The Ministry of Education and Science of Ukraine
NATIONAL MINING UNIVERSITY
Information Technology Faculty

ARTIFICIAL INTELLIGENCE METHODS AND SYSTEMS

Tutorial
for students of
122 "Computer Science"

Dnipro
NMU
2017

The manual: ARTIFICIAL INTELLIGENCE METHODS AND SYSTEMS for 122 specialty «Computer Science» / life .: I.M. Udovik, G.M. Korotenko, L.M. Korotenko, V.A. Trusov, A.T. Kharj. – D.: State University «National Mining University», 2017. – 100 p.

Authors: I.M .Udovik, G.M. Korotenko, L.M. Korotenko, V.A. Trusov, A.T. Kharj.

(The stamp was «Recommended by the Academic Council of the State University of Social Sciences» as a textbook for bachelors and masters of specialty 122 «Computer Science». Protocol No. 11, June 26, 2017)

Responsible for producing Chair Computer system's software

I.M. Udovik

Content

Introduction

At present, there is no single and universally recognized, accurate and comprehensive definition of the advanced scientific direction called ***"artificial intelligence" (AI)***, as, indeed, there is no universal definition of the human intellect.

Wikipedia, defines the field of research and applications of AI, (English *artificial intelligence*, *AI*), as follows.

Artificial intelligence is: 1) the science and technology of creating intelligent machines, especially intelligent computer programs; 2) the property of intellectual systems to perform creative functions, which are traditionally considered the prerogative of man.

Among many points of view, artificial intelligence is dominated by three. According to the first, researches in the field of AI are fundamental researches in which frameworks models and methods of the decision of the problems traditionally considered intellectual and not giving before formalization and automation are developed. According to the second point of view, the new direction is associated with new ideas for solving problems on computers, developing fundamentally different programming technologies and creating new computer architectures that reject classical architecture that goes back to the first computer engineering. Finally, the third point of view, which seems to be the most pragmatic, is that as a result of the work in the field of AI, many application systems are created that solve problems for which previously created systems were unsuitable.

The task of artificial intelligence as a science is the reconstruction with the help of artificial devices (mainly laptops, tablets, smartphones, portable devices such as watches, key chains, etc., as well as robotic devices: home robots, unmanned vehicles, etc.) of reasonable actions and reasoning . Studying the theoretical foundations of this science makes it possible to find the most expedient ways of developing new information technologies and solving promising intellectual problems.

In the discipline "Methods and systems of artificial intelligence" the fundamentals of knowledge representation in modern computers, the methodology of mathematical modeling and automation of logical and analytical thinking, known artificial intelligence systems are studied. The manual for this course consists of seven sections. The first section defines the main tasks of research in the field of AI. The second section is devoted to the study of basic methods and models of knowledge representation in computers. The third section outlines the logic of propositions and methods for solving problems in this field of knowledge. The fourth section deals with the fundamentals of predicate logic and gives examples of solving problems using the method of resolutions. The fifth section is devoted to the consideration of the stages.

The design of artificial intelligence systems and the development of expert systems. In the sixth and seventh sections, the principles of constructing natural language systems, speech communication systems and intelligent computer graphics systems are set out.

The materials on the theory and practice of AI in the manual will help in studying the course "Methods and systems of artificial intelligence" and will ensure the consolidation of knowledge in the course of laboratory work.

1. Goals and objectives of research on artificial intelligence

In addition to the classical applications of computer technology related to the implementation of engineering and economic calculations, the development of automated control systems, the creation of information retrieval systems, etc., a trend is currently actively developing, called *"**artificial intelligence**"* (AI). Any problem for which the decision algorithm is unknown is a priori applied to the field of artificial intelligence.

Currently, there are four main areas for research in the field of AI [1 - 6, 11]:

1) modeling of individual functions of creative processes;

2) external intellectualization of computers;

3) internal intellectualization of computers;

4) purposeful behavior of robots.

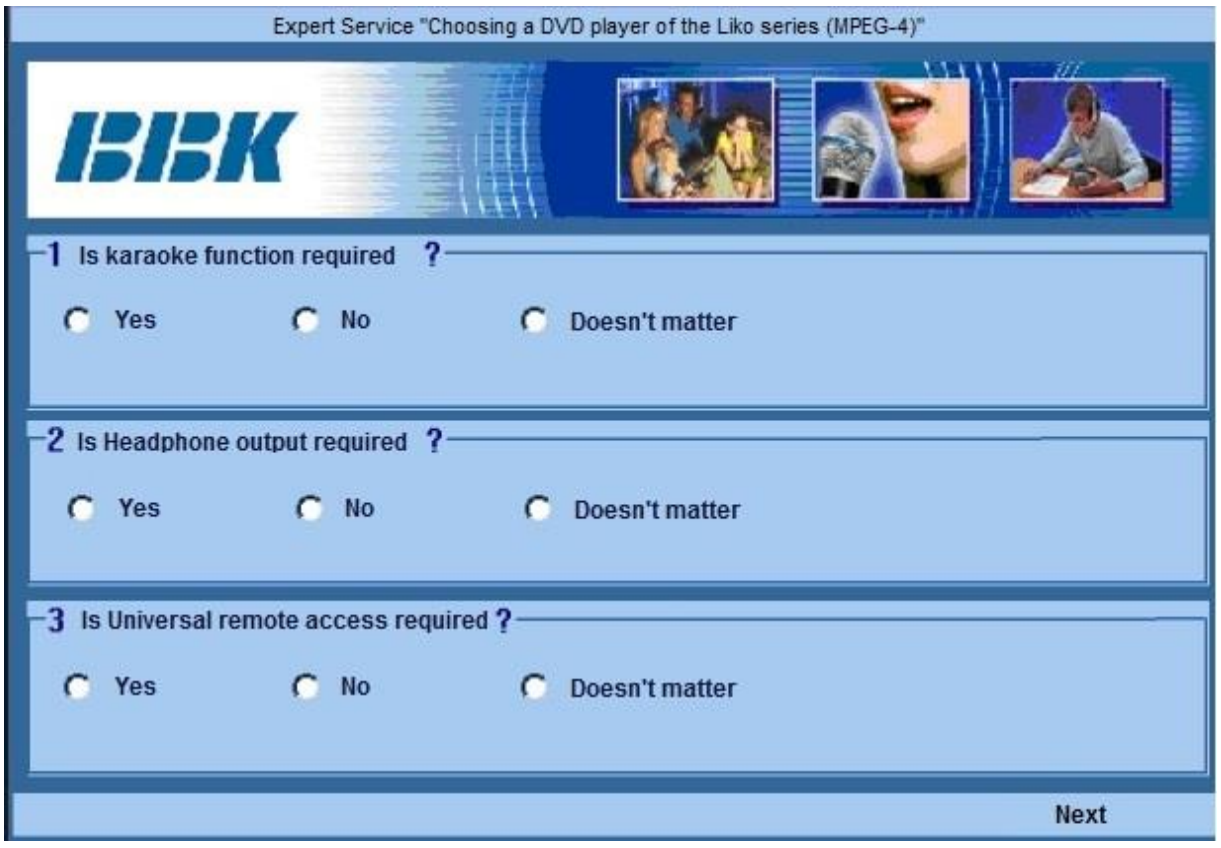
The first direction before others began to develop in AI; It was this term that gave birth to this term: modeling on computers of certain functions of creative processes (playing chess, checkers, dominoes, etc., automatic proof of theorems, automatic synthesis of programs, analysis and synthesis of musical works, automatic translation, pattern recognition, etc.).

The second direction is the fundamental and applied research aimed at increasing the level of human interaction with computer systems and devices within the framework of improving the functions of the interactive interface. The intelligent interface brings to a new level the efficiency of the use of automated control systems (ACS), computer-aided design (CAD), automated research systems (ASNI) and operational management of production as a whole.

