8

```
1:  while «,»∈ l[1 : left ] do                          ▷ Пока есть запятые
2:      if l[left −1] = «,» and l[left −3] = «,» then    ▷ Между запятыми одно слово
3:          res1← l[left−2].[pos, singular, cow]
4:          for from i = 0 to len(res1) do
5:              if res1[i][0] =part then
6:                  left← left−2
7:                  break
8:              end if
9:          end for
10:     else if l[left −1] = «,» and l[left −4] = «,» then    ▷ Между запятыми
    словосочетание из двух слов
11:         res1← l[left−3].[pos, singular, cow]
12:         for from i = 0 to len(res1) do
13:             if res1[i][0] =part then
14:                 r ← check(l[left−3 : left−1])
15:                 if «N» ∈ r then
16:                     return [«он», «писали» ]
```

Алгоритм 10 – Продолжение алгоритма 9

```
                else if «Y» ∈ r then
                    left← left−3
                    break
                end if
            end if
        end for
    else if l[left −1] = «,» and l[left −5] = «,» then
        res1← l[left−4].[pos, singular, cow]
        for from i = 0 to len(res1) do
            if res1[i][0] =part then
                r ← check(l[left−4 : left−1])
                if «N» ∈ r then
                    return [«он», «писали» ]
                else if «Y» ∈ r then
                    left← left−4
                    break
                end if
            end if
        end for
    else if l[left −1] = «,» and l[left −6] = «,» then
        res1← l[left−5].[pos, singular, cow]
        for from i = 0 to len(res1) do
            if res1[i][0] =part then
                r ← check(l[left−5 : left−1])
                if «N» ∈ r then
                    return [«он», «писали» ]
                else if «Y» ∈ r then
                    left← left−5
                    break
                end if
            end if
        end for
    else if l[left −1] = «,» and l[left −7] = «,» then
        res1← l[left−6].[pos, singular, cow]
        for from i = 0 to len(res1) do
            if res1[i][0] =part then
                r ← check(l[left−6 : left−1])
                if «N» ∈ r then
                    return [«он», «писали» ]
                else if «Y» ∈ r then
                    left← left−6
                    break
                end if
            end if
        end for
    else if l[left −1] = «,» and l[left −8] = «,» then
```

Алгоритм 11 – Продолжение алгоритма 10

```
63:         res1← l[left−7].[pos, singular, cow]
64:         for from i = 0 to len(res1) do
65:             if res1[i][0] =part then
66:                 r ← check(l[left−7 : left−1])
67:                 if «N» ∈ r then
68:                     return [«он», «писали» ]
69:                 else if «Y» ∈ r then
70:                     left← left−7
71:                     break
72:                 end if
73:             end if
74:         end for
75:     else
76:         Алгоритм 12
77:     end if
78: end while
```

Итак, в результате работы фрагментов 9, 10 и 11 будет сдвинута граница до «первой» запятой.

Следующий этап — поиск начала перечисления. Соответствующий фрагмент описан алгоритмом 12.

Индикатор end отвечает за нахождение начала перечисления (изначально был инициализирован (−1), а после нахождения начала перечисления будет равен 1). Как и раньше, проверяем первый найденную подстроку на выполнение правил в ней.

Алгоритм 12 – Фрагмент алгоритма 10

```
1:  if left−2 ≥ 0 and l[left−1] =«,» and end= (−1) then
2:      res1← l[left−2].[pos, singular, cow]
3:      for from i = 0 to len(res1)−1 do
4:          if res1[i][0] =«6» then
5:              left← left−2
6:              break
7:          end if
8:      end for
9:  end if
10: if left−3 ≥ 0 and l[left−1] =«,» and end= (−1) then
11:     res1← l[left−3].[pos, singular, cow]
12:     for from i = 0 to len(res1)−1 do
13:         if res1[i][0] =«6» then
14:             r ← check(l[left−3 : left−1])
15:             if «N»∈ r then
16:                 return [«он», «писали» ]
17:             else if «Y» ∈ r then
```

Алгоритм 13 – Продолжение алгоритма 12

```
                left← left−3
                end ← 1
                break
            end if
        end if
    end for
end if
if left−4 ≥ 0 and l[left−1] =«,» and end= (−1) then
    res1← l[left−4].[pos, singular, cow]
    for from i = 0 to len(res1)−1 do
        if res1[i][0] =«6» then
            r ← check(l[left−4 : left−1])
            if «N»∈ r then
                return [«он», «писали» ]
            else if «Y» ∈ r then
                left← left−4
                end ← 1
                break
            end if
        end if
    end for
end if
if left−5 ≥ 0 and l[left−1] =«,» and end= (−1) then
    res1← l[left−5].[pos, singular, cow]
    for from i = 0 to len(res1)−1 do
        if res1[i][0] =«6» then
            r ← check(l[left−5 : left−1])
            if «N»∈ r then
                return [«он», «писали» ]
            else if «Y» ∈ r then
                left← left−5
                end ← 1
                break
            end if
        end if
    end for
end if
if left−6 ≥ 0 and l[left−1] =«,» and end= (−1) then
    res1← l[left−6].[pos, singular, cow]
    for from i = 0 to len(res1)−1 do
        if res1[i][0] =«6» then
            r ← check(l[left−6 : left−1])
            if «N»∈ r then
                return [«он», «писали» ]
            else if «Y» ∈ r then
                left← left−6
```

Алгоритм 14 – Продолжение алгоритма 13

```
                end ← 1
                break
            end if
        end if
    end for
end if
if left−7 ≥ 0 and l[left−1] =«,» and end= (−1) then
    res1← l[left−7].[pos, singular, cow]
    for from i = 0 to len(res1)−1 do
        if res1[i][0] =«6» then
            r ← check(l[left−7 : left−1])
            if «N»∈ r then
                return [«он», «писали» ]
            else if «Y» ∈ r then
                left← left−7
                end ← 1
                break
            end if
        end if
    end for
end if
```

Итак, после выполнения алгоритма 12 будут определены границы заменяемой подстроки, после чего необходимо вставить вместо перечисления инфинитивов одиночный инфинитив. Нами было выбрано слово *«учить»* (для данной цели можно было выбрать любой инфинитив, так как мы решаем проблему согласования единственного и множественного числа).

Алгоритм 15 – Фрагмент алгоритма 8

```
l← l[ : left]+ [«учить»]+ l[ right+1 :id1]
```

Если же при помощи союза «и» перечисляются личные глаголы, то упрощение идёт согласно алгоритму 16. В зависимости от длины буквосочетания, возможны различные варианты словосочетаний:

1. Словосочетание длины 7. Например, личн. глаг. + инф. + сущ. + сущ. + сущ. + + сущ.: *«Хотел организовать проверку знаний требований охраны труда»*.

