Scientific Visualization, 2018, volume 10, number 5, pages 57 - 85, DOI: 10.26583/sv.10.5.05

The image analysis of the geometric bodies with supplemental interactive block for training new knowledge in a limited natural language

N.G. Volchenkov

National Research Nuclear University "MEPhI"
ORCID: 0000-0002-1474-6853, NGVolchenkov@mephi.ru

Annotation

The prototype of the system for image analysis of geometric bodies, using the interface of logical and visual programming, was developed by the author for experiments with an intelligent robot that is equipped with a video camera. It is assumed that the robot is designed to plan its actions - capture and transfer of bodies. The class of bodies considered by the author is colored polyhedra in local colors. The drawback of the results obtained in the author's previous publication is the absence of an analysis of the relative location of the identified set of polyhedra. Obvious are the difficulties that can arise when trying to automatically (without the participation of a person) to identify the mutual arrangement of bodies. The task is facilitated by the inclusion in the system of the block of interactive training of the robot by man. This person is an operator - a person who can formulate his description of a specific image in a limited natural language. The form of this description is the so-called surface structures of natural language phrases. In this article, the author presents the program of syntactic analysis of surface structures necessary for this purpose. This program is implemented on Prolog - the language of logical programming. To illustrate the results of the training block implemented on Prolog, the author, like in his previous publication, offers a visual programming interface (Visual Basic language) and logical programming (Prolog language). The article presents an example of a concrete image, on which 5 bodies of different colors are revealed. This example allowed us to demonstrate the most typical cases of the relative positioning of bodies, their description on the surface structures proposed by the author and the syntactic analysis of the phrases of this language. An important side effect of syntactic analysis is also presented - the construction of deep structures of a limited natural language. These structures are represented in the form of structures of the language Prolog - on the developed by the author language of deep structures. This view can later be used by the robot directly to plan its actions.

Key words: logical programming (LP); language Prolog; visual programming; the Visual Basic language; database of Prolog; limited natural language (LNL); the surface structure of the phrase on the LNL; learning as introducing new knowledge into the Prolog database; syntactic analysis of the LNL; the language of deep structures (DSL); the definite clause grammar (DCG) – the mechanism in Prolog.

Introduction

The content of this article is the development of the topic presented by the author's previous publication in this journal – the article "The application logical and visual programming interface for image analysis of the geometric bodies" [1]. In this publication, the urgency of the "intellectual" task of analyzing the images obtained from the video camera of objects that are in the field of view of an industrial robot manipulator, whose purpose are: the choice of the desired object; estimation of its size; his location relative to other objects and planning his actions to capture and transfer this subject.

In this publication, in particular, the possibility of using a joint ap-

plication of *logical programming* (Prolog language) and *visual programming* (Visual Basic language) for analyzing a set of images of geometric bodies with flat faces was justified.

Important stages of analysis have been described, from the stage of transforming a tone (raster) image into a vector image (Figure 1), and ending with the logical analysis of a vector image, as a result of which an exact number of bodies with a number of characteristics is revealed in the image (Figure 2).

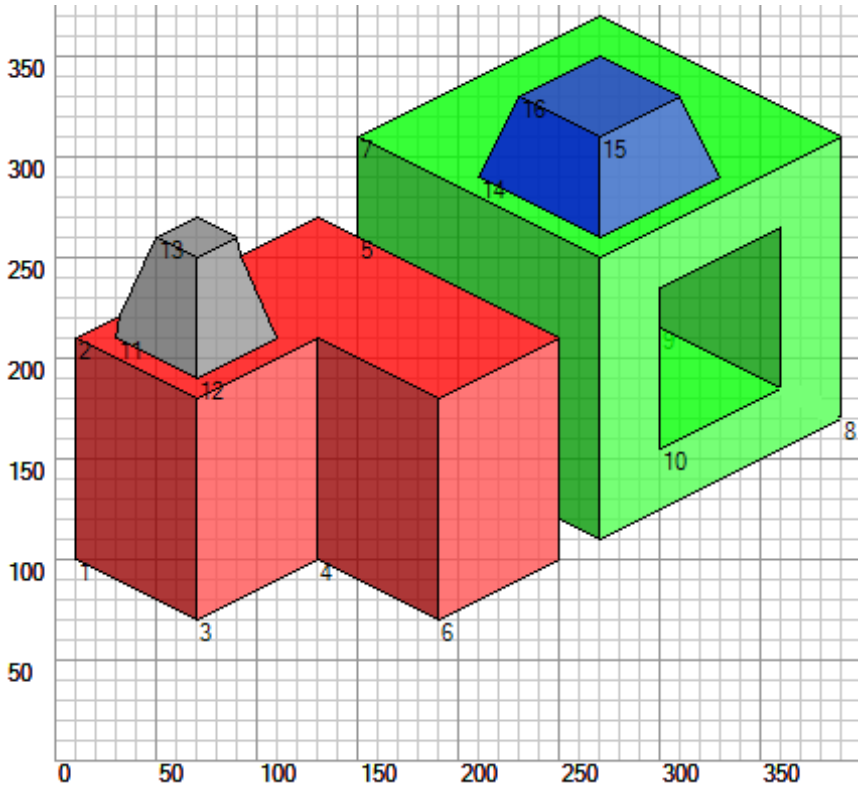

Figure 1. An example of a "picture" obtained during the conversion of a bitmap image into a vector image. The numbers of all detected faces are shown.

```
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
```

Figure 2. The result of the analysis of the image shown in Figure 1.

The characteristics obtained as a result of the analysis of images of bodies of this class are as follows:

(1) the number of detected bodies ("bodies" in this case are conventionally called sets of images of polygons corresponding to the faces of polyhedrons);
(2) the coordinates of the conventional "centers" of the identified bodies (the coordinates of the centers of the rectangles, in each of which images of all the faces of one polyhedron are inscribed);
(3) conditional "sizes" of identified bodies in units proportional to the areas of the rectangles indicated in the previous paragraph (2);
(4) the "objective" color value of each body as the value of the function **Argb**, the arguments of which are calculated as the mean values of this function for all faces of this body, and the "subjective" value of the linguistic variable **Color** (**red**, **green**, etc.).

The publication [1] presented the "logical-visual" approach formulated by the author to the solution of the problem of analyzing images of geometric bodies with planar faces. The main component of this approach is the syntactic (structural) method based on logical programming [2, 3, 4], which is a particular case of a wider paradigm of declarative programming [5], fundamentally different from the traditional paradigm of imperative (operator) programming. The problem of analyzing images of geometric bodies considered in this publication can serve as a fairly vivid example of the problems of this class. Therefore, the results obtained by the author on the creation of the demonstrational prototype of the "intellectual" component of the system of image analysis of geometric bodies can serve as a good basis for mastering the skills of practical application of the declarative approach to programming many problems of artificial intelligence [6, 7].

As noted in the publication [1], the experience of using functional and logical programming languages “to solve symbolic processing problems led to an understanding of the need to visualize the results of solving many practical problems of this class. Without visualization, the interpretation of these results is extremely difficult. To solve the problem of visualization, the author proposed an interface of two programming languages - the language of logical programming (**Prolog**) and the visual imperative programming language (**Visual Basic**), more precisely, of two programming systems for these languages” [8, 9]. Language Prolog is used to solve the main, "intellectual" part of the task. The Visual Basic language is used to perform computations that supplement the logic inference, as well as to form a visual representation of the results of this inference.

Of course, most of the information that is needed by the "intelligent robot" to solve the tasks of planning its actions is not contained in the results, the example of which is shown in Figure 2. They are "seen" by the person, but they are not "visible" to the robot, since it is not has the meta-knowledge that a person has. For example, knowledge of the mutual arrangement of bodies, which can be very important in making decisions about the capture of the desired object (if necessary, its transfer to a new location).

