ITMO University

Hangzhou Dianzi University

Program SEMANTIC

Release 1.5

USER MANUAL

Prof. Igor Bessmertny

Last update 13.03.2020

TABLE OF CONTENT

1. Purpose and conditions of application of the program

The program is designed to represent and visualize knowledge in the form of semantic networks, as well as to access knowledge bases using a graphical interface and query language.

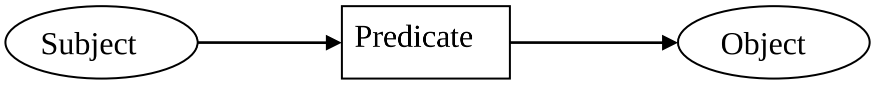
The program can be used in the training system mode, in which the knowledge base contains an implicit hypothesis, for example, a diagnosis, and the user, asking questions, must identify this hypothesis, spending a minimum of questions. In addition, the program can work in the expert system mode, comparing the available facts with the rules in the knowledge base.

To run the program requires a personal computer running Windows XP. There are no special requirements for resources. To store the program, you need at least 16 MB on your hard drive.

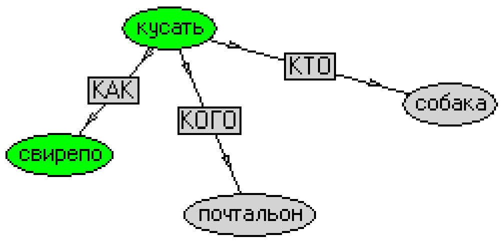
The program is designed to teach students the basics of building semantic networks in the framework of the discipline "Artificial Intelligence". The program is written in VISUAL PROLOG V 7.2 Personal Edition (www.pdc.dk). This program is intended solely for educational purposes. The use of this program for commercial purposes is not allowed.

2. Characteristics of the program

The program provides the creation and use of semantic networks, as one of the ways of representing knowledge. In this version of the program, networks with binary relations are supported, i.e. connecting exactly two concepts, one of which is the subject, the second - the object. The triplet subject-predicate-object forms a fact.



One and the same concept can be present in several facts, which determines the network structure. The program also allows the creation of semantic networks based on Rastier graphs (graphs with a verb in the center), as well as a combination of ordinary relational graphs with Rastier graphs. Rastier graph does not contain information about the relations between subjects and objects, but about processes (actions). The following is a commonly used example of a description using Count Rastier of the following event: A dog fiercely bit a postman. The network is built around the verb BITE. The agent (subject) of biting is a dog, the object is a postman. UGLY - the manner of biting.



The advantages of Rastier graph lies in the possibility of establishing properties related not to an object or subject, but to a process. UGLY is not a property of a dog, but a property of the postman's biting process.
The program allows you to create semantic networks in different languages, including multilingual networks. The text user interface allows you to extract facts from the knowledge base in a simplified natural language. The modular structure of knowledge representation, i.e. placement of different subject areas in different files.

The visualization of knowledge is provided using a graphical representation of the semantic network. The graphical user interface allows you to expand the graph in depth and width, as well as roll back, change the topic of the dialogue, i.e. manage context. The context hereinafter refers to a set of facts already extracted from the knowledge base.
To simplify the debugging of created semantic networks, all program actions are logged using messages in the “Messages” window.

3. Input Data

The knowledge bases of semantic networks are stored in text files and contain knowledge in the syntax of the Prolog language, since they are loaded into the program as a fact base by the predicate consult. One knowledge base corresponds to one subject area. File names are not restricted. You can edit the database file using any text editor. However, if you use the editor as part of the Prolog compiler, this will allow you to detect errors using the capabilities of the Prolog debugging environment. Then, for the editor as part of SWI-Prolog, it is advisable to provide the extension “.pl”, and for Visual Prolog, “.pro”.

Each knowledge base file describing a subject area may contain links to ontology files, which store term dictionaries and rules, with the help of which new facts can be extracted from facts. In addition, the knowledge base file may contain links to other knowledge base files, i.e. other subject areas. In this edition, in order to simplify the writing of paths, all files used by the program should be stored in the same directory.

Within the same subject area, the uniqueness of naming concepts should be respected. With an increase in the number of facts, this problem can complicate the addition of new facts, since it is necessary to observe the uniqueness of names and impair the readability of knowledge. If the knowledge base becomes too cumbersome, it is advisable to split the file into several knowledge bases that are placed in separate files. Each file corresponds to a separate dialogue topic. This technique allows you to switch from one topic of the dialogue to another in the same way as in a real conversation. Such a switch will be called a context change.

3.1. File of the Knowledge Base

The knowledge base file may have an arbitrary name. This file may contain the following entries:

Name	Purpose	Format	Example
t	Name of the main topic	t(<subject>).	t(″Jack″).
pic	The name of the startup image in BMP format. Optional.	pic(<file name>).	pic(″mailman.bmp″).
onto	The name of the ontology file.	onto(<file name >).	onto(″common_eng.pro″).
e	External link description	e(<subject >,< file name >).	e(″engine″, ″engine.txt″).
f	Description of fact	f(<subject >, <predicate>, <object>).	f(″hare″, ″is_a″, ″mammal″).
hypo	Hypothesis description	hypo(<subject >).	hypo(″anaemia″).

The following are explanations of these descriptions.

All entries in knowledge base files obey the Prolog language predicate syntax. Each expression begins with a name starting with a lowercase Latin letter. Arguments are placed in brackets, separated by commas. The expression ends with a period.
The fact predicate f is a basic element of the knowledge base. All elements of the triple must be enclosed in quotation marks (the ″ sign, but not the “or“). It is not allowed to use spaces and separator characters in them, if you intend to use them in queries, since query parsing is performed as parsing text into words. Spaces between separators are allowed: f (″ hare ″, ″ one_of ″, ″ mammal ″).

3.2. Ontology File

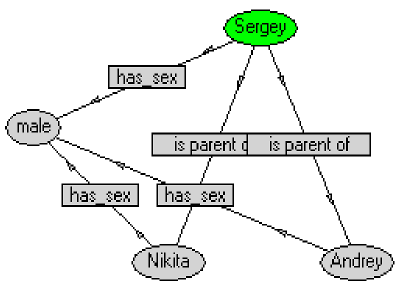
The following types of records can also be located in the ontology file.

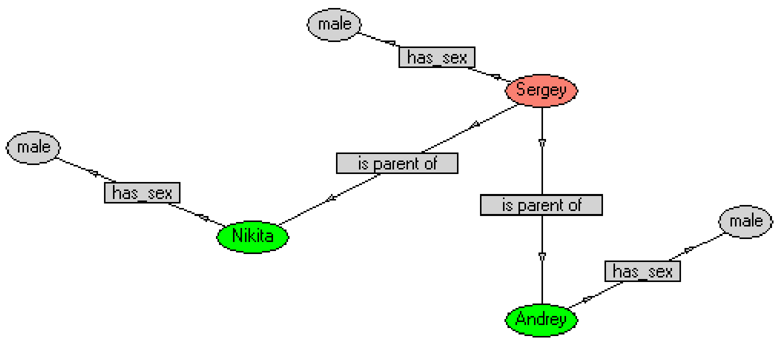
Name	Purpose	Format	Example
c	Class description	c(<word>).	c("person"). c("everybody"). c("thing").
o	Fact. The same as f(...) in the knowledge base file.	o([<субъект>,<предикат>, <объект>]).	d(["everybody","a_kind_of","thing"]).
d	Description of objects and subjects (dictionary)	d([<word >,< word >,…, < word >]).	d(["car","автомобиль","voiture"]).
p	Description of predicates	p([<word >,< word >,…, < word >]).	p(["ERG","КТО","Агент"]).
pr	Description of properties	p([<word >,< word >,…, < word >]).	p(["has_name"]). p(["мужской”, “male"]).
g	Preferred communication orientation for better graph readability	g(<direction>,[< word >,< word >,…, < word >]).	g(down,["is_parent", ”is_grandparent", "is_uncle","is_father"]). g(side,["is_cousin", "is_sibling"]). g(up, ["is_a", "is_child"]).
j	List of ignored words (junk).	j([<word >,< word >,…, < word >]).	j(["the", "a", "an", "one", "your"]).
q	List of question words (questions);	q([<word>,< word >,…, < word >]).	q([“кто”,"who",]). q(["what", "when", "why", "where", "how"]).
r	Rule Description Predicate	r(<cause>),<consequence>),	r([t("?x", "brother", "?y")], [t("?x", "rekative", "?y")]).