Industrial robots have become not only one of the driving forces of automation, but also the most important means of implementing profound socio-economic changes in the world of work. The development and production of industrial robots with a high level of intelligence, multifunctionality and possessing strikingly high precision, made it possible to raise labor productivity to an unprecedented height. The number of industrial robots produced by Japanese firms (FANUC, KAWASAKI, MOTOMAN, OTC DAIHEN, PANASONIC), Germany (KUKA), USA (KC ROBOTICS, TRITON MANUFACTURING, KAMAN CORPORATION), Sweden (ABB) and a number of other countries are estimated at tens and hundreds of thousands (Figure 1.1).

Using knowledge accumulated in computers and databases on the development of their domains of interest, specialists in many branches of science have the opportunity, without going beyond the subject areas (sublanguages of the natural language), to recognize and diagnose processes in complex systems, to make optimal decisions, to formulate Plans of action, put forward hypotheses, and also to reveal patterns in the results of observations. These opportunities are realized by ***expert systems***, which began to spread intensively in hard-to-formalize branches of knowledge, for example, medicine, education, technical disciplines, etc. (Figure 1.2).

Figure. 1.1. The robots of the Japanese company FANUC, intended for welding, loading, sorting, transportation and other precision works



Expert Service "Choosing a DVD player of the Liko series (MPEG-4)"

1 Is karaoke function required ?

Yes No Doesn't matter

2 Is Headphone output required ?

Yes No Doesn't matter

3 Is Universal remote access required ?

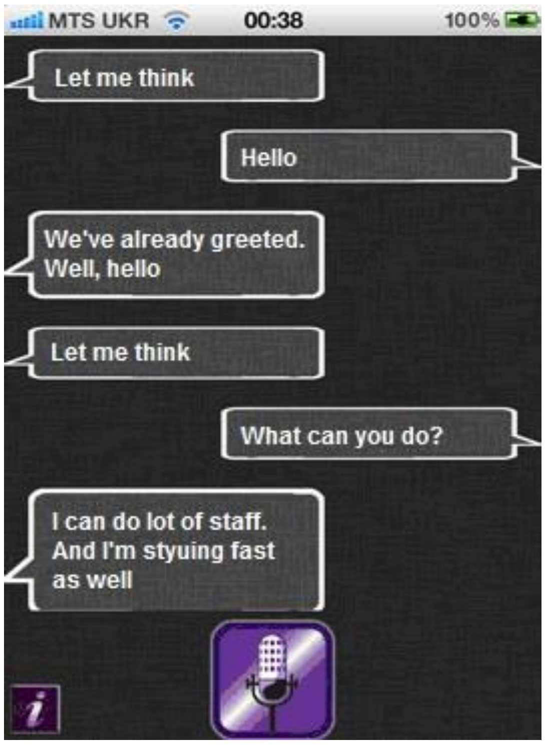
Yes No Doesn't matter

Figure 1.2. Example of expert system interface for selecting a DVD player

Third direction of using Artificial Intelligence solving problems of building new generation computers, because for solving problems of creating AI application, hardware as important as new method of symbol information processing. As usual, in this systems uses information, which presented in symbolic form: letters, words, signs, figures. It's makes difference between AI area and other, where traditionally computers processing information in numeric form.

In AI systems, there is a choice between many possible solutions under uncertainty, which requires fundamentally new architectural designs of computing devices and software components to manage them.

So, for example, for the operating system iOS smartphones iPhone (Apple), there's a personal assistant and the question-answer system, as well, *Siri* (*Speech Interpretation and Recognition Interface*) where developed. This application uses natural speech processing to answer questions and give recommendations. In addition, *Siri* adapts to each user individually, studying his preferences for a long time (Figure 1.3).



MTS UKR 00:38 100%

Let me think

Hello

We've already greeted.
Well, hello

Let me think

What can you do?

I can do lot of staff.
And I'm styuing fast
as well

Figure 1.3. An example of the result of interaction between a smartphone user and Siri - personal assistant

The fourth direction of AI applications is related to the creation of intelligent robots for various areas of human activity. This scientific and technical problem requires the development of both specialized computing facilities and a whole complex of complex technical, mechanical, software and energy systems: sensors, propellers, and so on. Like all AI systems, intelligent robots are focused on the use of knowledge. For example, knowledge of the external environment comes to the onboard computers of robots from numerous sensors (visual, acoustic, radar, tactile, etc.) (Figure 1.4).

Artificial intelligence, as the basis of the new information technology, multiplies the intellectual resources of society, because the interaction of the user with computing devices in his professional language increases the level of the user's intelligence and expands his capabilities in the formation of new elements of logical inference.

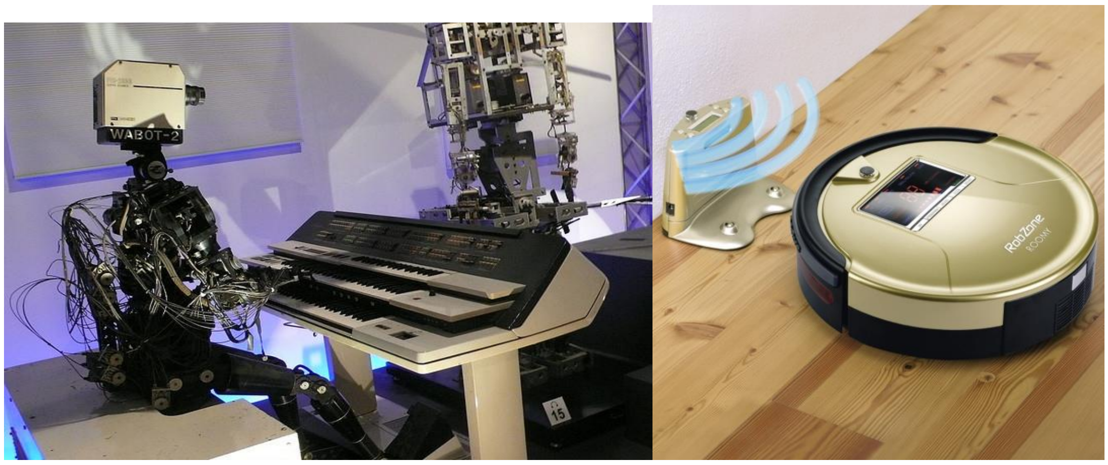


Figure 1.4 - piano-robot (left) and robot - vacuum cleaner (right)

And if earlier computers were the basis of the data processing industry, now, in connection with the active use of AI ideas and methods, it became legitimate to talk about the creation on the basis of computer devices of the industry of robotics and technologies for the design of intelligent systems.

Let's indicate the main areas of application of artificial intelligence:

1. Using the deductive reasoning (otherwise called ***calculus***) when solving with the help of cybernetic systems of intellectual problems, i.e. Problems that do not have a priori known solution algorithms;
2. Automated proof of theorems in axiomatic theories;
3. Recognition and understanding of speech and texts, visual images;
4. Understanding of the learning process and automated formation of scenarios of interaction between trainee and instructor;
5. Automated design of integrated (intelligent) robots;
6. Solving extreme problems with fuzzy and incomplete target functions;
7. Designing of systems of interaction of the person and databases and knowledge at their creation and operation;
8. Development of expert systems for different subject areas;
9. Designing decision-making systems for incorrect tasks and tasks with fuzzy setting;
10. Automated designing of interactive graphic systems with structured high-level graphic operands.

2. Representation of knowledge in artificial intelligence systems

At the heart of the development and using of computer technology are traditionally such concepts as ***programs*** and ***data***. In this case, the first are designed to process the second. At the early stages of the development of this industry, the programmer, as usual, developed the program by himself and entered the necessary data into it.

Then there was a major change in their interaction - with the appearance of ***databases*** of different structures (hierarchical, network, relational, object-oriented) and database management systems (DBMS), the ***data*** was separated from the

programs. To solve this problem, the data description tools contained in programming languages were used. Languages, such as FORTRAN and ALGOL, contained means for describing relatively simple data structures in the memory of electronic computers. More sophisticated means of describing hierarchical data structures are embedded in the languages of COBOL, C, PASCAL. MODULE-2, ADA, etc. In these languages, there are also tools for constructing data structures by the user.

