

## **Аннотация**

Тут будет аннотация

## Содержание

<b>1</b>	<b>Введение</b>	<b>4</b>
<b>2</b>	<b>Анализ проблемы</b>	<b>5</b>
<b>3</b>	<b>Описание подхода</b>	<b>6</b>
3.1	Идея . . . . .	6
3.2	Подготовительный этап . . . . .	7
3.3	Обработка предложения . . . . .	7
3.3.1	Упрощение предложения . . . . .	8
3.3.2	Непосредственная проверка предложения . . . . .	19
3.4	Примеры работы алгоритма . . . . .	19
<b>4</b>	<b>Заключение</b>	<b>20</b>
	<b>Список литературы</b>	<b>21</b>
	<b>ПРИЛОЖЕНИЕ А. Структура таблиц базы данных</b>	<b>22</b>
	<b>ПРИЛОЖЕНИЕ В. Система правил для анализа словосочетаний из двух слов</b>	<b>23</b>

## 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 [3].

## 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») [4]. 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» [5]. 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 [6]. 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 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, 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.

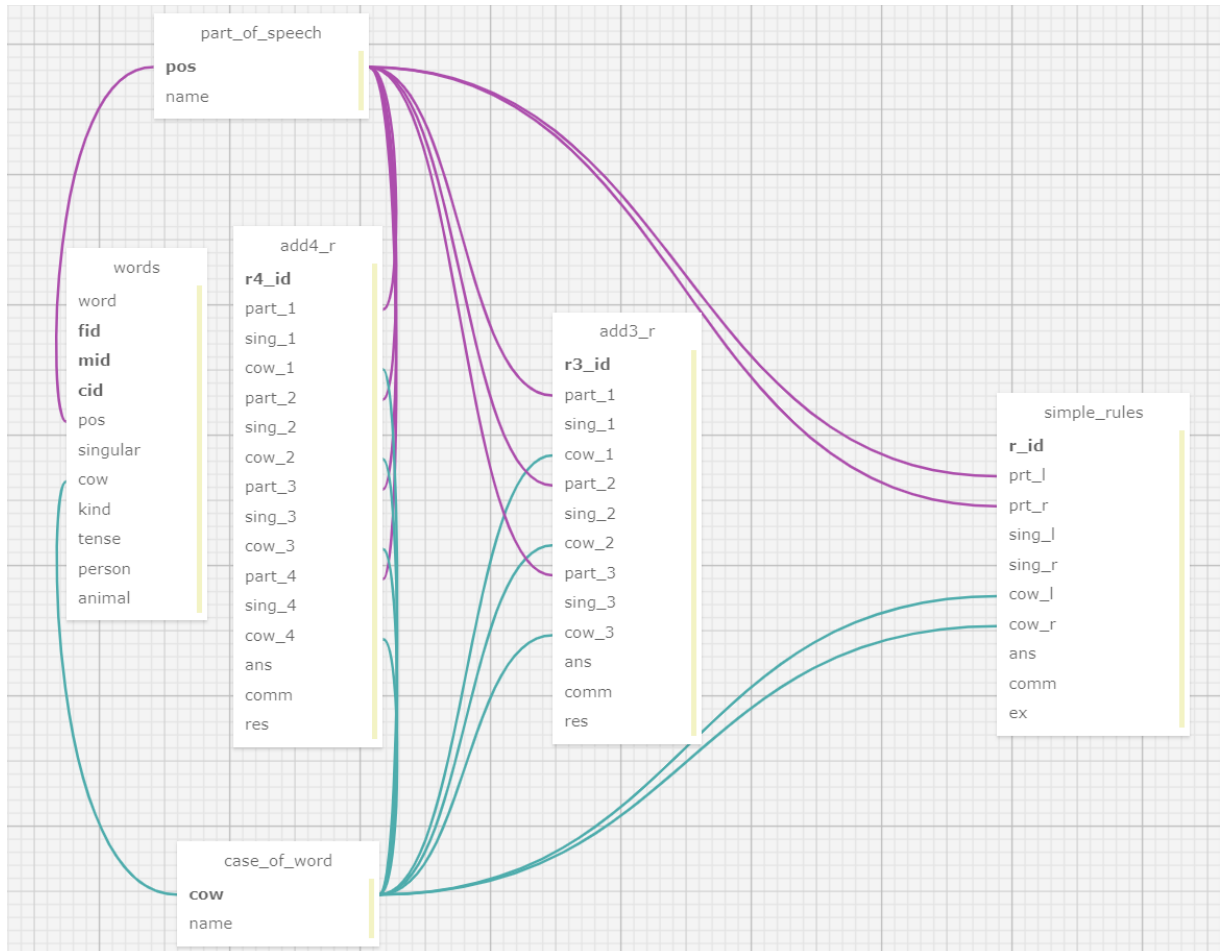


Рис. 1 – Схема усовершенствованной базы данных

В рамках предложенного нами подхода мы описываем слово при помощи трёх параметров: часть речи, число и падеж. Нами было выявлено около сотни правил для словосочетаний длины 2, 3 или 4, по которым можно определить, нет ли 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 [7].

### 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] ∈ {«.», «,», «.»} 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.

### 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.

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 СОММА(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)
6:   llen ← len(l)                                   ▷ Длина исходного списка
7:   if «и» ∈ l then
8:     w ← «и»                                       ▷ w хранит союз, использующийся в перечислении
9:   else if «или» ∈ l then
10:    w ← «или»
11:   end if
12:   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.

```

13:   if llen > id1 +1 then
