Software & Systems Received 04.08.17
DOI: 10.15827/0236-235X.120.583-592 2017, vol. 30, no. 4, pp. 583–592

INTERPRETATION OF THE MEANING OF NATURAL LANGUAGE PHRASES IN PROBLEM-ORIENTED SYSTEMS

V.A. Billig, Ph.D. (Engineering), Senior Researcher, Associate Professor, Vladimir-Billig@yandex.ru;
I.S. Smirnov, Master Student, dru0121@gmail.com
(Tver State Technical University, Nikitin Quay 22, Tver, 170026, Russian Federation)

Abstract. The organization of a natural language dialog with a computer is one of the most important problems in the field of artificial intelligence. The expressive power of a natural language makes it difficult to formalize and eliminate ambiguities in understanding phrase meanings.

This article considers the approach to interpreting natural language phrases based on the "Meaning–Text" theory. The key point is an intentional dialogue context limitation with a specified domain. This allows the system to conduct a more meaningful dialogue and to solve specific problems of the given domain.

The "Eliza–Student" software built upon the developed algorithms is oriented to a fairly broad subject domain, which includes a fair amount of tasks from the Unified State Exam (the Russian abbreviation is "ЕГЭ", an English analogue is SAT) in informatics.

The developed system is able to explain actions performed during the analysis of the text, which contains user questions on solving certain tasks in informatics, solve them and explain decision making in solving process. It seems that the reciprocal form of interaction seems to be the most natural in the learning process.

The analysis process is divided into a sequence of stages (preliminary, morphological, syntactic and semantic analysis). Each of them uses different models of language and subject domain. The proposed approach is based on the following ideas: abstraction from the subject area to the latest stages of analysis and focus on the result, i.e. the construction of the most probable, perhaps incomplete, representation of the meaning, despite the incompleteness of the initial information or possible errors in the analysis process. The developed algorithms can be applied to different subject domains.

Keywords: program systems, human-machine interface, natural language, tasks in informatics, morphological analysis, syntax analysis, semantic analysis, frame model, "Meaning–Text" model, dialogue with a computer, natural language interface, knowledge representation, task database, system explanation of solution.

The way people interact with computers changes with technology advancements. An interface with a large number of forms has become a common thing. But in some cases it becomes too complicated. This leads to the fact that the user needs to undergo certain training in order to interact with such system. The most progressive form of interaction with a computer system, which is devoid of such a drawback, is a natural language dialogue. Moreover, such dialogue has a fair amount of apparent advantages. For example, absence of a formalization step allows the user to make up an arbitrary request much faster.

Organization of a dialogue in natural language with a computer is one of the most popular problems related to *artificial intelligence* (AI) since 1950s. Actually, the main problem is natural language processing, i.e. extracting the sense that a user tries to convey using text and presentation it in a form suitable for further processing by a computer.

One of the first systems to interact using natural language was ELIZA developed by Joseph Weizenbaum [1]. It simulated a conversation with a psychotherapist using the active listening technique. The core of ELIZA was a set of patterns based on key words. Even now some chatterbots use similar algorithms. Another example could be PARRY [2], Eliza's "opponent", imitating the behavior of a paranoid schizophrenic. PARRY was a much more serious program implementing complicated interaction strategy. One of the most advanced systems of such class is ALICE [3], who is a several times winner of the Löbner Prize. ALICE-like programs do not limit the subject of conversations, but at the same time they are not able to conduct a meaningful dialogue involving obtaining any result or solving problems.

A natural language, unlike formal ones, was formed spontaneously. This led to emergence of ambiguities at all levels of perception, from phonetics and morphology to semantics itself. This makes the understanding of the NL in its entirety an incredibly difficult task. In connection, there has been formed a class of systems that offer using a subset of the natural language. This class includes the diagnostic system MYCIN [4], which used the questionnaire language ("yes/no" questions) as a form of communication.

This paper considers an intermediate stage between the extreme cases. By natural language we mean a problem-oriented subset of the Russian language, i.e. the part of the NL that is most likely to be used for a dialogue on a given domain.

When designing a system that could communicate on a given topic, it is necessary to introduce knowledge of the subject domain into it. Nowadays, there is a large number of knowledge representation formalisms [5], from semantic cases of C. Philmore [6] and the conceptual dependencies of R. Schenk [7] to network and logical models, for example, semantic networks and predicate logic. Despite all the diversity, there is no universal method suitable for any text.

However, nowadays there are platforms that allow building interactive systems capable of answering some questions ("Who…?", "When…?", "Where…?",

"Is … ?" and so on), which are formulated as one or several sentences. These platforms include IBM Watson [8] and Microsoft Cognitive Services [9]. They are already used to solve a large number of problems, for example, to diagnose cancer based on a patient's card. Of course, the system does not understand what a cancer is (as a doctor understands), but it can detect signs of cancer by the way it is described.

Despite the apparent intelligence of such systems, they cannot answer the question of "how", i.e. explain the course of decision-making. We believe that the ability to explain why it received such a solution and not another is the key to the operation of the systems based on expert knowledge. Explanations allow to acquire knowledge and gain experience. Therefore, we intentionally limit the range of tasks to be solved.

We choose the problem of explaining the algorithms for solving tasks of the Unified State Exam in Computer science to secondary school students as a subject domain. The scope of the problems involved includes knowledge from various areas of computer science (in particular, algorithmization): working with various units of information and different number notations, programming languages, logical problems, problems from set and graphs theories and etc. Most of them are solved quite simply, but some require a more "intelligent" approach. The other important thing is that most of the tasks are formulated in a text form. The developed system allows a user to formulate tasks either "as-is" or in the form of a question. This form of interaction is the most natural in the learning process.

The goal of this paper is to consider the approach to constructing systems with a dialogue interface in the Russian language on the example of a particular system in a given domain. Since the original system processes the Russian language, examples were translated into English. Such examples might seem wrong from the perspective of English grammar, so they should not be considered from that perspective. Russian analogues are given in the description.

The paper is organized as follows. Section 2 contains a description of the input data and a general principle of the text analysis. Sections 3–6 describe the sequential stages of the analysis. Section 7 describes the general structure of software implementation and its main components. Section 8 draws a conclusion.