In parallel with the above processes, the concepts of data representation in the external memory of computers developed which, with the improvement of the components of the element base and the transition from lamps, diodes and triodes to integrated microcircuits, were transformed into computers (Figure 2.1).

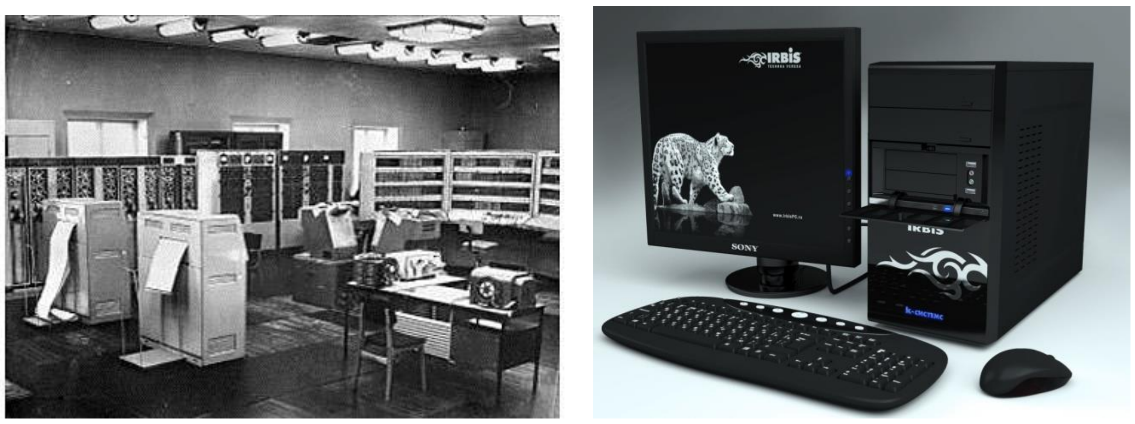


Figure 2.1. Computer BESM-6 (left) and a personal computer (right)

Another component that made it possible to finally separate the data from the programs was a device called the magnetic disk.

The revolutionary nature of his appearance was explained by the fact that after the computer was de-energized all the data present in it's memory was erased. The magnetic disk provided storage of information even after the computer was turned off. Moreover, a hard disk drive (HDD) can be extracted from one computer and connected to another, where it can provide not only data, but also operating systems and application programs written on it before (Figure 2.2).

At this stage, the fundamental concept of informatics was the notion of an information array - a ***file*** that is a specially organized data structure, recognized by the computer as a single unit, with a name and containing all the necessary structured records of data about various objects with which the computer is operating. The file can be interpreted as an information model of an object, such as a program, document, spreadsheet, musical works, etc. (Figure. 2.3).

Thus, the data representation in the external memory of the computer passed through three stages:

1. At the first stage, the methods of creating data records in files, maintaining files and organizing access to them were completely determined in the user's specific programs;
2. At the second stage, file management and organization of access to them began to be performed by computer operating systems;

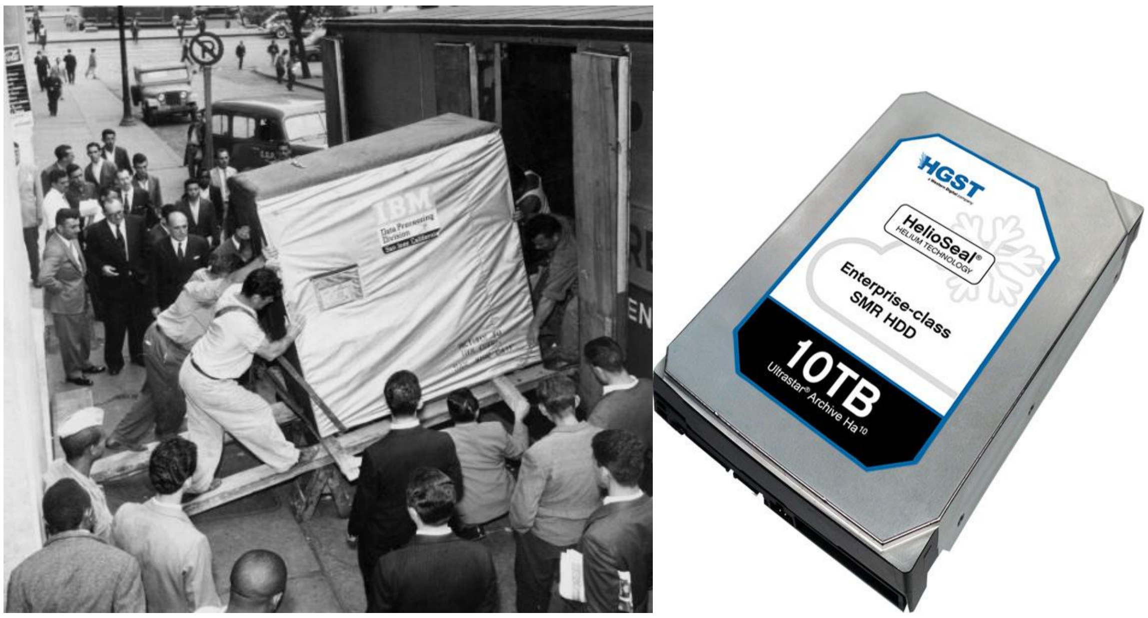
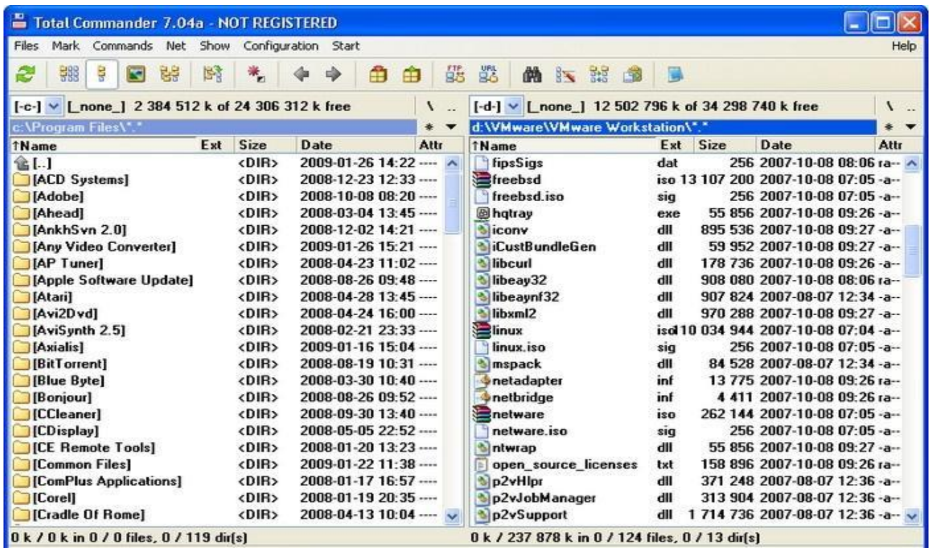


Figure 2.2 - Loading for the transferer of IBM's 5-megabyte hard drive, 1965 (USA) (left) and 10 terabyte Western Digital HDD, 2016 (right)


