Научная визуализация, 2018, том 10, номер 5, страницы 57 - 85, DOI: 10.26583/sv.10.5.05

Анализ изображений геометрических тел с дополнительным интерактивным обучением на естественном языке

Н.Г. Волченков

Национальный исследовательский ядерный университет «МИФИ», Москва, Россия
ORCID: 0000-0002-1474-6853, NGVolchenkov@mephi.ru

Аннотация

Прототип системы анализа изображений геометрических тел, использующий интерфейс логического и визуального программирования, был разработан автором для проведения экспериментов с интеллектуальным роботом, который снабжён видеокамерой. Предполагается, что робот предназначен для планирования своих действий – захвату и переносу тел. Класс рассмотренных автором тел – это раскрашенные в локальные цвета многогранники. Недостатком полученных в предыдущей публикации автора результатов является отсутствие анализа взаимного расположения выявленного множества многогранников. Очевидны трудности, которые могут возникнуть при попытке автоматического (без участия человека) выявления взаимного расположения тел. Задача облегчается путём включения в систему блока интерактивного обучения робота человеком. Этим человеком является оператор – человек, который может сформулировать своё описание конкретного изображения на ограниченном естественном языке. Форма этого описания – так называемые поверхностные структуры естественно-языковых фраз. В данной статье автор представляет необходимую для указанной цели программу синтаксического анализа поверхностных структур. Эта программа реализована на Прологе – языке логического программирования. Для наглядного представления результатов работы блока обучения, реализованного на Прологе, автор, как и в предыдущей своей публикации, предлагает интерфейс визуального программирования (язык Visual Basic) и логического программирования (язык Пролог). В статье представлен пример конкретного изображения, на котором выявлены 5 тел разного цвета. Этот пример позволил продемонстрировать наиболее типичные случаи взаимного расположения тел, их описание на предлагаемом автором языке поверхностных структур и синтаксический анализ фраз этого языка. Представлен также важный побочный эффект синтаксического анализа – построение глубинных структур ограниченного естественного языка. Эти структуры представлены в виде структур языка Пролог – на разработанном автором языке глубинных структур. Это представление в дальнейшем может быть использовано роботом непосредственно для планирования своих действий.

Ключевые слова: *логическое программирование* (ЛП); язык *Пролог*; *визуальное программирование*; язык *Visual Basic*; *база данных* (БД) Пролога; *ограниченный естественный язык* (ОЕЯ); *поверхностная структура* фразы на ОЕЯ; *обучение* как внесение новых знаний в БД Пролога; *синтаксический анализ* ОЕЯ; язык *глубинных структур* (ЯГС); механизм *definite clause grammar* (DCG) в Прологе.

Введение

Содержание данной статьи – это развитие темы, представленной предыдущей публикацией автора в настоящем журнале – статьей «Использование интерфейса логического и визуального программирования для анализа изображений геометрических тел» [1]. В указанной публикации обосновывалась актуальность «интеллектуальной» задачи анализа полученных с видеокамеры растровых изображений предметов,

находящихся в поле зрения промышленного робота-манипулятора, целью которого являются: выбор нужного предмета; оценка его размера; его расположения относительно других предметов и планирование своих действий по захвату и переносу этого предмета.

В указанной публикации, в частности, обосновывалась возможность использования совместного применения *логического программирования* (язык Пролог) и *визуального программирования* (язык Visual Basic) для анализа совокупности изображений геометрических тел с плоскими гранями.

Были описаны важные этапы анализа, начиная с этапа преобразования тонового (растрового) изображения в векторное изображение (рисунок 1), и заканчивая этапом логического анализа векторного изображения, в результате которого на изображении выявляется точное число тел с рядом характеристик (рисунок 2).

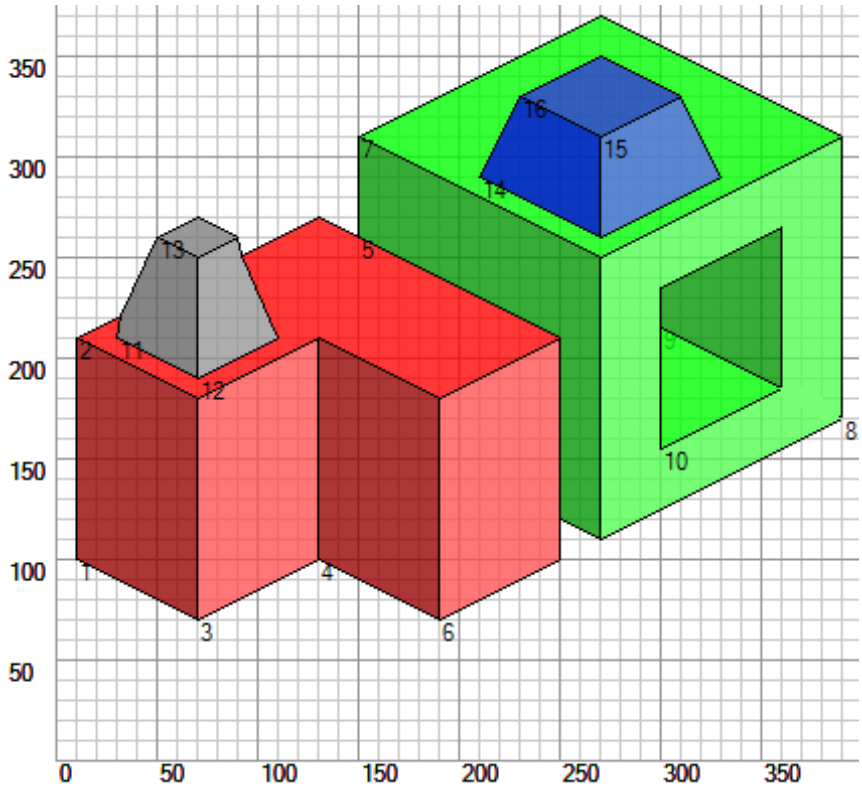

Рисунок 1. Пример «картинки», полученной в ходе преобразования растрового изображения в векторное. Показаны номера всех выявленных граней.

```
Result:
Тело 01:(126,155):054:c(200-213-032-032):red
Тело 02:(296,227):086:c(200-016-213-016):green
Тело 03:(069,239):007:c(200-125-125-125):grey
Тело 04:(270,309):011:c(185-037-037-255):blue
```

Рисунок 2. Результат анализа изображения, представленного на рисунке 1.

Характеристики, полученные в результате анализа изображений тел указанного класса, следующие:

1. число выявленных тел («телами» в данном случае условно называются совокупности изображений многоугольников, соответствующих граням многогранников);
2. координаты условных «центров» выявленных тел (координаты центров прямоугольников, в каждый из которых вписаны изображения всех граней одного многогранника);
3. условные «размеры» выявленных тел в единицах, пропорциональных площадям указанных в предыдущем пункте (2) прямоугольников;
4. «объективное» значение цвета каждого тела как значение функции **Argb**, аргументы которой вычислены как средние значения этой функции для всех граней этого тела, а также «субъективное» значение лингвистической переменной **Цвет** (**red**, **green** и т.д.).

В публикации [1] был представлен сформулированный автором «логико-визуальный» подход к решению задачи анализа изображений геометрических тел с плоскими гранями. Главной составляющей такого подхода является синтаксический (структурный) метод на базе логического программирования [2, 3, 4], которое является частным случаем более широкой парадигмы декларативного программирования [5], принципиально отличной от традиционной парадигмы императивного (операторного) программирования. Рассмотренная в упомянутой публикации задача анализа изображений геометрических тел может служить достаточно ярким примером задач указанного класса. Поэтому, полученные автором результаты по созданию демонстрационного прототипа «интеллектуальной» составляющей системы анализа изображений геометрических тел могут служить хорошей основой для усвоения навыков практического применения декларативного подхода к программированию многих задач искусственного интеллекта [6, 7].

Как было отмечено в публикации [1], опыт использования языков функционального и логического программирования для решения задач символьной обработки привел к пониманию необходимости визуализации результатов решения многих практических задач этого класса. Без визуализации интерпретация этих результатов бывает крайне затруднительной. Для решения проблемы визуализации автором был предложен интерфейс двух языков программирования – языка логического программирования (**Пролог**) и визуального императивного языка программирования (**Visual Basic**), точнее, двух систем программирования для указанных языков [8, 9]. Язык Пролог используется для решения основной, «интеллектуальной» части задачи. Язык Visual Basic используется для проведения вычислений, дополняющих логический вывод, а также для формирования наглядного представления результатов этого вывода.

Разумеется, большая доля сведений, которые необходимы «интеллектуальному роботу» для решения задач планирования своих действий, не содержатся в тех результатах, пример которых показан на рисунке 2. Их «видит» человек, но они не «видны» роботу, так как он не обладает метазнаниями, которыми обладает человек. Например, знаниями о взаимном расположении тел, которые могут быть очень важными при принятии решений о захвате нужного предмета (при необходимости его переноса на новое место). Предмет может быть «завален» другими предметами, которые необходимо убрать, чтобы добраться до нужного предмета. И т.д. Например, для захвата и переноса зелёного «ящика» на рисунке 1, необходимо освободить его от покоящегося на нём предмета – пирамиды синего цвета. При этом, робот «знает» лишь то, что пирамид на рисунке две; что одна из них серая, размер её меньше, чем размер другой, синей пирамиды, и что она помещена на самый левый предмет. Разумеется, для решения этой простой для человека задачи роботу надо, как ребёнку, кое-что объяс-

нить – в частности, научить его элементарным приёмам дедукции. В частности, делать такие, например, выводы: «из того, что обе пирамиды разного цвета; серая пирамида находится на красном объекте; зелёный ящик служит опорой для большей пирамиды, следует, что для освобождения зелёного ящика следует убрать с него синюю пирамиду».