A description of class c is necessary in order to distinguish a class from an instance.

An entry of type o is used in the same way as an entry of f in the knowledge base file. The need to assign another name is due to the limitations of the Prolog language. Since the ontology file is intended to store knowledge at the level of classes, rather than instances, a record of type o describes the relationship between classes with class properties.

If synonyms of terms are used as part of the semantic network, they must be described in records of types d and p. A record of type d describes the synonyms of objects and subjects (dictionary), and a record of type p describes synonyms of predicates. Records of type pr must be created even in the absence of synonyms, since the graph for properties is constructed differently than for relations. If you do not specify a property in the pr entry, then the same properties of all objects will be reduced to one vertex, for example, as shown below. In the left picture, the properties are reduced to one vertex, in the right - not.





In addition, the identification of properties is useful for filtering the graph, if the graph becomes too dense, it is advisable to exclude the display of properties on it, leaving only the relationship. A record of type pr can include both the name of the property and the value, for example, “has_sex” or “male” and “female”.

All synonyms of one term should be placed in one list. Using dictionaries makes it relatively easy to make a semantic network language-independent. In this case, the first element of each list will always be used for display on the graph.

Records of types q (questions) and j (junk) are used to process queries to the knowledge base in a simplified natural language. A list of interrogative words is needed to identify the interrogative sentence, and a list of ignored words is needed to bring the question asked into a form suitable for unification with facts. Thus, in particular, articles and prepositions are ignored.

Attention:
In this version of the program, only one list of synonyms is allowed for each concept. If in different ontologies connected to the knowledge base there will be a record of type d or p for a given concept, the program uses only one of them. For example, p ([“ISA”, “is_a”] is indicated in one ontology file, and p ([“one_of”, ”ISA”] in another), then the program will use only one of the lists, depending on the sequence onto d records of the knowledge base file.

3.3. Rules if the Knowledge Base

Rules allow you to establish patterns on the basis of which you can obtain new knowledge. The rules use variables that are assigned values in the process of unification with facts. Due to the fact that external Prolog files do not allow the use of variables, they are specially named here. Variables must be represented by text constants starting with a question mark, for example, ″? X ″, ″? Class ″. The rule consists of two lists of triplets. The second list contains new knowledge that becomes true if the facts from the first list are valid. Example:
r([t(″?x″, ″parent″ , ″?y″), t(″?y″, ″parent″ , ″?z″), t(″?z″, ″sex″ , ″male″)], t(″?z″, ″внук″ , ″?x″)]).

This entry is equivalent to the rule in Prolog:

```
grandchild( X, Y )  :-  parent( Y, Z ), parent( Z, X ), sex( X, male ).
```

Despite the fact that the Semantic program is written in Prolog, it is impossible to use such a rule directly, since it can only be written to the compiled text of the program, and not to an external knowledge base. Only the facts are allowed to be stored in the external database of the Prolog.

In cases where it is required to establish the identity / difference of two objects, the rule should use the special predicate “differs”. The following example shows a rule for a brother or sister relationship:

```
r( [t("?x","is_
parent","?y"),t("?x","is_parent","?z"),t("?y","differs","?z")],
   [t("?y","is_sibling","?z")] ).
```

If negation is required in the rule, then the construction n (<subject>, <predicate>, <object>) is used.
This construction is similar to negating the relation t (<subject>, <predicate>, <object>).
The following is an example of a rule that describes the stepfather / stepmother -> stepson / stepdaughter relationship.

```
r( [t("?x","is_parent","?y"),t("?x","is_spouse","?z"), n("?z", "is_parent", "?y")],
   [t("?z","is_step-parent","?y")] ).
```

Comment. Use negative predicates with caution. Firstly, this contradicts the Open World Assumption, according to which the lack of information about a fact in the knowledge base does not mean a denial of this fact. The above example demonstrates this. It is understood that? Z is not the parent of? Y, if there is no fact that he / she is the parent. The correct definition of this relationship can be written as follows:

```
r([t("?x","is_parent","?y"),t("?x","is_spouse","?z"),t("?a","is_parent","?y"), t("?a","differs",
"?z")],    [t("?z","is_step-parent","?y")] ).
```

As you can see, a negative predicate was not needed. Secondly, a negative fact may require a search across the entire knowledge base. Another incorrect definition of an orphan is given below:

```
r( [n("?x", "is_parent", "?y")],[t("?y", "is_a”, “orphan")]).
```

According to this rule, any object for which there is no “is_parent” relationship will be considered an orphan, and to derive from this rule, a search will be performed on all the facts of “is_parent”.
More examples of rules are in the Examples folder.

4. Types of Relation in the Semantic Network

An element of the semantic network [1] is a triplet of the form (Subject - Predicate - Object). Any object mentioned in one triplet can be an object or a subject in other triplets. Such connections form a network. The following types of relationships are distinguished:

1) Hierarchical;
2) Functional;

3) Quantitative;
4) Spatial;
5) Temporary;
6) Attributive;
7) Logical;
8) Linguistic.

This list is not exhaustive and may be supplemented.

4.1. Hierarchical relationship

Hierarchical relationships arise most often. These include:

1) **ISA** classification relation (from English “is a”). It is said that a multitude (class) classifies its specimens (for example, “Socrates is a man”). This relationship is sometimes referred to as “member of”. In Russian, it can be called "one_of", "is" (singular) or "essence" (plural). The inverse relationship is “example of” or “example”. In relation to the ISA, the subject is always an instance, and the object is a class.
2) The relationship between a plurality and a subset of **AKO** (“a kind of”), for example, “Masters is a subset of students”. The difference from the ISA relationship is that the classification is one-to-many, and the subset is many-to-many. In Russian - a "subset". In relation to AKO, both the object and the subject are classes.
3) The ratio of **meronomy** is the relation of the whole to the part (“has part”). Meronym - an object that is part of another object. The relation of holonymy is the relation of the part to the whole (“is a part”). The hand is a holonym for the body. The body is a meronym for the hand. In relation to meronymy, subject and object are either both classes or both instances.

4.2. Supportive Relationships

The following types of relationships are often used in semantic networks:

1) functional relationships (“produces”, “influences”, ...);
2) quantitative (“more”, “less”, “equal”, ...);
3) spatial (“far from,” “close to,” “for,” “above,” “below,” “above,” ...);
4) temporary (“earlier”, “later”, “simultaneously with”, ...);
5) attributive (“have a property”, “have a value”, ...);
6) logical (“and”, “or”, “not”).

The number of relationship types can be very large. The main problem with this is the possibility of identifying these relationships in queries to the knowledge base. In this regard, it is preferable to reduce the number of types of relationships (and vertices) by increasing the number of vertices. For example, instead of the “semantic network" - "intended" - "data_representation" relationship, you can use the "semantic network" - "has" - "purpose"; "Purpose" - "is" - "representation"; "Representation" - "what" - "data".