2. Словосочетание длины 6. Например, личн. глаг. + сущ. + сущ. + сущ. + сущ. + + сущ.: *«Организовывал проверку знаний требований охраны труда»*.

3. Словосочетание длины 5. Например, личн. глаг. + инф. + сущ. + сущ. + сущ.:

«Хотел изучить основы теории кодирования».

4. Словосочетание длины 4. Например, личн. глаг. + сущ. + сущ. + сущ.: *«Изучил основы теории кодирования»*.

5. Словосочетание длины 3. Например, личн. глаг. + инф. + сущ.: *«Желает знать правду»*.

6. Словосочетание длины 2. Например, личн. глаг. + инф.: *«Желает знать»*.

7. Одиночный личный глагол. Например: *«Желать»*.

Во многом алгоритм обработки перечислений личных глаголов похож на алгоритм обработки перечислений инфинитивов.

Однако, в отличие от последних, для личных глаголов определено понятие числа. И в данной ситуации возникает ***проблема омографии***. Так, слово *«спАли»* — личный глагол во множественном числе, а *«спалИ»* — личный глагол в единственном числе. В самом деле, данные слова совпадают по написанию, но различны по звучанию и значению. Заметим, что в единственном числе слово интерпретируется тогда и только тогда, когда оно в повелительном наклонении. Легко видеть, что на множестве рассматриваемых в данной работе частей речи перечисляются личные глаголы в повелительном наклонении тогда и только тогда, когда предложение начинается с глагола в повелительном наклонении. Потому сразу определим, является ли первое слово глаголом. Если да, однозначно ли определяется его число.

Алгоритм 16 – Продолжение алгоритма 8

```
if part=«5» then
    sng0 ← [ ]                ▷ Для определения числа первого слова в предложении
    pov ← (−1)                ▷ Индикатор повелительного наклонения
    res0← l[0].[cow, singular, cow]
    end← (−1)
    for from i = 0 to len(res0)−1 do
        if res0[i][0] =«5» then
            sng0 ← sng0 + list(res0[i][1])
        end if
    end for
    sng0 ← list(set(sng0))
    if len(sng0)> 1 then
        pov ← 1
        sng ←«Y»              ▷ Считаем единственным число перечисляемых личных
глаголов
```

Алгоритм 17 – Продолжение алгоритма 16

```
15:     end if
16:     if id1−7 ≥ 0 and left= (−1) and «,»∉ l[id1−7 : id1] then
17:         sng1 ← [ ]                          ▷ Для записи возможных значений числа
18:         r ← [ ]                        ▷ Для записи результата проверки по системе правил
19:         for from i = 0 to len(resl7)−1 do
20:             if resl7[i][0] =part then
21:                 sng1← sng1 + list(resl7[i][1])
22:                 r← r + list(check(l[id1−7 : id1 ]))
23:             end if
24:         end for
25:         if len(sng1)> 0 then    ▷ Если словосочетание действительно начинается с личного глагола
26:             if (len(sng1)= 1 and sng ∈ sng1 and «Y»∈ r) or (len(sng1)> 1 and (sng=«N» or pov= 1) and «Y» ∈ r) then
27:                 left← id1−7
28:             else
29:                 return [«он», «писали» ]
30:             end if
31:         end if
32:     end if
```

Здесь следует рассмотреть решение проблемы омографии. Оно представлено в строках 16.6 – 17.32. Для начала проверяем, является ли первое слово рассматриваемого предложения личным глаголом, число которого определено неоднозначно. Это имеет значение, поскольку в рамках поставленных ограничений если в предложении есть личные глаголы в повелительном наклонении, то с одного из них оно начинается. Например: *«Учите математику, высыпайтесь и будьте людьми»*.

Далее проверяем, что рассматриваемое словосочетание нужной длины и не содержит запятых. Если вдруг первое слово данного словосочетания оказалось личным глаголом, то мы запоминаем какого оно числа может быть; а также проверяем данное словосочетание на наличие или отсутствие ошибок в согласовании единственного и множественного числа.

Словосочетание не содержит ошибок в следующих случаях:

1. Число личного глагола опрелеляется однозначно и совпадает с числом правого операнда союза, проверка словосочетания на наличие ошибок в согласовании единственного и множественного числа прошла успешно (ошибок и незнакомых сочетаний не обнаружено).

2. Число личного глагола определяется неоднозначно, и имеет место повелитель-

ное наклонение.

3. Число личного глагола определяется неоднозначно, первое слово в предложении не является личным глаголом в повелительном наклонении и перечисляются личные глаголы во множественном числе.

Это мы и проверяем. В остальных случаях возвращаем заведомо неверное предложение. Совершенно аналогично рассматриваются словосочетания меньшей длины:

Алгоритм 18 – Продолжение алгоритма 17

```
33:     if id1−6 ≥ 0 and left= (−1) and «,»∉ l[id1−6 : id1] then
34:         sng1 ← []
35:         r ← []
36:         for from i = 0 to len(resl6)−1 do
37:             if resl6[i][0] =part then
38:                 sng1← sng1 + list(resl6[i][1])
39:                 r← r + list(check(l[id1−6 : id1 ]))
40:             end if
41:         end for
42:         if len(sng1)> 0 then
43:             if (len(sng1)= 1 and sng ∈ sng1 and «Y»∈ r) or (len(sng1)> 1 and
    (sng=«N» or pov= 1) and «Y» ∈ r) thenя
44:                 left← id1−6
45:             else
46:                 return [«он», «писали» ]
47:             end if
48:         end if
49:     end if
50:     if id1−5 ≥ 0 and left= (−1) and «,»∉ l[id1−5 : id1] then
51:         sng1 ← []
52:         r ← []
53:         for from i = 0 to len(resl5)−1 do
54:             if resl5[i][0] =part then
55:                 sng1← sng1 + list(resl5[i][1])
56:                 r← r + list(check(l[id1−5 : id1 ]))
57:             end if
58:         end for
59:         if len(sng1)> 0 then
60:             if (len(sng1)= 1 and sng ∈ sng1 and «Y»∈ r) or (len(sng1)> 1 and
    (sng=«N» or pov= 1) and «Y» ∈ r) then
61:                 left← id1−5
62:             else
63:                 return [«он», «писали» ]
64:             end if
65:         end if
```