В настоящей статье описана попытка дополнения системы анализа изображений указанного класса, представленного в предыдущей статье, компонентом обучения. Обучение предполагается в режиме общения с человеком – «учителем», который на ограниченном естественном языке интерпретирует полученные на предыдущем этапе результаты.

Например, относительно изображения, показанного на рисунке 1, «учитель» может на ограниченном естественном языке (ОЕЯ) ввести в базу данных логической части программы анализа изображений (на языке логического программирования Пролог) следующие утверждения:

1. Big blue pyramid is on the green box. (Большая синяя пирамида находится на зелёной коробке.)
2. Little grey pyramid is on the red object. (Маленькая серая пирамида стоит на красном предмете.)
3. Large object is to the right and behind the small object. (Большой объект расположен справа и позади маленького объекта.)
4. Big green object has the rectangular notch. (Большой зелёный объект имеет выемку.)
5. A red object does not have specificity. (Маленький красный объект не имеет особенностей.)
6. All four bodies highlighted on the right. (Все четыре тела освещены справа.) And the like. (И тому подобное.)

После ввода указанных фраз на ОЕЯ в виде списков Пролога, представляющих анализируемые цепочки языка, происходит их синтаксический анализа в соответствии с некоторой порождающей грамматикой. После чего, в БД (базе данных Пролога) должны появиться новые факты. Эти факты будут представлены в форме так называемых *глубинных* (канонических) *структур* – некоторых *утверждений* (*statements*), например, для приведённых выше фраз – это следующие факты Пролога:

1. statement(location(is(on), object(piramid1, char(size(big), color(blue), _), object(box1, char(size(_), color(green), _))).
2. statement(location(is(on), object(piramid2, char(size(little), color(grey), _), object(box2, char(size(_), color(blue), _))).
3. statement(location(is(right), object(box1, char(size(big), color(_), _), object(box2, char(size(little), color(_), _))).
4. statement(location(is(behind), object(box1, char(size(big),color(_), _), object(box2, char(size(little), color(_), _))).
5. statement(object(box1, char(size(big), color(green), spec(notch)))).
6. statement(object(box2, char(size(little), color(red), spec(no)))).
7. statement(object(all, char(number(4), lighting(right)))).

Здесь «_» – анонимная неозначенная переменная Пролога, char – сокращение от слова characteristics (характеристики), в данном случае, это характеристики объекта. Характеристик несколько: size(размер), color(цвет), spec – сокращение от слова specific (особенность).

Отметим, что число структур (здесь их 7) может не совпадать с числом исходных фраз на ОЕЯ (здесь их 6).

Отметим также, что одинаковые *глубинные структуры* могут, в силу разнообразия способов выражения одного и того же смысла и в силу замены многих понятий (например, **предмет**, **ящик**, **коробка**, **пирамида** и т.д.) единым *каноническим термином* (например, **object**), могут стать одинаковыми для различных исходных так называемых *поверхностных структур* – предложений

на ОЕЯ. Это приводит к тому, что число глубинных структур для данной, весьма специфической предметной области, существенно, на порядки, меньше числа поверхностных структур. Это даёт возможность существенно упростить процесс «понимания» *обучаемым* (*объектом обучения* – программой на Прологе) фраз на ОЕЯ, предоставляемых базе данных Пролога *обучающим* (*субъектом обучения* – человеком-оператором). Практически, это выливается в то, что формальные грамматики, порождающие фразы на ОЕЯ, могут быть компактными и лаконичными.

«Учитель» может «попросить» программу ответить на вопросы, сформулированные также на ОЕЯ:

1. How many objects in the image? (Сколько предметов на изображении?)
2. What color is the big object? (Какого цвета большая коробка)
3. What is left of the green object? (Что находится слева от зелёного объекта?)
4. What color is the body near a small body? (Какого цвета тело, находящееся рядом с маленьким телом?)
5. Less if red body green body? (Меньше ли красное тело зелёного тела?)
   And the like. (И тому подобное.)

После введения указанных фраз (вопросов) на ОЕЯ и их синтаксического анализа в базе данных Пролога должны появиться новые факты в форме так называемых «глубинных структур» *вопросов* (*questions*), например, для приведённых выше:

1. question (object(all, char(number(X), _))).
2. question (object(box1, char(size(big), color(X), _))).
3. question (location(is(left), object(Name1, Char1), object(Name2, char(_, color(green), _)))).
4. question (location(is(on), object(Name1, char(_, color(Color1), _)), object(Name2, char(size(little), _, _)))).
5. question (compar(less(Size1, Size2), object(Name1, char(size(Size1)_, color(red), _), object(Name2, char(size(Size2, color(green), _)))).

Главным «побочным эффектом» действий «учителя» (вводом утверждений и вопросов на ОЕЯ) должны быть новые знания, которые должны сохраниться в базе данных Пролога в виде фактов-утверждений и фактов-вопросов. В дальнейшем, факты-вопросы могут *интерпретироваться* с помощью запросов пользователя к базе данных Пролога. Результат этой интерпретации может оказаться неожиданным: содержащиеся в указанных структурах переменные могут «превратиться» из неозначенных в означенные! Такой эффект свидетельствует об элементах «интеллектуальности» формального логического (дедуктивного) вывода, который «самостоятельно» проявляет программа на Прологе в процессе её работы.

Например, на 3-й вопрос из представленного списка: question(location(is(left), object(Name1, Char1), object(Name2, char(_, color(green), _)))) («Что находится слева от зелёного объекта?») неожиданно может последовать «расширенный» ответ Пролога: statement(location(is(left), object(box2, char(size(little), color(red), spec(no))), object(box1, char(size(big), color(green), spec(notch)))). («Слева от большого зелёного объекта с именем box1, имеющего особенность (выемку), находится маленький красный объект с именем box2, не имеющий особенностей»).

Отметим, что, «между прочим», Пролог «самостоятельно уточнил»: исходный зелёный объект (1) имеет большой размер, (2) имеет имя box1 и (3) имеет особенность (выемку). Хорошо это, или плохо – получать избыточную информацию – судить пользователю.

1. Теоретические основы использования логического программирования для обработки текстов на ОЕЯ, описывающих изображения совокупности тел рассматриваемого класса

1.1. Использование разностных списков для повышения эффективности нисходящего синтаксического анализа на Прологе

Синтаксический анализ является важным приложением логического программирования (в целом) и языка Пролог (в частности).

Напомним представленные в разд. 1.1 статьи [1] определения: *порождающей грамматики*; формального *языка*, порождаемого такой грамматикой, а также мест, которые занимают контекстно-свободные и контекстно-зависимые грамматики в классификации формальных грамматик. (Классическую формализацию порождающих грамматик впервые предложил Ноам Хомский в 1950-х годах. [*Википедия*: https://en.wikipedia.org/wiki/Formal_grammar]).

Формальная грамматика – это следующая «четвёрка»:

$$\mathbf{G = <VT, VN, S, P>},$$

где **VT** и **VN** – терминальный и нетерминальный словари, $\mathbf{P} = \{\alpha_i \rightarrow \beta_i\}$ – множество правил вывода, α_i – цепочка, содержащая нетерминальный символ, β_i – произвольная цепочка из терминальных и нетерминальных символов, **S** – начальный символ.

Непосредственным порождением называется отношение

$\lambda \Rightarrow \mu$,

где $\lambda = \delta_1\ \alpha_i\ \delta_2$, $\mu = \delta_1\ \beta_i\ \delta_2$ и существует правило: $\alpha_i \rightarrow \beta_i$.

Порождением называется отношение

$\gamma \Rightarrow^* \gamma_n$, $n = 1, 2, \ldots$

если существует последовательность отношений

$\gamma \Rightarrow \gamma_1 \Rightarrow \ldots \Rightarrow \gamma_n$.

Языком, порождаемым грамматикой **G**, называется следующее множество:

$\mathbf{L(G)} = \{\gamma \mid S \Rightarrow^* \gamma\}$.

(множество терминальных цепочек γ – цепочек, состоящих только из символов терминального словаря **VT**).

Грамматики и языки подразделяют на типы. В практических задачах наиболее часто используются грамматики и языки следующих трех типов.

Типы формальных грамматик:

- Если нетерминальный символ в левой части какого-либо правила грамматики находится в окружении других символов (терминальных и/или нетерминальных), то такая грамматика называется *контекстно-зависимой*.
- Грамматика называется *контекстно-свободной*, если левая часть каждого правила вывода этой грамматики – это цепочка, состоящая из *единственного* нетерминального символа.
- Контекстно-свободная грамматика называется *автоматной*, если каждое правило вывода этой грамматики имеет следующий вид:

$$A \rightarrow aB \text{ или } A \rightarrow a \text{ или } A \rightarrow \lambda,$$

где A, B – нетерминальные символы, a – терминальный символ,

λ – пустая цепочка.

Типы формальных языков:

- Язык называется *автоматным*, если для его порождения можно построить автоматную грамматику. (Разумеется, для порождения этого языка можно построить грамматики и других типов.)
- Язык называется *контекстно-свободным*, если для его порождения можно построить контекстно-свободную грамматику, но нельзя построить автоматную грамматику.
- порождения его описание Язык называется *контекстно-зависимым*, если для его порождения нельзя построить контекстно-свободной грамматики.

Как было сказано выше, в язык Пролог встроен специальный механизм

DCG, позволяющий строить эффективные нисходящие синтаксические анализаторы для языков, определяемых формальными грамматиками различных типов. Его наличие позволяет создавать эффективно работающие нисходящие грамматические разборщики на Прологе.

Механизм DCG использует понятие *разностного списка*, с помощью которого можно избежать «комбинаторного взрыва» при недетерминированном разбиении исходной анализируемой цепочки, если язык достаточно сложен. Именно такое разбиение приходится производить в нисходящих синтаксических анализаторах.