Input structure and analysis principle

The most important part of an input is the text itself. The texts in question are of a small size. It is also assumed that the input text does not contain spelling or grammatical errors, i.e. belongs (The input text may not carry any meaning or it may belong to a completely different subject domain. Such texts are considered to be correct, but a relevant answer from the system should not be expected in this case.) to the considered natural language. Depending on a subject domain, an input text might contain other forms of information. In particular, exam tasks can contain formulas, tables, program texts in various programming languages and graphic information. However, in most cases such forms are rare or not used at all.

The most detailed model for describing the semantics of the Russian language is the "Meaning–Text" theory founded by A.K. Zholkovsky and I. Melchuk [10]. This model is prevalent due to the fact that it was initially oriented to computer implementation. In connection with the spread of homonymy and synonymy in a natural language, the model presents a multistep transition from a text to its meaning. Thus tier structure is used by analogy with a natural language. Most modern systems are based on the ideas of the "Meaning–Text" theory in one way or another. The linguistic processor ETAP-3 [11], which is used for a preliminary automatic syntax markup of the SinTagRus texts corpora, can be given as an example of such system.

Preliminary analysis stage

The goal of the preliminary stage is to extract additional information from the text and to form the text for further processing.

Tables and pictures are first to extract. Each of the extracted objects receives numbered alias (starting with a "table" keyword for tables and a "picture" keyword for pictures). At the current stage of system development only tables with textual information are processed. Pictures are extremely rare and require specific processing algorithms. Lists and formulas are detected using context free grammars. Lists are processed the same way as tables. Formulas receive special tag and remain in the text.

Processing algorithms of such additional information might vary depending on meaning and structure of the text. After extracting the remaining text is passed to a morphological analysis module.

Morphological analysis

As an input data for designing morphological model we used a Russian dictionary and a morphologically tagged corpora, that were provided by OpenCorpora project [12]. To define morphological analysis algorithm itself we need certain definitions.

A *word* is understood as a finite string of symbols of NL having a certain meaning in terms of NL grammar. Thus, a word means all words in the sense we are familiar with, as well as punctuation. The word can be in one of its *wordforms* or in an initial form (*lemma*).

A *sentence* is a finite sequence of words, the last of which is a punctuation.

A *sub-tag* is a morphological information unit (grammeme) represented as a string, which can have one of the fixed set of values. We use the OpenCorpora notation to denote sub-tags. For example, for a word "mom" the sub-tags are NOUN (noun), anim (ani-

mate), femn (feminine) etc. We chose 58 sub-tags (57 real and 1 fictitious) which are significant for subsequent stages of analysis.

A *tag* is a set of sub-tags for a given wordform. *Correct tag* refers to tag, which is correct in terms of NL grammar. For example, in the sentence "Mom washed a frame" the correct tag for the word "washed" is the following set: $T = \{$*VERB*(*verb*), *tran*(*transitive*), *sing*(*single*), *femn*(*feminine*), *pres*(*present*)$\}$.

Tags can be represented as constant length vectors (58 bits – by the number of sub-tags). This vector can also be represented as a number (58 bit unsigned integer). In the course of the study, it was revealed that the grammar of the Russian language allows about 500 tags using the above-mentioned number of sub-tags.

The goal of morphological analysis is to break up a text, obtained in the previous step, into sentences and then into words, as well as to associate with each word its correct tag and lemma.

The source text is divided into words using regular expressions. This approach is applicable for most cases, but it can allocate additional words (split numbers, allocating integer and fractional parts, separating abbreviations, etc.). Then, the resulting words are grouped into sentences on the basis of the following assumptions.

1. The sentence must end with a punctuation mark, namely a dot, an ellipsis, and interrogative or explanation point.
2. If a point separates two words that are numbers, then they are combined into one word.
3. The word preceding the final punctuation mark cannot have a length equal to one. Otherwise, if it is not a pronoun, it is believed to be reduction (or part of the name)
4. There should be a dictionary word (word which is in a systems' dictionary) or a number before a dot (otherwise use assumption no. 3).

The abovementioned assumptions are sufficient to handle most cases. An exception may be scientific texts or texts containing a many unknown words. Texts of tasks in informatics are correctly divided into sentences by this algorithm.

At the next stage each sentence is processed separately. The algorithm passes the sentence once from left to right. Each word of the sentence is assigned an *ambiguity class*. The *ambiguity class* of a wordform x refers to intersection of all possible tags of x. For example, for the word "washed" the ambiguity class will be the following (noun "soap" and the verb "wash"):

{*NOUN*(*noun*), *VERB*(*verb*), *inan*(*inanimate*), *nomn*(*nominative*), *accs*(*accusative*), *imp*(*imperfect*), *femn*(*feminine*), *neut*(*neutral*), *ind*(*indicative*), *sing*(*single*), *plur*(*plural*), *past*(*past*), *trans*(*transitive*)}

The function $f: X \to Y$ that associates the word form X with its ambiguity class Y is called ambiguity class model, as well as corresponding data structure. As a such data structure we use DAFSA (DAWG) implementation allowing storing pairs "string key – ambiguity class (number)".

The ambiguity class of a word can contain ambiguities from the point of view of the grammar of EH. For example, the ambiguity class, which is indicated above, contains two sub-tags that denote a part of speech. In order to build the correct tag for a given word, it is necessary to exclude some sub-tags from the ambiguity class. For this purpose sub-tags are divided into grammatical groups, except for four sub-tags, which are never excluded. This, for example, refers to the sub-tag Anph denoting an anaphoric pronoun. It appears only in pronouns, which are related to other parts of the text by the coreference relation. Sub-tags within a group are mutually exclusive.

We train one multiclass classifier for every sub-tag group. The number of classes is equal to the number of sub-tag groups increased by one (special class, meaning that a given word is absent from a given sub-tag group). Training algorithm use the SVM method and the "one-against-all" strategy. Parameter evaluation is done via grid search. Learning and tagging processes use the same algorithm of converting words into vector. The following information is used to construct the vector.

