*Громов Петр Александрович*

# ФУНКЦИОНАЛЬНОЕ ПРЕДСТАВЛЕНИЕ ТЕКСТА

**Аннотация**

Обычно программа, работающая с синтаксическими структурами естественного языка, должна содержать синтаксический анализатор – парсер. Альтернативой созданию своего парсера является использование существующих, которые могут не совсем хорошо подходить для конкретной задачи программы. Лучшее решение заключается в создании одного парсера, выдающего в результате работы такую структуру данных, которая могла бы быть применима к как можно большему количеству прикладных задач. В статье предложен такой формат выходных данных парсера, приведены его преимущества по сравнению с более традиционными структурами представления текста и показана его к ним сводимость, а также возможности этого формата по автоматическому разрешению лексических неоднозначностей.

**Ключевые слова:** парсер, структура непосредственно составляющих, структура зависимостей, синтаксис, семантика, разрешение неоднозначностей.

## ВВЕДЕНИЕ

В существующих анализаторах естественных языков (парсерах) результатом работы, как правило, является некая структура в памяти компьютера, отражающая важные для целевой задачи свойства предложения. Часто для этого применяются структура зависимостей [1, 2] и структура непосредственно составляющих [3]. Построенная структура затем используется для решения более практических задач обработки текста, таких как извлечение знаний, машинный перевод и т. п. Парсеры пишутся с целью построить совершенно конкретную структуру данных. К сожалению, ни одна из известных мне подобных структур не является универсальной и одинаково хорошо подходящей для любой задачи, что затрудняет переиспользование парсера.

Идеальный парсер выдавал бы результат, принимаемый в качестве входных данных остальными, более специфичными программами интеллектуальной обработки текста. Я расскажу о возможном способе представления такого результата. Это будет не какая-либо определенная структура данных, а набор инструкций, которые, будучи выполнены в правильном порядке, эту структуру соберут. Способ этот не нов, по крайней мере, Т. Виноград и В. Тузов применяли его при анализе текста [4, 5]. К сожалению, неявно: оба они описывают алгоритмы работы своих анализаторов и построения результата, при этом не сильно фокусируя внимание читателя на общей методологии. А ведь эта методология весьма красива и стоит того, чтобы ее использовать. Задача этой статьи – выявить ее и выделить как отдельный шаг работы любой программы обработки текстов.



## КОМПОЗИЦИЯ ФУНКЦИЙ

Предлагаю сосредоточиться не на том, что получается в результате работы лингвопроцессора, а на том, как и в какой последовательности это происходит. Для этого рассмотрим примеры разбора одного предложения в различных анализаторах. Предложением будет «Четыре черненьких чумазеньких чертенка чертили черными чернилами чертеж». Все алгоритмы записаны на псевдоязыке программирования, который является чем-то средним между Python и Groovy. Сам процесс

**Листинг 1**

```
devil = new Node("чертенка")
devil.propert.add(new Node("чумазеньких"))
devil.propert.add(new Node("черненьких"))
devil.quantit = new Node("4")

ink = new Node("чернилами")
ink.propert.add(new Node("черными"))

draw = new Node("чертили")
draw.sub = devil
draw.obj = new Node("чертеж")
draw.method = ink
```

*«Четыре черненьких чумазеньких чертенка чертили черными чернилами чертеж»*

анализа в коде не участвует, приведен только минимальный набор инструкций, строящих результат.

Таким, например, может быть алгоритм построения поверхностно-семантической структуры из проекта AOT [1] (см. рис. 1, листинг 1).