Как выяснилось, можно обойтись без неэффективного предиката **append/3**, использование которого «напрашивается» для недетерминированного разбиения цепочки на подцепочки. Для этого следует воспользоваться понятием *разностного списка* [3, 4] или (что эквивалентно) ввести дополнительный аргумент в предикаты анализатора.

Разностный список – это структура инфиксного типа с именем \ (косая черта с наклоном влево) и двумя компонентами: списком **[A1, …, An | T]** и списком **T**:

[A1, …, An | T] \ T.

Эта структура эквивалентна «обычному» списку, содержащему **n** элементов: **[A1, …, An]**. Но в записи разностного списка есть переменная **T**, значением которой может быть любой список. Можно привести аналогию: любое число, например **5**, равно значению выражения **(5 + X) – X**, содержащего переменную **X** с любым значением.

Использование понятия «разностный список» позволяет существенно повысить эффективность работы многих предикатов обработки списков – в нашем случае, предиката **append/3** (конкатенации или «склеивания»). Вместо определения

append([], L, L).

append([H|T], L, [H|R]) :- append(T, L, R).

можно воспользоваться гораздо более эффективным определением:

append_dl(X\Y, Y\Z, X\Z).

Иллюстрация данного определения – на рисунке 3.

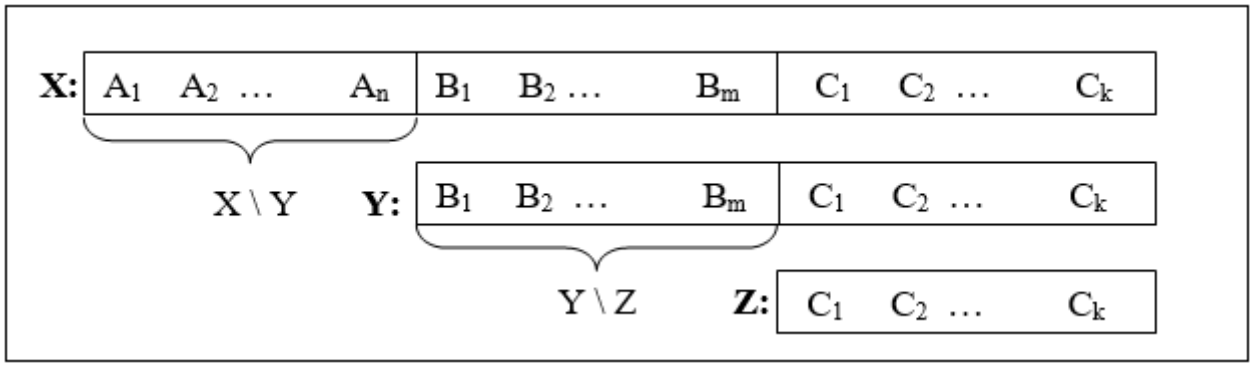

Рисунок 3. Иллюстрация конкатенации разностных списков:
append_dl(X\Y, Y\Z, X\Z).

Последнее определение позволяет «склеивать» списки (длина их может быть весьма большой) без многочисленных рекурсивных вызовов, – а только на уровне *сопоставления структур* (быстро работающего механизма *pattern matching* Пролога).

Пример 1.1.

Рассмотрим два списка: **L1 = [a, b, c]**; **L2 = [d, e, f]**. Необходимо найти конкатенацию этих списков.

Вызов **?- append([a, b, c], [d, e, f], Z).**

приводит к необходимости трёх рекурсивных вызовов, дающих искомый результат

Z = [a, b, c, d, e, f].

Вызов **?- append_dl([a, b, c|X] \ X, [d, e, f] \ [], Z).**

дает тот же результат **Z = [a, b, c, d, e, f] \ []** за единственный шаг логического вывода.

Эксперимент в системе **Win Prolog LPA** [8] демонстрирует следующий листинг как многострочный *комментарий* – фрагмент текста программы на Прологе, обрамлённый скобками /* и */.

Листинг 1

```
/*      Программа:
 :- op(200, xfx, '\').
 append_dl(X\Y, Y\Z, X\Z).
        Консоль:
 # 0.000 seconds to consult append_dl
[c:\prolog\lpa\]
 | ?- append_dl([a, b, c|X]\X, [d,e,f]\[],
Z).
 X = [d,e,f] ,
 Z = [a,b,c,d,e,f] \ []
*/
```

Пример 1.2.

Рассмотрим программу на Прологе, реализующую анализатор для фрагмента контекстно-свободной грамматики, порождающей фразы типа:

(1) Маленькая серая пирамида располагается на красном ящике.

(2) На красном ящике располагается маленькая серая пирамида.

с использованием разностных списков вместо предиката **append/3**.

Сначала, рассмотрим неэффективный вариант этого анализатора с использованием предиката **append/3.**

Перед написанием программы синтаксического анализа, работающего по принципу нисходящего разбора, рекомендуется построить "классическую" порождающую грамматику.

Желательно составить «минимальную» порождающую грамматику, которая обеспечивала бы генерацию фраз языка, не выходящих за рамки типов примера 1.2. Порождающая грамматика для анализа приведенных выше цепочек типов (1) и (2), может быть такой, которая представлена следующим листингом:

Листинг 2

```
/*
SLoc → NGr Verb Prep NGr
       для фраз типа (1);
SLoc → Prep NGr Verb NGr
       для фраз типа (2);
                нетерминал NGr
означает «группа существительного;
                нетерминал Verb
означает «глагол»;
                нетерминал Prep
означает «предлог»;
NGr  → Noun | Adj2 NGr
Verb  → находится | расположен | …
Prep  → на | под | левее | правее |
выше | ниже | …
Adj  → большой | маленький | зелё-
ный | красный | большого | малень-
кого | …
Noun → предмет | предмета | объ-
ект | объекта | …
                нетерминал
Noun означает «существительное»;
                нетерминал Adj
означает «прилагательное»
                (в именительном
или родительном падеже).
*/
```

Это контекстно-зависимая грамматика, так как члены предложения должны быть связаны по роду, числу и падежу. Контекстную зависимость можно реализовать с помощью переменных. Например так, как в двух изменённых строках листинга 2:

Листинг 3

```
SLoc → NGr(K1) Verb(K1) Adj1 NGr(K2)
       для фраз типа (1);
SLoc → Adj1 NGr(K1) Verb(K2) NGr(K2)
       для фраз типа (2);
```

Значения переменных **K1** и **K2** – это контекст, например, в рассмотренном примере:

K1 = k(падеж(именительный), род(мужской)) и **K2 =**

k(падеж(родительный), род(мужской)) – для фраз типа (1);

K1 = k(падеж(родительный), род(женский)) и **K2 = k(падеж(именительный), род(мужской))** – для фраз типа (2).

Связь по контексту **K1** реализуется между нетерминальными символами **NGr** и **Verb** в начале фразы (1), а по контексту **K2** – в конце фразы (2).

А теперь, главное: о замене неэффективно работающего предиката **append/3** разностными списками.

Листинг 4 представляет анализатор на Прологе с квази недетерминированным, медленно работающим предикатом **append/3** (пока, без привлечения разностных списков и механизма **DCG**).

Листинг 4

```
 an_SLoc( sloc( type(1), L1, X, Y, L2), Linput) :-
      append( L1, [X,Y|L2], Linput),
      an_NGr( K1, L1),                  % для фраз ОЕЯ 1-го типа
      an_Verb( K1, X),
      an_Adj1( Y),
      an_NGr( K2, L2).
 an_SLoc( sloc( type(2), X, L1, Y, L2), Linput) :-
      append( [X|L1], [Y|L2], Linput),
      an_Adj1( X),                      % для фраз ОЕЯ 2-го типа
      an_NGr( K1, L1),
      an_Verb( K2, Y),
      an_NGr( K2, L2).
 an_NGr( K, [X]) :- an_Noun( K, X).
 an_NGr( K, [X|L]) :- an_Adj2( K, X), an_NGr( K, L).
```

В этой программе на Прологе предполагается, что анализируемые цепочки ОЕЯ представляются в виде списков *лексем*. Здесь, это следующие списки:

[большой, зелёный, ящик, находится, правее, маленького, предмета];

[левее, зелёной, коробки, расположен, красный, объект].

Воспользуемся описанным выше определением разностного списка:

DList = [A1, …, An | T] \ T.

Этот список эквивалентен «обычному» списку **[A1, …, An]**.

Использование разностных списков вместо «обычных» списков позволяет существенно сократить время перебора вариантов недетерминированного разбиения цепочки, представленной в виде списка лексем, на подцепочки. Известно [2, 6, 7], что на таком разбиении основан алгоритм *нисходящего синтаксического разбора* (синтаксического анализа), реализованный в логическом программировании вообще, и на Прологе, в частности. Разностные списки позволяют избавиться от предиката **append/3** в правилах синтаксического анализатора следующим образом:

Пример 1.3. Пусть имеется грамматическое правило с тремя нетерминальными символами в правой части: **S → A B C**. При реализации нисходящего синтаксического анализатора на Прологе это правило требует разбиения исходной цепочки на 3 цепочки:

Листинг 5

```
      an_S( InputList) :- append( L1, L2, InputList),
                          append( L3, L4, L2),
                          an_A( L1), an_B( L3), an_C( L4).
```

Как было отмечено выше, использование предиката конкатенации двух списков (или разбиения одного списка на два) описывается без рекурсивных правил: **append_dl(X\Y, Y\Z, X\Z).** Это позволяет производить недетерминированное разбиение списков на подсписки без дорогостоящих рекурсивных вызовов – только на уровне отождествления структур данных.