The object can be "swamped" with other items that need to be removed to get to the desired object. Etc. For example, to capture and transfer the green "box" in Figure 1, you need to release it from the object resting on it - the pyramid of blue. In this case, the robot "knows" only that the pyramids in Figure 2; that one of them is gray, its size is smaller than the size of the other, the blue pyramid, and that it is placed on the leftmost object. Of

course, to solve this simple task for a human robot, it is necessary, as a child, to explain something - in particular, to teach it the elementary methods of deduction. In particular, to make such, for example, the conclusions: "from the fact that both pyramids of different colors; the gray pyramid is on the red object; the green box serves as a support for the larger pyramid, it follows that to release the green box, you must remove the blue pyramid from it".

In this article, we describe an attempt to complement the system of image analysis of the class indicated in the previous article with a learning component. Training is supposed in the mode of communication with a person - a "teacher", who in a limited natural language interprets the results obtained at the previous stage.

For example, with respect to the image shown in Figure 1, the "teacher" can enter the following statements into the database of the logical part of the image analysis program (in the logical programming language Prolog) in a limited natural language (LNL):

1. Big blue pyramid is on the green box.
2. Little gray pyramid stands on the red object.
3. Large object is to the right and behind the small object.
4. Big green object has the rectangular notch.
5. A red object does not have specificity.
6. All four bodies highlighted on the right.
   Etc.

After entering these phrases on the LNL (in the form of Prolog lists, for example: **[big, blue, pyramid, is, on, the, green, box]**) representing the analyzed language strings, their syntactic analysis takes place in accordance with some generative grammar. After that, new facts should appear in the database (Prolog database). These facts will be presented in the form of so-called *deep* (canonical) *structures* - some *statements*.

For example, for the above phrases - these are the following Prolog facts:

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)))).

Here, "_" is an anonymous unnamed Prolog variable, *char* is an abbreviation of the word *characteristics*, in this case it is the characteristics of the object. Characteristics are several: *size*, *color*, *spec* (short for the word *specific*).

Note that the number of structures (here their 7) may not coincide with the number of initial phrases on the LNL (here they are 6).

We also note that the same *deep structures* can, by virtue of the variety of ways of expressing the same meaning and by replacing many concepts (for example, **object**, **box**, **pyramid**, etc.) with a single *canonical term* (for example, **object**), can become the same for different initial so-

called *surface structures* – LNL sentences. This leads to the fact that the number of deep structures for a given, very specific subject area, is significantly, by orders of magnitude, smaller than the number of surface structures.

This makes it possible to significantly simplify the process of "understanding" by the *learner* (the *object of learning* - the program on Prolog) of the phrases on LNL provided to the Prolog database by the *instructor* (the *subject of training* is the human operator). In practice, this translates into the fact that the formal grammars that generate phrases on the LNL can be compact and concise.

"Teacher" can "ask" the program to answer the questions formulated also on LNL:

1. How many items are there in the image?
2. What color is the big box?
3. What is to the left of the green object?
4. What color is the body next to the small body?
5. Is the red body of a green body smaller?
   Etc.

After the introduction of these phrases (questions) to LNL and their parsing, new facts should appear in the Prolog database in the form of so-called "deep structures" of *questions*, for example, for the ones mentioned above:

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), _)))).

The main "side effect" of the actions of the "teacher" (input of statements and questions on the LNL) should be new knowledge, which should be preserved in the Prolog database in the form of facts-statements and facts-questions. In the future, facts-questions can be *interpreted* using user requests to the Prolog database. The result of this interpretation may turn out to be unexpected: the variables contained in these structures can "turn" from unrecognized into the ones indicated! This effect is indicative of the elements of "intellectuality" of formal logical (deductive) inference, which "the program" itself displays on Prolog in the process of its work.

For example, the third question from the list: question **(location (is (left), object (Name1, Char1), object (Name2, char (_, color, green), _))))** Prolog response can unexpectedly follow: **statement (location (is (left), object (box2, char (size (little), color (red), spec (no))), object (box1 , char (size (big), color (green), spec (notch))))**. (To the left of the large green object named box1, which has a feature (notch), there is a small red object named box2.)

Note that, by the way, Prolog "independently specified": the original green object (1) has a large size, (2) has the name box1 and (3) has a feature (notch). It's good or bad - to get redundant information - to judge the user.

1. Theoretical bases of the use of logical programming for processing texts on LNL, describing images of the totality of bodies of the class under consideration

1.1. The use of difference lists to improve the efficiency of downward parsing in Prolog

Parsing is an important application of logical programming (in general) and Prolog language (in particular).

We recall the ones presented in Sec. 1.1 of the article [1] definitions: *generating grammar*; formal *language* generated by such a grammar, as well as the place of context-free and context-dependent grammars in the classification of formal grammars. In the classic formalization of generative grammars first proposed by Noam Chomsky in the 1950s. [Wikipedia: https://en.wikipedia.org/wiki/Formal_grammar].

Formal generating grammar is the next "four":

G = <VT, VN, S, P>,

where **VT**, **VN** are terminal and non-terminal dictionaries, **P** = $\{\alpha_i \rightarrow \beta_i\}$ is the set of inference rules, where α_i is a chain containing a non-terminal symbol, β_i is an arbitrary chain of terminal and non-terminal characters, S is the initial symbol.

Direct derivation is a relation:

$\lambda \Rightarrow \mu$,

where $\lambda = \delta_1\ \alpha_i\ \delta_2$, $\mu = \delta_1\ \beta_i\ \delta_2$ and there exists a rule: $\alpha_i \rightarrow \beta_i$.

Derivation is the relation

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

if there is a sequence of relations

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

The *language* generated by the grammar G is the following set:

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

a set of terminal chains of γ - chains consisting only of terminal symbols – symbols of the terminal vocabulary VT).

Grammars and languages are divided into types. In practical problems, grammars and languages of the following three types are most often used.

Types of formal grammars:

- If a nonterminal symbol on the left of some grammar rule is surrounded by other symbols (terminal and / or nonterminal), then such a grammar is called *context-dependent*.
- Grammar is called *context-free* if the left-hand side of every rule of derivation of this grammar is a chain consisting of a *single* nonterminal symbol.
- A context-free grammar is called *automaton* if every rule of derivation of this grammar has the following form:

  $A \rightarrow aB$ or $A \rightarrow a$ or $A \rightarrow \lambda$,

  where A, B – nonterminal symbols, a – terminal symbol, λ – empty chain.

Types of formal languages:

- A language is called *automaton* if it is possible to construct an automaton grammar to generate it. (Of course, for the generation of this language, grammars of other types can also be constructed.)
- Language is called *context-free*, if it is possible to create a context-free grammar to generate it, but you can not construct an automaton grammar.
- Language is called *context-dependent*, if it is possible to construct a context-free grammar for its generation.

As mentioned above, the Prolog program has a built-in DCG mechanism that allows you to build effective downstream parsers for languages defined by formal grammars of various types. Its presence makes it possible to create efficiently working descending grammar analyzers on Prolog.

The DCG mechanism uses the notion of a *difference list*, by means of which it is possible to avoid a "combinatorial explosion" in the nondeterministic decomposition of the initial analyzed chain, if the language is sufficiently complex. It is this partition that must be made in descending syntactic analyzers.

As it turned out, you can do without an inefficient **append/3** predicate, the use of which "begs" for the nondeterministic partitioning of the chain into sub-strings. To do this, use the notion of a *difference list* [3, 4] or (equivalently) to introduce an additional argument into analyzer predicates.

A ***difference list*** is an infix type structure with the name \ (slash with a slope to the left) and two components: the list **[A1, ..., An | T]** and the list **T**:

[A1, ..., An | T] \ T.

This structure is equivalent to an "ordinary" list containing **n** elements: **[A1, ..., An]**. But there is a variable **T** in the difference list entry, whose value can be any list. An analogy can be given: any number, for example, **5**, is equal to the value of the expression **(5 + X) - X**, containing the variable **X** with any value.

Using the notion of "difference list" allows to significantly increase the efficiency of many list processing predicates - in our case, the **append/3** predicate (concatenation or "gluing").