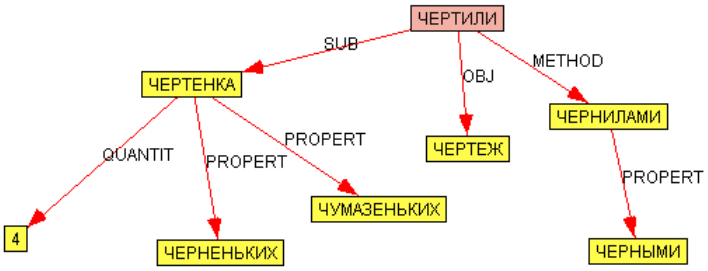



**Рис. 1**

**Листинг 2**

```
triplets = []
devil = new Devil()
triplets += new Triplet(devil, "Cleanliness", -5)
triplets += new Triplet(devil, "Color", "black")
triplets += new Triplet(devil, "Quantity", 4)

ink = new Ink()
triplets += new Triplet(ink, "Color", "black")

draw = new DrawAction()
triplets += new Triplet(draw, "Subject", devil)
drawing = new Drawing()
triplets += new Triplet(draw, "Object", drawing)
triplets += new Triplet(draw, "When", "past")

triplets += new Triplet(draw, "Instrument", ink)
```

А таким – формирование списка логических триплетов вида «субъект – предикат – объект» (см. рис. 2, листинг 2).

Можно еще построить структуру непосредственно составляющих (см. рис. 3, листинг 3).

Цели и результаты у этих программ довольно-таки сильно отличаются друг от друга, но во всех трех алгоритмах прослеживается одна и та же последовательность действий, а именно:

**1.** Обработать «чертенка».
**2.** Обработать «чумазеньких».
**3.** Связать результаты 2 и 1 («чумазеньких чертенка»).
**4.** Обработать «черненьких».
**5.** Связать результаты 4 и 3 («черненьких чертенка»).
**6.** Обработать «четыре».
**7.** Связать результаты 6 и 5 («четыре чертенка»).
**8.** Обработать «чернилами».
**9.** Обработать «черными».
**10.** Связать результаты 9 и 8 («черными чернилами»).
**11.** Обработать «чертеж».
**12.** Обработать «чертили».
**13.** Связать результаты 12 и 7 («(четыре чертенка) чертили»).
**14.** Связать результаты 13 и 11 («чертили чертеж»).
**15.** Связать результаты 14 и 10 («чертили чернилами»).

Алгоритм весьма абстрактен: под словами «обработать» и «связать» могут скрываться любые специфичные для задачи парсера операции. Его можно также выразить в виде одного единственного выражения, представляющего собой композицию функций. Для удобства чтения слова «обработать» и «связать» сокращены до одной буквы, и аргументы одного и того же вызова «с» («связать») помещены друг под другом:

```
c(c(c(o(чертили),
      c(o(четыре),
        c(o(черненьких),
          c(o(чумазеньких),
            o(чертенка)))))),
    o(чертеж)),
  c(o(черными),
    o(чернилами)))
```

Итак, наши три задачи обработки текста свелись к такой абстрактной композиции функций. Легко показать, что многие другие задачи тоже сводятся к исполнению такого же функционального выражения. Построение этого выражения – основная задача парсера, она непроста и выходит за рамки данной статьи.

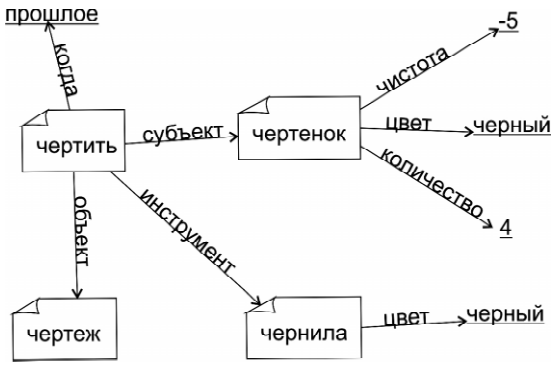



**Рис. 2**

Остается реализовать две функции: «обработать» и «связать». Функция «обработать» обычно реализуется нехитро и сводится к получению из словаря записи,