Реально, представленное выше правило в анализаторе будет выглядеть так:

Листинг 6

```
      an_S( InputList\RestList) :-
                          an_A( InputList\L1), an_B( L1\L2), an_C( L2\RestList).
```

Указанное включение понятия разностного списка в реализованные на Прологе синтаксические анализаторы, позволил создателям многочисленных

современных версий этого языка включать в эти версии специально разработанный около 40 лет назад [2] механизм DCG, подробно описанный в следующем разделе 1.2.

1.1. DCG – встроенный в Пролог механизм синтаксического анализа, реализующий идею разностных списков

Реализацией разностных списков является встроенный в любую Пролог систему механизм DCG.

В классической монографии Стерлинга и Шапиро [7] отмечено (с. 203): «Происхождение Пролога связано с попыткой использовать логику для выражения грамматических правил и формализации процесса синтаксического разбора. Наиболее распространённым подходом к реализации синтаксического разбора средствами Пролога является использование *грамматик, задаваемых определительными предложениями* (definite clause grammar, DCG). Такие грамматики являются некоторым обобщением контекстно-свободных грамматик. Они представляют собой нотационный вариант определённого класса программ на Прологе и потому являются *исполняемыми*».

DCG как нотационный вариант программ класса «нисходящих синтаксических анализаторов» на Прологе, в которых «напрямую» записаны правила порождающих грамматик, декларируется следующими правилами нотации :

1. Связка Пролога :- («обратная импликация»), которая читается как слово «if» («если»), между левой и правой частями правила Пролога, заменяется связкой -->, соответствующей стрелке порождающей грамматики.
2. Предикаты как левой части правила (до связки -->), так и правой части правила (после связки -->) в нотации DCG не должны содержать в явном виде (в виде входных параметров) ни входной цепочки (списка лексем), ни остатка этой цепочки, оставшейся после её анализа в соответствии с данным правилом грамматики.
3. Аргументами предикатов как левой, так и правой части правила в нотации DCG могут быть только выходные параметры.
4. Все сопровождающие синтаксический анализ дополнительные действия (например, арифметические вычисления), производимые с аргументами предикатов, должны обрамляться фигурными скобками: { и }.
5. Терминальные символы в правых частях грамматических правил должны быть представлены списками. В частности, если «читается» один символ, например, лексема, это список из одного элемента.

Пример 1.4.

Рассмотрим листинг 6 – программу синтаксического анализа в соответствии с правилом грамматики **S → A B C.** В единственном правиле этой программы, как в левой, так и в правой его части у всех предикатов только один входной параметр, представленный разностным списком. Поэтому в нотации DCG данное правило будет иметь следующий вид:

Листинг 7

```
an_S --> an_A,  an_B,  an_C.
…
```

Допустим, данное правило используется для анализа языка $\{a^n b^m c^k\}$, $n>0$, $m>0$, $k>0$. Это контекстно-свободный язык, поэтому не требует использования переменных для реализации контекстной зависимости.

Для написания анализатора с выдачей значения трёх выходных параметров N, M и K добавим к листингу 7 дополнительные правила в нотации DCG:

Листинг 8

```
an_S( N, M, K) --> an_A( N), an_B( M), an_C( K).
an_A( 1) --> [a].
an_A( N) --> [a], an_A( N1), { N is N1 + 1 }.
an_B( 1) --> [b].
```

```
        an_B( M) --> [b], an_B( M1), { M is M1 + 1 }.
        an_C( 1) --> [c].
        an_C( K) --> [c], an_C( K1), { K is K1 + 1 }.
        /*  Пример вызова цели и получения результата:
        # 0.000 seconds to consult a^nb^mc^k.pl [d:\НИЯУ-МИФИ-2018\]
        | ?- an_S( N, M, K, [a,a,a,b,b,c,c,c,c], []).
        Yes,  N = 3 , M = 2 , K = 4  */
```

И, наконец, внесём ещё одно усложнение: равенство значений переменных N, M и K. Это ограничение превращает язык в контекстно-зависимый язык $\{a^n b^n c^n\}$, $n>0$.

Для реализации анализатора этого языка в нотации DCG достаточно изменить только первое правило:

Листинг 9

```
        an_S( N) --> an_A( N),  an_B( M), { M = N },  an_C( K), { K = N }.
        …
        /*  Пример вызова цели и получения результата:
        # 0.000 seconds to consult a^nb^nc^n.pl [d:\НИЯУ-МИФИ-2018\]
        | ?-an_S( N, [a,a,b,b,c,c], []).
        Yes,  N = 2
        | ?-an_S( N, [a,a,a,b,b,c,c,c,c], []).
        No  */
```

Анализатор на Прологе для рассмотренного в разделе 1.1 Примера 2 (Листинг 4) без предиката **append/3** и с привлечением механизма **DCG** представлен листингом 10:

Листинг 10

```
an_SLoc( sloc( type1)) -->          %
для фраз ОЕЯ 1-го типа
        an_NGr(K1), an_Verb( K1),
        an_Adj1, an_NGr( K2).
an_SLoc( sloc( type2)) -->          %
для фраз ОЕЯ 2-го типа
        an_Adj1, an_NGr( K1),
        an_Verb( K2), an_NGr( K2).
an_NGr( K) --> an_Noun( K).
an_NGr( K) --> an_Adj2( K),
an_NGr( K).
```

В заключение данного раздела рассмотрим, в какое правило в традиционной нотации Пролога автоматически преобразуется правило в нотации DCG. На этот вопрос отвечает запрос: **?-listing(<имя предиката>/<арность предиката>).** Этот запрос следует ввести после компиляции предиката, введённого в нотации DCG.

Например, вызовем указанный запрос после компиляции определения предиката **an_S/1**, введённого в нотации DCG (листинг11), и после проверки работы этого предиката (листинг 12):

Листинг 11

```
an_S( N) --> [a], an_S( N1), [b,c],
{N is N1 + 3}.
an_S( 3) --> [a,b,c].
```

Листинг 12

```
| ?- an_S( N,
[a,a,a,a,b,c,b,c,b,c,b,c], []).
N = 12
| ?- listing(an_S/3).
% an_S/3
an_S( A, B, C ) :-
        'C'( B, a, D ), an_S( E, D, F ),
(  'C'( F, b, G ), 'C'( G, c, C ) ), A is E + 3.
an_S( 3, A, B ) :- 'C'( A, a, C ), 'C'(
C, b, D ), 'C'( D, c, B ).
```

Отметим, что использование встроенного предиката 'C'/3, который применяется только для выделения «головы» (Head) и «хвоста» (Tail) списка (List), демонстрирует применение разностных списков для синтаксического анализа. Его определение: 'C'([Head|Tail], Head, Tail).

Переопределим имена переменных, сформированных системой (системные имена A, B, C, D, E, F, G), на более содержательные (мнемонические имена).

В первом правиле заменим: A на N; B на Input; C на Rest; D на L1; E на N1; F на L2; G на N3. Во втором правиле заменим: A на Input; B на Rest; C на L1; D на L2.

Заменим также встроенный предикат 'C'/3 в соответствии с его определением.

После чего получим:

Листинг 13

```
an_S( N, Input, Rest) :-
        Input = [a|L1], an_S( N1, L1,
L2),
        L2 = [b|L3], L3 = [c|Rest],
        N is N1 + 3.
an_S( 3, Input, Rest) :-
        Input = [a|L1], L1 = [b|L2],
L2 = [c|Rest].
```

При трансляции из нотации DCG в нотацию Пролога система автоматически проводит дополнительную оптимизацию этих правил:

Листинг 14

```
an_S( N, [a|L1], Rest) :-
        an_S( N1, L1, [b,c|Rest]), N
is N1 + 3.
an_S( 3, [a,b,c|Rest], Rest).
```

Очевидно, что представление этих двух правил в нотации DCG (листинг 11) выглядит значительно лаконичней и выразительней, так как непосредственно отражает их связь с порождающей грамматикой.

2. Технология практической реализации и полученные результаты

Технологию практической реализации включения блока обучения на ОЕЯ в систему анализа рассматриваемого класса изображений удобней и наглядней продемонстрировать на конкретных примерах. При практической реализации рассмотренного в данной статье подхода был проведён эксперимент на следующем примере.

Пример 2.1.

Пусть анализируемое тоновое изображение после его преобразования в векторную форму (процесс преобразования описан в статье автора [1]) демонстрирует пять тел разного цвета с плоскими гранями (рисунок 4):

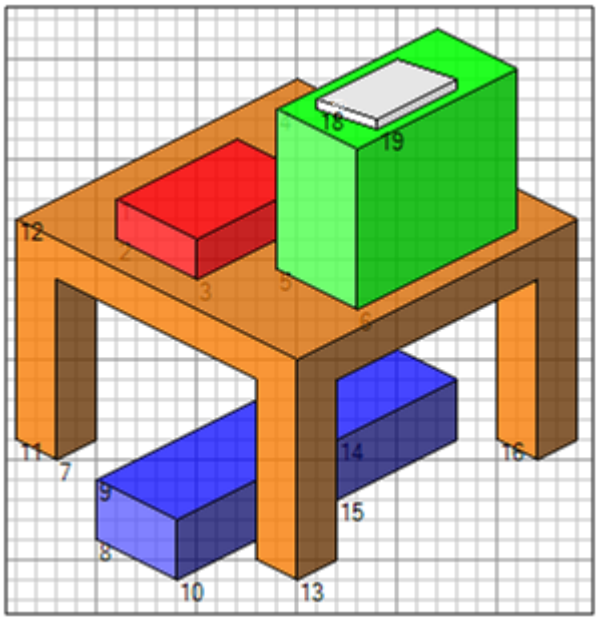

Рисунок 4. Пример векторного изображения пяти многогранников, полученного путём обработки исходного тонового изображения. Числа на рисунке – номера граней (полигонов одинакового цвета)

Система анализа изображения, описанная в упомянутой выше статье [1], не только преобразует тоновое изображение в векторное, выделяет вершины и рёбра представленных на изображении многогранников с помощью визуального языка Visual Basic, но и вычисляет некий результат с помощью программы на языке Пролог. Исходные данные и результат визуализируются на экранной форме. На рисунке 5 демонстрируется вид этой формы для данного примера.