4.3. Relations in the graphs of Rastier

Networks with a verb in the center (Rastier networks) operate with the following types of connections [5]:

Name	Type	Definition	Simplified name
(ACC)	accusative	Object of Impact	PATient
(ASS)	assumptive	Point of view	PERspective
(ATT)	attributive	Property, characteristic	CHARacteristic
(BEN)	benefactive	Beneficiary Entity	BENeficiary
(CLAS)	classitive	Class instance	CLASsitive
(COMP)	comparative	Items Combined by Comparison	COMParison
(DAT)	dative	Recipient	RECeiver
(ERG)	ergative	Ergative, process or action agent	AGEnt
(FIN)	final	Result or Expected Goal	GOAL
(INST)	instrumental	Used funds	MEAns
(LOC S)	spatial locative	Position (position) in space	SPAce
(LOC T)	temporal locative	Position (position) in time	TIME
(MAL)	malefactive	Affected party	MALeficiary
(PART)	partitive	Part of the whole	PARTitive
(RES)	resultative	Result, effect, effect	EFFect (или CAUse)

These types of relationships can be represented in global or local knowledge bases as follows:

```
p(["ACC", "patient"]).
p(["ASS", "perspective"]).
p(["ATT", "characteristic"]).
p(["BEN", "beneficiary"]).
p(["CLAS", "classitive"]).
p(["COMP", "comparison"]).
p(["DAT", "receiver"]).
p(["ERG", "agent"]).
p(["FIN", "goal"]).
p(["INST", "means"]).
p(["LOC_S", "space"]).
p(["LOC_T", "time"]).
p(["MAL", "maleficiary"]).
p(["PART", "partitive"]).
p(["RES", "effect", "cause"]).
```

4.4. Inheritance rules

The rules for inheritance of relations are automatically applied every time you access the knowledge base after all the facts have been exhausted. The following inheritance rules apply:

If X AKO Y and Y AKO Z then X AKO Z
If X ISA Y and Y AKO Z then X ISA Z
If X has_part Y and Y has_part Z then X has_part Z
If X ISA Y and Y has_part Z then X has_part Z
If X AKO Y and Y has_part Z then X has_part Z
If X ISA Y и and Y has_a Z then X has_a Z
If X AKO Y and Y has_a Z then X has_a Z

These rules are given in the file examples \ onto.pro.

4.5. Object Identification Principles

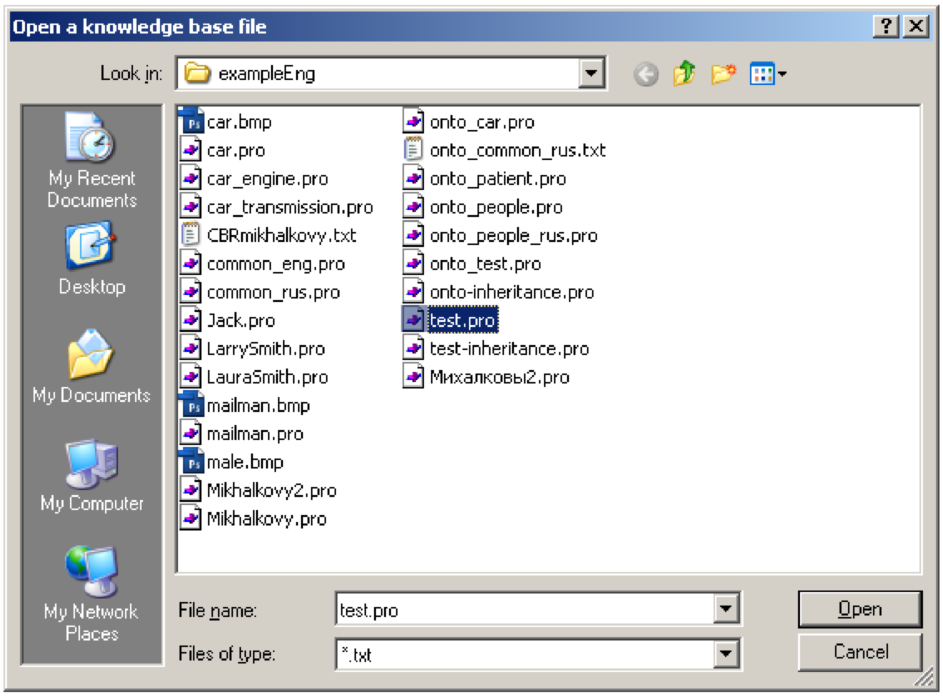
The designations of objects used in the facts are their local identifiers, which are valid only within the framework of one knowledge base file. For example, the fact f (“Sergey”, “is_parent”, “Nikita”) does not necessarily apply to the Mikhalkov family, even if there are facts f (“Sergey”, ”is_parent”, ”Andrey”), f (“Andrey” , ”Is_parent”, ”Egor”), which everyone is hearing. Its properties such as “has_name”, “has_surname”, “has_birth_date”, etc., including the passport number, fully identify the object. However, such a strict identification will greatly clutter requests, and it is not required for educational purposes. We will assume that the identification of the subject area is contained in the comments in the text of the file, which are available to the user.

The program requires unique identification of the vertices of the graph, and, therefore, unique identifiers of objects within the same file. In the graphs of Rastier it is difficult to observe uniqueness, since a verb (process) is put in the center, the variety of which is significantly less than objects. Here you can assign a unique identifier before the process name. For example, in the same column two verbs “live” are used, referring to different objects. In this case, instead of “live1” and “live2”, you can write “1: live” and “2: live”. Prefixes before the colon will not be displayed on the graph. In addition, as a result of applying the rules, objects with the same names may appear. Let the fact “person has_part hand” be in the database of ontologies (a person has a hand). After applying the inheritance rules to several instances, the same identifiers may appear: “Ivan has_part arm”, “Stepan has_part arm”, etc. On the graph, and not only this can be perceived as a fact that all these people have the same component (Siamese twins). To eliminate ambiguities, the program automatically substitutes the prefix for the name of the object - the name of the subject: “Ivan has_part Ivan: arm”, “Stepan has_part Stepan: arm”. Such identification avoids ambiguity; at the same time, prefixes are not displayed on the graph, since the location of the links allows identifying objects.

5. Program Run

Каталог программы должен хранить следующие файлы, требуемые для ее выполнения:
Semantic.exe
Vip6u2a.dll
Vip7edit.dll
Vip7kernel.dll
Vip7regexp2.dll
Vip7run.dll
Vip7vpi.dll

The executable module is Semantic.exe.
Immediately after start, the program displays a dialog box for selecting a global knowledge base file, which looks like this:



Open a knowledge base file

Look in: exampleEng

```
car.bmp                 onto_car.pro
car.pro                 onto_common_rus.txt
car_engine.pro          onto_patient.pro
car_transmission.pro    onto_people.pro
CBRmikhalkovy.txt       onto_people_rus.pro
common_eng.pro          onto_test.pro
common_rus.pro          onto-inheritance.pro
Jack.pro                test.pro
LarrySmith.pro          test-inheritance.pro
LauraSmith.pro          Михалковы2.pro
mailman.bmp
mailman.pro
male.bmp
Mikhalkovy2.pro
Mikhalkovy.pro
```

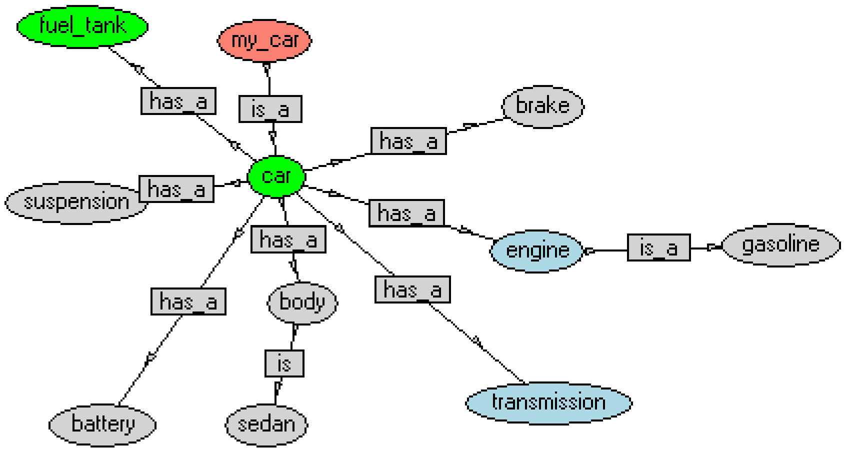
File name: test.pro

Files of type: *.txt

After the global knowledge base file is opened, the main program window appears, which is described in section. 7 "Program Interface".