1. Correct sub-tags of 3 preceding words. These words will already be recognized because the algorithm passes the sentence from left to right. If there is no word, then its tag is a zero vector.
2. The ambiguity class for the word is recognized. This vector is the starting point for a classification process.
3. A pseudo-ending of the recognized word. Getting a real ending without using special dictionaries is quite difficult. However, even a pseudo-ending can provide a lot of useful information about grammatical attributes of a word. To obtain a pseudo-ending, it is necessary to use a Porter's stemmer. Then the resulting ending is encoded as a vector of the constant length 10 (the maximum length of pseudo-ending we obtained during experiments with real texts and Porter's stemmer).
4. A word length. The length can clearly indicate that this part of the speech is official (prepositions, conjunctions, etc.).
5. The ambiguity classes of three following words. If there is no word, then the zero vector is used.
6. Most possible tags and their probabilities (using bigrams and trigrams). When constructing ngrams, we use a word's tag instead of a word itself. The practical use of ngrams with $n > 3$ doesn't prove to be useful. As it was said above, the Russian language grammar allows about 500 tags. If grammatical constraints on the compatibility of words are taken into account, the actual size of 3-gram would be much less than 500^3.

After applying all received classifiers to a recognized word, all unnecessary sub-tags will be excluded from its entropy class, and only those that enter the cor-

rect tag for the given word will remain. Sentence analysis ends when all words are processed.

The main properties of the algorithm include.

- It combines advantages of two approaches (dictionary and machine learning). If a word doesn't have homonyms it is processed unambiguously. For other words the problem with homonymy is solved using machine learning. The accuracy we acquired is shown in table 1. The training subset included 2 thousand sentences with no homonymy. Testing was conducted on a subcorpus of 10 thousand sentences. The accuracy shown was sufficient for practical use. Currently, we plan to use bigger corpora for training purposes.
- The algorithm passes sentence once left to right. The complexity is $O(n)$, where n denotes a number of words.
- The algorithm can process unknown words. But the tagging accuracy depends heavily on the number of dictionary words.
- The lexical context is taken into account. When resolving homonymy, a window of seven words is considered (a recognizable word and 3 words on both sides). Experiments have shown that a window of this length takes into account the nearest context in the best possible way.
- The result of the algorithm is a single parsing option, unlike most systems that provide all parsing options. This greatly facilitates syntax analysis step, although in some cases (if errors are present) does not allow parsing sentence at all.

Table 1

Tagging accuracy. A training set consists of 2 thousand sentences. Testing was conducted on a corpora of 10 thousand sentences

Sub-tag group	Accuracy (%)
Part of speech	90
Verb form	96
Gender	97
Mood	99
Case	85
Person	98
Tense	98
Transitivity	96
Voice	100
Animacy	100

Here are some examples of a morphological analysis. The sentences were taken from the text of tasks on Computer science from the FIPI website (fig. 1, 2).

Syntax analysis

Traditionally, the goal of syntax parsing is to determine if the input string belongs to a given language. For computer linguistics the more important result of analysis is a syntax tree. To construct such trees we decided to abandon machine learning methods and to use a more traditional grammar approach. Despite the fact that machine learning leads to more robust results, the choice fell on a traditional approach because it is easier to explain, supplement and correct.

The free order of words in the Russian language does not allow the use of grammars suitable for English [13–15]. For Russian the most popular approach is to use dependency grammars. Such grammars make it possible to construct non-projective dependencies. Such dependencies constitute a rather large part of corpora (approximately 10 %).

In general, the dependency grammar is specified by a set of rules. Such rules are called surface syntactic relations (SSR) within the "Meaning–Text" theory. SSR restricts grammatical attributes, position and other dependency types of main and dependent words. This paper considers a grammar build upon SSR set given by I.A. Melchuk. We selected 32 most important syntactic relations from 42 given ones (some were merged). The remaining SSRs represent rare language constructions that are unlikely to be used in a dialogue on the tasks in informatics.

The core of the "Meaning–Text" theory is an explanatory combinatorial dictionary. The construction of such dictionary for a given subject domain is a time-consuming task, so we decided to abandon it. Instead we divide SSR set into two features:

1) "Power": in absence of an explanatory combinational dictionary some SSRs can be built with errors (for example, construction inconsistent definitions). All SSRs are divided into:

a. "Strong": the ones that are built correctly all the time.

b. "Weak": the ones that can be built with errors. Building algorithms of such SSRs are built upon different heuristics.

how [Union]
to [Particle]
convert [Perfect, Infinitive, Transitive]
the [Particle]
binary [Accusative, Neutral, Single, Adjective]
number [Inanimate, Accusative, Neutral, Single, Noun]
11011 [Number]
into [Accusative, Preposition]
decimal [Accusative, Feminine, Single, Adjective, Qualitative]
notation [Inanimate, Accusative, Feminine, Single, Noun]
? [Punctuation]

Fig. 1. Morphological analysis of the sentence "How to convert the binary number 11011 into decimal notation?"

(x1->x2)&(x3|x4)|y=1[Formula]
how [Conjunction]
many [Particle]
solutions: решение [Inanimate, Genitive, Neutral, Plural, Noun]
does [Indicative, Single, Verb, Present]
the [Particle]
logical [Accusative, Neutral, Single, Adjective]
equation [Inanimate, Accusative, Neutral, Single, Noun]
have [Imperfect, Indicative, Single, Verb, ThirdPerson, Present, Transitive]
?: ? [Punctuation]

Fig. 2. Morphological analysis of the sentence "How many solutions does the logical equation (x1-> x2)&(x3|x4)|y=1 have?"

2) Use case: this feature characterizes the dependency with respect to use within a segment or between segments. A segment is defined here as a sequence of words in the sentence between punctuation marks. In this scope SSRs are divided into:

a. Internal
b. External

One SSR can have multiple use cases with different building algorithms as opposed to power. Every use case is treated like a separate SSR.

Thus, all SSRs can be divided into 4 groups. In this paper, these groups are called *working sets*. Working sets are represented in table 2.