Total Commander 7.04a - NOT REGISTERED

Files Mark Commands Net Show Configuration Start Help

[-c-] [_none_] 2 384 512 k of 24 306 312 k free

c:\Program Files*.*

↑Name	Ext	Size	Date	Attr
[..]		<DIR>	2009-01-26 14:22	----
[ACD Systems]		<DIR>	2008-12-23 12:33	----
[Adobe]		<DIR>	2008-10-08 08:20	----
[Ahead]		<DIR>	2008-03-04 13:45	----
[AnkhSvn 2.0]		<DIR>	2008-12-02 14:21	----
[Any Video Converter]		<DIR>	2009-01-26 15:21	----
[AP Tuner]		<DIR>	2008-04-23 11:02	----
[Apple Software Update]		<DIR>	2008-08-26 09:48	----
[Atari]		<DIR>	2008-04-28 13:45	----
[Avi2Dvd]		<DIR>	2008-04-24 16:00	----
[AviSynth 2.5]		<DIR>	2008-02-21 23:33	----
[Axialis]		<DIR>	2009-01-16 15:04	----
[BitTorrent]		<DIR>	2008-08-19 10:31	----
[Blue Byte]		<DIR>	2008-03-30 10:40	----
[Bonjour]		<DIR>	2008-08-26 09:52	----
[CCleaner]		<DIR>	2008-09-30 13:40	----
[CDisplay]		<DIR>	2008-05-05 22:52	----
[CE Remote Tools]		<DIR>	2008-01-20 13:23	----
[Common Files]		<DIR>	2009-01-22 11:38	----
[ComPlus Applications]		<DIR>	2008-01-17 16:57	----
[Corel]		<DIR>	2008-01-19 20:35	----
[Cradle Of Rome]		<DIR>	2008-04-13 10:04	----

0 k / 0 k in 0 / 0 files, 0 / 119 dir(s)

[-d-] [_none_] 12 502 796 k of 34 298 740 k free

d:\VMware\VMware Workstation*.*

↑Name	Ext	Size	Date	Attr
fipsSigs	dat	256	2007-10-08 08:06	ra--
freebsd	iso	13 107 200	2007-10-08 07:05	-a--
freebsd.iso	sig	256	2007-10-08 07:05	-a--
hqtray	exe	55 856	2007-10-08 09:26	-a--
iconv	dll	895 536	2007-10-08 09:27	-a--
iCustBundleGen	dll	59 952	2007-10-08 09:27	-a--
libcurl	dll	178 736	2007-10-08 09:26	-a--
libeay32	dll	908 080	2007-10-08 08:06	ra--
libeaynf32	dll	907 824	2007-08-07 12:34	-a--
libxml2	dll	970 288	2007-10-08 09:27	-a--
linux	iso	10 034 944	2007-10-08 07:04	-a--
linux.iso	sig	256	2007-10-08 07:05	-a--
mspack	dll	84 528	2007-08-07 12:34	-a--
netadapter	inf	13 775	2007-10-08 09:26	ra--
netbridge	inf	4 411	2007-10-08 09:26	ra--
netware	iso	262 144	2007-10-08 07:05	-a--
netware.iso	sig	256	2007-10-08 07:05	-a--
ntwrap	dll	55 856	2007-10-08 09:27	-a--
open_source_licenses	txt	158 896	2007-10-08 09:26	ra--
p2vHlpr	dll	371 248	2007-08-07 12:36	-a--
p2vJobManager	dll	313 904	2007-08-07 12:36	-a--
p2vSupport	dll	1 714 736	2007-08-07 12:36	-a--

0 k / 237 878 k in 0 / 124 files, 0 / 13 dir(s)

Figure. 2.3. The view in the file manager window of files related to the organization's database

3. On the third, the files became elements of the created databases and developed control systems for them (so-called DBMS - database management systems). At the same time, it was possible to work effectively with large databases (in particular, with integrated databases containing heterogeneous data), processed for the benefit of the whole enterprise, industry, etc., and intended for use in applied tasks.

Thus, at the first stage of the development of data processing methods, the creation, maintenance and organization of access to them, both at the logical and physical levels, were entirely entrusted either to the developer or to the user of each individual program. Working with data related to a specific program, much less their use in other programs, was extremely time-consuming and inefficient.

The situation improved in the second stage, when part of the concerns for processing data in the external memory of computers (mainly at the physical level) was taken over by the operating system (OS).

However, work with integrated data became a reality only at the third stage of the development of information systems (IS), when it became possible to effectively organize databases with complex structure, and within the framework of DBMS, powerful tools were developed for working with sets of various information about the world around. It makes the existence of data sets independent of application programs in which these data are created and used, and also allowed technologically to separate different programs (creating, maintaining and using data) from data systems that were justified and effective. There was an opportunity effectively to connect programs for data processing with these data and already to cause programs on the basis and proceeding from existing data, instead of on the contrary, as was earlier. To ensure the operation of huge amounts of data, accumulated and continuing to accumulate by organizations and research institutions, Data processing centers (data centers, server farms, "cloud" data centers, etc.) (Figure 2.4).

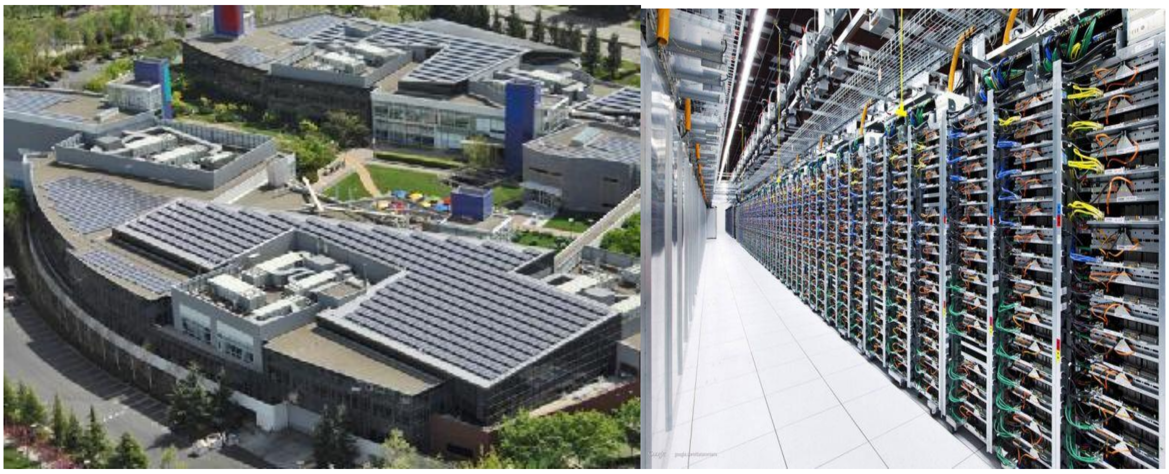


Figure 2.4. External (left) and internal view, with disk drives (right), one of Google's data centers

Finally, the DBMS provided the means for creating in each software suite an intermediate layer of software that separates the application software from the data used, effectively implementing the search, placement and other operations on the data, thereby freeing the application programmers from this activity. The intermediate layer is partly composed of system DBMS programs, partially completed by the user, the tools for which are provided by specialized data description languages and data manipulation languages included in the DBMS. These languages, such as SQL, supplement traditional programming languages with the tools for organizing large data sets. To ensure the possibility of direct use of these tools in applications in traditional languages, the data description language and the data manipulation language are often designed as an extension of the developed programming language - the so-called. "Including the language".

At the present time, we can talk about a new stage in the presentation of data in the computer's memory - the creation of information and computer networks and on their basis - distributed databases of shared use. This leads to both a reduction in the cost of creating and introducing databases, and to improve the quality of stored information, since it is possible to attract more qualified specialists to maintain databases. At the same time, the availability of this information for users is sharply increasing.

With the advent of AI systems, new concepts emerged - ***"knowledge"*** and ***"knowledge base"*** (KB). In the theory of artificial intelligence, knowledge is a collection of information about the world, the properties of objects, the laws of processes and phenomena, as well as the rules for using them for decision-making. The main difference between ***knowledge*** and ***data*** is their structuring and activity.

The emergence of new facts in the database or the establishment of new links can become a source of changes in decision-making. There was a need to somehow correlate the concepts "data and databases" that had become familiar with the concepts of "knowledge and knowledge". Undoubtedly, the data and structure of the database to some extent reflect knowledge of the subject area and its structure.

Nevertheless, there are specific features that distinguish knowledge from data. As knowledge of specific characteristics due to their representation in computers there are the following four characteristics:

- inner interpretability;
- structuring;
- connectivity;
- activity.