6. Semantic Network Graph

The semantic network graph displayed using the program is as follows:



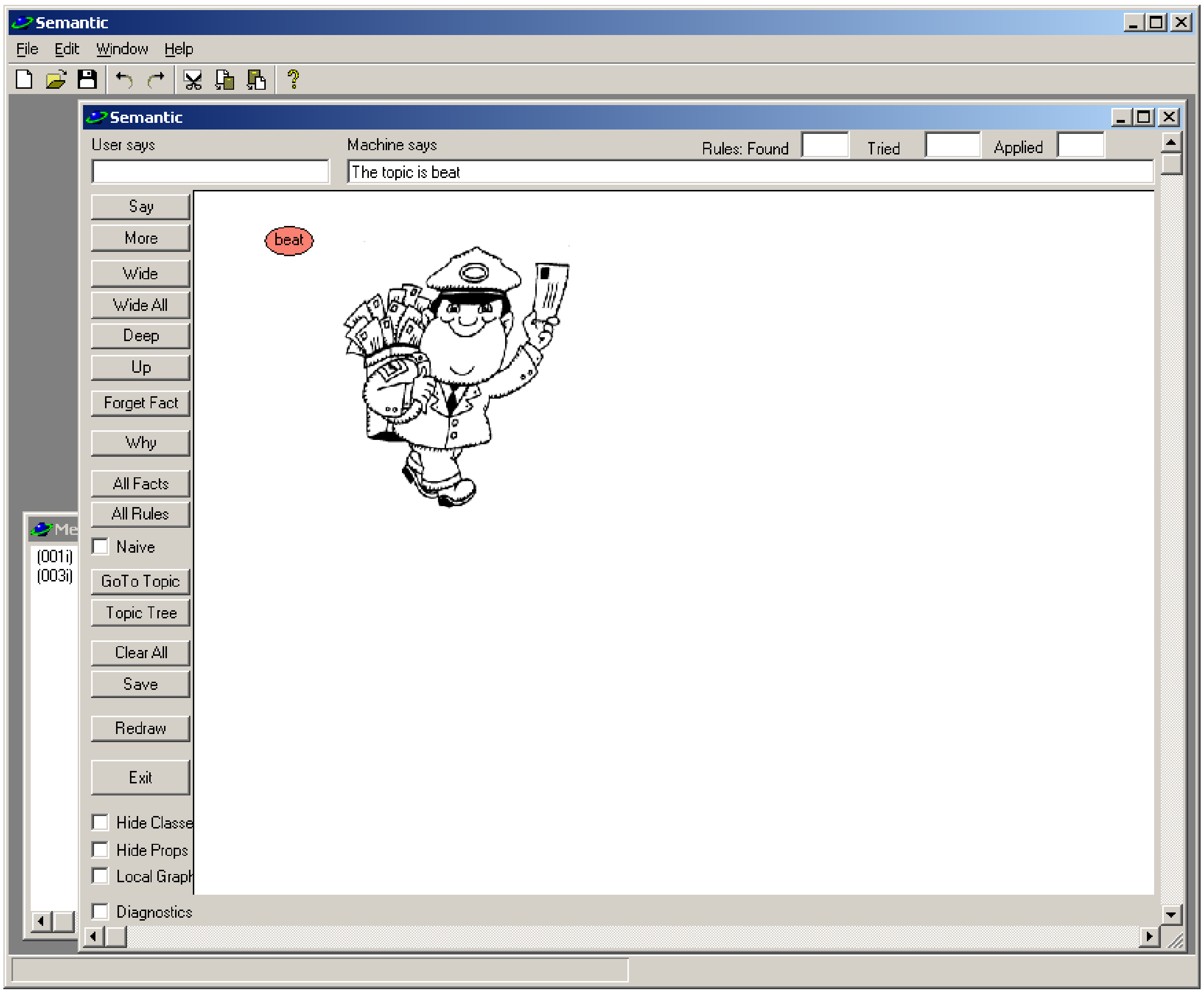
Subjects and objects are displayed as ellipses with inscriptions, and relations (predicates) are displayed as rectangles with inscriptions. For clarity, it is not recommended to give long names to subjects, objects and predicates, otherwise the figures will overlap each other. The following color codes are accepted:
1. Pink color indicates the main theme.
2. Classes are highlighted in yellow.
3. Red color indicates a fact or an individual object or subject selected by the mouse.
4. Blue color indicates an object or subject that has links to an external file.
5. The last fact, included in the context, is colored green.
6. Orange color indicates causal relationships.

The program does not fully implement redrawing the main window. If you close it with another window, the image is not restored. To restore the graph, you can click the Redraw button or resize or position the window on the screen. Window scrolling is also not supported yet.

7. Program Interface

5.1. Control Elements

After starting the program and downloading files with knowledge bases, the main program window is displayed on the screen, which looks like this:



The topic is beat

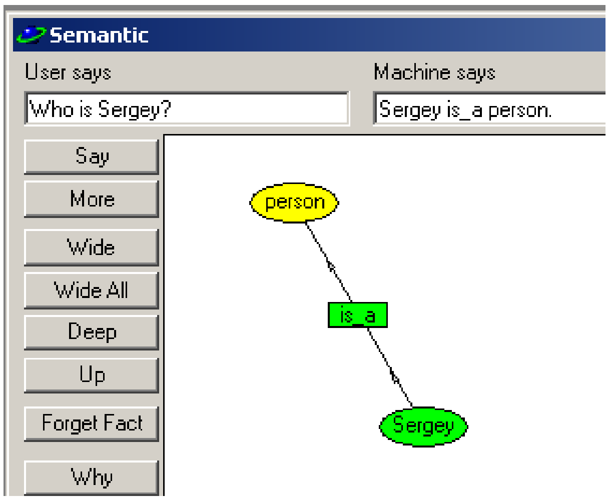
The program window has the main Semantic panel and the Messages window.
The “Semantic” panel has a field for entering text (under the line “User says”). Next to it is a dialog box (“Machine says”). The “Questions counter” field contains a counter of queries to the knowledge base and can be used to control knowledge in In addition to the question counter, there are rule counters: Number of rules found (Rules Found), number of attempts to apply them (Tried) and number of successful attempts (Applied).

The only effective option of the main menu of the program - “File / New” or the “New” button - allows you to open a new program window without restarting it. In parallel, you can open an unlimited number of windows. The limitation is associated only with the amount of available RAM.

The buttons of the main window have the following functions:

Say - Process the query string entered in the input field. The request may be in the form of an affirmative or interrogative sentence. The request must end with either a period or a question mark. Inside the query, these characters are naturally not allowed. The result is displayed in the dialog box, as well as in the form of a graph of the semantic network.

For example, for the query “Who is Sergey?”, We get the answer: “Sergey is_a person” and the corresponding fragment of the semantic network.



Who is Sergey?

Sergey is_a person.

Note that in order for the request to be processed in this form, then in addition to the fact
f (″ Sergey ″, ″ is_a ″, ″ person ″).

we must add a list of question words q ([″ Who ″]), and p ([″ is_a ″, ″ is ″]) in the list of predicate synonyms. If we want the structure to be processed

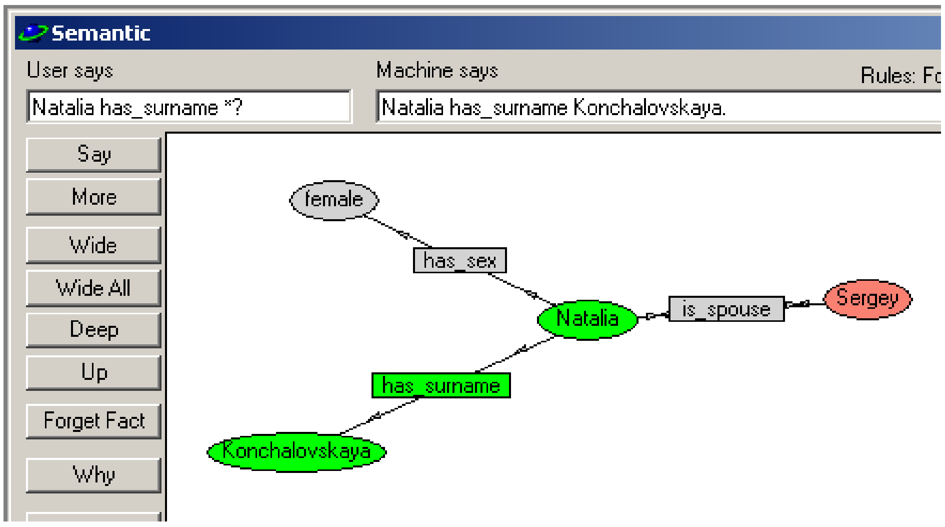
Who is Mr Sergey?