Instead of defining

append([], L, L).

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

you can use a much more effective definition:

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

The illustration of this definition is

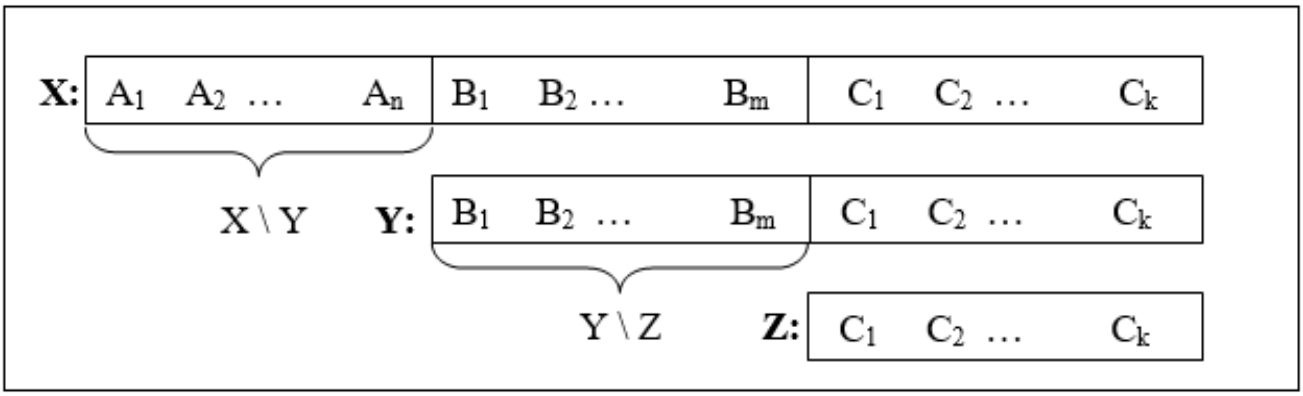

Figure 3. Illustration of the concatenation of difference lists:
append_dl (X \ Y, Y \ Z, X \ Z).

The last definition allows you to "glue" lists (their length can be very large) without numerous recursive calls - but only at the level of *matching patterns* (the fast working *pattern matching* Prolog).

Example 1.1.

Consider two lists: **L1 = [a, b, c]**; **L2 = [d, e, f]**. It is necessary to find the concatenation of these lists.

The call **? - append ([a, b, c], [d, e, f], Z).** leads to the need for three recursive calls, while the call **? - append_dl ([a, b, c | X] \ X, [d, e, f] \ [], Z).** gives the result **Z = [a, b, c, d, e, f] \ []** for the only step of inference.

Listing 1 is the multi-line comment of the small Prolog program. (Multi-line comment framed by brackets /* and */.) This listing demonstrates the experiment with system **Win Prolog LPA** [8].

Listing 1

```
/* Program:
:- op(200, xfx, '\').
append_dl(X\Y, Y\Z, X\Z).
Console:
# 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] \ []
*/
```

Example 1.2.

Consider a program on Prolog that implements an analyzer for a fragment of a context-free grammar that generates phrases (questions) like:

(1) Is a small gray pyramid located on a red box?

(2) The small gray pyramid is located on the red box?

using the *difference lists* instead of the **append/3** predicate.

Before writing a parsing program, that works on the principle of top-down parsing, it is recommended to build a "classical" generative grammar.

It is desirable to make a "minimal" generating grammar that would ensure the generation of language phrases that do not go beyond the types of Example 1.2. The generating grammar for the analysis of the above chains of types (1) and (2) can be one that is represented by the following listing:

Listing 2

```
/*
SLoc → Verb NGr Verb Prep NGr Qm          for phrases of the type (1);
SLoc → NGr Verb Verb Prep NGr Qm          for phrases of the type (2);
   non-terminal NGr means "noun group";
   non-terminal Verb means "verb";
   non-terminal Prep means "preposition";
   non-terminal Qm means " question mark".
NGr → Noun | Adj NGr
Verb → is | located | ...
Prep → on | under | to the left | to the right | higher | below | ...
Adj → big | little | green | red | large | small | ...
Noun → subject | object | box | body | pyramid | ...
   non-terminal Noun means "noun";
   non-terminal Adj means "adjective"
   in the nominative or the genitive case (no for English – only for Russian).
*/
```

Remarks (for the version in Russian).

This is a context-dependent grammar, since the sentence members must be related by gender, number and case. Context dependency can be implemented using variables. For example, as in the two changed lines of Listing 2:

Listing 3

```
SLoc → NGr(K1) Verb(K1) Verb(K1) Prep NGr(K2)          for phrases of the type (1);
SLoc → Verb(K1) NGr(K1) Verb(K1) Prep NGr(K2)          for phrases of the type
(2);
```

The values of the variables K1 and K2 are the context, for example, in the considered case:

K1 = k(case(nominative), gender(female)) and **K2 = k(case (prepositional), gender(female))** - for phrases of the type (1), for example: "**the red box is placed on a green box**" - *in Russian*;

K1 = k(case(nominative), gender(female)) and **K2 = k(case(genitive), gender(female))** - for phrases like (2), for example: "**was the red box placed on a green box?**" - *in Russian*.

The context connection **K1** is realized between the symbols **NGr** and **Verb** at the beginning of the phrase (1), and the context **K2** - between the symbols **Noun** and **Adj** at the end of the phrase (2).

End of remarks (for the version in Russian).

And now, the main thing: about replacing the ineffectively working predicate **append/3** difference lists.

Listing 4 represents a parser on Prolog with a quasi-nondeterministic, slowly working **append/3** predicate (for now, without using the difference lists and the **DCG** mechanism).

Listing 4

```
/* Grammar:
SLoc → Verb(K1) NGr(K1) Verb(K1) Prep NGr(K2)              for phrases of the type (1)
SLoc → NGr(K1) Verb(K1) Verb(K1) Prep NGr(K2)        for phrases of the type (2)
NGr(K) → Noun(K)
NGr(K]) → Adj(K) NGr(K)
Prolog: */
an_SLoc( location( type(2), X, L1, Y, L2), Linput) :-
      append( [X|L1], [Y, Z|L2], Linput),
      an_Verb( K1, X),                              % for phrases of the type (1)
      an_NGr( K1, L1),
      an_Verb( K1, Y),
      an_Prep( Z),
      an_NGr( K2, L2).
an_SLoc( location( type(1), L1, X, Y, L2), Linput) :-
      append( L1, [X,Y|L2], Linput),
      an_NGr( K1, L1),                              % for phrases of the type (2)
      an_Verb( K1, X),
      an_Verb( K1, X),
      an_Prep( Y),
      an_NGr( K2, L2).
 an_NGr( K, [X]) :- an_Noun( K, X).
 an_NGr( K, [X|L]) :- an_Adj( K, X), an_NGr( K, L).
```

In this program in Prolog's notation it is assumed that the analyzed LNL chains are represented as lists of tokens. Here, these are the following lists:

[is, small, gray, pyramid, located, on, red, box, '?'];

[small, gray, pyramid, is, located, on, red, box, '?'].

We use the above definition of a difference list:

DList = [A1, ..., An | T] \ T.

This list is equivalent to the "usual" list **[A1, ..., An]**.

Using difference lists instead of "normal" lists allows to significantly reduce the time for searching for variants of nondeterministic partitioning of the chain represented as a list of tokens into sub-strings.

It is known [2, 6, 7] that an algorithm of *downward parsing* (syntactic analysis from top to bottom) based on logical programming, in general, and on Prolog, in particular, is based on such a partitioning. The difference lists allow to get rid of the **append/3** predicate in the rules of the parser. This shown in the following example:

Example 1.3.