Table 2

Working sets

Use case	Power	
	Strong	**Weak**
Internal	Strong internal	Weak internal
External	Strong external	Weak external

Each working set can be considered as a separate grammar. The purpose of splitting into sets was that it allows you to initially build only those relationships that do not contain errors. It is also easier to maintain determinism in sets, rather than in the entire grammar. This property is necessary for the correct operation of the parsing algorithm.

Each sentence is parsed separately. First, the algorithm splits a sentence into segments and constructs internal strong dependencies using the appropriate working set and the algorithm developed by M. Covington [16]. This will result into one or more syntax trees for every segment. If there are several trees, then the merge algorithm (The merge algorithm is a modification of the algorithm developed by Michael Covington [16]. The original algorithm in the process of work maintains two lists of trees: headless (tree roots) and all (all vertices). The modification is to simulate the algorithm, starting with a certain step. To do this, the "headless" and "all" lists are passed to the algorithm as parameters instead of initializing. Let it be two trees which must be merged. Then we put a reference to the first tree in the "headless" list and references to all the vertices from the second one to the "all" list. If the "headless" list does not contain any elements after running the modified algorithm with these lists, then the first tree became the subtree of the second one, and the merge was successful. Otherwise, it is concluded that the merge is not possible.) with a weak internal working set. Each procedure iteration applies the merge algorithm to all trees in pairs. As soon as at least one merge occurs, two merged trees are replaced by the merge result and the algorithm moves to the next iteration. The number of trees decreases by 1 at each iteration where the merge occurred. The algorithm finishes its work when the number of trees on the current iteration does not change.

The same procedure is used to construct external dependencies (first with a strong set, then with a weak set). The result of the algorithm is the one surface dependency tree.

However, this tree is quite difficult to use for further analysis, because the number of connections is too big. The tree of deep dependencies (DD-tree) is of great interest. Using such trees, one can already see a predicate (action), an object over which the action is performed, an oblique semantic object, a semantic subject and etc. A formal definition of DD trees is the subject of controversy. This paper uses the definition given in [17]. The surface representation is translated into the deep by a simple recursive procedure. We omit the details of this procedure, because it is of an experimental nature and requires further development.

Instead, we give some examples of the surface and deep structures obtained (fig. 3–6).

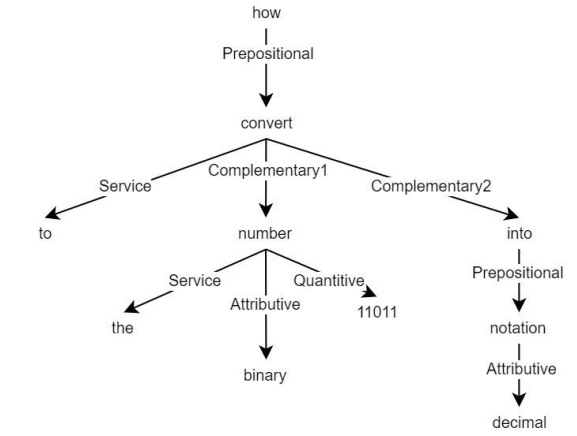

Fig. 3. Surface dependency tree of the sentence "How to convert the binary number 11011 into decimal notation?"

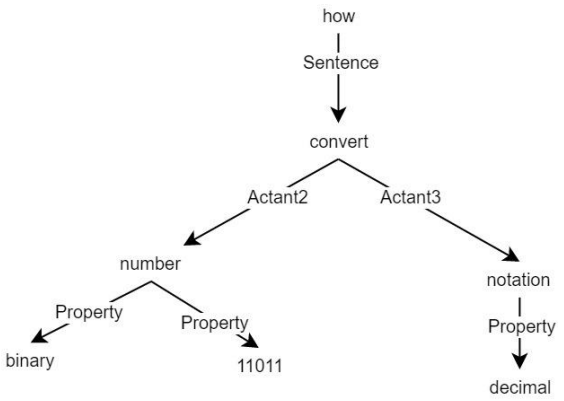

Fig. 4. Deep dependency tree of the sentence "How to convert the binary number 11011 into decimal notation?"

The set of DD-trees are passed to the semantics module.

Semantic analysis

In this paper semantic analysis refers to the problem of recognizing in an input text (a set of DD-trees) one of the tasks of the subject domain. If the task is found, system tries to extract its parameters and solve it. Otherwise there is a message for a user showing that it is necessary to reformulate the question.

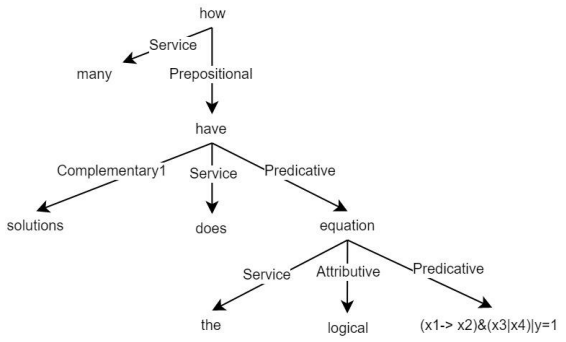

Fig. 5. Surface dependency tree of the sentence "How many solutions does the logical equation (x1-> x2) &(x3|x4)|y=1 have?"

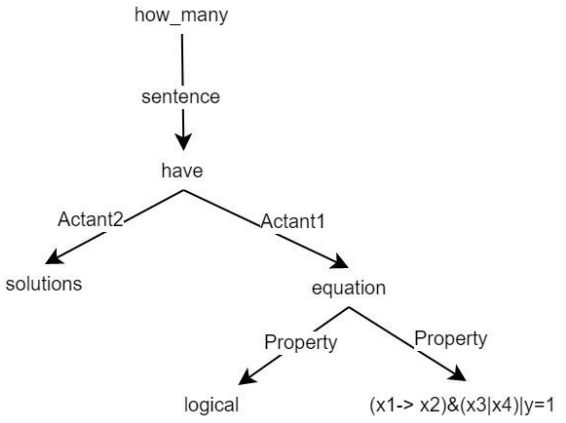

Fig. 6. Deep dependency tree of the sentence "How many solutions does the logical equation (x1-> x2) &(x3|x4)|y=1 have?"