If you apply to data sets, some of the above characteristics, inherent knowledge will be valid for them. For example, the first sign is an ***interpretability*** - clearly visible in a relational database, where the column names are the attributes of relations, which names are indicated in rows. Internal interpretability provides the ability for install element of data related with its system of names. Name System includes the individual's name, which is assigned to a given information unit. The presence of "excess" name allows to system of an artificial intelligence to know what is stored in its knowledge base, and, consequently, to be able to answer on indistinct questions about the contents of the knowledge base.

The second feature is a ***structuring*** - can be considered as the property of decomposition of complex objects into simpler and linking between simple objects, which means using the relationship "part-whole", "class-subclass", "genus-species", etc. Relationships of this kind are found in the hierarchical and network databases. The same of the relation can be implemented in the relational (tabular) databases.

The third feature of knowledge is a ***connectivity***, for which is almost impossible to find analogues in conventional databases. Our knowledge are connected not only in the sense of structure. They reflect the regularities of interaction of facts, processes, phenomena and cause-and-effect relationships between them. It characterizes the possibility of establishing connectivity between the information units a variety of relationships (clear, fuzzy, binary, composite, etc.), which define the semantic and pragmatic of communications phenomena and facts, as well as the relations that determine the meaning of the system as a whole.

As for the fourth feature – an ***activity***, the situation is such that the use of computers - new knowledge generated programs (i.e., programmatically) and data are passive stored in memory. For human intrinsics a cognitive activity, in other words, the knowledge of human are active. And this distinguishes knowledge from data. For example, the discovery of contradictions in knowledge becomes a motive for their overcomings and emergence new knowledge. The same stimulus of activity is the incompleteness of knowledge, which is expressed in the need for their replenishment. Thus, the main distinction knowledge from data is their connectedness and activity, and the emergence in the base of new facts or establishment new relationships can be a source of change in decision-making.

Knowledge that used to build AI systems that ensure its operation, stored, modified and produced in it, can be defined in different ways. Currently knowledge is using three definitions. Knowledge is a:

♦ result obtained knowledge;

♦ judgment system in principle and a unified organization, based on the objective laws;

♦ formalized information referenced or used in the process of inference.

The process of solving problems of simple model AI system is shown in Figure 2.5.

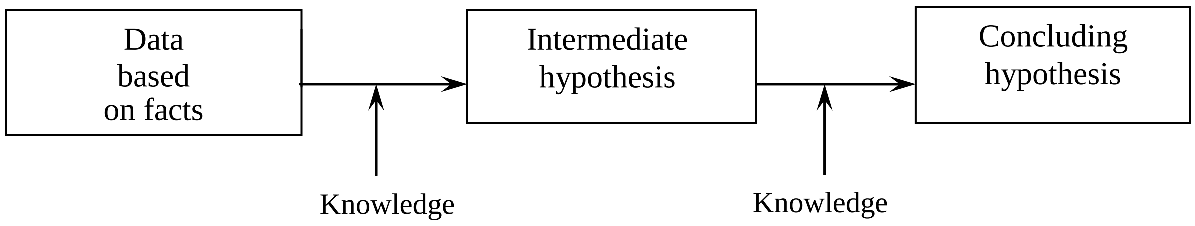


Figure 2.5. Link between knowledge and conclusion in solving of intellectual problems.

As shown in Figure of the task setting, the knowledge is the information that is referenced when are making different conclusions on the basis of the available data by ***logical conclusions***. If such actions are performed through the use of the software, the knowledge is the obligatory information, presented in a certain form.

It is important that the setting and solution of any problem related with the processing of data and knowledge re always associated with its "immersion" in the relevant subject field.

A ***subject area*** is a part of the real world that is considered in the predetermined (used developer) context. Usually, it is the set of all objects, properties and relations between them are considered in scientific theory or in industry of practice specialist's activity. Mentally, the subject area is considered that is composed of real or abstract objects called entities. For example, solving the problem of scheduling the processing of parts on machine tools, we engage in the subject area, on the one hand, such entities as specific machines, parts, intervals of treatment, and on the other - the general concept of "machine", "detail ", machine type ", etc. United totality of the subject area, its concepts and entities, as well as tasks that are solved in this field, is determined by the concept of ***problem areas***.

It should be understood that ***knowledge*** is regularities of subject area (principles, communications, laws), obtained as a result of practice activity and professional experience, enabling for professionals to formulate and solve problems in this area.

It is also important to note that the solution of problems in some subject area of knowledge, the latter is conveniently to divide into two large categories - ***facts and heuristics***[1] .

The first category indicates generally a well-known circumstances in the subject field, so knowledge of this category is sometimes called the text, referring sufficient illumination of them in the literature or textbooks.

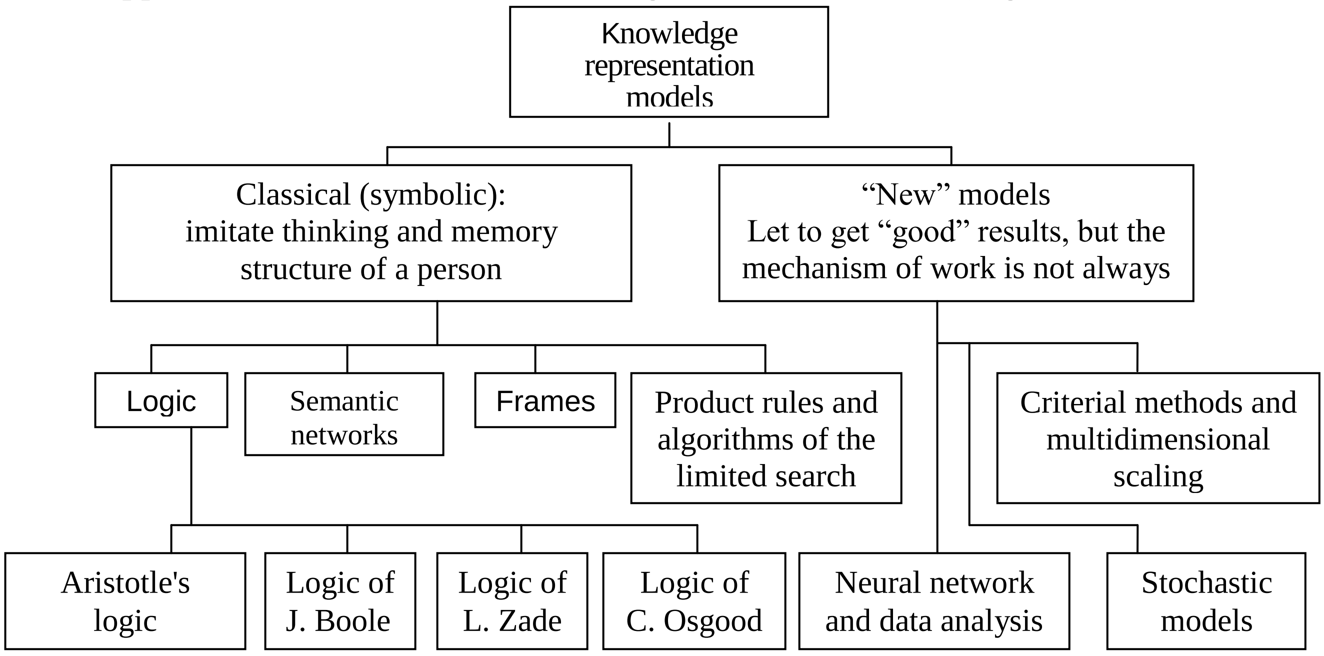
The second category of knowledge is based on their own professional experience in the subject area (the so-called experts), accumulated as a result of years of practice. In the so-called expert systems heuristic knowledge play a decisive role in improving the efficiency of the systems. In other words, this category includes such skills as "ways of concentration", "ways to remove useless ideas," "ways of using the fuzzy information", and so on, that are can decided to decide tasks with greater efficiency. However, due to lack of scientific validity and the lack of comprehensive information to use such knowledge is necessary cautiously.