Алгоритм 19 – Продолжение алгоритма 18

```
66:     end if
67:     if id1−4 ≥ 0 and left= (−1) and «,»∉ l[id1−4 : id1] then
68:         sng1 ← [ ]
69:         r ← [ ]
70:         for from i = 0 to len(resl4)−1 do
71:             if resl4[i][0] =part then
72:                 sng1← sng1 + list(resl4[i][1])
73:                 r← r + list(check(l[id1−4 : id1 ]))
74:             end if
75:         end for
76:         if len(sng1)> 0 then
77:             if (len(sng1)= 1 and sng ∈ sng1 and «Y»∈ r) or (len(sng1)> 1 and
    (sng=«N» or pov= 1) and «Y» ∈ r) then
78:                 left← id1−4
79:             else
80:                 return [«он», «писали» ]
81:             end if
82:         end if
83:     end if
84:     if id1−3 ≥ 0 and left= (−1) and «,»∉ l[id1−3 : id1] then
85:         sng1 ← [ ]
86:         r ← [ ]
87:         for from i = 0 to len(resl3)−1 do
88:             if resl3[i][0] =part then
89:                 sng1← sng1 + list(resl3[i][1])
90:                 r← r + list(check(l[id1−3 : id1 ]))
91:             end if
92:         end for
93:         if len(sng1)> 0 then
94:             if (len(sng1)= 1 and sng ∈ sng1 and «Y»∈ r) or (len(sng1)> 1 and
    (sng=«N» or pov= 1) and «Y» ∈ r) then
95:                 left← id1−3
96:             else
97:                 return [«он», «писали» ]
98:             end if
99:         end if
100:     end if
101:     if id1−2 ≥ 0 and left= (−1) and «,»∉ l[id1−2 : id1] then
102:         sng1 ← [ ]
103:         r ← [ ]
104:         for from i = 0 to len(resl2)−1 do
105:             if resl2[i][0] =part then
106:                 sng1← sng1 + list(resl2[i][1])
107:                 r← r + list(check(l[id1−2 : id1 ]))
108:             end if
109:         end for
```

Алгоритм 20 – Продолжение алгоритма 19

```
110:        if len(sng1)> 0 then
111:            if (len(sng1)= 1 and sng ∈ sng1 and «Y»∈ r) or (len(sng1)> 1 and
    (sng=«N» or pov= 1) and «Y» ∈ r) then
112:                left← id1−2
113:            else
114:                return [«он», «писали» ]
115:            end if
116:        end if
117:    end if
118:    if id1−1 ≥ 0 and left= (−1) then
119:        sng1 ← [ ]
120:        for from i = 0 to len(resl1) do
121:            if resl1[i][0] =part then
122:                sng1← sng1 + list(resl1[i][1])
123:            end if
124:        end for
125:        sng1 ← list(set(sng1))
126:        if (len(sng1)= 1 and sng ∈ sng1) or (len(sng1)> 1) then
127:            left← id1−1
128:        else
129:            return [«он», «писали» ]
130:        end if
131:    end if
```

Итак, по завершении работы алгоритма 20 будут определены границы левого и правого операндов союза.

Затем, как и в случае с инфинитивами, осуществляется поиск участков между двумя запятыми (при их наличии), а затем — поиск начала перечисления.

Алгоритм 21 – Продолжение алгоритма 20

```
132:    while «,»∈ l[1 : left ] do
133:        if left−3 ≥ 0 and l[left−3] = «,» and l[left−1] = «,» then
134:            sng1← [ ]
135:            res1← l[left−2].[pos, singular, cow]
136:            for from i = 0 to len(res1)−1 do
137:                if res1[i][0] =part then
138:                    sng1← sng1 + list(res[i][1])
139:                end if
140:            end for
141:            sng1← list(set(sng1))
142:            if (len(sng1 = 1) and sng∈ sng1) or (len(sng1)> 1 and (sng=«N» or
    pov = 1)) then
143:                left ← left−2
144:            else
145:                return [«он», «писали» ]
```

Алгоритм 22 – Продолжение алгоритма 21

```
146:            end if
147:        else if left−4 ≥ 0 and l[left−4] = «,» and l[left−1] = «,» then
148:            sng1← [ ]
149:            r← [ ]
150:            res1← l[left−3].[pos, singular, cow]
151:            for from i = 0 to len(res1)−1 do
152:                if res1[i][0] =part then
153:                    sng1← sng1 + list(res[i][1])
154:                    r← r + check(l[left −3 : left−1])
155:                end if
156:            end for
157:            sng1← list(set(sng1))
158:            if (len(sng1 = 1) and sng∈ sng1 and «Y»∈ r) or (len(sng1)> 1 and
    (sng=«N» or pov = 1) and «Y»∈ r) then
159:                left ← left−3
160:            else
161:                return [«он», «писали» ]
162:            end if
163:        else if left−5 ≥ 0 and l[left−5] = «,» and l[left−1] = «,» then
164:            sng1← [ ]
165:            r← [ ]
166:            res1← l[left−4].[pos, singular, cow]
167:            for from i = 0 to len(res1)−1 do
168:                if res1[i][0] =part then
169:                    sng1← sng1 + list(res[i][1])
170:                    r← r + check(l[left −4 : left−1])
171:                end if
172:            end for
173:            sng1← list(set(sng1))
174:            if (len(sng1 = 1) and sng∈ sng1 and «Y»∈ r) or (len(sng1)> 1 and
    (sng=«N» or pov = 1) and «Y»∈ r) then
175:                left ← left−4
176:            else
177:                return [«он», «писали» ]
178:            end if
179:        else if left−6 ≥ 0 and l[left−6] = «,» and l[left−1] = «,» then
180:            sng1← [ ]
181:            r← [ ]
182:            res1← l[left−5].[pos, singular, cow]
183:            for from i = 0 to len(res1)−1 do
184:                if res1[i][0] =part then
185:                    sng1← sng1 + list(res[i][1])
186:                    r← r + check(l[left −5 : left−1])
187:                end if
188:            end for
189:            sng1← list(set(sng1))
```

Алгоритм 23 – Продолжение алгоритма 22

```
190:            if (len(sng1 = 1) and sng∈ sng1 and «Y»∈ r) or (len(sng1)> 1 and
    (sng=«N» or pov = 1) and «Y»∈ r) then
191:                left ← left−5
192:            else
193:                return [«он», «писали» ]
194:            end if
195:        else if left−7 ≥ 0 and l[left−7] = «,» and l[left−1] = «,» then
196:            sng1← [ ]
197:            r← [ ]
198:            res1← l[left−6].[pos, singular, cow]
199:            for from i = 0 to len(res1)−1 do
200:                if res1[i][0] =part then
201:                    sng1← sng1 + list(res[i][1])
202:                    r← r + check(l[left −6 : left−1])
203:                end if
204:            end for
205:            sng1← list(set(sng1))
206:            if (len(sng1 = 1) and sng∈ sng1 and «Y»∈ r) or (len(sng1)> 1 and
    (sng=«N» or pov = 1) and «Y»∈ r) then
207:                left ← left−6
208:            else
209:                return [«он», «писали» ]
210:            end if
211:        else if left−8 ≥ 0 and l[left−8] = «,» and l[left−1] = «,» then
212:            sng1← [ ]
213:            r← [ ]
214:            res1← l[left−7].[pos, singular, cow]
215:            for from i = 0 to len(res1)−1 do
216:                if res1[i][0] =part then
217:                    sng1← sng1 + list(res[i][1])
218:                    r← r + check(l[left −7 : left−1])
219:                end if
220:            end for
221:            sng1← list(set(sng1))
222:            if (len(sng1 = 1) and sng∈ sng1 and «Y»∈ r) or (len(sng1)> 1 and
    (sng=«N» or pov = 1) and «Y»∈ r) then
223:                left ← left−7
224:            else
225:                return [«он», «писали» ]
226:            end if
227:        else if left−9 ≥ 0 and l[left−9] = «,» and l[left−1] = «,» then
228:            sng1← [ ]
229:            r← [ ]
230:            res1← l[left−8].[pos, singular, cow]
231:            for from i = 0 to len(res1)−1 do
```