We should add to the list another list of ignored words: j ([″ Mr ″, ...]). The rules for generating requests are described in more detail in the sub-section 7.2.

If the phrase in the query string was specified in the affirmative form, then if there is a relevant fact, a fact corresponding to the query appears in the response window. If there is no such fact in the knowledge base, then it is added to the knowledge base, after which it can be saved with the Save button. If the fact is added by mistake, then it can be deleted using the Forget Fact button, previously selecting the entire fact with the mouse (by clicking on the predicate).

Due to the fact that the analysis of phrases in natural language in the program is poorly implemented, you can build a query very simply: specify the triplet in the same form as it is supposedly contained in the knowledge base, replacing unknowns with asterisks. For example, we want to ask a question about what surname the “Natalia” object has. Edit English phrase “What is surname of Natalia?” or “What surname has Natalia?” or even “What surname does Natalia have?” quite difficult. Therefore, we can write the query “Natalia has_surname *?” and we get the answer: “Natalia has_surname Konchalovskaya”.

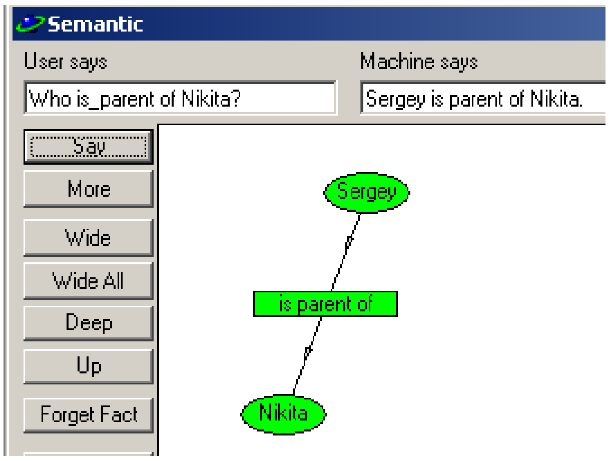
If there are no more facts relevant to this query in the database, the program applies the Back Chain Reasoning. This may take a long time.



Natalia has_surname *?

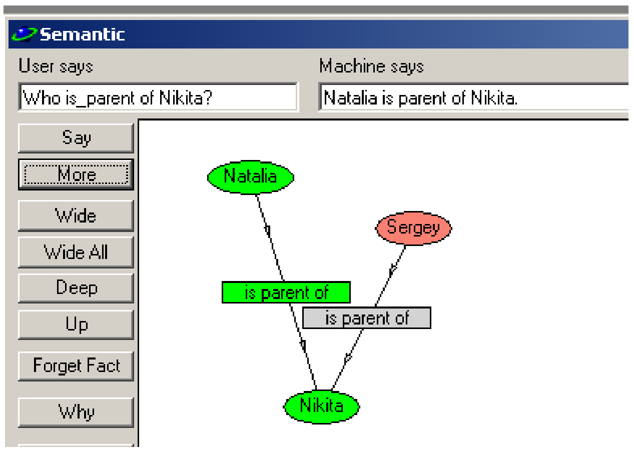
Natalia has_surname Konchalovskaya.

More - get an alternative response to the request by the Say button. For example, to the question “Who is_parent of Nikita?” the program gives the answer: “Sergey is parent of Nikita”. Pressing the More button gives an alternative answer: “Natalia is parent of Nikita”, as shown below. Just as with the Say button, in the absence of relevant facts, the reverse output is launched.



Who is_parent of Nikita?

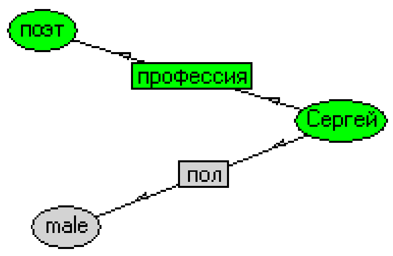
Sergey is parent of Nikita.

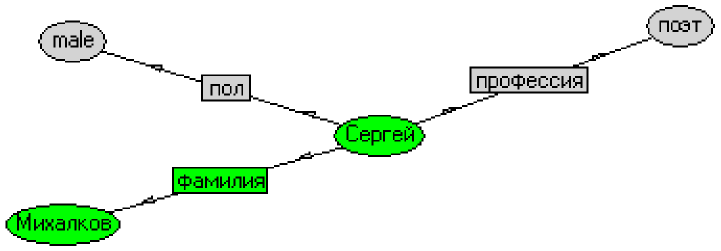


Who is_parent of Nikita?

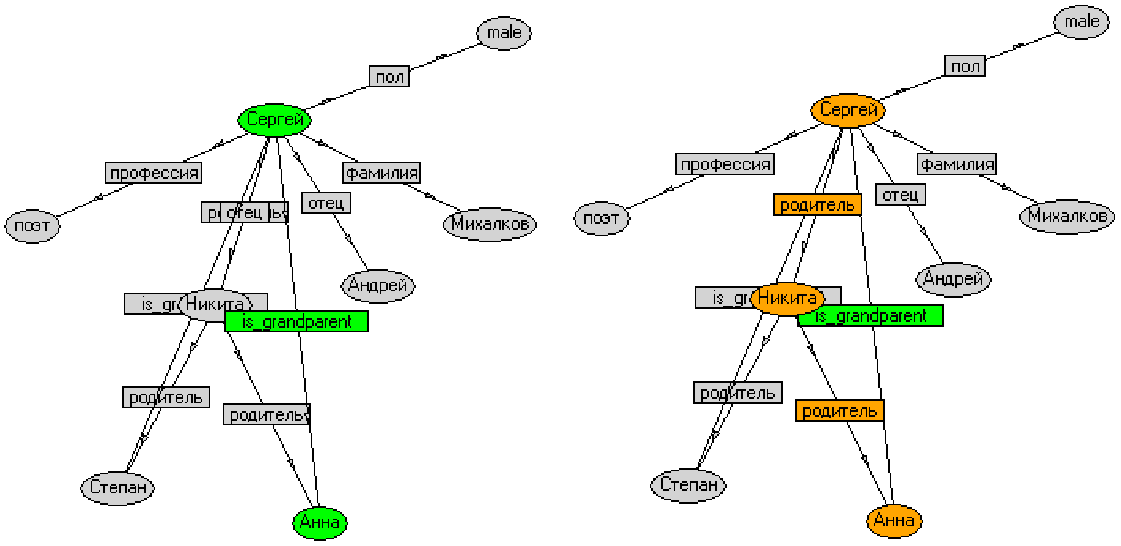
Natalia is parent of Nikita.

Wide - receive a fact about another object associated with the selected subject or subject received in the last appeal, i.e. Get a neighboring branch relative to the last. (The last fact included in the context is displayed on the graph in green and the object selected by double-clicking in red). For example, the last fact was “Sergey is a poet's profession”.If you click the Wide button again, the fact “car has_a transmission”, etc. will be received.





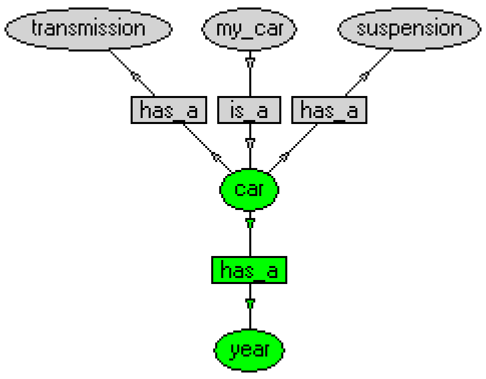
Each time you press the Wide button, the next fact will be extracted from the knowledge base and replenish the context. After all the facts related to this subject are retrieved, the program will begin to try to apply the rules. It is shown below that Sergey is the progenitor of Anna. If you click the Why button, then the facts that are the basis for the conclusion will be highlighted in orange,



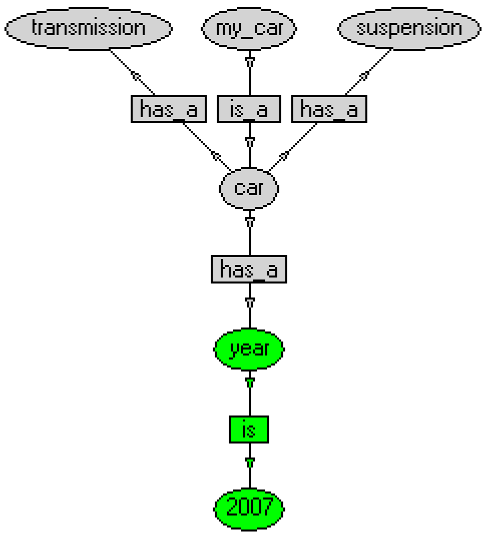
and in the dialog box and message box of the program - a text explanation: Sergey is_grandparent Anna BECAUSE Sergey is the parent of Nikita And Nikita is the parent of Anna.

