

Аннотация

Тут будет аннотация

Содержание

1	Введение	4
2	Описание подхода	5
2.1	Подготовительный этап	5
2.2	Обработка предложения	5
2.2.1	Упрощение предложения	6
	Список литературы	15
	ПРИЛОЖЕНИЕ А. Система правил для анализа словосочетаний из двух слов	16

1. Введение

Тут будет введение

2. Описание подхода

2.1. 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      ▷ Если последним элементом полученного
    списка оказался пробел
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.

2.2. Обработка предложения

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

В основе предложенного нами подхода лежит гипотеза, согласно которой одно 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 [2].

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 [1]. Чтобы избежать противоречивости разработанной системы, мы включили лишь те правила, которые встречаются на практике, а не перебрали все возможные комбинации используемых нами параметров.

2.2.1. 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. Механизм работы функции `comma()` описывает алгоритм 2.

Алгоритм 2 – Обработка перечислений

1: function COMMA(l)	▷ l — подготовленная строка в виде списка
2: if «и» in l then	
3: part ← []	▷ Список значений параметра «часть речи» для данного слова (изначально пустой)
4: end ← (−1)	▷ Индикатор нахождения начала перечисления
5: left ← (−1)	▷ Левая и правая границы заменяемого участка изначально
6: right ← (−1)	▷ инициализируем невозможными значениями: (−1)
7: llen ← len(l)	▷ Длина исходного списка
8: id1 ← l.index(«и»)	▷ Записываем индекс «и» в списке

Для начала инициализируем переменные, затем находим индекс вхождения «и» в список (при условии, что в списке есть «и»). После этого анализируем слова, находящиеся в окрестности слова «и»:

```

9:      if llen > id1 +1 then
10:         resp1 ← l[id1 +1].[pos, singular, cow]  ▷ Варианты интерпретации слова,
            стоящего за «и»
11:      end if
12:      if llen > id1 +2 then
13:         resp2 ← l[id1 +2].[pos, singular, cow]
14:      end if
15:      if llen > id1 +3 then
16:         resp3 ← l[id1 +3].[pos, singular, cow]
17:      end if
18:      if llen > id1 +4 then
19:         resp4 ← l[id1 +4].[pos, singular, cow]
20:      end if
21:      if id1 -1 ≥ 0 then
22:         resl1 ← l[id1 -1].[pos, singular, cow]
23:      end if
24:      if id1 -2 ≥ 0 then
25:         resl2 ← l[id1 -2].[pos, singular, cow]
26:      end if
27:      if id1 -3 ≥ 0 then
28:         resl3 ← l[id1 -3].[pos, singular, cow]
29:      end if
30:      if id1 -4 ≥ 0 then
31:         resl1 ← l[id1 -4].[pos, singular, cow]
32:      end if
33:      if id1 -5 ≥ 0 then
34:         resl5 ← l[id1 -5].[pos, singular, cow]
35:      end if
36:      if id1 -6 ≥ 0 then
37:         resl6 ← l[id1 -6].[pos, singular, cow]
38:      end if
39:      if id1 +1 < llen then
40:         for from i = 0 to len(resp1)-1 do
41:            if resp1[i][0] = «6» then  ▷ Слово оказалось инфинитивом
42:               part ← «6»
43:               right ← id1 +1
44:            end if
45:         end for
46:         if right = (-1) then  ▷ Если же это не инфинитив
47:            for from i = 0 to len(resp1)-1 do
48:               if resp1[i][0] = «5» then  ▷ Слово оказалось личным глаголом
49:                  right ← id1+1
50:                  part ← «5»
51:                  sng ← l[i][1]  ▷ Для личных глаголов важно число
52:                  break
53:               end if
54:            end for
55:         end if
56:      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

```

57:      if part = «6» then                ▷ Если перечисляемая часть речи — инфинитив
58:      if id1-6 ≥ 0 and left = (-1) and «,» ∉ l[id1-6 : id1] then ▷ Проверяем
      буквосочетания длины 6
59:      for from i = 0 to len(resl6)-1 do
60:      if resl6[i][0] = part then
61:      r ← check(l[id1-6 : id1])
62:      if «N» ∈ r then
63:      return [«он», «писали» ]           ▷ Заведомо неверное
      предложение

```

```

64:                else if «Y» ∈ r then
65:                    left ← id1-6                ▷ Инициализировали границу левого
операнда «и»
66:                    break
67:                end if
68:            end if
69:        end for
70:    end if
71:    if id1-5 ≥ 0 and left = -1 and «,» ∉ l[id1-5 : id1] then
72:        for from i = 0 to len(resl5)-1 do
73:            if resl5[i][0] = part then
74:                r ← check(l[id1-5 : id1])
75:                if «N» ∈ r then
76:                    return [«он», «писали» ]
77:                else if «Y» ∈ r then
78:                    left ← id1-5
79:                    break
80:                end if
81:            end if
82:        end for
83:    end if
84:    if id1-4 ≥ 0 and left = -1 and «,» ∉ l[id1-4 : id1] then
85:        for from i = 0 to len(resl4)-1 do
86:            if resl4[i][0] = part then
87:                r ← check(l[id1-4 : id1])
88:                if «N» ∈ r then
89:                    return [«он», «писали» ]
90:                else if «Y» ∈ r then
91:                    left ← id1-4
92:                    break
93:                end if
94:            end if
95:        end for
96:    end if
97:    if id1-3 ≥ 0 and left = -1 and «,» ∉ l[id1-3 : id1] then
98:        for from i = 0 to len(resl3)-1 do
99:            if resl3[i][0] = part then
100:                r ← check(l[id1-3 : id1])
101:                if «N» ∈ r then
102:                    return [«он», «писали» ]
103:                else if «Y» ∈ r then
104:                    left ← id1-3
105:                    break
106:                end if
107:            end if
108:        end for
109:    end if

```

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

Алгоритм 6 – Продолжение алгоритма 5

```

110:         if  $id1-2 \geq 0$  and left = -1 and «,<»  $\notin l[id1-2 : id1]$  then
111:             for from  $i = 0$  to  $len(resl2)-1$  do
112:                 if  $resl2[i][0] = part$  then
113:                      $r \leftarrow check(l[id1-2 : id1])$ 
114:                     if «N»  $\in r$  then
115:                         return [«он», «писали» ]
116:                     else if «Y»  $\in r$  then
117:                         left  $\leftarrow id1-2$ 
118:                         break
119:                     end if
120:                 end if
121:             end for
122:         end if
123:         if  $id1-1 \geq 0$  and left = (-1) then
124:             for from  $i = 0$  to  $len(resl1)-1$  do
125:                 if  $l[i][0] = part$  then
126:                     left  $\leftarrow id1-1$ 
127:                     break
128:                 end if
129:             end for
130:         end if
131:     end if
132:     Алгоритм 7.
133: end if
134: Алгоритм
135: 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 «и» связывает только два сочетания.
- Союз «и» используется для перечисления 3 и более словосочетаний.

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

```

Список литературы

- [1] 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.: ил.
- [2] LanguageTool — Проверка грамматики и стилистики [Электронный ресурс] — <https://languagetool.org/ru>

ПРИЛОЖЕНИЕ А

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