Алгоритм 24 – Продолжение алгоритма 23

```
232:                if res1[i][0] = part then
233:                    sng1← sng1 + list(res[i][1])
234:                    r← r + check(l[left −8 : left−1])
235:                end if
236:            end for
237:            sng1← list(set(sng1))
238:            if (len(sng1 = 1) and sng∈ sng1 and «Y»∈ r) or (len(sng1)> 1 and
    (sng=«N» or pov = 1) and «Y»∈ r) then
239:                left ← left−8
240:            else
241:                return [«он», «писали» ]
242:            end if
243:        else
244:            if left−9 ≥ 0 and l[left−1] = «,» and end = (−1) then ▷ Поиск начала
    перечисления
245:                sng1← [ ]
246:                r ← [ ]
247:                res1← l[left−9].[pos, singular, cow]
248:                res1← list(set(res1))
249:                for from i = 0 to len(res1)−1 do
250:                    if res1[i][0] = part then
251:                        sng1← sng1 + list(res[i][1])
252:                        r ← r + list(l[left −9 : left−1])
253:                    end if
254:                end for
255:                sng1← list(set(sng1))
256:                if (len(sng1) = 1 and sng∈ sng1 and «Y» ∈ r) or (len(sng1)> 1 and
    (sng=«N» or pov = 1) and «Y» ∈ r) then
257:                    left ← left−9
258:                    end ← 1
259:                else if len(sng1)> 0 and «Y» ∈ sng1 then
260:                    return [«он», «писали» ]
261:                end if
262:            end if
263:            if left−8 ≥ 0 and l[left−1] = «,» and end = (−1) then
264:                sng1← [ ]
265:                r ← [ ]
266:                res1← l[left−8].[pos, singular, cow]
267:                res1← list(set(res1))
268:                for from i = 0 to len(res1)−1 do
269:                    if res1[i][0] = part then
270:                        sng1← sng1 + list(res[i][1])
271:                        r ← r + list(l[left −8 : left−1])
272:                    end if
273:                end for
```

Алгоритм 25 – Продолжение алгоритма 24

```
274:                sng1← list(set(sng1))
275:                if (len(sng1) = 1 and sng∈ sng1 and «Y» ∈ r) or (len(sng1)> 1 and
    (sng=«N» or pov =  1) and «Y» ∈ r) then
276:                    left ← left−8
277:                    end ← 1
278:                else if len(sng1)> 0 and «Y» ∈ sng1 then
279:                    return [«он», «писали» ]
280:                end if
281:            end if
282:            if left−7 ≥ 0 and l[left−1] = «,» and end = (−1) then
283:                sng1← [ ]
284:                r ← [ ]
285:                res1← l[left−7].[pos, singular, cow]
286:                res1← list(set(res1))
287:                for from i = 0 to len(res1)−1 do
288:                    if res1[i][0] = part then
289:                        sng1← sng1 + list(res[i][1])
290:                        r ← r + list(l[left −7 : left−1])
291:                    end if
292:                end for
293:                sng1← list(set(sng1))
294:                if (len(sng1) = 1 and sng∈ sng1 and «Y» ∈ r) or (len(sng1)> 1 and
    (sng=«N» or pov =  1) and «Y» ∈ r) then
295:                    left ← left−7
296:                    end ← 1
297:                else if len(sng1)> 0 and «Y» ∈ sng1 then
298:                    return [«он», «писали» ]
299:                end if
300:            end if
301:            if left−6 ≥ 0 and l[left−1] = «,» and end = (−1) then
302:                sng1← [ ]
303:                r ← [ ]
304:                res1← l[left−6].[pos, singular, cow]
305:                res1← list(set(res1))
306:                for from i = 0 to len(res1)−1 do
307:                    if res1[i][0] = part then
308:                        sng1← sng1 + list(res[i][1])
309:                        r ← r + list(l[left −6 : left−1])
310:                    end if
311:                end for
312:                sng1← list(set(sng1))
313:                if (len(sng1) = 1 and sng∈ sng1 and «Y» ∈ r) or (len(sng1)> 1 and
    (sng=«N» or pov =  1) and «Y» ∈ r) then
314:                    left ← left−6
315:                    end ← 1
```

Алгоритм 26 – Продолжение алгоритма 25

```
316:                 else if len(sng1)> 0 and «Y» ∈ sng1 then
317:                     return [«он», «писали» ]
318:                 end if
319:             end if
320:             if left−5 ≥ 0 and l[left−1] = «,» and end = (−1) then
321:                 sng1← [ ]
322:                 r ← [ ]
323:                 res1← l[left−5].[pos, singular, cow]
324:                 res1← list(set(res1))
325:                 for from i = 0 to len(res1)−1 do
326:                     if res1[i][0] = part then
327:                         sng1← sng1 + list(res[i][1])
328:                         r ← r + list(l[left −5 : left−1])
329:                     end if
330:                 end for
331:                 sng1← list(set(sng1))
332:                 if (len(sng1) = 1 and sng∈ sng1 and «Y» ∈ r) or (len(sng1)> 1 and
    (sng=«N» or pov = 1) and «Y» ∈ r) then
333:                     left ← left−5
334:                     end ← 1
335:                 else if len(sng1)> 0 and «Y» ∈ sng1 then
336:                     return [«он», «писали» ]
337:                 end if
338:             end if
339:             if left−4 ≥ 0 and l[left−1] = «,» and end = (−1) then
340:                 sng1← [ ]
341:                 r ← [ ]
342:                 res1← l[left−4].[pos, singular, cow]
343:                 res1← list(set(res1))
344:                 for from i = 0 to len(res1)−1 do
345:                     if res1[i][0] = part then
346:                         sng1← sng1 + list(res[i][1])
347:                         r ← r + list(l[left −4 : left−1])
348:                     end if
349:                 end for
350:                 sng1← list(set(sng1))
351:                 if (len(sng1) = 1 and sng∈ sng1 and «Y» ∈ r) or (len(sng1)> 1 and
    (sng=«N» or pov = 1) and «Y» ∈ r) then
352:                     left ← left−4
353:                     end ← 1
354:                 else if len(sng1)> 0 and «Y» ∈ sng1 then
355:                     return [«он», «писали» ]
356:                 end if
357:             end if
358:             if left−3 ≥ 0 and l[left−1] = «,» and end = (−1) then
```