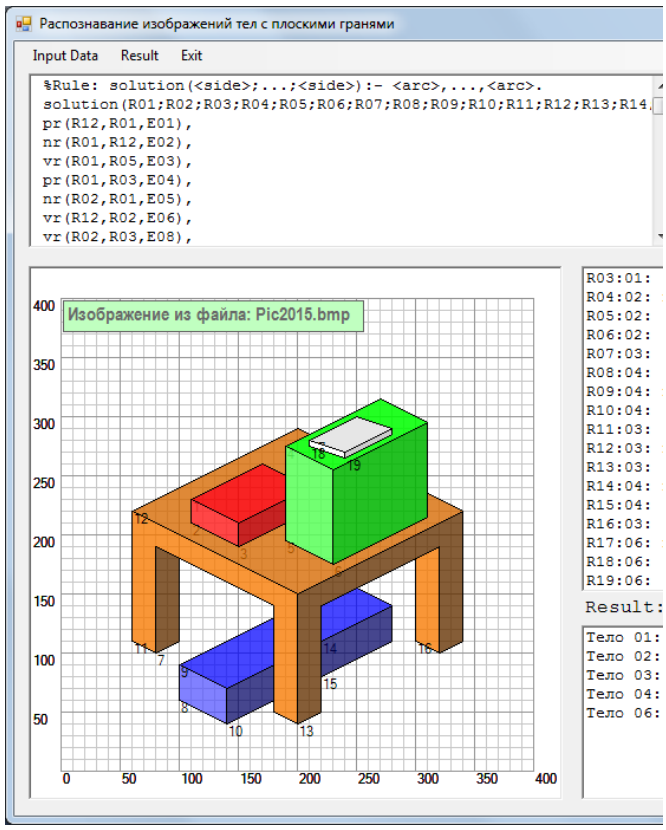

Рисунок 5. Пример экранной формы приложения, в котором реализован интерфейс программ на языках Visual Basic и Prolog. На форме видны как исходные данные, так и результаты анализа изображения: выявленный список тел (многогранников) с их характеристиками

Тела на рисунке 4 находятся в очевидных для неискушенного зрителя пространственных отношениях. Эти отношения человек (наблюдатель) в соответствии со своей традиционной моделью мира может легко интерпретировать, например, следующим образом:

- на рисунке зафиксирован предмет, который можно назвать «столом» коричневого цвета»;
- на «столе» расположены две «коробки» красного и зелёного цвета;
- под «столом» находится «ящик» синего цвета»;
- на зелёной «коробке» лежит белая «книга».

Как было сказано в предыдущей статье автора на рассматриваемую тему [1], конечной целью системы распознавания изображений данного класса может быть планирование действий интеллектуального робота по захвату и переносу предметов, зафиксированных видеокамерой робота. Программа планирования, разумеется, должна работать вместе с программой анализа изображений.

Очевидно, что для конкретизации плана указанных выше действий робота не хватает информации о взаимном расположении предметов выявленных на изображении. Представляется целесообразным включение в систему планирования действий робота дополнительного блока «обучения» тому, каково взаимное расположение объектов на изображении с тем, чтобы робот мог, например, освободить предмет, предназначенный для захвата и переноса на новое место, от мешающих этому других предметов. Очевидно также, что разумным было бы использовать опыт человека, который без труда мог бы оценить ситуацию и предложить необходимую для транспортировки работу (что-то убрать, что-то переставить и т.д.). Чтобы не требовать от человека знания формы представления робота о той сцены, которую фиксирует программа, разумно предложить человеку очень простой ограниченный естественный язык, о котором уже шла речь в настоящей статье.

Для создания указанного блока обучения, в первую очередь, следует разработать грамматику, порождающую фразы ОЕЯ – ограниченного естественного языка описания утверждений и вопросов относительно предметов, присутствующих на изображении. Продемонстрируем простой пример такой грамматики и программу на языке Пролог, которая реализует синтаксический анализ фраз, порождаемых этой грамматикой.

Разумно ограничиться представлением листинга разработанной автором программы синтаксического анализа ОЕЯ, в которую автор включил необходимые комментарии, существенно облегчающие понимание её работы даже для тех читателей настоящей статьи, ко-

торые не имеют значительного опыта логического программирования, в частности, на языке Пролог.

Назначение данной программы – преобразование выражения поверхностных структур *ограниченного естественного языка* (ОЕЯ) в выражения на *языке глубинных структур* (ЯГС), которые, в силу своей жёсткой и однозначно интерпретируемой формальной основы, могут быть легко использованы интеллектуальным роботом при его планировании своих действий с выявленными на изображении телами.

Пример: преобразование поверхностной структуры (списка лексем ОЕЯ):

[четвёртое, тело, находится, под, третьим, телом] в структуру:
st(6, location(is(under), object(box3, char(size(middle), color(blue))), object(table, char(size(big), color(brown))))).

Начало программы на Прологе (точнее, кода в смешанной нотации «чистого» Пролога встроенного в Пролог языка DCG) представлено в следующем листинге:

Листинг 15

```
/* - Левая скобка многострочного комментария.
Перевод утверждений и вопросов с ограниченного естественного языка (ОЕЯ)
на  язык глубинных структур (ЯГС).
Translation of Statements and Questions from Limited Natural Language (LNL) to Deep Structures Language (DSL)
В силу ориентированности нотации DCG на порождающие грамматики, грамматика, генерирующая поверхностные структуры (списки лексем на ОЕЯ)  здесь не описывается отдельно, а представлена в самой программе в виде правил в нотации DCG.
Запуск программы осуществляется вводом следующей цели Пролога:
        | ?- from_LNL_to_DSL.
Имя целевого предиката – это мнемоническое имя: «перевод с ограниченного естественного языка на язык глубинных структур».
Правая скобка многострочного комментария: */
```

После комментария – интерпретируемая часть кода:

Листинг 16

```
% Начало программы
:-dynamic(st/2).   :-dynamic(qu/2).
                % Декларации, снимающие защиту
                % с предикатов % st/2 и qu/2, что позволяет
                % записывать в базу данных Пролога новые утверждения и вопросы.
```

В качестве тестовой базы данных Пролога в данной программе представлены факты, соответствующие изображению на рисунке 4:

Листинг 17

```
% Утверждения
% представлены в виде трёх фактов, единственным аргументом каждого из которых
% является список из произвольного числа элементов – списков лексем,
% каждая из которых – это слово русского или английского языка:
% (1) утверждения для команд переименования всех тел на изображении:
ren_sts([[присвоим, первому, телу, имя, ящик1],
                % «ren» от слова «rename», «sts» от слова «statements».
        [присвоим, второму, телу, имя, ящик2],
        [присвоим, третьему, телу, имя, стол],
        [присвоим, четвёртому, телу, имя, ящик3],
        [присвоим, пятому, телу, имя, книга]]).
% (2) утверждения для уточнения значений характеристик тел (размера, цвета)
%       в терминах значений лингвистических переменных:
char_sts([[первое, тело, имеет, маленький, размер, и, красный, цвет],
                % «char» от слова «characteristics».
         [второе, тело, имеет, большой, размер, и, зелёный, цвет],
         [третье, тело, имеет, огромный, размер, и, коричневый, цвет],
         [четвёртое, тело, имеет, средний, размер, и, синий, цвет],
         [пятое, тело, имеет, маленький, размер, и, белый, цвет]]).
% (3) утверждения для уточнения расположения тел относительно друг друга:
loc_sts( [[четвёртое, тело, находится, под, третьим, телом],
                % «loc» от слова «location».
        [пятое, тело, лежит, на, втором, теле],
        [первое, тело, лежит, на, третьем, теле],
        [второе, тело, стоит, на, третьем, теле],
        [второе, тело, стоит, правее, первого, тела],
        [первое, тело, расположено, позади, второго, тела]]).
 % Вопросы
% представлены в виде одного факта, единственным аргументом которого
% является список из произвольного числа элементов – списков лексем,
% каждая из которых – это слово русского или английского языка:
 all_qus( [[где, находится, пятое тело, «?»],
                % «qus » от слова «questions»
         [где, лежит, четвёртое, тело, «?»],
         [что, находится, на, третьем, теле, «?»],
         [что, лежит, под, третьим, телом, «?»],
         [под, чем, лежит, четвёртое, тело, «?»],
         [на, чём, лежит, пятое, тело, «?»],
         [левее, чего, находится, первое, тело, «?»],
         [перед, чем, расположено, второе, тело, «?»],
         [каков, размер, пятого, тела, «?»],
         [какого, цвета, пятое, тело, «?»]]).
```

Основная часть представляемой здесь программы размещена в разделе данной статьи

Приложение. Программа синтаксического анализа языка утверждений и вопросов для изображений рассматриваемого класса**.**

Далее – лишь небольшой фрагмент программы.

В частности, это демонстрация проверки работоспособности процедуры присвоения новых, более выразительных имён тел, выявленных на изображении, – в поверхностных структурах ОЕЯ – как в утверждениях, так и в вопросах. Например:

- новое_имя(ящик1) → старое_имя(первое, тело);
- новое_имя(ящик2) → старое_имя(второе, тело);
- новое_имя(стол) → старое_имя(третье, тело);
- новое_имя(ящик3) → старое_имя(четвёртое, тело);
- новое_имя(книга) → старое_имя(пятое, тело).