Suppose that there is a grammatical rule with three nonterminal symbols on the right: **S → A B C**. When implementing a downward parser on Prolog, this rule requires breaking up the original chain into 3 chains:

Listing 5

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

As noted above, the use of the predicate of concatenating two lists (or splitting one list into two) is described without recursive rules: **append_dl(X \ Y, Y \ Z, X \ Z)**. This allows nondeterministic partitioning of lists into sub-lists without costly recursive calls – only at the level of identifying data structures (*pattern matching*).

Actually, the above rule in the analyzer will look like this:

Listing 6

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

This inclusion of the notion of a difference list in Prolog syntax analyzers allowed the creators of numerous modern versions of this language to include the DCG mechanism, specially developed about 40 years ago [2], which is described in detail in the next section 1.2.

1.2. DCG – the built-in parsing mechanism in Prolog that implements the idea of difference lists

The implementation of difference lists is the DCG mechanism built into any modern Prolog system.

In the classic monograph of Sterling and Shapiro [7] it is noted (p. 203): "The origin of Prolog is connected with the attempt to use logic to express grammatical rules and formalize the process of syntactic analysis. The most common approach to the implementation of parsing by Prolog is the use of *definite clause grammar* (DCG). Such grammars are some generalization of context-free grammars. They are a notation version of a certain class of programs on Prolog and therefore are *executable*.

The DCG as a notation version of the programs of the class of "downstream parsers" on Prolog, in which the rules of generating grammars are "written" directly, is declared by the following rules of notation:

(1) Prolog bundle **:-** ("reverse implication"), which is read as the word "if", between the left and right parts of the Prolog rule, is replaced by a bundle **-->** corresponding to the arrow of the generating grammar.

(2) The predicates of both the left part of the rule (before the binding -->)

and the right side of the rule (after the link -->) in the DCG notation should not contain the input chain (the lexeme list) or the remainder (in the form of input parameters) This chain, left after its analysis in accordance with this rule of grammar.

(3) The arguments of the predicates of both left and right of the rule in DCG notation can only be output parameters.

(4) All additional actions accompanying the syntactic analysis (for example, arithmetic calculations), produced with the arguments of the predicates, must be surrounded by curly braces: {and}.

(5) The terminal symbols in the right parts of the grammar rules must be represented by lists. In particular, if one character is "read", for example, a token, this is a list of one element.

Example 1.4.

Let's look at Listing 6 - the parse program according to the grammar rule:

S → A B C.

In the only rule of this program, in both the left and right part of it all predicates have only one input parameter represented by the difference list. Therefore, in DCG notation, this rule will have the following form:

Listing 7

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

Let's say that this rule is used for analysis language: **L = {$a^n b^m c^k$}, n>0, m>0, k>0**. This is a context-free language, so it does not require the use of variables to implement a context dependency. To write the analyzer with the output of the three output parameters **N**, **M** and **K**, add additional rules to Listing 7 in the DCG notation:

Listing 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 }.
/*  Example of calling the goal and getting the result:
# 0.000 seconds to consult a^nb^mc^k.pl [d:\2018\]
| ?- an_S( N, M, K, [a,a,a,b,b,c,c,c,c], []).
Yes,  N = 3 , M = 2 , K = 4  */
```

And, finally, we introduce one more complication: the equality of the values of the variables **N**, **M** and **K**. This restriction turns the language into a context-dependent language **{$a^n b^n c^n$}, n>0**. To implement the analyzer of this language in DCG notation, it is enough to change only the first rule:

Listing 9

```
 an_S( N) --> an_A( N),  an_B( M),  { M = N },
                an_C( K), { K = N }.
...
/* Example of calling the goal and getting the result:
# 0.000 seconds to consult a^nb^nc^n.pl [d:\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  */
```

The Prolog analyzer for the example 1.2 in Listing 4 without the **append/3** predicate and using DCG is shown in Listing 10:

Listing 10

```
an_SLoc( location( type1)) -->                  % for phrases of the type (1)
   an_Verb( K1), an_NGr(K1), an_Verb( K1), an_Prep, an_NGr( K2).
an_SLoc( location( type2)) -->                  % for phrases of the type (2)
   an_NGr( K1), an_Verb( K1), an_Verb( K1), an_Prep, an_NGr( K2).
an_NGr( K) --> an_Noun( K).
an_NGr( K) --> an_Adj( K), an_NGr( K).
```

In conclusion of this section, let's consider what rule in the traditional Prolog notation the rule is automatically converted into DCG notation. This question is answered by the query: **? - listing (<predicate name> / <arity of predicate>).** You must enter this query after compiling the predicate that you entered in the DCG notation.

For example, we call the specified query after compiling the definition of the predicate **an_S/1**, entered in the DCG notation (Listing 11), and after testing the operation of this predicate (Listing 12):

Listing 11

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

Listing 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 ).
```

Note that using the built-in predicate **'C'/3**, which is used only to separate the head (**Head**) and the tail (**Tail**) of the list (**List**), demonstrates the use of difference lists for parsing. Its definition is: **'C' ([Head | Tail], Head, Tail)**.

We redefine the names of the variables generated by the system (system names A, B, C, D, E, F, G), to more meaningful ones (mnemonic names). In the first rule, replace: A to N; B to Input; C to Rest; D to L1; E to N1; F to L2; G to N3. In the second rule, replace: A to Input; B to Rest; C to L1; D to L2. We also replace the built-in predicate **'C' / 3** in accordance with its definition. Then we get:

Listing 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].
```

When translating from DCG notation to Prolog's notation, the system automatically performs an additional optimization of these rules:

Listing 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).
```

Obviously, the presentation of these two rules in DCG notation (Listing 11) looks much laconic and expressive, as it directly reflects their relationship to the generating grammar.

2. Technology of practical implementation and the results obtained

The technology of practical implementation of the inclusion of the train

ing block on the LNL into a system of analysis of the considered class of images is more conveniently and clearly demonstrated on concrete examples. In the practical implementation of the approach considered in this article, an experiment was conducted using the following example.

Example 2.1.

Let the analyzed tone image after its transformation into a vector form (the transformation process described in the author's article [1]) demonstrate five bodies of different colors with flat faces (Figure 4).

The image analysis system described in the above article [1] not only converts the tone image into a vector image, selects the vertices and edges of the polyhedra represented in the image using the visual Visual Basic language, but also calculates a certain result using the Prolog program. The original data and the result are visualized on the screen form (Figure 5).

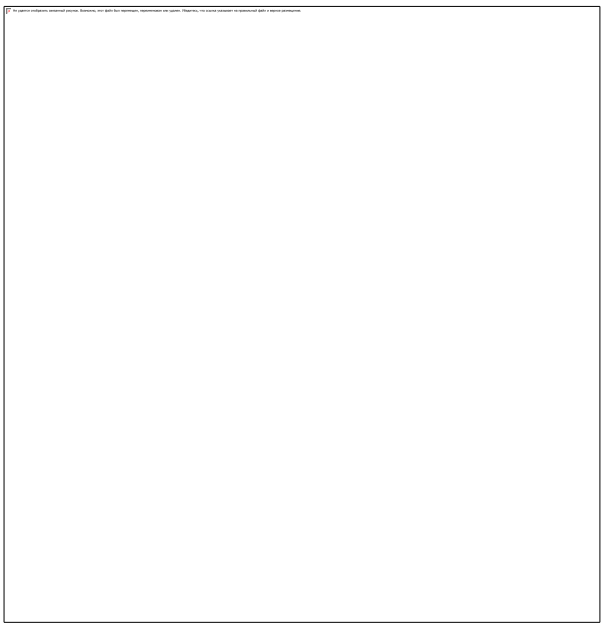

Figure 4. Example of a vector image (obtained from a tone image) of five bodies with flat faces. The faces of all bodies on this image are numbered