Алгоритм 27 – Продолжение алгоритма 26

```
359:                sng1← [ ]
360:                r ← [ ]
361:                res1← l[left−3].[pos, singular, cow]
362:                res1← list(set(res1))
363:                for from i = 0 to len(res1)−1 do
364:                    if res1[i][0] = part then
365:                        sng1← sng1 + list(res[i][1])
366:                        r ← r + list(l[left −3 : left−1])
367:                    end if
368:                end for
369:                sng1← list(set(sng1))
370:                if (len(sng1) = 1 and sng∈ sng1 and «Y» ∈ r) or (len(sng1)> 1 and
    (sng=«N» or pov = 1) and «Y» ∈ r) then
371:                    left ← left−3
372:                    end ← 1
373:                else if len(sng1)> 0 and «Y» ∈ sng1 then
374:                    return [«он», «писали» ]
375:                end if
376:            end if
377:            if left−2 ≥ 0 and l[left−1] = «,» and end = (−1) then
378:                sng1← [ ]
379:                res1← l[left−2].[pos, singular, cow]
380:                res1← list(set(res1))
381:                for from i = 0 to len(res1)−1 do
382:                    if res1[i][0] = part then
383:                        sng1← sng1 + list(res[i][1])
384:                    end if
385:                end for
386:                sng1← list(set(sng1))
387:                if (len(sng1) = 1 and sng∈ sng1) or (len(sng1)> 1 and (sng=«N» or
    pov = 1)) then
388:                    left ← left−2
389:                    end ← 1
390:                else if len(sng1)> 0 and «Y» ∈ sng1 then
391:                    return [«он», «писали» ]
392:                end if
393:            end if
394:        end if
395:    end while
```

Итак, после выполнения окончания выполнения алгоритма 27 в переменной left будет записан индекс списка, соответствующий левой границе перечисления; в переменной right — индекс, соответствующий правой границе перечисления; в переменной sng — число перечисляемых личных глаголов (данный фрагмент будет выполнен только в том случае, если имеет место перечисление личных глаголов).

Далее, в зависимости от числа заменяем перечисление одним личным глаголом в таком же числе. Мы решили заменять перечисление личных глаголов на слово *«учил»* или *«учили»* в зависимости от числа.

Так, *«Он* ***писал диплом, ел и спал»*** будет преобразовано в *«Он* ***учил»***.

Данный этап описан алгоритмом 28.

Алгоритм 28 – Продолжение алгоритма 27

```
396:    if sng = «Y» then
397:        l← l[ : left]+ [«учил»] + l[ right +1 : ]
398:    else if sng = «N» then
399:        l← l[ : left]+ [«учили»] + l[ right +1 : ]
400:    end if
401: end if
```

Итак, на этом завершается обработка перечисления. Далее будет описан алгоритм, как же проверяются предложения без перечислений.

3.3.2. Непосредственная проверка предложения

Проанализировав предложения, состоящие из существительных, местоимений, личных глаголов и (или) инфинитивов, мы обнаружили, что важно проанализировать словосочетания длины 2, 3 или 4 во всём списке. Словосочетания бОльшей длины декомпозируются на составляющие указанной длины.

Так, предложение *«Он тебя видит и слышит»* разбивается на части: *«Он тебя видит»* (длины 3) и *«слышит»* (длины 1). Союз «и» в данном случае выполняет роль связки.

Нами были написаны функции two() (алгоритм 29), three() (алгоритм 32) и four() (алгоритм 34), которые анализируют словосочетания соответствующей длины.

Алгоритм 29 – Анализ словосочетаний длины 2

```
1: function TWO(lst)
2:     result ← [ ]                                  ▷ Результат выполнения проверки
3:     pov ← (−1)          ▷ Идентификатор наличия повелительного наклонения
4:     sng ← [ ]
5:     ans1← [ ]
6:     for from q = 0 to len(lst)−2 do
7:         res_l ← lst[q].[pos, singular, cow]
8:         k← 0
9:         uns← (−1)
10:        for from i = 0 to len(res_l)−1 do
```

Алгоритм 30 – Продолжение алгоритма 29

```
            if res_l[i][0] = «1» or res_l[i][0] = «b» then
                if res_l[i][1] = «N» then
                    uns← 1
                end if
            end if
        end for
        res_r ← lst[q + 1].[pos, singular, cow]
        for from i = 0 to len(res_r)−1 do
            if res_r[i][0] = «5» then
                k← k+1
            else if res_r[i][0] = «1» or res_r[i][0] = «b» then
                if res_r[i][1] = «N» then
                    uns ← 1
                end if
            end if
        end for
        if k > 1 then
            pov← 1
        end if
        for from i = 0 to len(res_l)−1 do
            for from j = 0 to len(res_r)−1 do
                res ← (ans).simple_rules.(res_l[i])(res_r[j])        ▷ Записываем ответ для данного словосочетания из таблицы simple_rules, в которой хранятся правила для словосочетаний длины 2
                if len(res) > 0 then
                    for from r = 0 to len(res)−1 do
                        ans← ans + res[r][0]
                    end for
                end if
            end for
        end for
        ans1← list(set(ans1))
        if pov= 1 and uns = (−1) then        ▷ Если правый операнд может быть в повелительном наклонении, а слева стоит подлежащее в единственном числе
            if «N» ∈ ans1 then
                result← result + [«N»]
            else if «Y» ∈ ans1 then
                result ← result + [«Y»]
            else
                result ← result + [«E»]        ▷ «E» — незнакомое сочетание
            end if
        else
            if «Y» ∈ ans1 then
                result ← result + [«Y»]
            else if «N» ∈ ans1 then
```

Алгоритм 31 – Продолжение алгоритма 30

```
53:                 result← result + [«N»]
54:             else
55:                 result ← result + [«E»]                ▷ «E» — незнакомое сочетание
56:             end if
57:         end if
58:     end for
59:     return result
60: end function
```

Подобным образом устроена проверка словосочетаний длины 3.