14:       resp1 ← l[id1 +1].[pos, singular, cow]
15:   end if
16:   if llen > id1 +2 then
17:       resp2 ← l[id1 +2].[pos, singular, cow]
18:   end if
19:   if llen > id1 +3 then
20:       resp3 ← l[id1 +3].[pos, singular, cow]
21:   end if
22:   if llen > id1 +4 then
23:       resp4 ← l[id1 +4].[pos, singular, cow]
24:   end if
25:   if id1 -1 ≥ 0 then
26:       resl1 ← l[id1 -1].[pos, singular, cow]
27:   end if
28:   if id1 -2 ≥ 0 then
29:       resl2 ← l[id1 -2].[pos, singular, cow]
30:   end if
31:   if id1 -3 ≥ 0 then
32:       resl3 ← l[id1 -3].[pos, singular, cow]
33:   end if
34:   if id1 -4 ≥ 0 then
35:       resl1 ← l[id1 -4].[pos, singular, cow]
36:   end if
37:   if id1 -5 ≥ 0 then
38:       resl5 ← l[id1 -5].[pos, singular, cow]
39:   end if
40:   if id1 -6 ≥ 0 then
41:       resl6 ← l[id1 -6].[pos, singular, cow]
42:   end if
43:   if id1 -7 ≥ 0 then
44:       resl7 ← l[id1 -7].[pos, singular, cow]
45:   end if
46:   if id1 +1 < llen then
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 3 будет определено, слова (словосочетания) какой части речи перечисляются (если в предложении присутствует перечисление с союзом «и»).

Заметим, что в случае перечисления с союзом «и» 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 4.

Прежде всего, нужно определить начало левого операнда «и». 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 4 – Продолжение алгоритма 3

---

```

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
    «и»
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
101: end if
102: if id1-3 ≥ 0 and left= -1 and «,»∉ l[id1-3 : id1] then
103:     for from i = 0 to len(resl3)-1 do
104:         if resl3[i][0] = part then
105:             r ← check(l[id1-3 : id1])
106:             if «N» ∈ r then
107:                 return [«он», «писали» ]
108:             else if «Y» ∈ r then
109:                 left ← id1-3
110:                 break
111:             end if
112:         end if
113:     end for
114: end if

```

---

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

---

Алгоритм 6 – Продолжение алгоритма 5

---