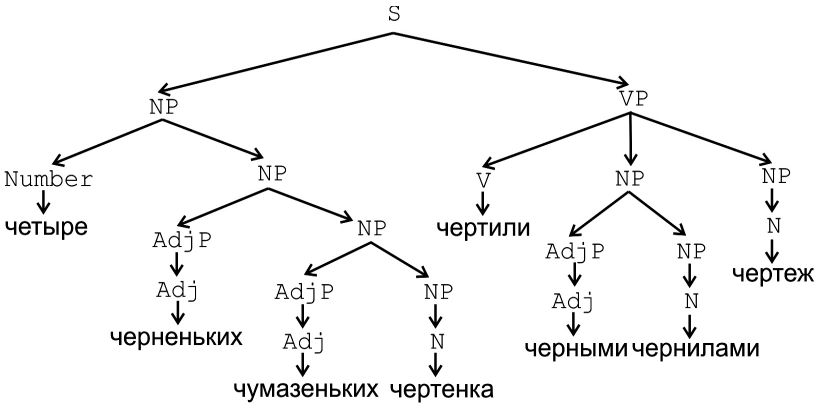



**Рис. 3**

**Листинг 3**

```
devils = new NP(new N("чертенка"))
dirtyDevils = new NP(devils, new AdjP(new Adj("чумазеньких")))
blackDevils = new NP(dirtyDevils, new AdjP(new Adj("черненьких")))

fourOfThem = new NP(blackDevils, new Number("четыре"))

ink = new NP(new N("чернилами"))
blackInk = new NP(ink, new AdjP(new Adj("черными")))

drawing = new NP(new N("чертеж"))

drew = new VP(new V("чертили"))
clause = new S(fourOfThem, drew)
drew.add(drawing)
drew.add(ink)

return clause
```

соответствующей слову-параметру, в наших трех примерах соответствующий код даже не всегда удостоен отдельной строчки. Для удобочитаемости в дальнейшем «o(X)» будем заменять на «X».

Функция связывания же намного интереснее, поскольку во многих задачах именно связи между словами являются главными при обработке. Принимает эта функция два параметра и должна их должным образом обработать. При этом нужно каждую комбинацию параметров рассматривать отдельно и вручную определять поведение программы для этих параметров. Количество возможных пар слов или синтактико/семантических категорий достаточно велико, чтобы функция приобрела очень значительные размеры. В связи с этим нелишним будет разбить этот монолит на части. Сделать это можно, объявив один из элементов каждой из взаимодействующих пар функцией, а второй из них – аргументом этой функции. В результате наш пример примет такой вид (см. листинг 4).

Стоящее в начале выражение «черными(чернилами)», после которого в скобках идет «чертили(...)(чертеж)», означает всего лишь, что первое выражение вернет функцию, которую мы и применим ко второму. Аналогичным образом чертили(...) возвращает функцию, применяемую потом к аргументу «чертеж». Это эквивалентно тому, как если бы функции принимали два аргумента: «чертили(четыре(...), чертеж)», просто для общности бывает удобно, когда все функции имеют ровно один аргумент. Запись в виде композиции функций, соответствующих словам, будем называть *функциональной структурой* предложения. Именно ее и предлагается строить обобщенному парсеру естественного языка, чтобы быть применимым к множеству задач.

Может возникнуть резонный вопрос: как именно я в каждом случае переписывания на новый лад операции «связать» определял, какой из ее параметров будет функцией, а какой – аргументом функции. В частности, почему прилагательное «черными» есть функция от существительного «чернилами» и почему их объединение есть функция от глагола «чертили», а не наоборот. Ведь в традиционных структурах составляющих и зависимостей как раз существительное и глагол являются главными в соответствующих отношениях.

Тут нужно вспомнить, зачем мы разбивали операцию связывания на части: чтобы получившиеся функции слов были маленькими и обозримыми. Следовательно, нужно стараться уменьшить количество возможных альтернатив их аргументов. Если сделать существительное функцией, ответственной за обработку всех своих зависимых слов: прилагательных, предложных групп, определений в виде придаточных предложений и т.п. («деревянный дом с изразцами, стоящий на берегу реки»), то функция эта получится очень большой, сложной для реализации, и нужно будет разбирать огромное количество случаев. Прилагательное же сочетается в основном с управляющими существительными (при этом должно присутствовать согласование), поэтому логично взвалить бремя обработки данной связи на него. В результате функция-существительное не будет знать о прилагательных вообще, а функция-прилагательное будет твердо знать, что ей могут «скармливать» только существительные. Значит, обе функции будут просты и невелики по размеру.