Knowledge, in addition, can be divided into ***facts*** (factual knowledge) and ***rules*** (knowledge for decision-making). Knowledge are implied by the facts of the type "A is A", they are characteristic for the databases. Rules are implied a knowledge of the kind "If A then B" under the rules. Besides them, there are the so-called meta-knowledge (knowledge about knowledge). The concept of "meta-knowledge" indicates to knowledge concerning ways of using knowledge, and knowledge concerning the properties of knowledge. This concept is necessary for manage of data knowledge, logical conclusion, identification, training, etc.

Data and data structures are not fully reflect features of subject areas. Although, generally speaking, it is not always possible to carry out a clear distinction between data and knowledge, however, there are differences between the data and knowledge, and these differences have led to the emergence of special formalisms in the form of models representation of knowledge in computers, reflecting in a greater or lesser degree the basic features that characterize knowledge.

1 Heuristics (lat."Heuristic" - from the "eureka!" - "I found it!") - other than an algorithmic method of solving complex problems, which is based on informal rules of experienced professionals. The concept of heuristics in solving of the tasks includes a list of methods, conjectures, methods aimed at reducing the amount of computational work or obtaining result in cases where the mathematically sound methods are not available or useless.

The development of methods of describing of knowledge led to the creation of a wide range of various models using a variety of theories and approaches. One of the approaches to their classification might look like this [4] (Figure 2.6).



Knowledge representation models
Classical (symbolic): imitate thinking and memory structure of a person
"New" models Let to get "good" results, but the mechanism of work is not always
Logic
Semantic networks
Frames
Product rules and algorithms of the limited search
Criterial methods and multidimensional scaling
Aristotle's logic
Logic of J. Boole
Logic of L. Zade
Logic of C. Osgood
Neural network and data analysis
Stochastic models

Figure. 2.6. Classification of the most common models of knowledge

Currently enough widely are used at least four kinds of models and therefore knowledge representation languages [9,11,14]

1) models and languages of the semantic networks;
2) systems of frames;
3) logical models and languages;
4) production systems.

Let us consider these approaches to the description and presentation of knowledge in computers.

2.1 Semantic[2] networks

Under ***semantic network*** is defined an information model of the subject area having the form of a directed graph whose vertices correspond to the object of the subject area and the arcs (edges) define the relationships between them.

The objects can be concepts, events, properties and processes. The first network models appeared in the 60s of the last century. Examples of its can be RX-codes, syntagmatic chains, and, finally, the semantic networks. The basis of the

2 Semantics, in the narrow sense of the word is a meaningful (semantic) side of the language units, their meaning. Each semantic unit has its own semantics: the most elementary one is the semantics of the morpheme, it is conscious of the composition of the inclusive words (meaning "a person that is committed to someone": an atheist, an antiglobalist). Semantics, in the broad sense of the word, is an analysis of the relationship between linguistic expressions and the world, real or imaginary, and this relation too (Cf. Expression type semantics of the word) and the totality of such relations (for example, we can talk about the semantics of some language). This relation is that language expressions (words, phrases, sentences, texts) denote what is in the world - subjects, qualities (or properties), actions, ways of performing actions, attitudes, situations and their sequences.

best known model is the concept of semantic network formed labeled vertices and arcs.

The top of the network are some entities (objects[3] , events[4] , processes[5] , the phenomenons), and an arc connecting them - relationships between these entities. For this reason, the language of semantic networks sometimes called relational language of relations. Imposing limitations on the description of vertices and arcs, you can get different types of networks.

If the tops does not have its own internal structure, the corresponding network called ***simple networks***. If some vertices have its own structure, such networks are called ***hierarchical networks***. At the initial stage of development of AI systems were used only simple networks. Currently, the majority of applications, that using semantic networks, are hierarchical.

Relationships in the networks may be of different types that allowing to sufficiently ensure in semantic network such indication knowledge as connectivity. In general, this means that with help of the semantic network, you can display the knowledge presented in texts in the natural language.

For example, consider the following sentence: [1] "Fisherman (***a1***) sat in a boat (***a2***), swam across to the other side (***a3***) and took the basket (***a4***) with fish (***a5***).These objects are related by the following relationships: “sat in” (***r1***), “swam across” (***r2***), “took” (***r3***) and the “is” (***r4***). The network that is corresponding to this text is shown in the Figure 2.7.

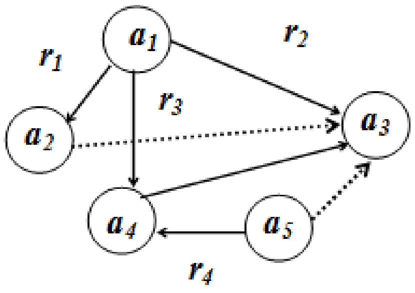


Figure 2.7. Representation of the objects (fisherman, boat, beach, basket, fish) and events (sat, swam across, took, is) in the form of a semantic network (example 1)

3 Object is an subject or phenomenon, existing in reality. Something that has clearly defined boundaries. Tangible entity has clearly defined behavior.

4 The event is something that takes place, occurs at an arbitrary point in space-time; significant incident, event or other activity as a fact of public or private life; a subset of the experimental outcomes.

5 Standard «ISO 9000: 2000 Quality Management Systems" defines a process as a set of interrelated or interacting activities that transform input data into outgoing.
Process (informatics) - performed program of the computer system.
Process (organization theory) - steady and purposeful set of interrelated activities, which in certain technologies are transforming inputs into outputs for getting predetermined products, results or services that are representing a value for the consumer.

Proceeding from the logic of real-world events and the accepted way of describing all possible situations , it is possible to consider data as some other relations that are obviously not present in the source text.

These additional relationships are shown in Fig. 7 in dotted lines.The expanded variant of the text will be as follows: "The fisherman sat in the boat and on the boat crossed over to the other shore. On the other side was a fish. The fish was in the basket. The fisherman took a basket of fish . "

One important circumstance should be emphasized. As shown by studies conducted on the example of Indo-European languages, there are no more than 200 different kinds of relations that are not reducible to each other. The combinations of these basic relations make it possible to express all other relationships fixed in texts in natural language. This circumstance underlies the formation of models of so-called situational management . In addition, the presence of a finite set of basic relations makes it possible to represent in any knowledge base any subject area and, moreover, to automatically construct semantic networks based on any text.

For example, the knowledge of a man with the surname Ivanov, who owns a Volga car of red color, can be displayed by the semantic network shown in Fig. 2.8.

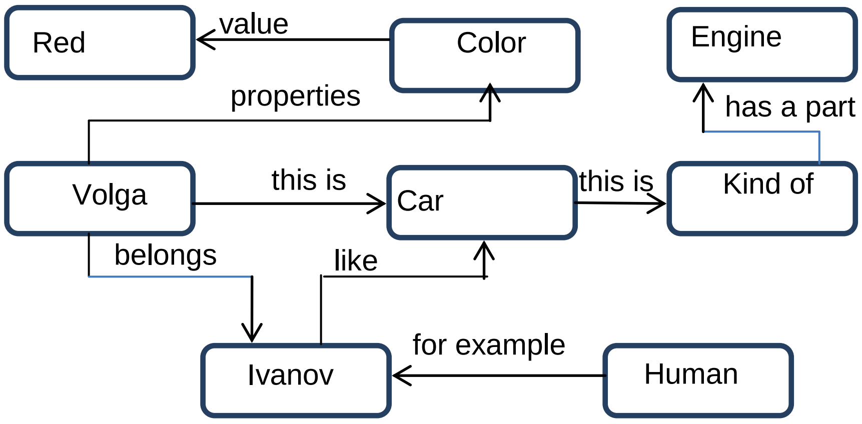


Figure 2.8. Semantic network (example 2)

The network also contains vertices and arcs that are not mentioned in the previous sentence and reflect relationships such as "part-whole", links like "it", "belongs", "likes", etc.