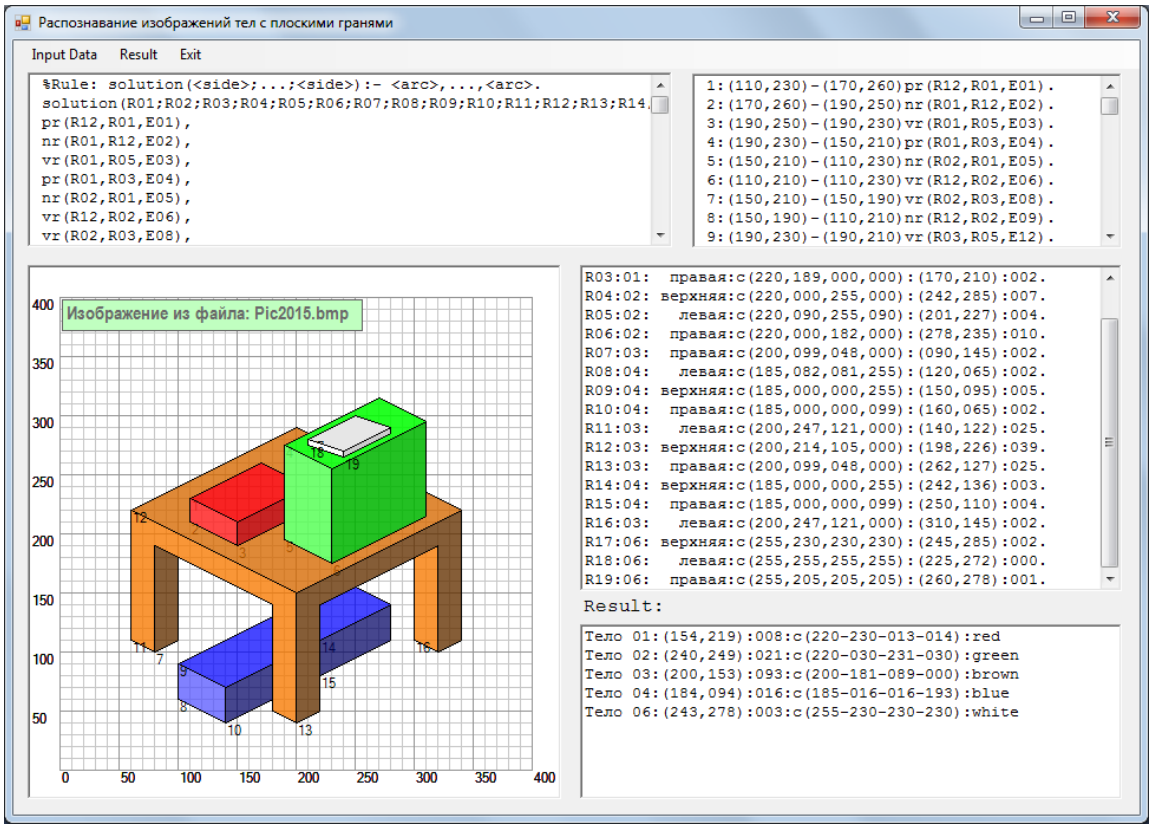

Figure 5. An example of the application's screen form, which implements the interface of programs in Visual Basic and Prolog. The form shows both the original data and the results of the image analysis: the revealed list of bodies (polyhedra) with their characteristics

The bodies in Figure 4 are in the spatial relationships that are obvious to the unsophisticated viewer. These relations man (observer) in accordance with his traditional model of the world can easily interpret, for example, as follows:

- in the figure an object is fixed, which can be called a "table" of brown color;

- on the "table" are two "boxes" of red and green color;
- under the "table" is a "box" of blue color;
- on the green "box" is a white "book".

As it was said in the previous article of the author on the topic under consideration [1], the ultimate goal of the image recognition system of this class can be the planning of the actions of an intelligent robot to capture and transfer objects fixed by the robot's video camera. The planning program, of course, should work together with the image analysis program.

Obviously, to specify the plan of the above actions of the robot, there is not enough information about the relative location of objects identified in the image. It seems advisable to include in the planning system of the robot's actions an additional "learning" block to what is the relative position of the objects on the image so that the robot can, for example, free an object intended for capture and transfer to a new location from other objects that interfere with it.

It is also obvious that it would be reasonable to use the experience of a person who could easily assess the situation and offer the work necessary for transportation (something to remove, something to rearrange, etc.). In order not to require a person to know the form of the robot's representation of the scene that the program fixes, it is reasonable to offer a very simple, limited natural language to the person, which we have already discussed in this article.

To create this block of training, first of all, it is necessary to develop a grammar that generates phrases LNL - a limited natural language for describing statements and questions regarding objects present in the image. Let's demonstrate a simple example of such a grammar and a program in the Prolog language that implements the syntactic analysis of phrases generated by this grammar.

It is reasonable to limit ourselves to the presentation of the small parser program, developed by the author of this article. In this program the author has included the necessary comments, which greatly facilitate the understanding of its work even for those readers of this article who do not have a significant experience in logical programming, in particular, in the Prolog language.

The purpose of this program is to convert the *surface structures* expressions of the *limited natural language* (LNL) into the *deep structures* expressions of the *deep structures language* (DSL) which, by virtue of their rigidly and unambiguously interpreted formal basis, can easily be used by an intelligent robot when planning its actions.

Example 2.2.

Transformation of the surface structure (the list of *lexemes* – list of English words):

[the, fourth, body, is, under, the third, body]

into the N-th structure for these bodies with detected values of names, sizes and colors:

st (N, location (is (under), object (box3, char (size (middle), color (blue))),
object (table, char (size (big), color (brown))))).

The beginning of the program on Prolog (more precisely, beginning of the code in the mixed of the "pure" Prolog notation and the DCG notation) is presented in the listing:

Listing 15

```
/* - Left bracket of a multi-line comment.
Translation of Statements and Questions from Limited Natural Language (LNL) to Deep Structures Language (DSL)
Due to the orientation of DCG notation to generative grammars, the grammar generating
surface structures (lists of lexemes on LNL) is not described here separately,
but is represented in the program itself in the form of rules in DCG notation.
The program is launched by entering the following Prolog goal:
    | ?- from_LNL_to_DSL.
The name of the target predicate is a mnemonic name: "translation from a limited natural language into the language of deep structures".
Right bracket of a multi-line comment:  */
```

After the comment - the interpreted part of the code:

Listing 16

```
% Start of the program
:- dynamic(st/2).    :- dynamic(qu/2).
                % These declarations remove protection
                % of the predicates st/2 and qu/2. This allows to write
                % new statements and questions into the Prolog database.
```

The Prolog test database in this program represents the facts corresponding to the image in Figure 4:

Listing 17

```
% Statements
% are presented in the form of three facts with the only argument.
% Each of these fact is a list of an arbitrary number of elements - lists of tokens.
% each of these token is a Russian or English word:
% (1) statements for the rename commands of all bodies in the image:
ren_sts([[assign, the, first, body, a, name, box1],
        [assign, the, second, body, a, name, box2],
        [assign, the, third, body, a, name, table],
        [assign, the, fourth, body, a, name, box3],
        [assign, the, fifth, body, a, name, book],]]).
                % "ren" from the word "rename",  "sts " from the word "statements".
% (2) statements to refine the values of the characteristics of bodies (size, color)
% in terms of values of linguistic variables:
char_sts([[first, body, is, small, and, red],
         [second, body, has, large, size, and, green, color],
         [third, body, has, huge, size, and, brown, color],
         [fourth, body, has, average, size, and, blue, color],
         [fifth, body, has, small, size, and, white, color]]).
                % "char" from the word "characteristics".
% (3) statements to clarify the location of bodies relative to each other:
loc_sts( [[fourth, body, is, under, third, body],
        [fifth, body, lies, on, second, body],
        [first, body, lying, on, third, body],
        [second, body, standing, on, third, body],
```

```
            [second, body, stands, right, first, body],
            [first, body, located, behind, second, body]]).
                    % "loc" from the word "location".
 % Questions
% are presented as one fact, the only argument of which
% is a list of an arbitrary number of elements - lists of tokens,
% each of which is a word of Russian or English:
all_qus( [[where, is, fifth, body, "?"],
            [where, lies, fourth, body, "?"],
            [what, is, on, third, body, "?"],
            [what, lies, under, third, body,"?"],
            [under, what, fourth, body, lies,"?"],
            [on, what, fifth, body, lies,"?"],
            [left, what, first, body, is, located, "?"],
            [before, what, second, body, is, located, "?"],
            [what, size, fifth, body, have, "?"],
            [what, color, fifth, body, have, "?"]]).
                    % "qus" from the word "questions".
```

The main part of the program presented here is located in the section of this article **Appendix. Prolog program that implements the top-down method of parsing for analysis of English LNL statements and questions about bodies with flat faces.**

Further - only a small fragment of the program.