Можно обобщить это правило. В каждой бинарной связи в языке хотя бы один элемент сам по себе самостоятельно в полных предложениях не встречается, требуя наличия второго элемента. При этом первый элемент часто можно убрать, сохранив грамматичность предложения. Примеры несамостоятельных элементов русского языка: прилагательное без существительного; предлог без подчиненной имен-

**Листинг 4**

```
черными(чернилами) (чертили(четыре(черненьких(чумазеньких(чертенка)))) (чертеж))
```

ной группы; предложная группа без того, что она описывает; существительное в косвенном падеже без того, что описывает оно; переходный глагол без прямого дополнения и т. п. Во всех таких случаях кажется разумным сделать функцией менее самостоятельный элемент взаимодействующей пары. Из предложения про чертят вполне можно выкинуть косвенное дополнение «черными чернилами», а вот глагол уже сложнее – значит, в данном случае именно «черными(чернилами)» есть функция от «чертили».

Кроме того, немаловажным свойством функции является то, что ее результат сильно зависит от аргумента. Предлог «в» может принимать на вход слова «лесу» или «полдень», при этом результаты могут очень сильно различаться, особенно если анализатор строит какую-нибудь семантическую структуру. Глагол «читать» будет иметь разный смысл в зависимости от прямого дополнения («лекцию» или «книгу»); «переводить» можно «текст» (с языка эсперанто на хинди) или «бабушку» (через дорогу). Таким образом, получаем еще один критерий распределения ролей – если один из аргументов может принимать разные семантические значения в зависимости от другого – он и есть функция. Прямые дополнения, в отличие от косвенных, часто определяют семантику глагола и не мыслятся в предложении сами по себе, то есть подпадают и под первое правило, являясь, таким образом, аргументами функций-глаголов.

В итоге мы получили два критерия выделения функций:

1. Несамостоятельный участник взаимодействия является функцией самостоятельного.

2. Семантически зависимый участник взаимодействия является функцией, аргумент которой при этом определяет его значение.

Эти критерии не очень четки, могут в принципе противоречить друг другу, и не покрывают всего разнообразия случаев. При этом они неплохо описывают регулярные взаимодействия основных частей речи. Вместе они выдают нечто похожее на известное в лингвистике деление слов, зависимых от некоторого слова, на *актанты*, которые непосредственно фигурируют в семантическом значении этого слова, и отсутствие которых часто делает предложение неполным, и *сирконстанты* – все остальные [6]. Например, в предложении «Свидетелей убирают в полночь» дополнение «свидетелей» является актантом глагола «убирать», а обстоятельство «в полночь» – сирконстантом.

Актанты обычно являются аргументами функции, соответствующей слову, а сирконстанты – функциями от значения, выданного ею. При этом функции-сирконстанты, а также функции, соответствующие связям от зависимого слова к главному (прилагательное-существительное, наречие-глагол и прочие), имеют важную особенность: их результат очень похож на их аргумент. Прилагательное принимает именную группу и выдает именную группу, при этом род, число и падеж сохраняются, да и семантика меняется не сильно (были «чернила», стали тоже «чернила», только еще и «черные»). Будем говорить, что такие функции *модифицируют* свой аргумент (при этом не меняя его основных свойств).

В языке довольно много нерегулярностей, которые обычно несколько хуже описываются формальными лингвистическими моделями. Обычно это всяческие устойчивые или идиоматические выражения и фразеологизмы, представляющие собой единое целое, где ни одна часть не главнее других, вроде «баш на баш», «во всю Ивановскую», «короче говоря». Их тоже можно представить как набор взаимодействующих пар, только часто оба элемента пары немыслимы друг без друга. В этом случае можно либо объединять такие выражения в одно целое и сопоставлять им одну сложную функцию, либо, если уж взаимодействие хочется оставить, назначить функцию и ее аргумент произвольно, либо добавить в каждый из элементов пары возможность быть как функцией, так и аргументом в этой паре.