For a dialogue to be constructive, the system must have a knowledge base in a corresponding subject domain. In this case, we decided to use a frame model, as it is suitable for modelling of static subject domains and allows to model it in the form of a very compact structure. At the top of the hierarchy is frame-class "Task", which defines the abstract task in the given domain. The names of the slots and their description are shown in table 3.

For each task class (logical equations solving, finding parameters of program written in one of programming languages, etc.) there is a corresponding frame-template which inherits the frame "Task". Each frame-template sets its own value for the "Parameters" set, defining types and possible values of task parameters for a given class of tasks. Each such frame-template allows generating an unlimited number of instance frames corresponding to specific tasks with given parameters, result, explanations, etc.

The slots from table 3 correspond to the concepts of the domain, except for the ones associated with the templates. The templates within the framework of this paper represent some form of a case analysis for solving a particular problem of a semantic analysis. They specify the expected structure or parts of the structure of the DD trees obtained in the parsing phase.

Table 3

Slots of frame-class «Task»

Slot name	Description
Identifier	This slot represents the string identifier of the task
Parameters	A list of task parameters. Each child frame contains its own list of parameters. Parameters can have default values
Input	Represents a list of DD trees obtained as a result of the previous step. It is set at the beginning of the work with the frame
Patterns	Templates, the purpose of which is to find matches in the "Input" slot. Templates allow to keep some values for subsequent assignment to other slots, in particular to the slot "Parameters"
Match	Checks the list of templates to match the input. If at least one template has found matches, then the value of this slot is set to True, and the value of the slot "Matching pattern" becomes equal to a specified template
Matching pattern	It corresponds to the syntactic pattern with which a match is found the input discourse. Sets the value of the "Parameters" slot
Result	The result of the task. Requires the value of the slot "Matching pattern". Calls the corresponding task from the task module. Sets the value of the "Explanations" slot
Explanations	Step-by-step textual explanations of the task soling algorithm. Requires the value of the "Result" slot. The default value is "empty string"

The following templates are supported.

- Lemma matching pattern. This pattern checks if tree vertex contains the same lemma as the pattern provides. In order to avoid describing a large number of patterns, synonymous series are defined. Each series contains lemmas that are understood by system in the same way.
- Dependency matching pattern. This pattern checks if two given tree vertexes are connected using the specified dependency (DD-tree definition used allows only 10 dependency types).
- Branching pattern. This pattern allows analyzing a tree width structure.
- Chained pattern. This pattern allows analyzing a tree height structure.
- Skip pattern. This pattern allows skipping any amount of vertexes in case if only a partial information is needed (for example if we are looking for a number to translate from one notation to another one).

The algorithm is sequential checking of each frame-template to match the set of input DD-trees and generating an instance frame for a corresponding task. An instance frame contains all the necessary information to form a request to a solution and explanation modules for obtaining a relevant response. A part of the frame model structure is shown in figure 7.

Below there are some answers to user questions.

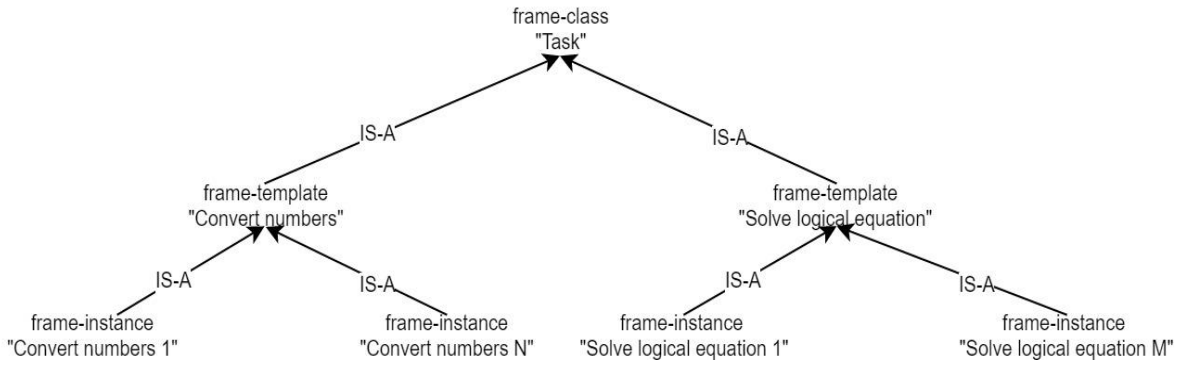

Fig. 7. The fragment of a frame knowledge base

Implementation

In order to achieve our goal of constructing a software system with a natural language interface in a specified subject domain, we developed the "Eliza–Student" system in the C# language. At the moment the system contains 10 projects that solve various tasks: language models generating, processing of the corpora, replenishment of expert knowledge, problem solving, etc. The total amount of the program code is approximately 5 thousand lines. The main components of the system are shown in figure 8.

The ElizaInterface component represents the program interface (figures 9, 10). The interface allows entering and displaying a text, customizing dialogue settings, as well as using interactive Help.

The OGESolver component allows solving and explaining tasks in Computer science. This component is developed using the "factory" design pattern, which allows you to quickly and easily expand the tasks set. The main class here is AlgorithmFactory, which allows creating an instance of a task by ID and parameters that contain an algorithm for solving and explaining the problem.

The library Eliza is responsible for processing requests in the natural language. In the morphological analysis phase, the tags are modeled using the Tag flags enumeration. This allows you to check whether the tag contains the set of sub-tags, and add and remove subtags almost instantly (for several logical operations). The stages of the preliminary and morphological analysis are performed by the class MorphologyModel. In addition, this class allows you to build a new morphological model, using the corpora and dictionary files. The LIBSVM [18] library is used as the implementation of support vector machines.