In particular, it is a demonstration of the efficiency of the procedure for assigning new, more expressive names of bodies identified in the image - in the superficial structures of the LNL - in both statements and questions. *For example*:

- new_name(box1) → old_name(first, body);
- new_name(box2) → old_name(second, body);
- new_name(table) → old_name(third, body);
- new_name(box3) → old_name(fourth, body);
- new_name(book) → old_name(fifth, body).

The procedure on the following listing:

Listing 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).
```

The result of the testing is on the following listing ("photos" of the console):

Listing 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:\mephi-2018\]
| ?- from_old_to_new_names.
  new(box1)->old(first, body)
[assign, first, body name,box1]
  new(box2)->old(second, body)
[assign, second, body, name, box2]
  new(table)->old(third, body)
[assign, third, body name, table]
  new(box3)->old(fourth, body)
[assign, fourth, body, name, box3]
  new(book)->old(fifth, body)
[assign, fifth, body, name, book]
...
Yes
```

After the assignment of names to all bodies in the image in the Prolog database, the changed structures - facts-statements and facts-questions with new names and numbers assigned to them are recorded:

Listing 20

```
test_st (1, [assign, first, body, name, box1]).
test_st (2, [assign, second, body, name, box2]).
test_st (3, [assign, third, body, name, table]).
test_st (4, [assign, fourth, body, name, box3]).
test_st (5, [assign, fifth, body, name, book]).
test_st (6, [box1, has, small, size, and, red, color]).
test_st (7, [box2, has, large, size, and, green, color]).
test_st (8, [box3, has, average, size, and, blue, color]).
test_st (9, [table, has, huge, size, and, brown, color]).
test_st (10, [book, has, small, size, and, white, color]).
test_st (11, [box3, located, under, table]).     test_st (12, [the book, lies, on, the box2]).
test_st (13, [box1, lies, on, the table]).       test_st (14, [box2, stands, on, the table]).
test_st (15, [box2, stands, to the right, of a box1]).
test_st (16, [box1, located, behind, box2]).
test_qu (1, [where, is, the book]).              test_qu (2, [where, lies, box3]).
test_qu (3, [what, is, on, the table]).          test_qu (4, [that, lies, under, the table]).
test_qu (5, [under, what, lies, box3]).          test_qu (6, [on what, lies, the book]).
test_qu (7, [to the left, which, is, box1]).     test_qu (8, [before, what, is, box2]).
test_qu (9, [what, size, book, have]).           test_qu (10, [what, color, book, have]).
```

After calling the goal: | **? - setof (N, s (N), L).** the following results are obtained (on the console of the Prolog interpreter of LPA [8]) - statements in the form of canonical deep structures:

Listing 21

```
test_st (1, [assign, first, body, name, box1]).
test_st (2, [assign, second, body, name, box2]).
test_st (3, [assign, third, body, name, table]).
test_st (4, [assign, fourth, body, name, box3]).
test_st (5, [assign, fifth, body, name, book]).
test_st (6, [box1, has, small, size, and, red, color]).
test_st (7, [box2, has, large, size, and, green, color]).
test_st (8, [box3, has, average, size, and, blue, color]).
test_st (9, [table, has, huge, size, and, brown, color]).
test_st (10, [book, has, small, size, and, white, color]).
test_st (11, [box3, located, under, table]).   test_st (12, [the book, lies, on, the box2]).
test_st (13, [box1, lies, on, the table]).     test_st (14, [box2, stands, on, the table]).
test_st (15, [box2, stands, to the right, of a box1]).
test_st (16, [box1, located, behind, box2]).
test_qu (1, [where, is, the book]).            test_qu (2, [where, lies, box3]).
test_qu (3, [what, is, on, the table]).        test_qu (4, [that, lies, under, the table]).
test_qu (5, [under, what, lies, box3]).        test_qu (6, [on what, lies, the book]).
test_qu (7, [to the left, which, is, box1]).   test_qu (8, [before, what, is, box2]).
test_qu (9, [what, size, book, have]).         test_qu (10, [what, color, book, have]).
```

Note that statements with numbers 12-17 appeared automatically during parsing using logical deductive output (for example, "if X is to the left of Y, then Y is to the right of X" or "if X stand of front of Y, then Y stand of behind of X" etc.).

To create this block of training, first of all, it is necessary to develop a grammar that generates phrases LNL - a limited natural language for describing statements and questions regarding objects present in the image. Let's demonstrate a simple example of such a grammar and a program in the Prolog language that implements the syntactic analysis of phrases generated by this grammar.

Note the following:

- in all statements (here, they are not 22, but only 17, since 5 duplicate assertions are automatically deleted);
- in the processed statements, all variables are automatically evaluated.

In all ten questions, variables, just like in all statements, are evaluated as a result of the program's work.

Questions have acquired the following canonical form (kind of deep structures):

Listing 22

```
    | ?- 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)))))
```

The impressive result of the experiment is that none of the ten phrases-questions examined left any variables that have not been evaluated, which indicates a sufficient "intellectuality" (of course, within the framework) of the program proposed by the author.

The software package developed for experiments with the image analysis system, supplemented by the training component considered in the article, is based on two interacting software *platforms*:

1. Microsoft .NET version 4 (specifically, version 4.0, Visual Studio 2010, and the Visual Basic 2010 language it supports);

2. Windows Prolog - versions of Prolog language interpreters and compilers developed and supported by LPA [8].

Portability to other platforms is currently not relevant, as the platforms discussed above are popular and accessible. In particular, later versions of the Microsoft .NET platform are compatible with version 4.0, and LPA is currently continuing to improve and commercialize Prolog language compiler interpreters, which are focused on the Windows operating system.

The functionality for the user demonstrates the following note.

The block of training, which is implemented in the prologue language, considered in this article, can be easily included in the image analysis system described in the previous article of the author [1]. It is enough to include in the menu of this system a command that causes the appearance of a new screen form.

This form, shown in figure 6, shows the original data in two text box: test statements and test questions about the characteristics and relative positions of the geometric bodies in the image. The third window shows the deep structures of answers to all test questions obtained as a result of training.