The Wide All button finds all objects associated with this subject, including trying to apply all the rules.

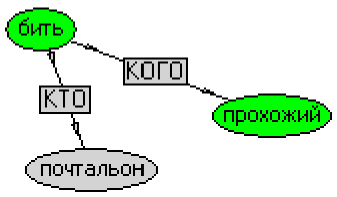
Deep - get the following fact about the selected object or object received in the last call, i.e. delve into the current branch of the graph. For example, the last fact was “car has_a year”.



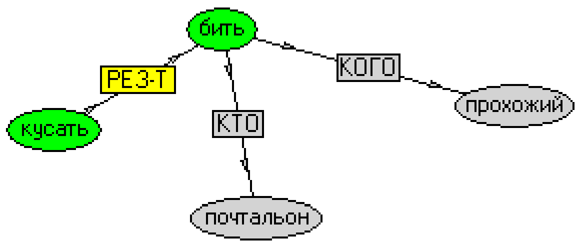
When deepening along this branch, we get the fact “year is 2007”.



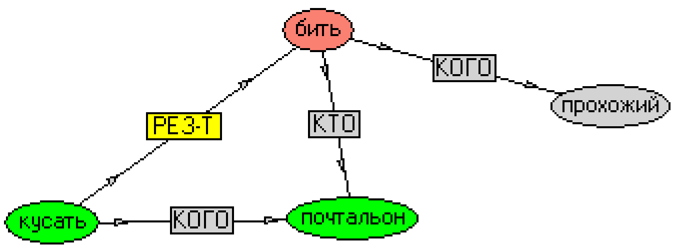
Up - get a fact in which the last or selected subject is an object, i.e. rise up the graph**.**



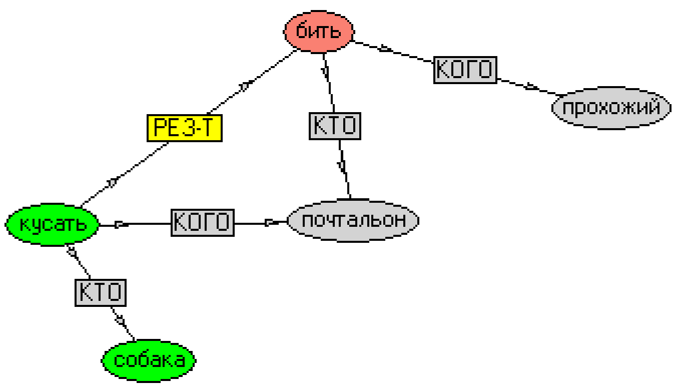
If we press the Up button now, we will get another subject associated with the “beat” object:



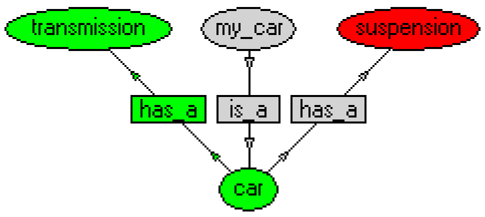
This means that the postman beat the passerby as a result of some kind of bite. Press the Wide button and find that the postman is the victim of the bite,



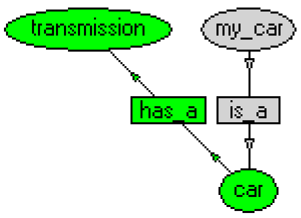
Pressing Wide again, we learn that the dog is the biting agent.



Forget Fact - remove a fact from the context whose object is previously highlighted in red with a double click. For example, we want to "forget" the fact "car has_a suspension". Select the object in red with a double click.



After clicking the Forget Fact button, this fact will be excluded from the context.



If the whole fact was selected (by clicking the mouse button on the predicate), then this fact will be removed not only from the context, but also from the knowledge base. If after that click the Save button, the fact will be deleted from the file.

All Facts - find and display all the facts. By pressing this button, the program includes in the context all the facts from the knowledge base, and also tries to apply inheritance rules to all these facts, adds new facts to the current context, established on the basis of these rules.

All Rules - Apply all known rules to the knowledge base. By pressing this button, the program sequentially applies each rule to all facts of the knowledge base, i.e. Performs direct output (Forward Chain Reasoning). This function may take a long time to complete. If the ontology database has 10 rules, two triplets each, and the knowledge base contains 100 facts, then the number of attempts to apply these rules will lie in the range 10 * 100 - 10 * 100 * 100, i.e. from 1000 to 100000. Processing 15000 attempts to apply the rules takes about 5 minutes.

In order to accelerate inference, the program implements an indexing fact algorithm that reduces the processing time of the rules by approximately two orders of magnitude. To study the temporal characteristics of the algorithm, the program also provides a non-indexing mode - naive output (Naïve switch), as well as counters showing the number of rules found (Rules Found), attempts to apply (Tried) and successful attempts (Applied), i.e. rules that produce relevant facts.

Counters are at the top of the screen. When calling the rules with the Wide button, the counters show increasing values, and with the All Rules button, the counters are previously reset.

It should be noted that one cycle of application of the rules does not always give all possible facts, since many facts are the result of the execution of chains of facts. The program logic here implements forward chaining, i.e. from known facts to the goal. At the same time, all the rules are sorted out, in the body of which there are triplets, unified by known facts. The search ends when a fact relevant to the query is found. An alternative to this approach is back chaining, in which the resulting part of each rule is unified for a given purpose. The return output is implemented in the Say button.

The All Rules button, at first glance, implements the descent through the search tree one level. However, this is not quite true. Each fact found is immediately included in the facts used by the following rule, and the total number of clicks of this button to guarantee the extraction of all possible facts largely depends on the order of the rules in ontologies. If the facts are the first, the processing results of which are used by the following facts, then all the facts can be obtained in one pass.

Goto Topic - go to the subject or object highlighted in red with a double click. At the same time, the accumulated context is “forgotten”, but retained for subsequent restoration, and the selected subject or object becomes the main topic from which the graph begins to unfold.

Topic Tree - show a tree of topics that were basic, and the order of transition from topic to topic. On the topic tree, you can use the mouse to select any topic and go to it with the Goto Topic button. In this case, the previously saved context is restored.

Redraw - Redraw the graph. It is advisable to use this button if the network nodes are located unsuccessfully.

Clear All - clear the context and start over.

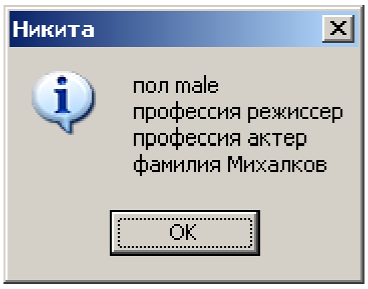
Save - save the current knowledge base. The program suggests choosing a file name. It is advisable to save the file if the knowledge base was replenished using the Say button. In addition to the facts of the knowledge base, the Say button causes the current context to be saved to a file, incl. facts obtained through the rules. Thus, the knowledge base can be updated so that the next time you do not start the long process of applying the rules.

Exit - exit from the task. Using the File / New menu option, you can start work again without restarting the program.

In addition to the buttons, there are switches: Hide Classes, Hide Props, Local Graph and Diagnostics.
The first two switches allow you to hide classes or properties of objects on the graph, respectively, to thin out an overloaded graph. The Local Graph switch allows you to restrict the graph to the selected object and relations only with this object. The Diagnostics switch enables the delivery of diagnostic messages in the message window.