## СИНТАКСИЧЕСКИЕ КАТЕГОРИИ В ФУНКЦИОНАЛЬНОМ ПРЕДСТАВЛЕНИИ

Рассмотрим способы представления основных частей речи русского языка в виде функций:

- *Имя существительное.* В именительном и винительном падежах чаще всего является аргументом глагола, в остальных падежах – либо актант (аргумент), либо сирконстант (функция) чего либо, чаще всего глагола. Является также функцией от своих актантов, буде таковые есть.
- *Имя прилагательное.* Функция от согласованной с ним именной группы.
- *Глагол.* Функция от своих актантов.
- *Наречие.* Функция от того, что оно модифицирует: прилагательного, глагола, причастия, деепричастия или другого наречия.
- *Причастие.* Функция от актантов и от согласованной именной группы.
- *Деепричастие.* Функция от актантов и глагола подчиняющей клаузы.
- *Местоимение.* Ведет себя как та часть речи, которую оно заменяет: прилагательное, существительное, наречие.
- *Предлог.* Функция от подчиненной именной группы, выдает некоторое значение, которое может быть как актантом чего-либо (тогда оно будет аргументом), либо сирконстантом (тогда оно само будет функцией).
- *Сочинительный союз.* Функция, принимает первое связуемое, выдает функцию, принимающую второе связуемое. Эквивалентно функции от двух аргументов.
- *Подчинительный союз или союзное слово.* Функция, принимает подчиненную клаузу (глагол), возвращает функцию-сир-константа на родителе (например, именной группе).

Знаки препинания также можно представлять функционально. Точка, вопросительный и восклицательный знаки будут принимать законченные предложения и обрабатывать их нужным образом. Двоеточие – обрабатывать перечисления. Запятая может выполнять несколько задач: это и разделение однородных членов, и отделение причастных и деепричастных оборотов, вводных слов, и многое другое – все это тоже можно выразить через функции.

*«дом, который построил Джек»*

Интересно представление сложноподчиненных предложений. Во фрагменте «дом, который построил Джек» слово «который» играет две роли. Во-первых, оно является вершиной подчиненной клаузы: придаточного предложения, определяющего существительное «дом». Во-вторых, оно играет роль прямого дополнения, подчиненного глаголу «построил». Структуру зависимостей наличие двух родителей у узла вводит в легкий ступор, структура составляющих вынуждена вводить трансформации. Наше же представление легко справится с данной проблемой:

```
«запятая»(который(построил)(Джек))(Дом)
```

Первым делом «который» применяется к «построил». Результат будет во многих отношениях похож на аргумент. На глаголе появится пометка, что глагол находится в придаточном предложении. Его актант, соответствующий прямому дополнению, будет также заполнен. Далее, «построил» обретает актора, применяясь к слову «Джек». Запятая, встретив глагол, все еще специально помеченный как вершина придаточной клаузы, не может на это не среагировать и не выдать, наконец, функцию-модификатор, применяемую к именным группам типа «дом».

## ДОСТОИНСТВА

### *Нейтральность*

Анализатору языка предлагается выдавать обобщенную композицию абстрактных функций. Чтобы получить потом нужную задаче конкрет-

ную структуру, необходимо реализовать собственную версию всех функций-слов. Это не так страшно, как может показаться, потому что в системе может существовать разумная их реализация по умолчанию. Функция, соответствующая прилагательному, может просто возвращать свой аргумент-существительное, тем самым никак не влияя на семантику, но сохраняя при этом синтаксические свойства существительного. Тогда нужно будет добавить только тот код, который действительно нужен для конкретной задачи.