The main classes in the parsing phase are SyntaxModel, which contain the basic methods for building and merging dependency trees, and DependencyGrammar, which contain the working sets described in the syntax analysis section that are initialized when the program is started. Each set is represented by a set of

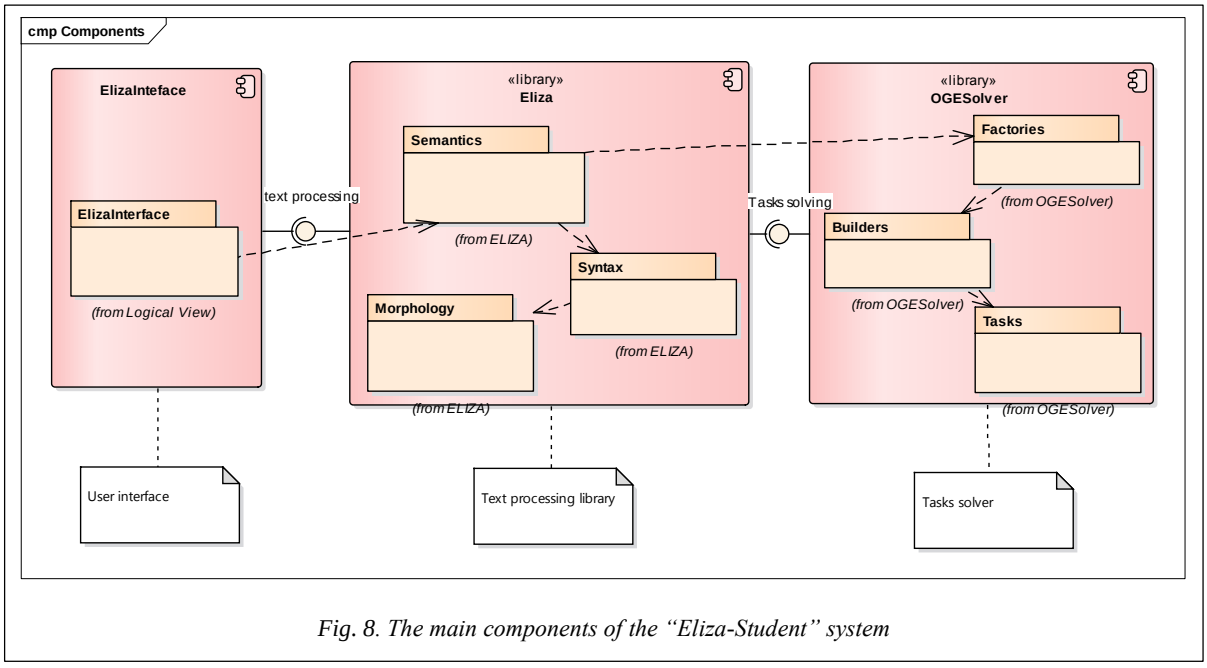

Fig. 8. The main components of the "Eliza-Student" system

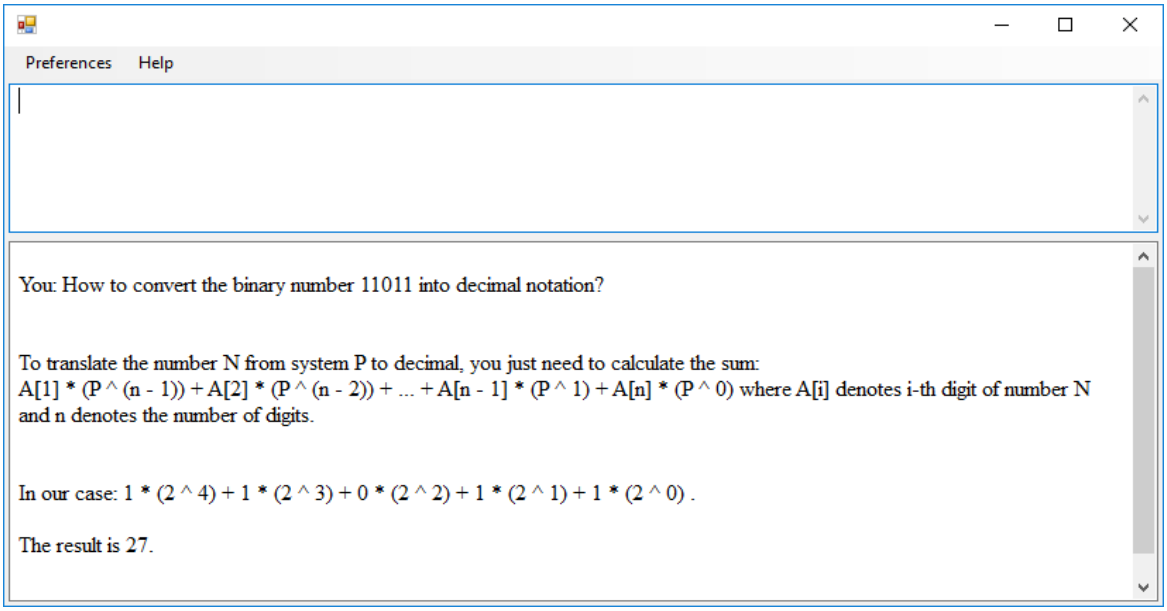

Fig. 9. The system answers the question "How to convert the binary number 11011 into decimal notation?"