Handling mouse clicks has the following options. If you click on the ellipse, it is highlighted in red and can later be used to establish new or delete established facts (buttons Deep, Wide, Wide All, Up, Why), as well as go to other topics (Goto Topic). If a mouse click on a predicate is made, then the whole fact is highlighted in green and acquires the status of the last established fact. Double-clicking on an ellipse causes an attempt to switch to a new knowledge base, a link to which should be contained in this object (record of type e).

In the event that the filter for displaying properties (Hide Props) is installed, a single click on the object also causes a window with a list of properties of this object to be displayed, as shown below.


Никита

пол male
профессия режиссер
профессия актер
фамилия Михалков

5.2. Query Language

The following language constructs can be entered in the query field:

A triplet ending with a dot, for example, “semantic network has history.”. If such a fact occurs, then this fact will be displayed in the output field (in practice, a repeat of the request). If there is a desire to bring this query closer to the natural language “Semantic network has a history.”, Then synonyms should be added to the dictionary entry, for example,
d (["semantic network", "semantic network"]).
d (["history", "history"]).
Note that the unification of the query with the database is performed in the case insensitive mode, i.e. case of input of characters does not matter.

Instead of any of the elements of a triplet, you can use an asterisk. This symbol is successfully unified with any of the values and is similar to a variable. For example, the query “* has a story.” will give the answer "semantic network has a history."; inquiry "* * *." will give out the first order fact from the knowledge base.

The following request forms are used if necessary to bring the request closer to natural language.

A triplet in interrogative form, ending with a question mark, for example, “Has a semantic network history?”. This word order for interrogative sentences is used in most languages. We can write the query in English “Has the semant_net a history?”. For this request to work, you need to add / change the following entries in the knowledge base:
d (["semantic network", "semantic network", "semant_net"]).
d (["history", "history", "history"]).
p (["has", "has", "have"]).
j (["the", "a"]).

A triplet in the interrogative form allowed in the Russian language, ending with a question mark, for example, “Does the semantic network have a history?”. For English, this construction is also relevant if we ask the question this way: “Does the semant_net have a history?”. In terms of parsing the query, the word “does” is garbage and should be included in the junk list: j (["the", ”a”, “do”, “does”]).

A triplet in question form with a question word ending with a question mark, for example, “What has a semantic network?”. The answer will be: "the semantic network has a history.". For this query to work, you must add the question words to the question: q (["what", "what", ...]) entry.

After successful unification of the query with the knowledge base, a triplet satisfying the query is displayed in the output field, and this fact is displayed in the graphic field, as well as the entire chain of facts from the main topic. For example, the semantic network is the main topic, and the first question was: "Does the graph have a vertex?" The corresponding fact is presented to the knowledge base, so the answer will be: "the graph has a vertex."

8. Control of Context

As the complexity of the created semantic network increases, problems arise. The first problem is related to the complexity of an expanding graph. Note that the screen does not display the entire network graph, but only that part of it that has already taken part in the dialogue, which we will call the context. As new facts become involved in the dialogue, the graph may become unreadable.

The second problem is that it becomes more difficult to assign unique names to concepts. Within the context, the same names correspond to the same concepts. There are two problems here: polysemy and synonymy. Polysemy - several meanings of the same word. For example, a field is an agricultural concept for a farmer, physical for an engineer, and legal for a lawyer. Polysemy can lead to the fact that fragments of the semantic network will not merge at all as expected. Synonymy - several terms for the same concept, for example, a helicopter and a helicopter. Due to synonymy, on the contrary, network fragments will be incoherent. The problem of synonymy also makes it difficult to combine fragments of the network written by different authors.

In a real dialogue, this problem is solved at the context level, which both interlocutors assume are the same. In cases where dialogue participants mean different contexts, misunderstandings usually occur. For example, you make an appointment at the metro in the usual place. For you, this place is at the exit of the metro, and the interlocutor thinks that this place is near the trains. It will be impossible not to miss each other.

Context management allows you to bring the dialogue to a form familiar to humans. Firstly, the current context is displayed on the screen, which allows you to keep abreast of what is at stake, if the user has forgotten. In particular, the current values are displayed. Secondly, changing the context, you can temporarily “forget” about the current topic by starting the dialog again. After that, you can again return to the topic under discussion, prompting the computer to "remember" what was mentioned earlier. For this, the program features are implemented using the buttons Goto Topic, Forget Fact, Topic Tree.

When displaying concepts for which there is a separate file with a local knowledge base, they are highlighted in blue. Selecting such an object with a single click, you can click the Topic Tree button. After that, the current context is remembered, the knowledge base for the new topic is loaded, and the remembered context is recalled from the memory if this topic has already been discussed.

So, by clicking the Topic Tree button, we get a graph of transitions from topic to topic that occurred during the dialogue. Thus, you can return to any previously discussed topic, "remembering" its context.

Note: Strictly speaking, the transition graph will be tree-like only if the dialogue develops accordingly. If you skip from topic to topic in an erratic way, the topic graph will have a network structure.

9. Messages of the Program

The program displays messages related to the dialog in the output field, and all other messages in the **Messages** window.

7.1. Messages it the Output Window

Message	Cause	User's action
The sentence must be terminated by "." or "?'	There is no terminal character at the end of the request	Check the query string
No more facts in the current branc h.	The DEEP button was pressed, but the last object is a leaf	Try to go on another branch
No more facts about the current s ubject.	The WIDE button was pressed, there are no more branches from the last subject	Try to go on another branch or go deeper into the current
OK, let us talk about <Topic>. Re peat your question.	he last request was not recognized. The program automatically tries to change the theme to <Topic>.	Press the SAY button
The fact about <Topic> is forgott en forever.	The FORGET FACT button is pressed. A fact that includes <Topic> is excluded from the context and from the knowledge base. If after that click SAVE, then from the file of this topic	Any
The fact about <Topic> cannot be forgotten.	The FORGET FACT button is pressed, but there are no facts including <Topic> in context	Possible error in the program. Contact the author
Double click the mouse to choose an object, then push Forget Fact b utton.	FORGET FACT button pressed, subject or object not selected	Double-click on subject or object
Click the mouse to choose an obje ct, then push Change Topic button	GOTO TOPIC button pressed and subject or object not selected	Double-click on subject or object
The topic <Topic> already exists in the tree	GOTO TOPIC button pressed to go to the <Topic> topic already discussed	Any
No external knowledge base abou t the topic <Topic>	GOTO TOPIC button pressed to go to <Topic> theme. No external file for	Check if an external file was really created but not connected

	this topic	
The context is cleared. You may s tart from scratch.	The CLEAR ALL button is pressed. The context is cleared. The knowledge base has been reloaded.	Any
Cannot re-initialize the knowledge base!	The CLEAR ALL button is pressed. The context is cleared. The knowledge base cannot be reloaded.	Possible error in the program. Contact the author
You are right. This is the true diag nose.	The hypothesis selected by the user matches the specified	Set user work
You fail. This is the wrong diagno se	The hypothesis selected by the user does not match the specified	Do not count user work