```

115:      if  $\text{id1}-2 \geq 0$  and  $\text{left} = -1$  and «,<»  $\notin l[\text{id1}-2 : \text{id1}]$  then
116:          for from  $i = 0$  to  $\text{len}(\text{resl2})-1$  do
117:              if  $\text{resl2}[i][0] = \text{part}$  then
118:                   $r \leftarrow \text{check}(l[\text{id1}-2 : \text{id1}])$ 
119:                  if «N»  $\in r$  then
120:                      return [«он», «писали» ]
121:                  else if «Y»  $\in r$  then
122:                       $\text{left} \leftarrow \text{id1}-2$ 
123:                      break
124:                  end if
125:              end if
126:          end for
127:      end if
128:      if  $\text{id1}-1 \geq 0$  and  $\text{left} = (-1)$  then
129:          for from  $i = 0$  to  $\text{len}(\text{resl1})-1$  do
130:              if  $l[i][0] = \text{part}$  then
131:                   $\text{left} \leftarrow \text{id1}-1$ 
132:                  break
133:              end if
134:          end for
135:      end if
136:  end if
137:  Алгоритм 7
138:  Алгоритм 11
139:  Алгоритм 12
140: end function

```

---

В алгоритмах 5 и 6 неоднократно фигурирует функция `check()`. В данном случае она используется для проверки предложения, не содержащего знаки пунктуации. Её описание будет в следующем параграфе.

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 7.

---

Алгоритм 7 – Фрагмент алгоритма 6

---

```

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» ]
17:        else if «Y» ∈ r then
18:          left ← left - 3
19:          break
20:        end if
21:      end if
22:    end for
23:  else if l[left - 1] = «,» and l[left - 5] = «,» then
24:    res1 ← l[left - 4].[pos, singular, cow]
25:    for from i = 0 to len(res1) do
26:      if res1[i][0] = part then
27:        r ← check(l[left - 4 : left - 1])
28:        if «N» ∈ r then
29:          return [«он», «писали» ]
30:        else if «Y» ∈ r then
31:          left ← left - 4
32:          break
33:        end if
34:      end if

```

---

```
35:     end for
36:     else if l[left - 1] = «,» and l[left - 6] = «,» then
37:         res1 ← l[left - 5].[pos, singular, cow]
38:         for from i = 0 to len(res1) do
39:             if res1[i][0] = part then
40:                 r ← check(l[left - 5 : left - 1])
41:                 if «N» ∈ r then
42:                     return [«он», «писали» ]
43:                 else if «Y» ∈ r then
44:                     left ← left - 5
45:                     break
46:                 end if
47:             end if
48:         end for
49:     else if l[left - 1] = «,» and l[left - 7] = «,» then
50:         res1 ← l[left - 6].[pos, singular, cow]
51:         for from i = 0 to len(res1) do
52:             if res1[i][0] = part then
53:                 r ← check(l[left - 6 : left - 1])
54:                 if «N» ∈ r then
55:                     return [«он», «писали» ]
56:                 else if «Y» ∈ r then
57:                     left ← left - 6
58:                     break
59:                 end if
60:             end if
61:         end for
62:     else if l[left - 1] = «,» and l[left - 8] = «,» then
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 9
77:     end if
78: end while
```

---

Итак, в результате работы фрагментов 7 и 8 будет сдвинута граница до «первой» запятой.

Следующий этап — поиск начала перечисления. Соответствующий фрагмент описан алгоритмом 9.

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 9 – Фрагмент алгоритма 8

---

```

1: if left-2  $\geq$  0 and l[left-1] = «,» and end =  $(-1)$  then
2:   res1  $\leftarrow$  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  $\leftarrow$  left-2
6:       break
7:     end if
8:   end for
9: end if
10: if left-3  $\geq$  0 and l[left-1] = «,» and end =  $(-1)$  then
11:   res1  $\leftarrow$  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  $\leftarrow$  check(l[left-3 : left-1])
15:       if «N»  $\in$  r then
16:         return [«он», «писали» ]
17:       else if «Y»  $\in$  r then
18:         left  $\leftarrow$  left-3
19:         end  $\leftarrow$  1
20:         break
21:       end if
22:     end if
23:   end for
24: end if
25: if left-4  $\geq$  0 and l[left-1] = «,» and end =  $(-1)$  then
26:   res1  $\leftarrow$  l[left-4].pos, singular, cow
27:   for from i = 0 to len(res1)-1 do
28:     if res1[i][0] = «6» then
29:       r  $\leftarrow$  check(l[left-4 : left-1])
30:       if «N»  $\in$  r then
31:         return [«он», «писали» ]
32:       else if «Y»  $\in$  r then
33:         left  $\leftarrow$  left-4
34:         end  $\leftarrow$  1
35:         break
36:       end if

```

---