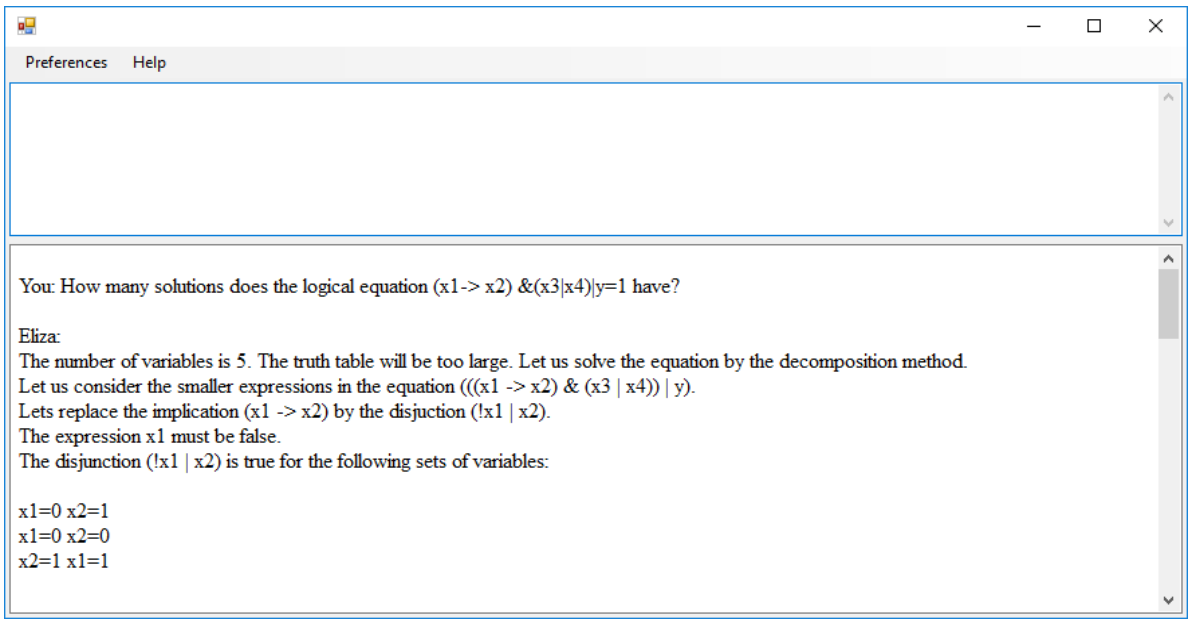

Fig. 10. The fragment of the system's answer to the question "How many solutions does the logical equation (x1-> x2) &(x3|x4)|y=1 have?"

rules, each of which is an instance of one of the classes inherited from the AbstractSSR class. Each of these classes contains a simple method for constructing a particular SSR (usually the complexity of such methods is $O(1)$) that looks like this:

```
protected override bool TryBuildRelation(Tree<Lexem, SurfaceRelationName> first, Tree<Lexem, SurfaceRelationName> second, out Tree<Lexem, SurfaceRelationName> head)
    {
        Lexem f = first.Key;
        Lexem s = second.Key;
        head = first;
        if ((s.Tag & (Tag.NounLike | Tag.Noun)) != 0 && (s.Tag & (Tag.Instrumental | Tag.Genitive)) != 0 && (s.Tag & Tag.Animated) != 0)
        {
            if (((f.Tag & (Tag.Gerund | Tag.Infinitive | Tag.Participle | Tag.ShortParticiple)) != 0) || (f.Tag & (Tag.Noun | Tag.Noun)) != 0)
            {
                head.AddChild(second, SurfaceRelationName.Agentive);
                return true;
            }
        }
        return false;
    }
```

Such methods check the morphological tags of tree roots. In rare cases, it is required to check for the existence of a certain dependency or heirs in the tree, for example, for constructing a tree for the phrase “each of which”. It is extremely difficult to establish all the links on the first pass, so we see the need to separate the grammar into working sets and perform multipass analysis.

At the stage of semantic analysis, the Semantics-Model class analyzes the belonging of the input text of one of the tasks, passing through all frames and setting the value of the “Input” slot. If the value of the “Match” slot has now become true, the values of the “Result” and “Explanations” slots, that contain all the information necessary for the user, are returned.

Conclusion

The described software system exists in the form of a prototype, which performs the basic functions and is able to conduct a constructive dialogue about some tasks. The developed algorithms in the early stages of text analysis (morphological and syntactic analysis) can be used to process texts from other subject domains with minor modifications, if any. The semantic analysis is the most problematic part. We tried to build the system in such a way that it was independent of a particular subject domain as much as possible. At the next stage we plan to isolate a purely linguistic part of the meaning extracting process (which can be extracted from the text itself using a lexical context) from extralinguistic one (including knowledge of the problem area, inference, etc.). For this purpose we plan to use the LRA semantics [17].

The constructed prototype proves the consistency of the general idea. Expert knowledge is much deeper than in systems that respond to keywords and their order. Although this approach requires a lot of work to replenish the expert knowledge, even with a small knowledge base, the system is able to “surprise” and answer the question “logically” while doing a qualitative work to solve the problem behind the scenes. In addition, it is able to explain the course of solving the problem. This important feature distinguishes it favorably among systems that receive the result of solving a complex problem without providing explanation. Explanation allows one to gain experience and acquire new knowledge. We believe that this approach is the future.

References

1. Weizenbaum J. *Computer power and human reason: from judgment to calculation.* W.H. Freeman and Company Publ., 1976, 300 p.
2. Güzeldere. Dialogues with colorful personalities of early AI. *Stanford Humanities Review*. 1995, no. 4, pp. 161–169.
3. *ALICEBOT*. Available at: http://www.alicebot.org/ (accessed March 3, 2017).
4. Buchanan B.G., Shortliffe E.H. *Rule Based Expert Systems: The MYCIN Experiments of the Stanford Heuristic Programming Project*. Addison-Wesley Publ., ed. Reading, 1984.
5. Batura T.V. Semantic analysis and meaning representation method in computer linguistics. *Programmnye produkty i sistemy* [Sofrware & Systems]. 2016, no. 4, pp. 45–57 (in Russ.).
6. Filmore C.J. The Case for Case. *Universals in Linguistic Theory*. E. Bach, C.J. Filmore (Eds.). London, Holt, Rinehart ad Winston Publ., 1968, pp. 1–25.
7. Schenk R.C., Goldman N.M., Rieger III C.J., Riesbeck C.K. *Conceptual information processing*. Norh-Holland Publ., 1975. (Russ. ed.: Moscow, Energiya Publ., 1980, 360 p.).
8. *Watson*. Available at: https://www.ibm.com/watson/ (accessed March 3, 2017).
9. *Cognitive-Services*. Available at: https://azure.microsoft.com/ru-ru/services/cognitive-services/ (accessed March 3, 2017).
10. Melchuk I.A. *Opyt teorii lingvisticheskikh modeley "Smysl–Tekst"* [Experience of "Text–Meaning" lingustic models]. Moscow, Shkola "Yazyki russkoy kultury" Publ., 1999, 2nd ed., 346 p. (in Russ.).
11. *Computer lingustics laboratory*. Available at: http://cl.iitp.ru/ru/etap3/ (accessed March 3, 2017).
12. *Opencorpora*. Available at: http://www.opencorpora.org/ (accessed March 3, 2017).
13. Chomsky N. *Syntactic Structures*. Mouton, Walter de Gruyter Publ., 1957, 117 p.
14. Winograd T. *Understanding natural language*. NY, Academic Press, 1972, 294 p.
15. Woods W.A. Transition network grammars for natural language analysis. *Communications of the Association for Computing Mchinery*. 1970, no. 10, pp. 591–606.
16. Convington M.A. A Fundamental algorithm for dependency parsing. *Proc. 39th Annual ACM Southeast Conf.* 2001, pp. 95–102.
17. Dikovsky A. *Linguistic↔Rational Agents' Semantics*. Jour. of Logic, Language and Information, pp. 1–97. Available at: https://doi.org/10.1007/s10849-017-9258-y (accessed 10 June, 2017).
18. *LIBSVM – A Library for Support Vector Machines*. Available at: https://www.csie.ntu.edu.tw/~cjlin/libsvm/ (accessed 10 June, 2017).