Процедура на следующем листинге:

Листинг 18

```
from_old_to_new_names :- ren_sts(L1), char_sts(L2), loc_sts(L3), all_qus(L4),
            L1 = [X1|T],
            an_Phrases1(K1, K2, Y1, X1, L1, []),  an_Phrases2(K1, K2, Y2, X2, [X2|_], []),
            an_Phrases3(K1, K2, Y3, X3, [X3|_], []),  an_Phrases4([X4|_], []).
an_Phrases1(K1, K2, Y,  X) --> [].
an_Phrases1(K11, K21, Y1,  X1) --> [X1], {X1=[A1,A2,A3,A4,A5], Y1=[A2,A3,A5],
                 K11=new(A5), K21=old(A2,A3),
                write(K11), write('->'), write(K21),nl},
                an_Phrases1(K12, K22, Y2, X2).
an_Phrases2(K1, K2, Y,  X) --> [].
an_Phrases2(K1, K2, Y1,  X1) --> [X1], {X1=[A1,A2|_], Y1=[A1,A2|_]},
                {write(K1), write('->'), write(K2), nl, write(X1), write('->'), write(Y1), nl},
                  an_Phrases2(K1, K2, Y2, X2).
an_Phrases3(K1, K2, Y,  X) --> [].
an_Phrases3(K1, K2, Y1,  X1) --> [X1], {X1=[A1,A2,A3,A4,A5,A6], Y1=[A1,A2,A5,A6]},
                {write(K1), write('->'), write(K2), nl, write(X1), write('->'), write(Y1), nl},
                  an_Phrases3(K1, K2, Y2, X2).
an_Phrases4(K1, K2, Y,  X) --> [].
an_Phrases4(K1, K2, Y1,  X1) --> [X1], {X1=[A1,A2,A3|_], Y1=[A1,A2,A3|_]},
               {write(X1), nl},  an_Phrases4(K1, K2, Y1, X1).
```

Результат проверки – на следующем листинге:

Листинг 19

```
LPA WIN-PROLOG 4.200 - S/N 0111615934 - 24 Oct 2001
Copyright (c) 2001 Logic Programming Associates Ltd
Licensed To: Nickolay Volchenkov
B=64 L=64 R=64 H=255 T=386 P=1163 S=63 I=64 O=64 Kb
-------------------------------------------------
| ?-
# 0.000 seconds to consult 2018_from_lnl_to_dsl_new.pl [d:\МИФИ - 2009-2017\]
| ?- from_old_to_new_names.
  new(ящик1)->old(первому,телу)
        [присвоим,первому,телу,имя,ящик1]
  new(ящик2)->old(второму,телу)
        [присвоим,второму,телу,имя,ящик2]
  new(стол)->old(третьему,телу)
        [присвоим,третьему,телу,имя,стол]
  new(ящик3)->old(четвёртому,телу)
        [присвоим,четвёртому,телу,имя,ящик3]
  new(книга)->old(пятому,телу)
        [присвоим,пятому,телу,имя,книга]
        …
Yes
```

После проведённого присвоения имён всем телам на изображении в базу данных Пролога заносятся следующие структуры – факты-утверждения и факты-вопросы с присвоенными им номерами:

Листинг 20

```
% Утверждения:
test_st( 1,   [присвоим, первому, телу, имя, ящик1]).
test_st( 2,   [присвоим, второму, телу, имя, ящик2]).
test_st( 3,   [присвоим, третьему, телу, имя, стол]).
test_st( 4,   [присвоим, четвёртому, телу, имя, ящик3]).
test_st( 5,   [присвоим, пятому, телу, имя, книга]).
test_st( 6,   [ящик1, имеет, маленький, размер, и, красный, цвет]).
test_st( 7,   [ящик2, имеет, большой, размер, и, зелёный, цвет]).
test_st( 8,   [ящик3, имеет, средний, размер, и, синий, цвет]).
test_st( 9,   [стол,  имеет, огромный, размер, и, коричневый, цвет]).
test_st(10,  [книга, имеет, маленький, размер, и, белый, цвет]).
test_st(11,  [ящик3, находится, под, столом]).
test_st(12,  [книга, лежит, на, ящике2]).
test_st(13,  [ящик1, лежит, на, столе]).
test_st(14,  [ящик2, стоит, на, столе]).
test_st(15,  [ящик2, стоит, правее, ящика1]).
test_st(16,  [ящик1, расположен, позади, ящика2]).
% Вопросы:
test_qu( 1,   [где, находится, книга]).          test_qu( 2,   [где, лежит, ящик3]).
test_qu( 3,   [что, находится, на, столе]).      test_qu( 4,   [что, лежит, под, столом]).
test_qu( 5,   [под, чем, лежит, ящик3]).         test_qu( 6,   [на, чём, лежит, книга]).
test_qu( 7, [левее, чего, находится, ящик1]).
test_qu( 8, [перед, чем, расположен, ящик2]).
test_qu( 9, [каков, размер, книги]).             test_qu(10,  [какого, цвета, книга]).
```

После вызова цели: | **?- setof(N, s(N), L).** были получены следующие результаты (на консоли интерпретатора Пролога фирмы LPA [8]) – утверждения в виде канонических глубинных структур:

Листинг 21

```
| ?- setof(N, s(N), L).
st(1,object(box1,char(size(small),color(red))))
st(2,object(box2,char(size(big),color(green))))
st(4,object(box3,char(size(middle),color(blue))))
st(3,object(table,char(size(big),color(brown))))
st(5,object(book,char(size(small),color(white))))
st(6,location(is(under),object(box3,char(size(middle),color(blue))),
        object(table,char(size(big),color(brown)))))
st(12,location(is(on),object(table,char(size(big),color(brown))),
        object(box3,char(size(middle),color(blue)))))
st(7,location(is(on),object(book,char(size(small),color(white))),
        object(box2,char(size(big),color(green)))))
st(13,location(is(under),object(box2,char(size(big),color(green))),
        object(book,char(size(small),color(white)))))
st(8,location(is(on),object(box1,char(size(small),color(red))),
        object(table,char(size(big),color(brown)))))
st(14,location(is(under),object(table,char(size(big),color(brown))),
        object(box1,char(size(small),color(red)))))
st(9,location(is(on),object(box2,char(size(big),color(green))),
        object(table,char(size(big),color(brown)))))
st(15,location(is(under),object(table,char(size(big),color(brown))),
        object(box2,char(size(big),color(green)))))
st(10,location(is(right),object(box2,char(size(big),color(green))),
        object(box1,char(size(small),color(red)))))
st(16,location(is(left),object(box1,char(size(small),color(red))),
        object(box2,char(size(big),color(green)))))
st(11,location(is(behind),object(box1,char(size(small),color(red))),
        object(box2,char(size(big),color(green)))))
st(17,location(is(front),object(box2,char(size(big),color(green))),
        object(box1,char(size(small),color(red)))))
```

В полученном списке следует отметить следующее:

- во всех утверждениях (а их не 16, а только 11, так как 5 утверждений-дубликатов автоматически удалены);
- в обработанных утверждениях автоматически означены все переменные.

После вызова цели | **?- setof(N, q(N), L).** во всех десяти вопросах переменные, так же, как в утверждениях, означены в результате работы программы. Вопросы приобрели следующий канонический вид (вид глубинных структур):

Листинг 22

```
| ?- setof(N, q(N), L).
qu(1,location(is(on),object(book,char(size(small),color(white))),
        object(box2,char(size(big),color(green)))))
qu(2,location(is(under),object(box3,char(size(middle),color(blue))),
        object(table,char(size(big),color(brown)))))
qu(3,location(is(on),object(box1,char(size(small),color(red))),
        object(table,char(size(big),color(brown)))))
qu(4,location(is(under),object(box3,char(size(middle),color(blue))),
        object(table,char(size(big),color(brown)))))
qu(5,location(is(under),object(box3,char(size(middle),color(blue))),
        object(table,char(size(big),color(brown)))))
qu(6,location(is(on),object(book,char(size(small),color(white))),
        object(box2,char(size(big),color(green)))))
qu(7,location(is(left),object(box1,char(size(small),color(red))),
        object(box2,char(size(big),color(green)))))
qu(8,location(is(front),object(box2,char(size(big),color(green))),
        object(box1,char(size(small),color(red)))))
qu(9,object(book,char(size(small),color(white))))
qu(10,object(book,char(size(small),color(white))))
```

Впечатляющим результатом проведённого эксперимента является то, что ни в оном из десяти рассмотренных вопросов не осталось ни одной неозначенной переменной, что свидетельствует о достаточной «интеллектуальности» (разумеется, в рассмотренных рамках) предлагаемой автором программы.

Программный комплекс, разработанный для проведения экспериментов с системой анализа изображений, дополненной рассмотренным в статье компонентом обучения, базируется на двух взаимодействующих программных *платформах*:

1. Microsoft .NET – версия 4 (в частности, версия 4.0, Visual Studio 2010 и поддерживаемый ей язык Visual Basic 2010);
2. Windows Prolog – версии интерпретаторов-компиляторов языка Пролог, разрабатываемые и поддерживаемые фирмой LPA [8].

Переносимость на другие платформы в настоящее время представляется неактуальной, так как рассмотренные выше платформы являются популярными и доступными. В частности, более поздние версии платформы Microsoft .NET являются совместимыми с версией 4.0, а фирма LPA в настоящее время продолжает совершенствование и коммерческое распространение ориентированных на операционную систему Windows интерпретаторов-компиляторов языка Пролог.

Функциональные возможности для пользователя демонстрирует следующее замечание.

Рассмотренный в данной статье блок обучения, реализованный на языке Пролог, можно легко включить в систему анализа изображений, описанную в предыдущей статье автора [1]. Для этого достаточно включить в меню этой системы команду, вызывающую появление новой экранной формы.

На этой форме, показанной на рисунке 6, в двух окнах демонстрируются исходные данные – тестовые утверждения и вопросы относительно характеристик и взаимного расположения геометрических тел на изображении. В третьем окне показаны полученные в результате обучения глубинные структуры ответов на все тестовые вопросы.