Алгоритм 32 – Анализ словосочетаний длины 3

```
1: function THREE(lst)
2:     result ← [ ]
3:     for from i = 0 to len(lst)−3 do
4:         res1 ← lst[i].[pos, singular, cow]
5:         res2 ← lst[i + 1].[pos, singular, cow]
6:         res3 ← lst[i + 2].[pos, singular, cow]
7:         for from i1 = 0 to len(res1)−1 do
8:             for from i2 = 0 to len(res2)−1 do
9:                 for from i3 = 0 to len(res3)−1 do
10:                    res ← (ans).add3_r.(res1[i1])(res1[i2])(res3[i3])   ▷ Записываем ответ для данного словосочетания из таблицы add3_r, в которой хранятся правила для словосочетаний длины 3
11:                    if len(res)> 0 then
12:                        for from r = 0 to len(res)−1 do
13:                            result← result + res[r][0]
14:                        end for
15:                    end if
16:                end for
17:            end for
18:        end for
19:        result ← list(set(result))
20:        if len(result)> 0 then
21:            if «N» ∈ result then
22:                return [«N»] ▷ Если на каком-то этапе сработало правило ошибки, не проверяем дальше
23:            end if
24:        end if
25:    end for
26:    return result
27: end function
```

Аналогично устроена проверка словосочетаний длины 4.

Алгоритм 33 – Анализ словосочетаний длины 4

```
1: function FOUR(lst)
2:     result ← [ ]
3:     for from i = 0 to len(lst)−4 do
4:         res1 ← lst[i].[pos, singular, cow]
5:         res2 ← lst[i + 1].[pos, singular, cow]
6:         res3 ← lst[i + 2].[pos, singular, cow]
7:         res4 ← lst[i + 3].[pos, singular, cow]
8:         for from i1 = 0 to len(res1)−1 do
9:             for from i2 = 0 to len(res2)−1 do
10:                for from i3 = 0 to len(res3)−1 do
11:                    for from i4 = 0 to len(res4)−1 do
12:                        res ← (ans).add4_r.(res1[i1])(res1[i2])(res3[i3])(res4[i4])  ▷ Записываем ответ для данного словосочетания из таблицы add4_r, в которой хранятся правила для словосочетаний длины 4
13:                        if len(res)> 0 then
14:                            for from r = 0 to len(res)−1 do
15:                                result← result + res[r][0]
16:                            end for
17:                        end if
18:                    end for
19:                end for
20:            end for
21:        end for
22:        result ← list(set(result))
23:        if len(result) then
24:            if «N» ∈ result then
25:                return [«N»] ▷ Если на каком-то этапе сработало правило ошибки, не проверяем дальше
26:            end if
27:        end if
28:    end for
29:    return result
30: end function
```

Всё это запускается при помощи функции check() (см. алгоритм 34).

Алгоритм 34 – Описание функции check()

```
1: function CHECK(l)
2:     result ← [ ]
3:     while «и» ∈ l or «или» ∈ l do  ▷ Упрощение предложения посредством замены перечислений
4:         l ← comma(l)
5:     end while
6:     if len(l)> 3 then
7:         result ← result + four(l)
8:     end if
```

Алгоритм 35 – Продолжение алгоритма 34

```
    if len(l)> 2 then
        result ← result + three(l)
    end if
    if len(l)> 1 then
        result ← result + two(l)
    end if
    if len(l)= 1 then        ▷ Одиночное слово не может быть ошибочным с точки зрения согласования единственного и множественного числа
        result← result + [«Y»]
    end if
    return result
end function
```

Анализирует полученные результаты функция res() (алгоритм 36).

Алгоритм 36 – Описание функции res()

```
function RES(r)
    if «N» ∈ r then
        print(«Ошибка в согласовании единственного и множественного числа»)
    else if «Y» ∈ r then
        print(«Ошибок в согласовании единственного и множественного числа не обнаружено»)
    else
        print(«Есть незнакомые сочетания»)
    end if
end function
```

3.3.3. Пример работы алгоритма

Рассмотрим предложение *«Он хотел организовать проверку знаний требований охраны труда, купить себе пони и выспаться»*. Запишем его в строку str. Чтобы проверить согласование единственного и множественного числа, необходимо вызвать res(check(space(str))).

Фунция space() преобразует str в список l = [*«он»*, *«хотел»*, *«организовать»*, *«проверку»*, *«знаний»*, *«требований»*, *«охраны»*, *«труда»*, *«,»*, *«купить»*, *«себе»*, *«пони»*, *«и»*, *«выспаться»*].

Далее, обрабатывая список l, фунция comma() определит id1 = 12, left = 2, right = 13. Проверит, что одиночный инфинитив *«выспаться»* не является ошибочным, как и словосочетания *«купить себе пони»* (правило 16 таблицы 8) и *«организовать проверку знаний требований охраны труда»* (правила 5, 69, 78, 79, 65 табли-

цы 7).

После этого функцией comma() список будет преобразован в [*«он»*, *«хотел»*, *«учить»*], который будет проверен функцией check(). В результате срабатывания правил 4, 6 таблицы 7 будет получен список result = [*«Y»*, *«Y»*].

Анализируя result, функция res() примет решение, что «Ошибок в согласовании единственного и множественного числа не обнаружено».

4. Заключение

Мы надеемся, что поставленная задача согласования единственного и множественного числа в рамках заданных ограничений была решена нами целиком и полностью.

Для тех, на наш взгляд, редких случаях, когда предложенный нами алгоритм и созданная для него система правил (более сотни) не позволяла бы найти ошибки в согласовании числа, нами предложен механизм для создания нового правила, при этом, никаких изменений в коде программы не требуется.

Мы считаем, что создан хороший задел для решения проблемы согласования в полном объёме и без всяких ограничений.

Список литературы

[1] **Дзюбенко, В.А. Согласование единственного и множественного числа в русском предложении**: бакалаврская диссертация: 03.03.01 / Дзюбенко Василий Александрович. – Долгопрудный, 2020. – 20 с.

[2] **Журавлёв, Ю.И. Дискретный анализ. Формальные системы и алгоритмы**: Учебное пособие / Ю.И. Журавлёв, Ю.А. Флёров, Н.М. Вялый – М.: ООО Контакт Плюс, 2010. – 336 с.

[3] **Марков, А. А. Пример статистического исследования над текстом «Евгения Онегина», иллюстрирующий связь испытаний в цепь** // Известия Имп. Акад. наук, серия VI. – 1913. – Т. 10. – №. 3. – С. 153.

[4] **Тестелец, Я. Г. Введение в общий синтаксис.** — Федеральное государственное бюджетное образовательное учреждение высшего образования Российский государственный гуманитарный университет, 2001.

[5] **Чесебиев, И. А. Компьютерное распознавание и порождение речи**: монография. – Москва: Спорт и Культура-2000, 2008. – 125 с.

[6] **Chomsky, N. Syntactic structures.** – De Gruyter Mouton, 2009.

[7] **Comrie, B. Language universals and linguistic typology: Syntax and morphology**. – University of Chicago press, 1989.

[8] **Conway, D. An algorithmic approach to English pluralization** // Proceedings of the Second Annual Perl Conference. – 1998.

[9] **Hutchins, W. J. The Georgetown-IBM experiment demonstrated in January 1954** // Conference of the Association for Machine Translation in the Americas. — Springer, Berlin, Heidelberg, 2004. — С. 102-114.