К примеру, если задачей является ответ на вопрос «где Ася?» на основе текста, то решение может состоять в том, чтобы специально обрабатывать только те узлы, которые могут дать необходимую информацию. Однозначно нужно обрабатывать «Асю» во всех падежах, предлог «в» в сочетании с описаниями мест, а также глаголы передвижения «находиться», «уехать», «вернуться» и т. п., вполне могущие иметь общую семантику, и, следовательно, общие части в реализации. Возможно, понадобится также разрешать ссылки, как, например, в предложении «Ася предупреждала, что в Москву она уедет тринадцатого ноября». Понимание, на что ссылается «она», очевидно, относится к операциям общеязыкового уровня, и, скорее всего, это задача парсера, клиенты не должны ею заниматься. Таким образом, программа, отвечающая на вопросы о местонахождении Аси, будет сама по себе невелика, обрабатывая только необходимые ей фрагменты текста. Остальное будет обеспечиваться парсером в виде функциональной структуры с реализацией большинства функций по умолчанию.

***Отсутствие канонического представления***

В теории составляющих нет однозначного ответа на вопрос о структуре словосочетания «больной безумный мир». Именных групп может быть либо две: NP[больной [NP безумный N[мир]]], либо одна: NP[больной безумный N[мир]]. Первый случай хорош тем, что узел NP имеет фиксированное количество детей (два), в отличие от второго, где число определений может быть в принципе любым. С другой стороны, тот же первый вариант предполагает, что слово «безумный» теснее связано со словом «мир», чем «больной», что в данном случае явно неправда, оба прилагательных характеризуют существительное в равной степени.

В нашей же системе отсутствуют строгие правила, под которые надо подгонять абсолютно все многообразие языка. Представление определяется удобством использования, и для большинства предложений - существует несколько равноправных эквивалентных структур. Композиция функций для словосочетания выше будет, скорее всего, такой: «больной(безумный(мир))». Или «безумный(больной(мир))». Или даже «(безумный(больной))(мир)», где функция «безумный», будучи примененной к прилагательному, объединяет оба значения связкой «И». Можно даже ввести специальную функцию для такого рода связки, получив при этом «(И(больной, безумный))(мир)». Но для большинства задач не имеет большого значения, какую из функций применять первой. Чаще всего функции слов «больной» и «безумный» будут просто делать где-нибудь в аргументе-существительном пометки касательно степени его болезненности и безумности, при этом никак не конфликтуя. Если при описании функций-модификаторов придерживаться разумных ограничений, что они должны влиять только на строго определенные аспекты своих аргументов, проблем с порядком применения не будет.

Кроме того, никто не накладывает ограничения, что функциям должны соответствовать именно слова. Это могут делать и морфемы, которые в данном примере добавляют синтаксическую характеристику рода к семантике корня: «ой(больн)(ый(безумн)(мир))». С помощью такого подхода легко анализируются сложные слова («теоретико-числовой») и морфемы, находящиеся вне слова («теле- и радиотрансляция»). А фразеологизм «ни

*«Эти типы стали есть в цехе»*

рыба ни мясо» не имеет никакого смысла собирать из четырех функций-слов, лучше ввести для него одну специфическую функцию.

Есть проблема: отсутствие каноничности противоречит основной цели введения функциональной структуры – легкости в переиспользовании парсера. Конечно, конкретный парсер должен стремиться быть последовательным в структуре своего результата, чтобы не взваливать лишнюю работу на клиентов-получателей этой структуры. Но при этом он может изначально выбрать любую степень детализации базовых функций – от букв и морфем до словосочетаний и предложений, при этом имея возможность получить сходные результаты, независимо от сделанного выбора.