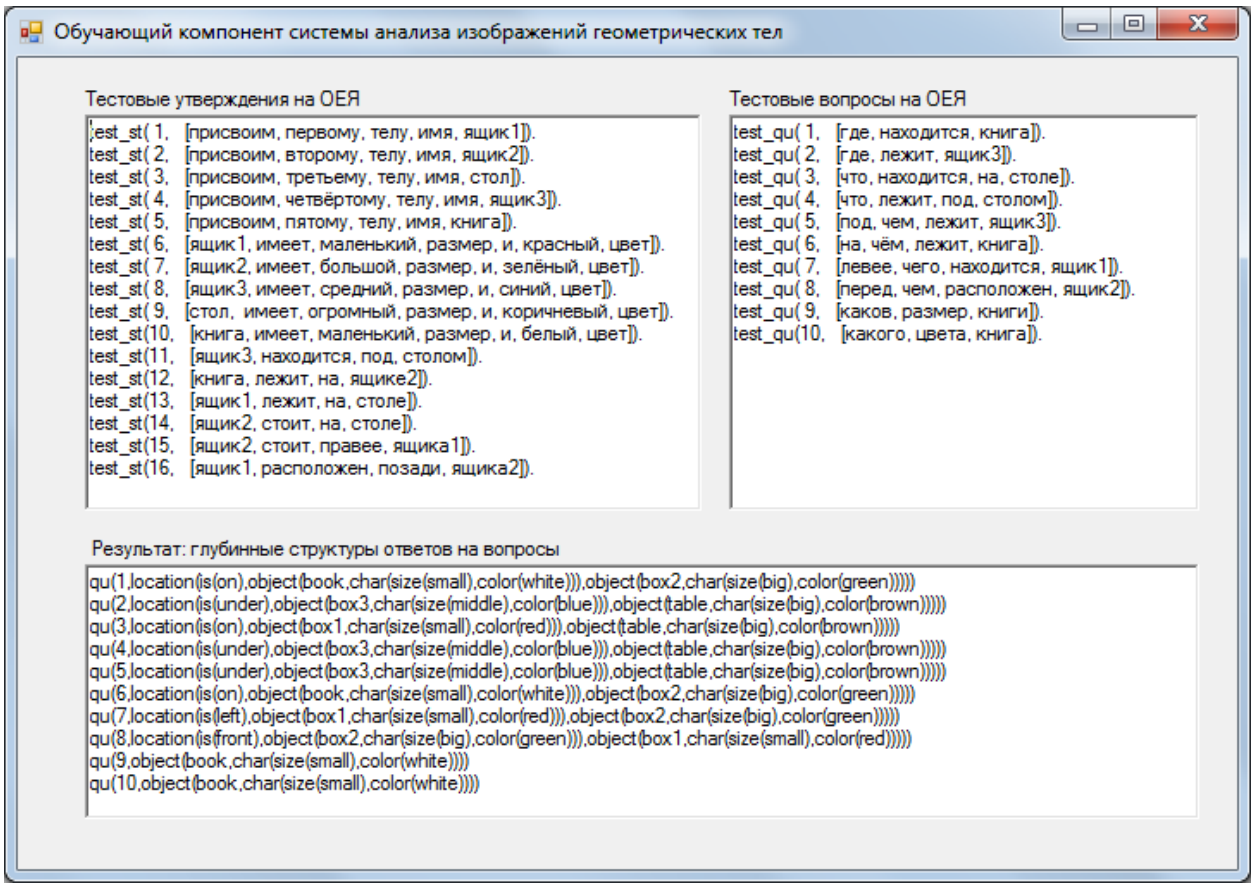

Рисунок 6. Экранная форма с результатами обработки обучающих тестов, вызываемая, в процессе работы системы анализа изображений

Государственная регистрация программы

Представленный в настоящей статье программный продукт послужил важным дополнением к разработанной автором программной системе анализа изображений многогранников и цилиндрических тел.

Эта система прошла государственную регистрацию в 2015 году, а автор получил свидетельство об указанной государственной регистрации соответствующей программы для ЭВМ «Демонстрационная программа по анализу изображений тел с плоскими гранями» № 2015611531 (дата регистрации 30 января 2015 г.) [10].

Заключение

Рассмотрена принципиальная возможность дополнения системы анализа изображений, в которой совмещены средства логического (структурного) подхода со средствами визуального программирования, описанной в публикации [1], блоком обучения. В этом блоке предлагается реализовать общение программы с человеком на ограниченном естественном языке с использованием реализованного автором на языке Пролог синтаксического анализатора. Цель указанного обучения – получение системой знаний о взаимном расположении тел. Эти знания должны помочь роботу, оснащенному системой анализа изображений, планировать свои действия, связанные с захватом и переносом тел, выявленных на изображении.

Экспериментальная проверка представленного в статье синтаксического анализатора продемонстрировала хорошие результаты: все рассмотренные утверждения и вопросы относительно зафиксированных на изображении нескольких тел, сформулированные на ограниченном естественном языке, были успешно проанализированы. Побочным результатом явились значения переменных в вопросах, преобразован-

ных в глубинные (канонические) структуры. Ответы на эти вопросы предоставляют важные знания роботу-манипулятору по планированию своих действий по захвату и переносу тел.

Предлагаемый в данной статье подход, по мнению автора, может быть полезным для обучения автономных роботов правильно ориентироваться при решении задачи захвата и транспортировки геометрических тел, выявленных при анализе их изображений.

Список литературы

1. Volchenkov N.G. The application logical and visual programming interface for image analysis of the geometric bodies. «Scientific Visualization». National Research Nuclear University MEPhI, 2015. Q3, V7, N3. Pages 84 – 97. [Электронный ресурс]. URL: http://sv-journal.org/2015-3/09.php?lang=eng (дата обращения 03.12.2017).
2. Warren D.H.D., Pereira L.M., Pereira F. PROLOG – the language and its implementation. Proceedings of the Symposium on Artificial Intelligence. SIGPLAN Notes, 12(8), 1977.
3. Волченков Н.Г. Логическое программирование. Язык Пролог: Тексты лекций. Изд. второе, испр. и доп. – М.: МИФИ, 2015. – 160 с.
4. Сергиевский Г.М., Волченков Н.Г. Функциональное и логическое программирование: Учеб. пособие для студ. высших учеб. заведений. – М.: Издательский центр «Академия», 2010. – 320 с.
5. Волченков Н.Г. Использование современных информационных технологий в обучении студентов по направлению «Информатика и вычислительная техника», различным парадигмам программирования: «Современные научные исследования и инновации». 2015. № 3 [Электронный ресурс]. URL: http://web.snauka.ru/issues/2015/03/47345 (дата обращения 28.03.2015).
6. Братко Иван. Алгоритмы искусственного интеллекта на языке PROLOG, 3-е издание.: Пер. с англ. – М.: Издательский дом «Вильямс», 2004. – 640 с
7. Стерлинг Л., Шапиро Э. Искусство программирования на языке Пролог. – М.: Мир, 1990. – 235 с.
8. Сайт компании Logic Programming Associates Ltd. [Электронный ресурс]. URL: http://www.lpa.co.uk/ (дата обращения 21.03.2015).
9. Зиборов В.В. Visual Basic 2010 на примерах. – СПб.: БХВ-Петербург, 2010. – 336 с.
10. Свидетельство о гос. регистрации программы для ЭВМ № 2015611531 «Демонстрационная программа по анализу изображений тел с плоскими гранями». Автор: Волченков Н.Г. (RU). [Электронный ресурс]. URL: http://www1.fips.ru/Archive/EVM/2015/2015.02.20/DOC/RUNW/000/002/015/611/531/document.pdf (дата обращения 07.04.2015).
11. Волченков Н.Г. Синтаксический анализ изображений совокупности тел некоторого класса, использующий интерфейс логического и императивного визуального языков программирования. Журнал «Новые информационные технологии», ФГУП «ЦНИЛОТ», 2010, № 1, с.58 – 62.

Приложение. Программа синтаксического анализа языка утверждений и вопросов для изображений рассматриваемого класса

Здесь представлена основная часть программы, фрагмент которой рассмотрен в разделе **Технология практической реализации и полученные результаты** данной статьи. В указанном фрагменте был представлен процесс предварительной обработки поверхностных структур утверждений и вопросов на ограниченном естественном языке. В результате этой обработки в базе данных Пролога появляются пронумерованные факты-утверждения и факты-вопросы, которые незамедлительно подвергаются синтаксическому анализу.

Программа синтаксического анализа, приведённая ниже в нотации DCG Пролога, не только реализует проверку правильности исходных выражений с точки зрения их синтаксиса, но и решает основную задачу: создаёт в процессе синтаксического анализа (в качестве побочного эффекта) новые факты. Эти факты, представляющие собой глубинные структуры исходных фраз, записываются программой в базу данных Пролога.

П1. Начало кода (комментарии)

Разумно ограничиться представлением листинга, разработанной автором программы парсера LNL, в который автор включил необходимые комментарии. Они значительно облегчают понимание работы этой программы даже для тех читателей этой статьи, у которых нет значительного опыта в логическом программировании, в частности, на языке Пролог.