[10] **Jiang B., Yin J., Liu Q. Zipf's law for all the natural cities around the world** //International Journal of Geographical Information Science. – 2015. – Т. 29. – №. 3. – С. 498-522.

[11] **The world atlas of language structures** / M. Haspelmath [and others]. – Oxford Univ. Press, 2005.

[12] Морфологический анализатор pymorphy2 [Электронный ресурс] — https://pymorphy2.readthedocs.io/en/latest/

[13] LanguageTool — Проверка грамматики и стилистики [Электронный ресурс] — https://languagetool.org/ru

ПРИЛОЖЕНИЕ А

Структура таблиц базы данных

Таблица 1 – words

№	Имя столбца	Тип данных	Комментарий
1	word	NVARCHAR2(60)	слово в нижнем регистре
2	mid	NUMBER(10)	id семейства слов
3	fid	NUMBER(6)	id формы слов
4	cid	NUMBER(6)	id слова данной формы
5	pos	CHAR(1)	часть речи
6	singular	CHAR(1)	число
7	cow	CHAR(1)	падеж
8	tense	CHAR(1)	время
9	kind	CHAR(1)	пол
10	animal	CHAR(1)	одушевлённость
11	person	CHAR(1)	лицо

Таблица 2 – case_of_word

№	Имя столбца	Тип данных	Комментарий
1	cow	CHAR(1)	id падежа
2	name	VARCHAR2(30)	название падежа

Таблица 3 – part_of_speech

№	Имя столбца	Тип данных	Комментарий
1	pos	CHAR(1)	id части речи
2	name	VARCHAR2(30)	название части речи

Таблица 4 – simple_rules

№	Имя столбца	Тип данных	Комментарий
1	r_id	NUMBER(10)	id правила
2	prt_l	VARCHAR2(1)	часть речи левого слова в словосочетании
3	sing_l	VARCHAR2(1)	число левого слова в словосочетании
4	cow_l	VARCHAR2(1)	падеж левого слова в словосочетании

5	prt_r	VARCHAR2(1)	часть речи правого слова в словосочетании
6	sing_r	VARCHAR2(1)	число правого слова в словосочетании
7	cow_r	VARCHAR2(1)	падеж правого слова в словосочетании
8	ans	VARCHAR2(1)	ответ
9	comm	VARCHAR2(1000)	правило
10	ex	VARCHAR2(100)	пример

В таблицах 5 и 6 подразумевается нумерация слов слева направо.

Таблица 5 – add3_r

№	Имя столбца	Тип данных	Комментарий
1	r3_id	NUMBER(10)	id правила
2	prt_1	VARCHAR2(1)	часть речи первого слова в словосочетании
3	sing_1	VARCHAR2(1)	число первого слова в словосочетании
4	cow_1	VARCHAR2(1)	падеж первого слова в словосочетании
5	prt_2	VARCHAR2(1)	часть речи второго слова в словосочетании
6	sing_2	VARCHAR2(1)	число второго слова в словосочетании
7	cow_2	VARCHAR2(1)	падеж второго слова в словосочетании
8	prt_3	VARCHAR2(1)	часть речи третьего слова в словосочетании
9	sing_3	VARCHAR2(1)	число третьего слова в словосочетании
10	cow_3	VARCHAR2(1)	падеж третьего слова в словосочетании
11	ans	VARCHAR2(1)	ответ
12	comm	VARCHAR2(1000)	правило
13	ex	VARCHAR2(100)	пример

Таблица 6 – add4_r

№	Имя столбца	Тип данных	Комментарий
1	r4_id	NUMBER(10)	id правила

2	prt_1	VARCHAR2(1)	часть речи первого слова в словосочетании
3	sing_1	VARCHAR2(1)	число первого слова в словосочетании
4	cow_1	VARCHAR2(1)	падеж первого слова в словосочетании
5	prt_2	VARCHAR2(1)	часть речи второго слова в словосочетании
6	sing_2	VARCHAR2(1)	число второго слова в словосочетании
7	cow_2	VARCHAR2(1)	падеж второго слова в словосочетании
8	prt_3	VARCHAR2(1)	часть речи третьего слова в словосочетании
9	sing_3	VARCHAR2(1)	число третьего слова в словосочетании
10	cow_3	VARCHAR2(1)	падеж третьего слова в словосочетании
11	prt_4	VARCHAR2(1)	часть речи третьего слова в словосочетании
12	sing_4	VARCHAR2(1)	число третьего слова в словосочетании
13	cow_4	VARCHAR2(1)	падеж третьего слова в словосочетании
14	ans	VARCHAR2(1)	ответ
15	comm	VARCHAR2(1000)	правило
16	ex	VARCHAR2(100)	пример

ПРИЛОЖЕНИЕ В

Система правил для анализа словосочетаний из двух слов

Обозначения:

- p — часть речи: 1 — существительное, 5 — личный глагол, 6 — инфинитив, *b* — местоимение;
- s — число: «N» — множественное, «Y» — единственное, «–» — не определено (инфинитивы);
- c — падеж: 1 — именительный, 2 — родительный, 3 — дательный, 4 — винительный, 5 — творительный, 6 — предложный, «–» — не определено (личные глаголы и инфинитивы);
- A — ответ: «Y» — верно, «N» — неверно.

_r и _l — указатель правого и левого операнда соответственно.

Таблица 7 – Система правил для словосочетаний из двух слов

№	p_l	s_l	c_r	p_r	s_r	c_l	A	Пример
1	b	N	1	5	N	–	Y	мы делали
2	1	Y	1	5	N	–	N	собака лаяли
3	1	Y	1	5	Y	–	Y	самолёт летит
4	b	Y	1	5	Y	–	Y	я делаю
5	6	–	–	1	Y	4	Y	делать дело
6	5	Y	–	6	–	–	Y	хочет есть
7	6	–	–	b	Y	2	Y	знать его
8	6	–	–	1	N	5	Y	гордиться детьми
9	b	Y	1	6	–	–	Y	я есть
10	b	N	1	6	–	–	Y	вы есть
11	5	N	–	6	–	–	Y	пришли договориться
12	b	N	1	5	Y	–	N	мы писал
13	5	Y	–	b	Y	2	Y	победил меня
14	5	Y	–	b	N	1	N	вздохнул мы