### *Разрешение неоднозначностей*

Теоретически для того, чтобы разрешить неоднозначности в предложении, достаточно самого этого предложения. Но с функциональной структурой работать несравненно удобнее. Дело, в частности, в том, что ее достаточно для разрешения некоторых лексико-семантических неоднозначностей. Рассмотрим пример «Эти типы стали есть в цехе». В этом предложении немало неоднозначных слов:

- «Эти» может быть указательным местоимением-прилагательным в именительном или винительном падеже.
- «Типы» – неодушевленным существительным в одном из тех же двух падежей, либо одушевленным в именительным.
- «Стали» – существительным родительного падежа, либо глаголом прошедшего времени.
- «Есть» – глаголом «быть» в личной форме или инфинитивом,
- «В» имеет очень много значений.

Предложение допускает два понимания: в первом речь идет о стали, а во втором – о каких-то вкушающих типах. Можно показать, как в обоих случаях разрешаются вышеуказанные неоднозначности. Начнем с первой структуры:

```
есть(эти(стали(типы)))(в(цехе))
```

Применение «стали(типы)» может быть понято двояко. Либо глагол «стали» берет себе в подлежащие слово «типы», которое в таком случае может быть только в именительном падеже, либо же «сталь» в родительном падеже модифицирует существительное «типы», падеж которого остается при этом неопределенным, зато одушевленный вариант отметается. Функция «эти» к глаголам неприменима, посему становится понятным, что речь идет именно о стали, правда, падеж ее типов до сих пор не определен. Наконец, «есть» съедает то, что получилось, тем самым определяя и падежи слов «эти» и «типы» (именительный, конечно же). При этом «есть» разрешает собственную неоднозначность, ибо смысл «поглощать пищу» слабо применим к типам стали, хотя бы они и были прямым дополнением. «В» мгновенно из всех своих смыслов выбирает местный, исходя из падежа и значения слова «цехе». Глагол «быть» (у нас он в третьем лице) имеет право на существование сам по себе, но при добавлении ему еще одного аргумента с местным значением приобретает другой смысл, «быть в».

Вторая структура:

```
в(цехе)(стали(эти(типы)))(есть))
```

С конструкцией «стали(эти(типы))» все происходящее похоже на первый вариант. В результате возникает либо финитный вспомогательный глагол «стали» с актором, либо типы сплава «сталь» в не-

известном падеже. Но применение этой конструкции к слову «есть» ставит все на свои места, поскольку ни один падеж «типов стали» не способен применяться ни к какому глаголу. Остается только вариант, где «стали» представляет из себя вспомогательный глагол, а все остальное имеет именительный падеж. Вспомогательный глагол умеет применяться только к инфинитивам, поэтому бытийный вариант «есть» остается не у дел. И «в(цехе)», являющееся в этом варианте сирконстантом, победно завершает процесс, модифицируя получившуюся клаузу.

Мы видим, что оба варианта синтаксической неоднозначности получили в функциональной структуре разные записи, и обе эти записи благополучно разрешили все лексические неоднозначности самим своим содержанием и порядком применения функций. Если, к примеру, добавить к предложению еще и слово «вилкой», оно сможет примениться (как сирконстант) только ко второй альтернативе структуры, исключив при этом первую. В конечном счете критичным вопросом для выявления неоднозначностей в этом предложения является то, применяется ли «стали» к «есть» или наоборот.

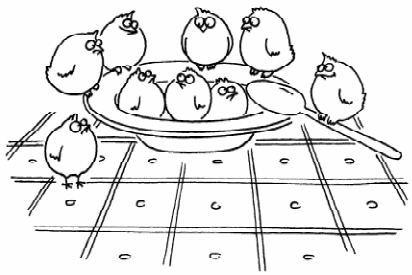
Следует заметить, что некоторые неоднозначности при функциональном подходе все равно будут иметь идентичные записи. «Цыплята готовы к обеду», допускающее двоякое понимание роли цыплят: то ли едоков, то ли блюда» – будет иметь структуру «готовы(цыплята)(к(обеду))», выражения с родительным падежом вроде «книга отца» – структуру «отца(книга)», из которой совершенно не вытекает, владеет ли отец книгой или просто написал ее. Тем не менее, это не мешает конкретным реализациям этих абстрактных функций получить из более широкого контекста информацию, по которой и сделать нужный вывод. Кстати, тот же самый контекст может позволить выбрать предпочтительную функциональную структуру, даже если парсер выдал несколько вариантов.