```

37:     end if
38:   end for
39: end if
40: if left-5 ≥ 0 and l[left-1] = «,» and end = (-1) then
41:   res1 ← l[left-5].[pos, singular, cow]
42:   for from i = 0 to len(res1)-1 do
43:     if res1[i][0] = «6» then
44:       r ← check(l[left-5 : left-1])
45:       if «N» ∈ r then
46:         return [«он», «писали» ]
47:       else if «Y» ∈ r then
48:         left ← left-5
49:         end ← 1
50:         break
51:       end if
52:     end if
53:   end for
54: end if
55: if left-6 ≥ 0 and l[left-1] = «,» and end = (-1) then
56:   res1 ← l[left-6].[pos, singular, cow]
57:   for from i = 0 to len(res1)-1 do
58:     if res1[i][0] = «6» then
59:       r ← check(l[left-6 : left-1])
60:       if «N» ∈ r then
61:         return [«он», «писали» ]
62:       else if «Y» ∈ r then
63:         left ← left-6
64:         end ← 1
65:         break
66:       end if
67:     end if
68:   end for
69: end if
70: if left-7 ≥ 0 and l[left-1] = «,» and end = (-1) then
71:   res1 ← l[left-7].[pos, singular, cow]
72:   for from i = 0 to len(res1)-1 do
73:     if res1[i][0] = «6» then
74:       r ← check(l[left-7 : left-1])
75:       if «N» ∈ r then
76:         return [«он», «писали» ]
77:       else if «Y» ∈ r then
78:         left ← left-7
79:         end ← 1
80:         break
81:       end if
82:     end if
83:   end for
84: end if

```

---

Итак, после выполнения алгоритма 9 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 11 – Фрагмент алгоритма 6

---

1:  $l \leftarrow l[: \text{left}] + [\text{«учить»}] + l[\text{right}+1 : \text{id1}]$

---

Если же при помощи союза «и» перечисляются личные глаголы, то упрощение идёт согласно алгоритму 12. 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 12 – Продолжение алгоритма 6

---

```

1: if part=«5» then
2:   sng0 ← []                                ▷ Для определения числа первого слова в предложении
3:   pov ← (-1)                               ▷ Индикатор повелительного наклонения
4:   res0 ← l[0].[cow, singular, cow]
5:   end ← (-1)
6:   for from i = 0 to len(res0)-1 do
7:     if res0[i][0] = «5» then
8:       sng0 ← sng0 + list(res0[i][1])
9:     end if
10:  end for
11:  sng0 ← list(set(sng0))
12:  if len(sng0) > 1 then
13:    pov ← 1
14:    sng ← «Y»                                ▷ Считаем единственным число перечисляемых личных
    глаголов
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» and «Y» ∈ r) then
27:        left ← id1-7
28:      else
29:        return [«он», «писали» ]
30:      end if
31:    end if
32:  end if
33: end if

```

---

Здесь следует рассмотреть решение проблемы омографии. В строках 16 – 32 алгоритма 12

### **3.3.2. Непосредственная проверка предложения**

### **3.4. Примеры работы алгоритма**

#### **4. Заключение**

Тут будет заключение

## 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] **Чесебиев, И. А. Компьютерное распознавание и порождение речи:** монография. – Москва: Спорт и Культура-2000, 2008. – 125 с.
- [4] **Comrie, B. Language universals and linguistic typology: Syntax and morphology.** – University of Chicago press, 1989.
- [5] **Conway, D. An algorithmic approach to English pluralization //** Proceedings of the Second Annual Perl Conference. – 1998.
- [6] **The world atlas of language structures** / M. Haspelmath [and others]. – Oxford Univ. Press, 2005.
- [7] LanguageTool — Проверка грамматики и стилистики [Электронный ресурс] — <https://languagetool.org/ru>

ПРИЛОЖЕНИЕ А

**Структура таблиц базы данных**

## ПРИЛОЖЕНИЕ В

### Система правил для анализа словосочетаний из двух слов

Таблица 1 – Система правил для словосочетаний из двух слов

№	prt_l	sing_l	cow_r	prt_r	sing_r	cow_l	ans	example
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 догово- риться
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