15	5	Y	–	1	N	1	N	вздохнул люди
16	5	Y	–	1	Y	1	Y	бежал человек
17	5	N	–	1	Y	1	N	бегут собака
18	5	N	–	1	N	1	Y	бежали собаки
19	5	N	–	b	Y	1	N	бежали я
20	5	N	–	b	N	1	Y	бежали мы
21	6	–	–	b	N	2	Y	укусить нас
22	6	–	–	5	N	–	N	видеть хотели
23	6	–	–	5	Y	–	N	быть хотел
24	1	Y	3	6	–	–	Y	чуду быть
25	1	N	1	5	N	–	Y	люди делали
26	1	N	1	5	Y	–	N	люди учил
27	1	N	3	6	–	–	Y	праздникам быть
28	b	Y	1	5	N	–	N	я делали
29	b	Y	2	5	N	–	Y	меня ранили
30	b	Y	3	5	N	–	Y	мне позвонили
31	5	N	–	b	Y	2	Y	переиграли меня
32	5	N	–	b	N	2	Y	позвали нас
33	5	Y	–	1	N	4	Y	вижу кошек
34	5	N	–	1	N	2	Y	позвали друзей
35	1	Y	1	1	Y	3	Y	человек собаке
36	6	–	–	1	N	2	Y	кормить свиней
37	6	–	–	6	–	–	Y	хотеть пить
38	5	Y	–	b	N	2	Y	отчитал их
39	6	–	–	1	Y	3	Y	дать человеку
40	1	Y	1	1	Y	4	Y	дурак дурака
41	6	–	–	b	Y	3	Y	купить себе
42	6	–	–	b	Y	5	Y	быть собой
43	5	N	–	1	Y	4	Y	украли сердце
44	5	Y	–	1	Y	4	Y	кормил собаку
45	5	Y	–	b	Y	3	Y	помог мне

46	5	N	–	b	Y	3	Y	помогли мне
47	6	–	–	1	Y	5	Y	быть учёным
48	b	Y	1	b	Y	4	Y	он меня
49	1	Y	4	5	Y	–	Y	человека увидел
50	b	Y	1	1	Y	4	Y	он руку
51	1	Y	4	5	N	–	Y	человека ценят
52	1	N	4	1	Y	1	Y	(губит) людей вода
53	b	Y	1	1	Y	4	Y	я храм (воздвиг)
54	1	Y	1	b	Y	3	Y	ребёнок себе (приготовил)
55	b	Y	3	5	Y	–	Y	себе купил
56	5	Y	–	b	N	5	Y	играет нами
57	5	Y	–	b	Y	5	Y	играет тобой
58	5	N	–	b	Y	5	Y	гордятся тобой
59	b	Y	1	1	N	4	Y	он руки (моет)
60	1	N	1	1	Y	4	Y	руки руку
61	6	–	–	1	N	4	Y	проверять знания
62	1	N	1	b	Y	4	Y	дети его (знали)
63	b	Y	1	b	Y	3	Y	ты мне (должен)
64	5	Y	–	b	Y	1	Y	получился ты
65	1	Y	2	1	Y	2	Y	певца любви
66	b	Y	4	5	Y	–	Y	(оно) тебя видит
67	1	Y	1	1	N	2	Y	игра слов
68	1	N	1	1	Y	2	Y	порывы ветра
69	1	Y	4	1	N	2	Y	игру слов
70	b	Y	2	1	Y	4	Y	его игру
71	1	Y	5	1	Y	2	Y	сном младенца
72	b	Y	2	1	N	4	Y	его таблетки
73	b	N	2	1	Y	4	Y	их игру
74	5	Y	–	1	Y	3	Y	понравился собаке
75	5	Y	–	5	Y	–	Y	пойду схожу

76	1	Y	4	1	Y	2	Y	икру нерки
77	5	Y	–	b	N	4	Y	звал их
78	1	N	2	1	N	2	Y	знаний требований
79	1	N	2	1	Y	2	Y	требований охраны
80	5	N	–	1	N	5	Y	будьте людьми

Для таблицы 8 номера 1, 2 и 3 соответствуют первому, второму и третьему словам соответственно при их чтении слева направо.

Таблица 8 – Система правил для словосочетаний из трёх слов

№	**p1**	**s1**	**c1**	**p2**	**s2**	**c2**	**p3**	**s3**	**c3**	**A**	**Пример**
1	b	Y	1	1	Y	4	5	N	–	N	он руку моют
2	b	Y	1	1	N	4	5	N	–	N	он руки моют
3	1	N	1	1	Y	4	5	Y	–	N	руки руку моет
4	b	Y	1	1	N	4	5	Y	–	Y	он руки моет
5	b	Y	1	b	Y	4	5	Y	–	Y	оно тебя видит
6	6	–	–	b	Y	2	1	Y	5	Y	жить его жизнью
7	1	Y	1	1	Y	2	5	N	–	N	образ человека понравились
8	b	Y	2	1	Y	1	5	N	–	N	его образ понравились
9	b	N	2	1	Y	1	5	N	–	N	их образы понравились
10	1	Y	1	b	Y	2	5	Y	–	Y	человек тебя видит
11	1	N	1	b	Y	2	5	Y	–	N	люди тебя видит
12	b	N	1	b	Y	2	5	Y	–	N	они тебя видит
13	b	N	1	1	N	4	5	Y	–	N	они собак видит
14	1	Y	1	1	N	4	5	Y	–	Y	дурак дураков видит
15	1	Y	1	1	N	4	5	N	–	N	дурак дураков видят
16	6	–	–	b	Y	3	1	Y	4	Y	купить себе пони
17	6	–	–	b	N	3	1	Y	4	Y	купить нам пони
18	b	Y	1	b	Y	4	5	N	–	N	он меня видят
19	b	Y	1	b	N	4	5	N	–	N	он нас видят

20	b	N	1	b	N	4	5	Y	–	N	они нас видит
21	1	N	1	b	N	4	5	Y	–	N	люди нас видит
22	1	Y	1	b	N	4	5	N	–	N	человек нас видят
23	1	Y	1	b	Y	4	5	N	–	N	человек меня видят
24	1	N	1	b	N	4	5	N	–	Y	люди нас видят
25	b	Y	1	b	Y	3	5	N	–	N	он себе купил
26	b	N	1	b	Y	3	5	Y	–	N	они себе купил

Для таблицы 9 номера 1, 2, 3 и 4 соответствуют первому, второму, третьему и четвёртому словам соответственно при их чтении слева направо.

Таблица 9 – Система правил для словосочетаний из четырёх слов

№	**p1**	**s1**	**c1**	**p2**	**s2**	**c2**	**p3**	**s3**	**c3**	**p4**	**s4**	**c4**	**A**	**Пример**
1	b	Y	1	5	Y	–	6	–	–	1	N	5	N	он хотел быть учёными
2	b	Y	1	1	Y	4	b	Y	3	5	N	–	N	я памятник себе воздвигли
3	b	N	1	5	N	–	6	–	–	1	Y	5	N	они хотели быть учёным
4	1	Y	1	5	Y	–	6	–	–	1	N	5	N	Борис хотел быть учёными
5	1	N	1	5	N	–	6	–	–	1	Y	5	N	птицы хотели быть рыбой
6	1	Y	1	1	N	2	5	Y	–	b	Y	3	Y	игра слов нравилась мне