The main difference between hierarchical semantic networks and simple networks is the ability to divide the network into subnet spaces and establish relationships not only between vertices, but also between spaces.All vertices and arcs are elements of at least one space. Note that the concept of space is analogous to the notion of brackets in mathematical notation. Different spaces existing in the network can be ordered in the form of a tree of spaces, the vertices of which correspond to spaces, and arcs to the "visibility" relationships.

Figure 2.9 shows an example of a tree of spaces, according to which, for example, all the vertices and arcs lying in the ancestor spaces P4, P2 and P0 are visible from the child P6, and the remaining spaces are "invisible".

The "visibility" relation allows you to group spaces into ordered sets - "perspectives". A perspective is usually used to restrict network entities that are "visible" by some procedure that works with the network.

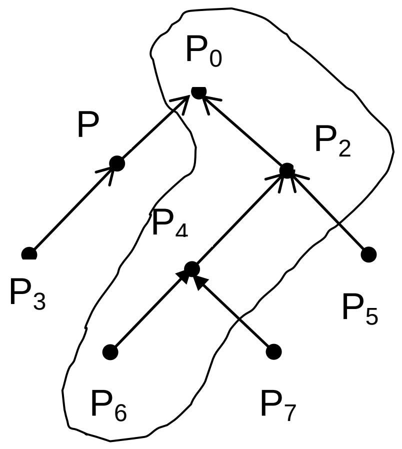


Figure 2.9. The space of states.

A particular case of semantic networks are ***scripts*** or ***homogeneous semantic networks*** [8, 11].

In such networks, objects are connected by a single relation of strict or non-strict order with different semantics.

If, for example, the objects-concepts are jobs (or individual operations), and the only strict-order relation is the sequence relation, then we come to the well-known network schedule of the complex of works with the so-called "French view". Obviously, scenarios are a convenient means of drawing up plans. The subject domain is the set of admissible states of its components. Presented through the general concepts and relations between them, this set forms a knowledge base (KB) - in the form of a so-called intensional semantic network. On the other hand, depending on the situation, the domain components will have specific values, properties, characteristics. All these specific domain data will be displayed in the so-called extensional semantic network or database (DB) of the network structure. The terms ***"intensional"*** and ***"extensional"*** are borrowed from semantics - the science of sign systems.

Intensional – the definition or description of a concept through its properties.***Extensional*** - a set of concrete facts that correspond to this concept. Using the example of a general semantic web, you can distinguish between a database and a knowledge base (see Fig. 2.10). In general, the ***intensional*** is a set of general concepts and relationships that characterize a multitude of objects, objects, phenomena. ***Extensional*** calls specific characteristics of each element of this set of concepts and relationships. For example, the concept of a "light car" with relations "body", "engine" and "management" will be ***intensional*** with respect to a variety of extensions - brands of cars ("Volvo", "Mercedes", "Zhiguli", "Ford", " Muscovite ") with their specific characteristics. In turn, if, for example, the "intranet" is, for example, "Zhiguli", the extensions may be their models (2101, 21301, 2303, 2309, etc.) with specific characteristics. Thus, the concepts of intensional and extensional are themselves relative.

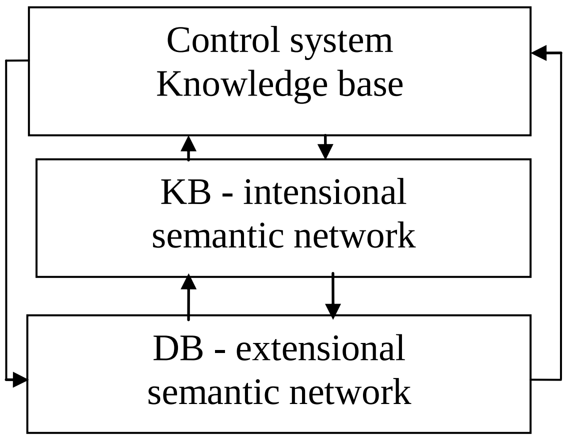


Figure 2.10. Data and knowledge in the semantic model.

In computers, the semantic network is implemented as an intra-machine data structure that is used to represent individual words and their semantics. For example, [11], the semantic net "NOC is the smallest common multiple" (see Figure 2.11) can be rewritten on the basis of the results of the structural analysis in the form of a semantic network data structure, as shown in Fig. 2.12.

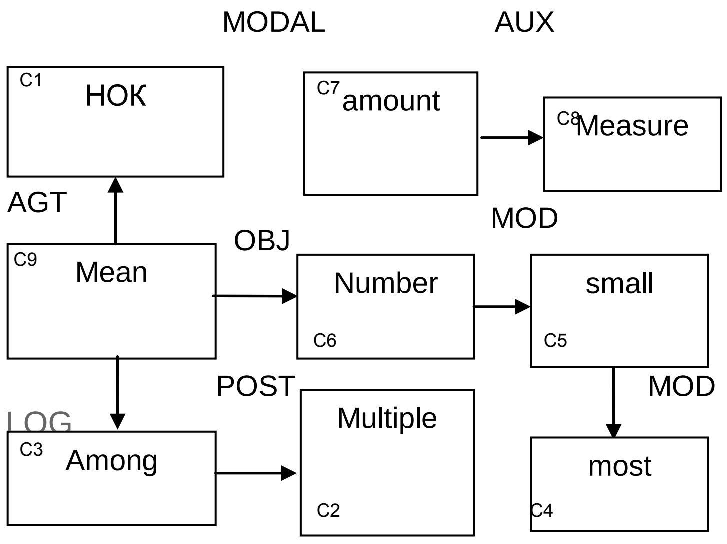


Figure 2.11. Semantic network "NOC is the least common multiple"

Here C1, C2, ..., C9 are nodes of the semantic network, which are pointers of the list structure.

The abbreviations for some grammatical terms have the following meaning:

CURRENT - printing the corresponding semantics, with some active word being written in the original form;

MODAL - time (estimate) and conjugation of the verb;

MOD - modified words;
POST - with the help of the union shows the modified word;
AUX - auxiliary.

The positive side of the knowledge representation of semantic networks is that it is a very simple and understandable way of describing based on the relationship between elements (nodes and arcs). However, with the increase in the size of the network, the time for finding solutions is significantly increased in comparison with methods that do not have a strategy. In addition, they inherent the problem of guaranteeing the suitability of output results, which also includes the problem of property inheritance.

C9	TOK	MEAN	C4	TOK	MOST
	AGT	C1		MOD	ZERO
	LOG	C3	C5	TOK	SMALL
	OBJ	C6		MOD	C4
	MODAL	C7	C6	TOK	NUMBER
C1	TOK	HOK		MOD	C5
	MOD	ZERO	C7	PRES	AMOUNT
C2	TOK	MULTIPLE		AUX	C8
	TOK	ZERO	C8	TOK	MEASURE
C3	TOK	AMONG			
	POST	C2			
	MOD	ZERO			

Figure2.12. Semantic network data structure

2.2. Frame Models

Semantic networks, in spite of their great opportunities, connected with the wealth of available means, for displaying the relations between concepts and objects, however, also have some drawbacks. The implementation of models with arbitrary structure and different types of vertices requires a wide variety of procedures for processing information, which complicates the software of computers. This led to the emergence of a number of specific types of semantic networks, such as: syntagmatic chains, scenarios, frames, and so on. Let's consider in more detail frame representations.

A frame is a way of representing knowledge in artificial intelligence, which is a scheme of actions in a real situation. Initially, the term "frame" was introduced by Marvin Minsky in the 1970s [15] to indicate the structure of knowledge in modeling the perception of spatial scenes. A frame is a model of an abstract image, a minimal possible description of the essence of an object, phenomenon, event, situation or process. There are sample frames, instance frames, frame-structures, frame-roles, frame-scenarios, frame-situations. In this case, the system of connected frames can form ***a semantic network***.

A frame structure is understood as a way of using a scheme, a typical sequence of actions, or a situational modification of a frame. The frame, among other things, includes a certain knowledge by default, which is called ***a presumption***.