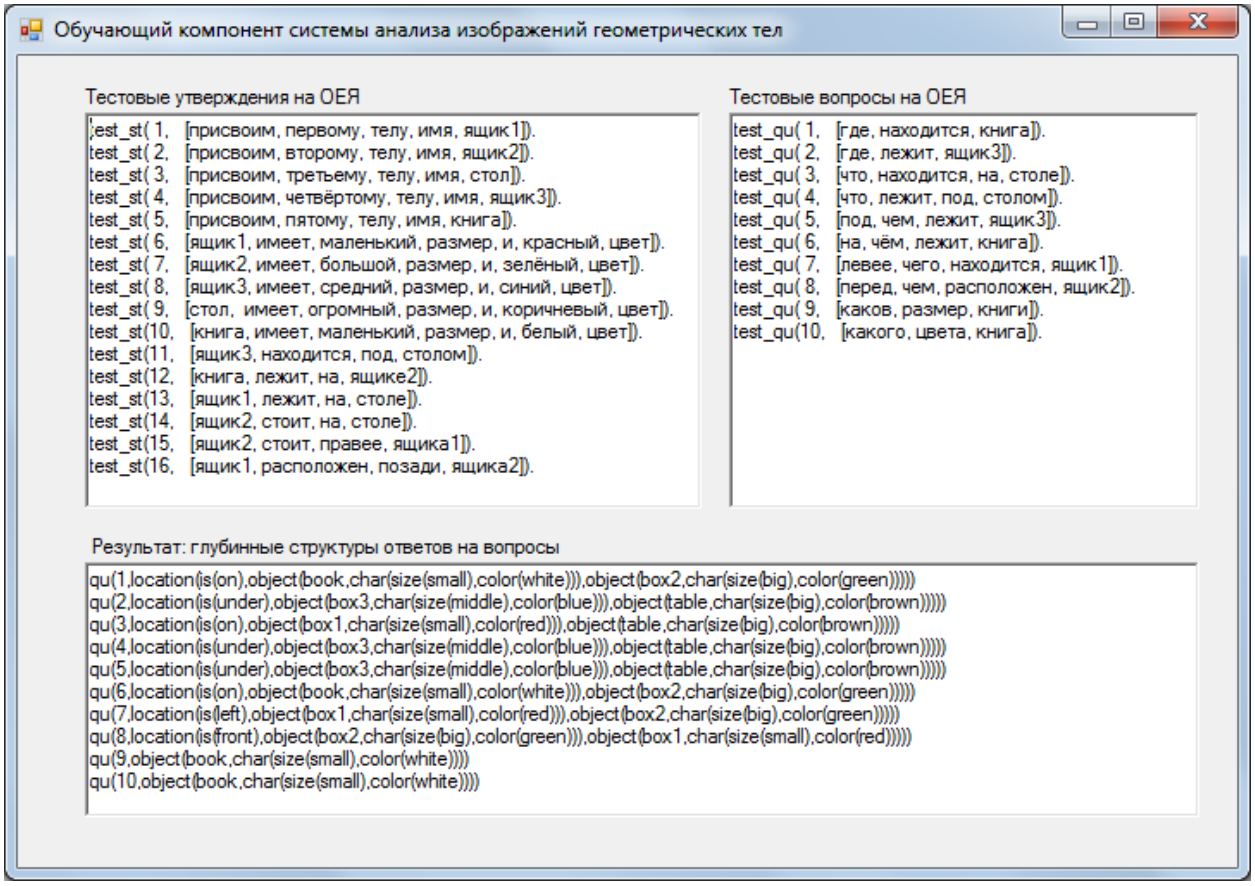

Figure 6. Screen form with the results of training tests processing, called up during the operation of the image analysis system

State registration of the program

The software product presented in this article served as an important addition to the software system developed by the author for image analysis of polyhedra and cylindrical bodies. This system passed state registration in 2015, and the author received a certificate of this state registration of the corresponding computer program "Demonstration program for the analysis of images of bodies with flat faces" No. 2015611531 (registration date January 30, 2015) [10].

Conclusion

In this article, the principal possibility of adding an image analysis system in which the logical (structural) approach is combined with the visual programming tools described in the publication [1], by the training unit, was considered. In this block it is proposed to implement the program communication with a person in a lim-

ited natural language using the syntactic analyzer realized by the author in the Prolog language. The purpose of this training is to get the system knowledge of the relative position of the bodies. This knowledge should help the robot equipped with the image analysis system to plan its actions related to capturing and transferring the bodies detected on the image.

The experimental check of the parser presented in the article showed good results: all the considered statements and questions regarding the images of several bodies fixed in the limited natural language were successfully analyzed. The side result was the values of variables in questions transformed into deep (canonical) structures. Answers to these questions provide important knowledge to the robot manipulator in planning its actions to capture and transfer bodies.

The approach proposed in this article, in the author's opinion, can be useful for teaching autonomous robots to properly navigate in solving the problem of capturing and transporting geometric bodies identified in the analysis of their images.

Bibliography

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. [Electronic resource]. URL: http://sv-journal.org/2015-3/09.php?lang=eng (circulation date 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. Volchenkov, N.G. Logical programming. Language Prologue: Texts of lectures. Ed. second, corrected. and additional. - Moscow: MEPhI, 2015. – 160 p.

4. Sergievsky G.M, Volchenkov N.G. Functional and logical programming: Proc. allowance for stud. higher education. - Moscow: Publishing Center "Academy", 2010. - 320 p.

5. Volchenkov, NG. The use of modern information technologies in teaching students in the field of "Informatics and computer technology", various programming paradigms: "Modern scientific research and innovation." 2015. № 3 [Electronic resource]. URL: http://web.snauka.ru/issues/2015/03/47345 (circulation date 28.03.2015).

6. Bratko Ivan. Algorithms of artificial intelligence in the language PROLOG, 3rd edition .: Trans. English. - Moscow: Williams Publishing House, 2004. - 640 pages

7. Sterling L., Shapiro E. The art of programming in the language Prolog. - Moscow: Mir, 1990. - 235 p.

8. The website of Logic Programming Associates Ltd. [Electronic resource]. URL: http://www.lpa.co.uk (circulation date: 21/03/2015).

9. Ziborov V.V. Visual Basic 2010 on examples. - St. Petersburg .: BHV-Petersburg, 2010. - 336 p.

10. Certificate of state. registration of the computer program № 2015611531 "Demonstration program for the analysis of images of bodies with flat faces." Author: Volchenkov N.G. (RU). [Electronic resource]. URL: http://www1.fips.ru/Archive/EVM/2015/2015.02.20/DOC/RUNW/000/002/015/611/531/ document.pdf (circulation date 07/04/2015).

11. Volchenkov, N.G. Syntactic analysis of images of a set of bodies of a certain class, using the interface of logical and imperative visual programming languages. Journal of New Information Technologies, FSUE "TsNILOT", 2010, No. 1, p.58 - 62.

Appendix. Prolog program that implements the top-down method of parsing for analysis of English LNL statements and questions about bodies with flat faces

A1. Beginning of the code (comments)

It is reasonable to limit ourselves to the presentation of the listing developed by the author of the LNL parser program, in which the author included the necessary comments, which greatly facilitate understanding of its work even for those readers of this article who do not have a significant experience in logical programming, in particular, in the Prolog language.

So, the beginning of the code interpreted by Prolog:

Listing A1

```
/*      Left bracket of a multi-line comment.
  Translation of Statements and Questions
  from Limited Natural Language (LNL) to Language of Deep Structures (LDS)
  Generating grammar:
Phrase →    Statement  |  Question    % A phrase is a statement or a question.
Statement →
        Verb NGroup Noun              % for example: " call a brown object a table"
        AdjSize Noun AdjColor  |      % for example: " the big box is green"
        AdjColor Noun AdjSize |       % for example: " the red box is small"
        NounGroup Verb Relation NounGroup
                                      % for example: " the white book is on the green box "
Question  →  QWord1  Verb  Relation NGroup |
                                      % for example: " what is under the table "
        QWord2  Verb  NGroup |        % for example: " where is the blue box "
        Relation QWord3 Verb NGroup |
                                      % for example: " under what is the blue box "
        QWord4 WSize NGroup |         % for example: " what size of the red object "
        QWord4 WColor NGroup |        % for example: " what is the color of the little book "
The rest of the grammar rules are given below, in the program - in DCG notation.
And, in conclusion of this introduction, an indication of how this program is launched.
The launch is carried out by entering two goals of Prolog:
        | ?- bagof(N, s(N), L).
        | ?- bagof(N, q(N), L).
The specification of the predicate bagof/3 is:
  The first argument is the form in which the result is produced, in this case,
        it's just the number of the target statement;
  The second argument is a query to the Prolog database (or target statement),
        in this case, either s (N) or q (N);
  The third argument is a list of all the answers to a query.
The definitions of the predicates s (N) and q (N) are lower, at the beginning of the program.
Right bracket of a multi-line comment:
*/
```

A2. Interpreted part of the code

After the comments - the interpreted part of the code:

Listing A2

```
% ---------- Beginning of the program ----------%
:-dynamic(st/2).        :-dynamic(qu/2).
    % Declarations that remove protection from st / 2 and qu / 2 predicates, which allows
    % write new statements and questions to Prolog DB (database)
```

```
% Statements – facts in the Prolog DB:
test_st( 1,   [assign, first, body, name, box1]).
test_st( 2,   [assign, second, body, name, box2]).
test_st( 3,   [assign, third, body, name, table]).
test_st( 4,   [assign, fourth, body, name, box3]).
test_st( 5,   [assign, fifth, body, name, book]).
test_st( 6,   [box1, has, small, size, and, red, color]).
test_st( 7,   [box2, has, big, size, and, virid, color]).
test_st( 8,   [box3, has, average, size, and, blue, color]).
test_st( 9,   [table, has, huge, size, and, brown, color]).
test_st(10,  [book, has, little, size, and, white, color]).
test_st(11,  [box3, is, under, table]).
test_st(12,  [book, lay, on, box2]).
test_st(13,  [box1, lies, on, table]).
test_st(14,  [box2, stands, on, table]).
test_st(15,  [box2, stands, to_the_right, box1]).
test_st(16,  [box1, located, behind, box2]).
```

```
% Questions – facts in the Prolog DB:
test_qu( 1,   [where, is, book, "?"]).
test_qu( 2,   [where, lies, box3, "?"]).
test_qu( 3,   [what, is, on, table, "?"]).
test_qu( 4,   [what, lies, under, table, "?"]).
test_qu( 5,   [under, what, lies, box3, "?"]).
test_qu( 6,   [on, what, lies, book, "?"]).
test_qu( 7,   [left, what, is, box1, "?"]).
test_qu( 8,   [before, what, is, box2, "?"]).
test_qu( 9,   [what, size, had, book, "?"]).
test_qu(10,  [what, color, had, book, "?"]).
```

```
        % Implementing the transition to DCG notation:
statement(N)  :- test_st(N, Linput), an_St(T, [N|Linput], []), !.
question(N)   :- test_qu(N, Linput), an_Qu(T, [N|Linput], []), !.
        % Next - the rules of the parser in DCG notation:
an_St(N) -->    [N],   [assign], an_AdjNumb (N), [body, name], an_Noun(Name),
        {St =.. [st, N, object(Name, char(size(Size), color(Color)))],
         assert(St), write(St), nl}.      % assert/1 – predicate of adding a term
                                          %  to the Prolog database.
                                          % Here is an old statement with a new name.
an_St(N) -->    [N],    an_Noun(W), [имеет],
```

```
        an_AdjSize(Size), [размер], [и],
        an_AdjColor(Color), [цвет],
        {retract(st(NX, object(W, _))), % retract/1 – predicate of removing statement
                                        % from the Prolog database.
                                        % Here is an outdated statement.
         St =.. [st, NX, object(W, char(size(Size), color(Color)))],
         assert(St), write(St), nl}.       % - adding of the new statement into place
                                        %   of the removed outdated statement.
an_St(N) -->            % This is the realization of symmetric relations:
                        % if X is "to the left" Y, then Y is "to the right" X. Etc.
        [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}.
                    % The addition to this rule in the pure Prolog notation:
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_QWord1, an_WSize, an_NGroup(Name, _), ["?"],
          {st(_, object(Name, Char)),
           Qu =.. [qu, N, object(Name, Char)],    assert(Qu), write(Qu), nl}.
                    % Interrogative words "What size", etc.
an_Qu(N) -->   [N],   an_QWord1, an_WColor, an_NGroup(Name, _), ["?"],
          {st(_, object(Name, Char)),
           Qu =.. [qu, N, object(Name, Char)],     assert(Qu), write(Qu), nl}.
                    % Interrogative words "What color", etc.
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}.
                     % The interrogative words "What lies on, under ...", etc.
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}.
                  % The interrogative words "Where lies ...", etc.
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}.
              % Interrogative words "On what lies, what lies ...", etc.
```

```
an_QWord1 --> [what].      an_QWord2 --> [where].
an_WSize   -->  [size].       an_WColor  -->  [color].
```

```
an_Verb --> [X],
   {(L = [call, rename, assign];
     L = [lies, stands, is]), member(X, L)}.
an_Relation(on)  -->      [on].
an_Relation(under) -->   [under].
an_Relation(near) -->     [X], {member(X, [near, around])}.
an_Relation(behind) --> [X], {member(X, [abaft, behind])}.
an_Relation(front) -->    [front].
an_Relation(left)  -->     [X], {member(X, [left, to_the_left])}.
an_Relation(right) -->    [X], {member(X, [right, to_the_right])}.
```

```
an_Noun(W)  -->  [X], {name(X, UX),              % numbered noun
            reverse(UX, [UN|_]), member(UN, "0123456789"),
            append("box", [UN], UW),  name(W, UW)}.
             % convert to the canonical word «boxN», where N = 1, 2, …
an_Noun(body) --> [X], {name(X, UX),           %  convert to the canonical word «body»
            (L = "object"; (L = "body"; (L = "polyhedron"; L = "box"))),
            append(L, _, UX)}.
an_Noun(pyramid) --> [X], {name(X, UX),      % convert to the canonical word «pyramid»
            L = "pyramid",
            append(L, _, UX)}.
an_Noun(book) --> [X], {name(X, UX),          % convert to the canonical word «book»
             (L = "book"; (L = "brifcase"; (L = "notebook"; L = "tablet"))),
            append(L, _, UX)}.
an_Noun(table) --> [X], {name(X, UX),          % convert to the canonical word «table»
           (L = "desk"; L = "table"),
            append(L, _, UX)}.
an_NGroup(Name, char(size(Size),color(_))) -->     % Analysis of the noun group
          an_AdjSize(Size), an_Noun(Name).          % with indication of only the size
an_NGroup(Name, char(size(_),color(Color))) -->  % Analysis of the noun group
          an_AdjColor(Color),  an_Noun(Name).    % with indication of only the color
an_NGroup(Name, char(size(_),color(_))) -->       % Analysis of the noun group
           an_Noun(Name).                              % without indication of color and size
an_NGroup(Name, char(size(Size),color(Color))) -->  % Analysis of the noun group
           an_AdjSize(Size), an_AdjColor(Color),        % with indication of both color and size
           an_Noun(Name).
```

```
an_AdjNumb(N) --> [X],                  % Analysis of words for the serial number of the body
           {Y =.. [X, N], member(Y,
           [first(1), second(2), third(3), fourth(4), fifth(5)])}.
```

```
an_AdjSize(big) --> [X], {name(X, UX),       % Analysis of words for a large size
            (((L = "large"; L = "big"); L = "great"); L = "huge"),
             append(L, _, UX)}.
an_AdjSize(middle) --> [X], {name(X, UX),  % Analysis of words for a middle size
             ((L = "middle"; L = "medium"); L = "average"),
             append(L, _, UX)}.
an_AdjSize(small) --> [X], {name(X, UX),    % Analysis of words for a little size
            ((L = "small"; L = "little"); L = "weeny"),
             append(L, _, UX)}.
an_AdjColor(red) --> [X], {name(X, UX),     % Analysis of words for red color
             L = "red",
             append(L, _, UX)}.
an_AdjColor(green) --> [X], {name(X, UX),  % Analysis of words for green color
            (L = "green"; L = "virid"),
             append(L, _, UX)}.
an_AdjColor(blue) --> [X], {name(X, UX),    % Analysis of words for blue color
            (L = "blue"; L = "sapphirine"),
            append(L, _, UX)}.
an_AdjColor(brown) --> [X], {name(X, UX), % Analysis of words for brown color
            (L = "brown"; L = "fulvous"),
            append(L, _, UX)}.
an_AdjColor(white) --> [X], {name(X, UX), % Analysis of words for white color
            L = "white",
            append(L, _, UX)}.
%------------------------------------ End of the program
%                                                    © Nikolay Volchenkov. 2018
```