9.2. Messages in the Window 'Messages'

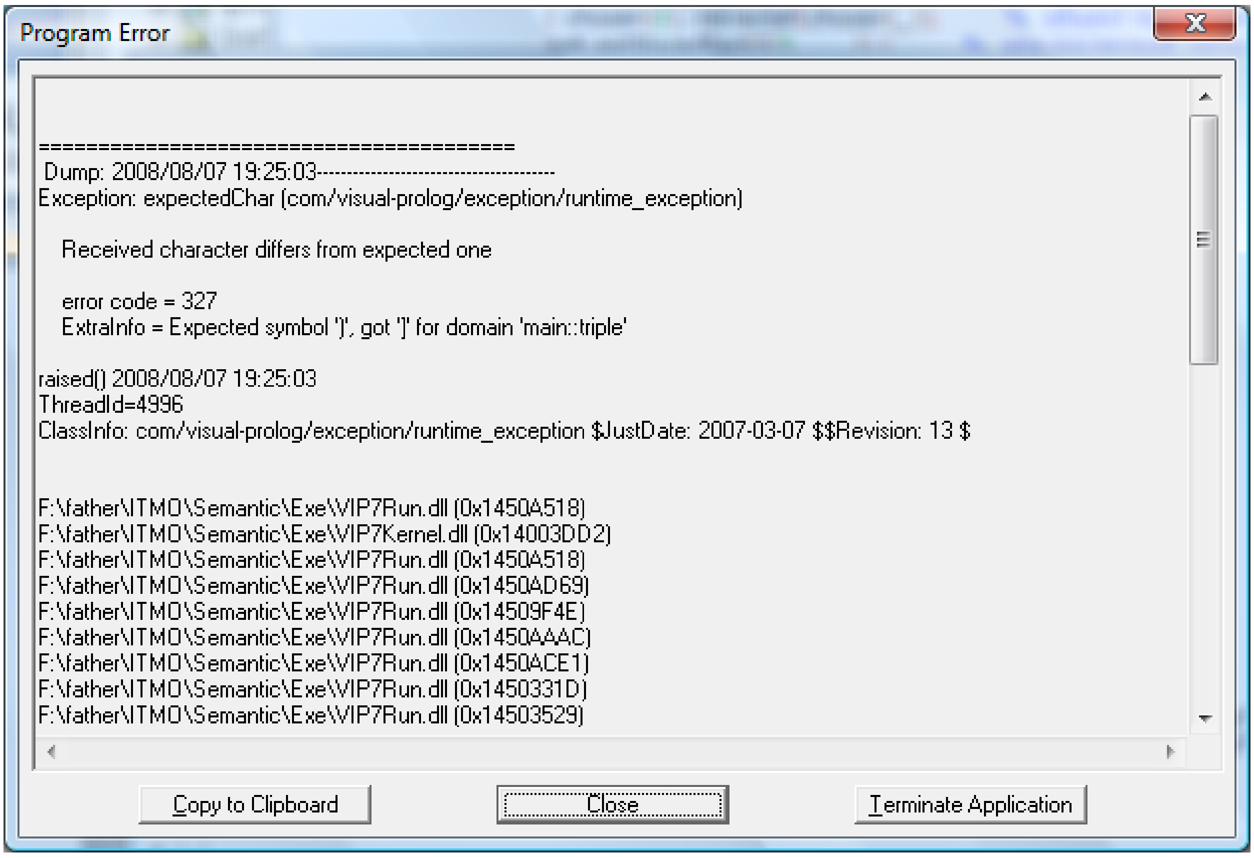
Сообщения в окне Messages имеют буквенную и цифровую идентификацию. Буква обозначает характер сообщения: i – информационное, W – предупреждение о возможной ошибке, E – сообщение об ошибке.

ID	Text	Cause	Action
001i	The file <path\name.txt > is loading….	Knowledge Base file is loading	Not required
002E	Error loading a knowledge base!	Unable to load knowledge base file	Check the presence and name of the file specified earlier in the message 001i
003i	The file <path\name.txt > is loading….	Ontology file is loading.	Not required
004E	Error initialization a knowl edge base!	Error loading ontologies	Check files specified in entries
005E	Error indexing a new fact!	Error indexing fact. Most likely, a program error	Contact the developer
007i	The sentence parsed: <Term1> <Term2> <Term3> <Sign>	The user's request is parsed into 4 components - three words and a sign	Not required
008i	The sentence SPO <Term1> <Term2> <Term3> unified with the fact <Subj ect> <Predicate> <Object>	User's proposal is unified by the fact <Subject> <Predicate> <Object>	Not required
009i	The fact SPO <Term1>	User's suggestion enriched the context	Not required

	<Term2> <Term3> replenished the knowledge base <Subject> <Predicate> <Object>	with the fact <Subject> <Predicate> <Object>	
010i	The question <Term1> <Term2> <Term3> identified as QPS	User’s proposal is identified in interrogative form (for example, word – predicate – subject)	Not required
011i	The question <Term1> <Term2> <Term3> identified as SPQ	The user's sentence is identified in interrogative form (subject – predicate – question word)	Not required
014E	Error unification stmt <stri ng> <string> <string> <string>.	Error unifying the query string.	Check query string
025i	A new fact established <Subject> <Predicate> <Object>	The rule written in the corresponding line of type r is applied to the current context and a new fact is received <Subject> <Predicate> <Object>	Not required
030i	The fact <Subject> <Predicate> <Object> replenished the context.	Fact <Subject> <Predicate> <Object> populated the context	Not required
040W	Pre-selection the fact <Subject> <Predicate> <Object> failed	There are no facts in the knowledge base for this triplet.	Not required
041i	чч.мм.сс.Forward chaining started	Processing of all rules started	Not required
042i	чч.мм.сс.Forward chaining ended. Rules found %, trie d %, applied %	Rules processing completed. Found, used, used successfully.	Not required
071E	Cannot draw fact <Subject > <Predicate> <Object>	Unable to display fact <Subject> <Predicate> <Object>	Contact the author
072E	Cannot draw <Object> at <X1,Y1 >	Unable to display <Object>	Contact the author
080i	The last fact <Subject> <Predicate> <Object> discarded.	<Subject> <Predicate> <Object> fact removed from context	Not required
081i	Current topic is <Topic>.	The main theme is now <Topic>.	Not required
082E	Cannot draw the current to pic in green!	It is not possible to repaint the current topic on the topic tree in green. Possible program error.	Contact the author
083i	Selected fact by a click is < Subject> <Predicate> <Object>	Selected fact Subject> <Predicate> <Object>	Not required
090i	The topic <Topic> already exists in the tree	The <Topic> theme is already on the topic tree	Not required
091W	No external knowledge ba se about the topic <Topic>.	There is no external database for the <Topic> theme. If the file should be, then a file name error is possible	Check file name, if any
092i	No context for the topic <Topic>.	There is no saved context for the <Topic> theme. If the topic has	Contact the author

		already been discussed, then a program error is possible	
093i	The topic <Topic> has not been discussed yet.	The topic <Topic> has not yet been discussed.	Not required
094i	Now we talk about <Topic>.	The main theme is now <Topic>.	Not required
095i	The topic "", Topic, "" has been restored.	Context main topic <Topic> restored.	Not required
096i	We talk about "", Topic, "" again.	You are redirected to a previously saved <Topic> theme.	Not required
097i	External knowledge base < base> about the topic <topi c> is loaded.	External knowledge base for <Topic> theme loaded	Not required

In case of errors in the syntax of the local or global database files, messages of the following form are issued:



```
========================================
 Dump: 2008/08/07 19:25:03------------------------------------
Exception: expectedChar (com/visual-prolog/exception/runtime_exception)

   Received character differs from expected one

   error code = 327
   ExtraInfo = Expected symbol ')', got ']' for domain 'main::triple'

raised() 2008/08/07 19:25:03
ThreadId=4996
ClassInfo: com/visual-prolog/exception/runtime_exception $JustDate: 2007-03-07 $$Revision: 13 $


F:\father\ITMO\Semantic\Exe\VIP7Run.dll (0x1450A518)
F:\father\ITMO\Semantic\Exe\VIP7Kernel.dll (0x14003DD2)
F:\father\ITMO\Semantic\Exe\VIP7Run.dll (0x1450A518)
F:\father\ITMO\Semantic\Exe\VIP7Run.dll (0x1450AD69)
F:\father\ITMO\Semantic\Exe\VIP7Run.dll (0x14509F4E)
F:\father\ITMO\Semantic\Exe\VIP7Run.dll (0x1450AAAC)
F:\father\ITMO\Semantic\Exe\VIP7Run.dll (0x1450ACE1)
F:\father\ITMO\Semantic\Exe\VIP7Run.dll (0x1450331D)
F:\father\ITMO\Semantic\Exe\VIP7Run.dll (0x14503529)
```

Unfortunately, this message does not indicate a specific line in which an error was detected. A hint can be the line “Expected symbol‘) ’, got‘] ’”, which implies that somewhere there is a square bracket instead of a round one. To speed up the search for such errors for editing knowledge base and ontology files, it is advisable to use editors as part of the Prolog compilers, for example, SWI-Prolog or Visual Prolog.