Итак, начало кода, интерпретируемого Прологом:

Листинг П1

```
/*      Левая скобка многострочного комментария.
  Перевод Утверждений и Вопросов
        с Ограниченного Естественного Языка (ОЕЯ) на  Язык Глубинных Структур (ЯГС)
  Порождающая грамматика:
Phrase →    Statement  |  Question     % Фраза – это утверждение или вопрос.
Statement →
        Verb NGroup Noun                % "назовём коричневый объект столом"
        AdjSize Noun AdjColor  |        % "большой ящик зелёный"
        AdjColor Noun AdjSize |         % "красный ящик небольшой"
        NounGroup Verb Relation NounGroup
                                        % "белая книжка лежит на зелёном ящике"
Question  →  QWord1  Verb  Relation NGroup |
                                        % "что лежит под столом"
        QWord2  Verb  NGroup |          % "где находится синяя коробка"
        Relation QWord3 Verb NGroup |
                                        % "под чем лежит голубая коробка"
        QWord4 WSize NGroup |           % "какого размера красный предмет"
        QWord4 WColor NGroup |          % "каков цвет маленькой книги"
Остальные правила грамматики приведены ниже, в программе - в нотации DCG.
И, в заключение данного вступления, указание, как запускается данная программа.
Запуск осуществляется вводом двух целей Пролога:
        | ?- bagof(N, s(N), L).             | ?- bagof(N, q(N), L).
Спецификация предиката bagof/3 такова:
1-й аргумент – форма, в которой выдаётся результат, в данном случае,
        это просто  номер целевого утверждения;
```

2-й аргумент – запрос к базе данных Пролога (или целевое утверждение),
в данном случае, это либо утверждение s(N), либо вопрос q(N);
3-й аргумент – это список всех ответов на запрос
Определения предикатов s(N) и q(N) – ниже, в самом начале программы.
Правая скобка многострочного комментария: */

П2. Интерпретируемая часть кода

После комментариев – интерпретируемая часть кода:

Листинг П2

```
        % Программа синтаксического анализа
        % с побочным эффектом – записью в БД Пролога глубинных структур
        % утверждений и вопросов на языке DSL.
        % Реализация перехода к нотации DCG:
statement(N)  :- test_st(N, Linput), an_St(T, [N|Linput], []), !.
question(N)   :- test_qu(N, Linput), an_Qu(T, [N|Linput], []), !.

        % Далее – правила синтаксического анализатора в нотации DCG:
an_St(N) -->   [N],   [присвоим], an_AdjNumb(N), [телу, имя], an_Noun(Name),
       {St =.. [st, N, object(Name, char(size(Size), color(Color)))],
        assert(St), write(St), nl}.      % assert/1 – предикат записи терма в БД Пролога.
                                         % Здесь – это старое утверждение с новым именем.
an_St(N) -->   [N],   an_Noun(W), [имеет],
       an_AdjSize(Size), [размер], [и],
       an_AdjColor(Color), [цвет],
       {retract(st(NX, object(W, _))), % retract/1 – предикат удаления
                                        % устаревшего утверждения из БД Пролога.
        St =.. [st, NX, object(W, char(size(Size), color(Color)))],
        assert(St), write(St), nl}.      % - запись нового утверждения на место
                                         %   удалённого устаревшего утверждения.
an_St(N1) -->          % Реализация симметричных отношений типа:
                       % если X «левее» Y, то Y «правее» X. И так далее.
       [N],   an_NGroup(Name1, Char1),
       an_Verb, an_Relation(R1),
       an_NGroup(Name2, Char2),
       {st(_, object(Name1, Char1)), st(_, object(Name2, Char2)),
        St1 =.. [st, N, location(is(R1),
              object(Name1, Char1), object(Name2, Char2))],
              assert(St1), write(St1), nl,
              opposite_location(R1, R2),
        St2 =.. [st, N, location(is(R2),
              object(Name2, Char2), object(Name1, Char1))],
              assert(St2), write(St2), nl}.
                    % Добавка этому правилу в нотации Пролога:
opposite_location(left, right) :- !.
opposite_location(right, left) :- !.
opposite_location(on, under) :- !.
opposite_location(under, on) :- !.
opposite_location(front, behind) :- !.
opposite_location(behind, front) :- !.
```

```
an_Qu(N) -->    [N],   an_QWord4, an_WSize, an_NGroup(Name, _),
          {st(_, object(Name, Char)),
           Qu =.. [qu, N, object(Name, Char)],    assert(Qu), write(Qu), nl}.
                    % Вопрос с вопросительным «Какой размер» и тому подобным.
an_Qu(N) -->   [N],   an_QWord4, an_WColor, an_NGroup(Name, _),
          {st(_, object(Name, Char)),
           Qu =.. [qu, N, object(Name, Char)],     assert(Qu), write(Qu), nl}.
                    % Вопрос с вопросительным «Какой цвет» и тому подобным.
an_Qu(N)-->   [N],    an_QWord1, an_Verb, an_Relation(R), an_NGroup(Name, Z),
          {st(_, location(is(R), object(Name1, Z1), object(Name, Z))),
           st(_, object(Name1, Z1)),
           Qu =.. [qu, N, location(is(R),object(Name1, Z1), object(Name, Z))],
           assert(Qu), write(Qu), nl}.
                    % Вопрос с вопросительным «Что лежит на, под …» и т.д.
an_Qu(N) -->   [N],    an_QWord2, an_Verb,  an_NGroup(Name, Z),
           {st(_, location(is(R), object(Name, Z), object(Name1, Z1))),
            st(_, object(Name1, Z1)),
            Qu =.. [qu, N, location(is(R),object(Name, Z), object(Name1, Z1))],
            assert(Qu), write(Qu), nl}.
                    % Вопрос с вопросительным «Где лежит …» и т.д.
an_Qu(N) -->   [N],    an_Relation(R), an_QWord3, an_Verb, an_NGroup(Name, Z),
          {st(_, location(is(R), object(Name, Z), object(Name1, Z1))),
           st(_, object(Name1, Z1)),
           Qu =.. [qu, N, location(is(R),object(Name, Z), object(Name1, Z1))],
           assert(Qu), write(Qu), nl}.
                    % Вопрос с вопросительным «На чём лежит, под чем лежит …» и т.д.
```

```
an_QWord1 --> [что].
an_QWord2 --> [где].
an_QWord3 --> [X], {member(X, [чем, чём, чего])}.
an_QWord4 --> [X], {member(X, [какой, каков, какого])}.
an_WSize  --> [X], {member(X, [размер, размера])}.
an_WColor --> [X], {member(X, [цвет, цвета])}.
```

```
an_Verb --> [X], {((L = [назовём, переименуем]; L = [лежит, стоит, находится]);
                    L = [расположен, расположена, расположено]), member(X, L)}.
an_Relation(on)  -->       [на].
an_Relation(under) -->    [под].
an_Relation(left)  -->     [левее].
an_Relation(right) -->     [правее].
an_Relation(near) -->      [X], {member(X, [около, возле])}.
an_Relation(behind)-->    [X], {member(X, [за, позади, сзади])}.
an_Relation(front) -->     [X], {member(X, [перед, спереди])}.
```

```
an_Noun(W)  -->  [X], {name(X, UX),         % Анализ существительного
            reverse(UX, [UN|_]), member(UN, "0123456789"),
            append("box", [UN], UW),  name(W, UW)}.
             % для преобразования в каноническое слово «boxN», где N = 1, 2, …
an_Noun(body) --> [X], {name(X, UX),       % для преобразования в слово «body»
           (L = "объект"; (L = "тело"; (L = "предмет"; L = "коробк"))),
            append(L, _, UX)}.
an_Noun(pyramid) --> [X], {name(X, UX),  % для преобразования в слово «pyramid»
            L = "пирамид",
            append(L, _, UX)}.
an_Noun(book) --> [X], {name(X, UX),      % для преобразования в слово «book»
           (L = "книг"; L = "книж"),
            append(L, _, UX)}.
an_Noun(table) --> [X], {name(X, UX),      % для преобразования в слово «table»
            L = "стол",
            append(L, _, UX)}.
```

```
an_NGroup(Name, char(size(Size),color(_))) -->     % Анализ группы существительного
          an_AdjSize(Size), an_Noun(Name).          % с указанием только размера
an_NGroup(Name, char(size(_),color(Color))) -->   % Анализ группы существительного
          an_AdjColor(Color),  an_Noun(Name).     % с указанием только цвета
an_NGroup(Name, char(size(_),color(_))) -->        % Анализ группы существительного
          an_Noun(Name).                            % без цвета и без размера
an_NGroup(Name, char(size(Size),color(Color))) -->  % Анализ группы существительного
          an_AdjSize(Size), an_AdjColor(Color),        % и с цветом, и с размером
          an_Noun(Name).
```

```
an_AdjNumb(N) --> [X],                          % Анализ слова для порядкового номера тела
          {Y =.. [X, N], member(Y,
          [первому(1), второму(2), третьему(3), четвёртому(4), пятому(5)])}.
an_AdjSize(big) --> [X], {name(X, UX),          % Анализ слова для большого размера
          (L = "больш";  L = "огромн"),
           append(L, _, UX)}.
an_AdjSize(middle) --> [X], {name(X, UX),     % Анализ слова для среднего размера
           L = "средн", append(L, _, UX)}.
an_AdjSize(small) --> [X], {name(X, UX),      % Анализ слова для маленького размера
          (L = "маленьк"; L = "небольш"),
           append(L, _, UX)}.
an_AdjColor(red) --> [X], {name(X, UX),       % Анализ слова для красного цвета
           L = "красн",
           append(L, _, UX)}.
an_AdjColor(green) --> [X], {name(X, UX),    % Анализ слова для зелёного цвета
          (L = "зелён"; L = "зелен"),
           append(L, _, UX)}.
an_AdjColor(blue) --> [X], {name(X, UX),    % Анализ слова для синего цвета
          (L = "голуб"; L = "син"),
          append(L, _, UX)}.
```

```
an_AdjColor(brown) --> [X], {name(X, UX), % Анализ слова для коричневого цвета
```

```
            (L = "коричнев"; L = "бур"),
            append(L, _, UX)}.
an_AdjColor(white) --> [X], {name(X, UX),  % Анализ слова для белого цвета
            L = "бел",
            append(L, _, UX)}.
%-------------------------------------------------------------------------Конец программы.

%                                                                © Николай Волчёнков. 2018.
```