УДК 004.512.4
DOI: 10.15827/0236-235X.120.583-592

Дата подачи статьи: 04.08.17
2017. Т. 30. № 4. С. 583–592

ИНТЕРПРЕТАЦИЯ СМЫСЛА ФРАЗ ЕСТЕСТВЕННОГО ЯЗЫКА В ПРОБЛЕМНО-ОРИЕНТИРОВАННЫХ СИСТЕМАХ

***В.А. Биллиг**, к.т.н., доцент, vladimir-Billig@yandex.ru;*
***И.С. Смирнов**, магистрант, dru0121@gmail.com*
(Тверской государственный технический университет, наб. Аф. Никитина, 22, г. Тверь, 170026, Россия)

Организация диалога с компьютером на естественном языке является одним из важнейших направлений в области искусственного интеллекта. Мощь и выразительная сила естественного языка затрудняют его формализацию и устранение неоднозначностей в понимании смысла фраз.

В данной статье рассматривается подход к интерпретации смысла фраз на основе теории «Смысл–Текст». Ключевым моментом является намеренное ограничение контекста диалога заданной предметной областью. Это позволяет системе вести более содержательный диалог и решать конкретные задачи из заранее определенного множества.

Построенная на основе разработанных авторами алгоритмов программная система «Элиза–Школьник» ориентирована на достаточно широкую предметную область, включающую множество задач, предлагаемых на едином государственном экзамене по информатике.

Система способна объяснять свои действия по анализу текста фраз, содержащих запросы на решение тех или иных задач, решать запрашиваемые задачи и давать объяснения хода решения. Форма взаимодействия «вопрос–ответ» представляется наиболее естественной в процессе обучения.

Процесс анализа разбивается на последовательность этапов (предварительный, морфологический, синтаксический и семантический анализ), на каждом из которых используются различные модели языка и предметной области. Предлагаемый подход базируется на следующих идеях: абстрагирование от предметной области до самых поздних этапов анализа и направленность на результат, то есть построение наиболее вероятного, может быть, неполного, представления смысла, несмотря на неполноту исходной информации или возможные ошибки в процессе анализа. Разработанные алгоритмы могут быть применены и к другим проблемным областям.

***Ключевые слова**: программные системы, человеко-машинный интерфейс, естественный язык, задачи по информатике, морфологический анализ, синтаксический анализ, семантический анализ, фреймовая модель, модель «Смысл–Текст», диалог с ЭВМ, текстоориентированный интерфейс, представление знаний, БД задач, система объяснения решения.*

Литература

1. Weizenbaum J. Computer power and human reason: from judgment to calculation. W.H. Freeman and Company Publ., 1976, 300 p.

2. Güzeldere. Dialogues with colorful personalities of early AI. Stanford Humanities Review. 1995, no. 4, pp. 161–169.

3. ALICEBOT. URL: http://www.alicebot.org/ (дата обращения: 3.03.2017).

4. Buchanan B.G., Shortliffe E.H. Rule Based Expert Systems: The MYCIN Experiments of the Stanford Heuristic Programming Project. Addison-Wesley Publ., ed. Reading, 1984.

5. Батура Т.В. Семантический анализ и способы представления смысла текста в компьютерной лингвистике // Программные продукты и системы. 2016. № 4. С. 45–57.

6. Filmore C.J. The Case for Case. Universals in Linguistic Theory. E. Bach, C.J. Filmore (Eds.). London, Holt, Rinehart ad Winston Publ., 1968, pp. 1–25.

7. Schenk R.C., Goldman N.M., Rieger III C.J., Riesbeck C.K. Conceptual information processing. Norh-Holland Publ., 1975. (Russ. ed.: Moscow, Energiya Publ., 1980, 360 p.).

8. Watson. URL: https://www.ibm.com/watson/ (дата обращения: 3.03.2017).

9. Cognitive-Services. URL: https://azure.microsoft.com/ru-ru/services/cognitive-services/ (дата обращения: 3.03.2017).

10. Мельчук И.А. Опыт теории лингвистических моделей Смысл–Текст. М.: Школа «Языки русской культуры», 1999. 346 с.

11. Computer lingustics laboratory. URL: http://cl.iitp.ru/ru/etap3/ (дата обращения: 3.03.2017).

12. Opencorpora. URL: http://www.opencorpora.org/ (дата обращения: 3.03.2017).

13. Chomsky N. Syntactic Structures. Mouton, Walter de Gruyter Publ., 1957, 117 p.

14. Winograd T. Understanding natural language. NY, Academic Press, 1972, 294 p.

15. Woods W.A. Transition network grammars for natural language analysis. Communications of the Association for Computing Mchinery. 1970, no. 10, pp. 591–606.

16. Convington M.A. A Fundamental algorithm for dependency parsing. Proc. 39th Annual ACM Southeast Conf. 2001, pp. 95–102.

17. Dikovsky A. Linguistic↔Rational Agents' Semantics. Jour. of Logic, Language and Information, pp. 1–97. Available at: https://doi.org/10.1007/s10849-017-9258-y (дата обращения: 10.07.2017).

18. LIBSVM – A Library for Support Vector Machines. URL: https://www.csie.ntu.edu.tw/~cjlin/libsvm/ (дата обращения: 10.07.2017).