## ЗАКЛЮЧЕНИЕ

Итак, функциональное представление предложения достаточно хорошо подходит в качестве промежуточного представления при анализе текстов. Как видно из приведенных примеров, оно легко сводимо к другим популярным форматам представления синтаксической структуры текста. При этом оно достаточно абстрактно, чтобы его внутренняя структура не создавала искусственных препятствий этому сведению. Это представление номинально решает проблему анализа порядка слов в тексте, но при этом при его помощи можно, например, эффективно разрешать лексическую омонимию.

Из широко используемых структур синтаксического представления текста ближе всего к описываемой находится представление в виде синтаксических групп [7]. В обоих подходах слова взаимодействуют попарно, причем результат взаимодействия может быть не похож ни на один из компонентов. Этот результат затем участвует в других взаимодействиях. Основное различие заключается в перспективе. Синтаксические группы видятся как просто способ статической разметки текста, структура данных. В функциональном же представлении взаимодействие представляет собой активный процесс, алгоритм, исполнение некоторой функции, реализация которой может сильно разниться в зависимости от задачи.

На основе функционального представления работает синтаксический анализатор В. Тузова [5]. Его подход несколько отличается от представленного в этой статье. Во-первых, композиция функций-слов



*«Цыплята готовы к обеду»*

является не окончательным результатом работы, а лишь промежуточной ступенью к построению другой, уже семантической структуры. Как следствие, эта композиция не видна явно в выводе программы. Во-вторых, и актанты, и сирконстанты у него являются аргументами функции, соответствующей главному слову. В результате нередки глаголы с 6 и более аргументами, некоторые из которых (сирконстанты, конечно) появляются и в других местах, играя там ту же роль. Почти все существительные имеют аргументом другое существительное в родительном падеже. С моей точки зрения, замена таких аргументов на функции от главного слова сильно сэкономит объем словаря и усилия по его поддержке и дополнению.

Анализатор, выдающий в качестве результата функциональную структуру, имеет меньший размер и сосредоточен на решении ровно одной задачи – задачи связывания слов, обеспечивая большую модулярность и переносимость лингвопроцессора. Программы же, имеющие дело с семантикой, в свою очередь, могут сосредоточиться именно на ней, не вникая в тонкости синтаксического анализа, если их входными данными будет не текст, а его функциональная структура.

Благодарю Илью Посова, Дану Корнишову и Игоря Кураленка за ценные замечания. Илье Посову также отдельно крайне признателен за подготовку иллюстраций к статье.

**Литература**

1. АОТ // *http://www.aot.ru* .
2. ЭТАП-3 // *http://proling.iitp.ru/etap/index.html* .
3. COMPERE. Mahesh, K. A theory of interaction and independence in sentence understanding. Technical Report GIT-CC-93/34. Georgia Institute of Technology, 1993.
4. *Winograd, Terry.* Language As A Cognitive Process, Volume 1, Syntax Addison-Wesley, 1982.
5. *Тузов В.А.* Компьютерная семантика русского языка. СПб.: Из-во СПбГУ, 2004.
6. *Тестелец Я.Г.* Введение в общий синтаксис. М.: РГГУ, 2001.
7. *Гладкий А.В.* Синтаксические структуры естественного языка в автоматизированных системах общения. М.: Наука, 1985.

**Abstract**

Typically, an NLP tool which has to work with syntactic structures should contain a parser. An alternative to developing one's own parser is to use existing parsers that are likely to be not so well suited for the tool's particular goals. A better solution would be to have one parser to produce some output that would be able to fit as many domain problems as possible. Such output format is presented, and its advantages are discussed, including backward compatibility to other widely used structures and lexical ambiguity resolution.

.



***Громов Петр Александрович,***
***аспирант кафедры системного программирования математико-механического факультета СПбГУ,***
***gromopetr@gmail.com***