Usually it consists of a name and individual units, called ***slots***, and, as a rule, has a homogeneous structure (Figure 2.13).

```
FRAME NAME

First slot name: 1st slot value
Second slot name: 2nd slot value
..................................
Name of the Nth slot: value of the Nth slot
```

Fig. 2.13. Structure of the frame

Slots are some unfilled frame substructures, after filling with specific data, the frame will represent one or another situation, phenomenon or object of the domain. When the frame is specified, it and its slots are assigned specific names and the slots are filled. As the values of the slots, the names of other frames can act, which makes it possible to build a network of frames. Formally, a frame is understood as a structural record of the following form:

[< f > , < V1 , g1 > , < V2 , g2 > , , < V3 , g3 > ,….., <Vn , g3 >] (2.1)

Here: f is the name of the frame, the pair <Vi, gi> is the i-th slot, where VI is the name of the slot, and gI is its value. The value of a slot can be almost anything (numbers or mathematical relationships, texts in natural language, programs, rules of inference or references to other slots of a given frame or other frames, etc.).

Frames are often divided into two groups: ***frame-descriptions*** and ***role frames***. Let us consider a number of examples.

For example, a frame-description of knowledge about cargoes of several types of fruit received on the base can be written as follows:

[<Fruit>, <grapes, Bulgarian 20 t>, <apples, Jonathan 10 t>, <cherry, Vladimirskaya 200 kg>].

An example of a role frame can be a set of knowledge about the goods being carried:

[<Transport], <what, rent 300 t>, <from where, Krivoy Rog>, <where, Odessa>, <what, by rail>, <when, in December 2017>].

In the role frame, the names of the slots are interrogative words, the answers to which are the values of slots. If in the examples in the general expression for the frame remove all the values of the slots, leaving only the names, we get a design, called ***protoframe*** (prototype frame). Frames with specific values are called instance frames.Let's consider examples of these types of frames.

Example of ***protoframe***:

[<Staff List>, <last name (value of slot 1)>, <year of birth (slot 2 value)>, <specialty (value of slot 3)>, <experience (value of slot 4)>].

Example of an instance frame:

[<List of employees>, <surname (Popov-Sidorov-Ivanov-Petrov)>, <year of birth (1965-1968-1987-1958)>, <specialty (locksmith-turner-plumber)>, <experience (5 -21-32-23)>].

Frames have the property of nesting. In these cases, the slot name can be a system of names for slots of a deeper level. The nesting property, the ability to have, as slot values, links to other frames and to other slots of the same frame provide framing languages with the satisfaction of the structuredness and connectivity requirements of knowledge [11].

A frame slot can contain not only a specific value, but also the name of a procedure that allows it to be computed from a given algorithm, as well as one or more products (heuristics) by which this value is determined. A slot can contain not one but several values. Sometimes this slot includes a component called ***a facet***, which specifies a range or a list of its possible values. The facet also indicates the boundary values of the slot filler.

In addition to the specific value in the slot, procedures and rules can be stored that are called when this value needs to be calculated. Among them are ***procedures-demons*** and ***procedures-servants***. The first are started automatically when some condition is met, and the second ones are activated only on special request. If, for example, the frame describing the person includes the slots <BIRTHDAY> and <AGE>, and in the first of them there is a certain value, then in the second slot there can be a name of the procedure-demon calculating age by date of birth and current date and activating Every time the current date is changed.

Thus, taking into account the possibility of inheritance, the structure of the frame data can look like this (see Figure 2.14).

The frame name is the identifier assigned to the frame. A frame must have a unique name in the system. The frame consists of slots.

Slots in a frame can be any number. Some of them are determined by the system itself for performing specific functions, and the rest are determined by the user. These include the IS-A slot, which shows the frame-parent of the frame, the slot of child frame pointer, which is a list of the pointers of these frames, the slot for entering the user name, the date of the change, the text of the comment and other slots. Each slot, in turn, is also represented by a specific data structure.

The slot name is the identifier assigned to the slot; The slot must have a unique name in the frame. Some slots are called system slots and are used when editing the knowledge base and controlling the output.

The indexes of inheritance concern only frame systems of the hierarchical type, based on the relations "abstract - concrete". They show what information about the attributes of slots in the top-level frame inherits slots with the same names in frames of the lower level. Typical indexes of inheritance are shown in Fig. 2.15.

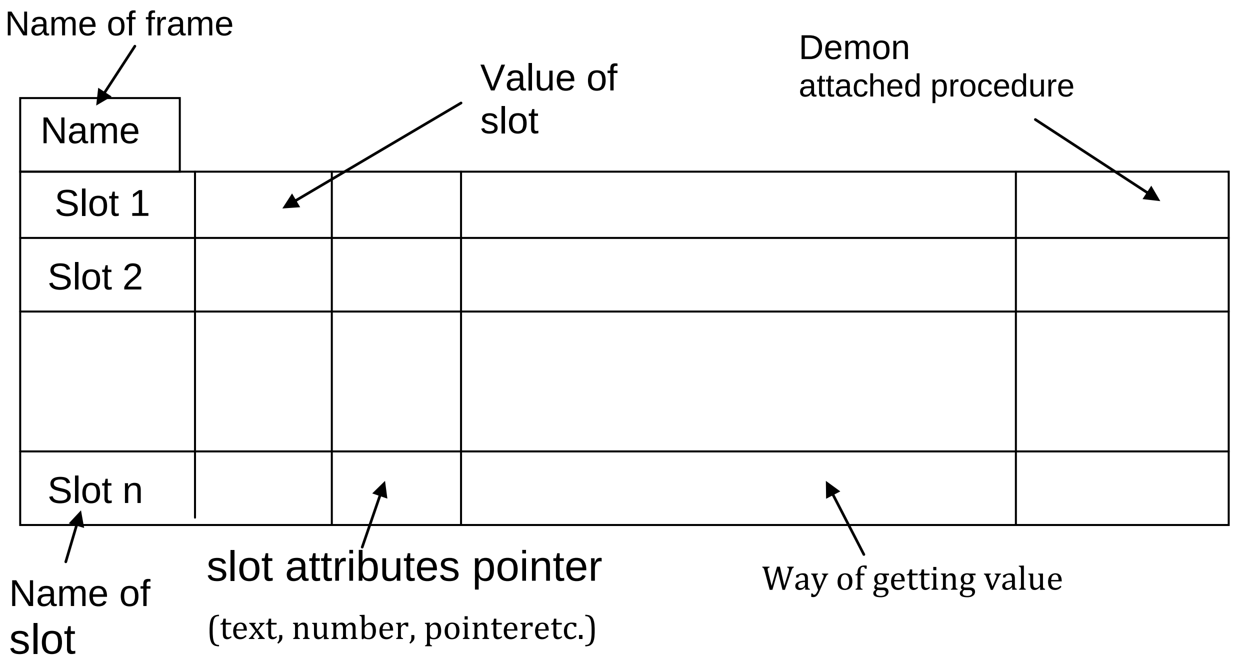


Figure. 2.14. Structure of data placement in a frame

- ***U*** (The first letter of the word Unique is unique) - each frame can have different slots with different values;
- ***S***- all slots must have the same values;
- ***R***- the values of the lower-level frame slots should be within the limits indicated by the values of the top-level slots;
- ***O***- if there is no indication, the value of the upper-level frame slot becomes the value of the lower-level frame slot, but in the case of determining the new value of the lower-level frame slots, they are indicated as slot values.

Figure 2.15. Basic indexes of inheritance

A pointer with the name O performs simultaneously the functions of pointers with the names U and S. Although most systems allow for several variants of indicating inheritance, there are many and such that only one option is allowed. In this case, you can assume that you use the default pointer O. An example of using pointers is shown in Fig. 2.16.

The arrows in the figures indicate the values returned after the call.

It should also be noted that the presence of framing names and slots names in formed structures means that the values stored in frames have the character of references and thus are internally interpreted. The possibility of placing orders for calls to one or another procedure for execution (so-called ***demons***) as slots allows you to activate programs based on existing knowledge.

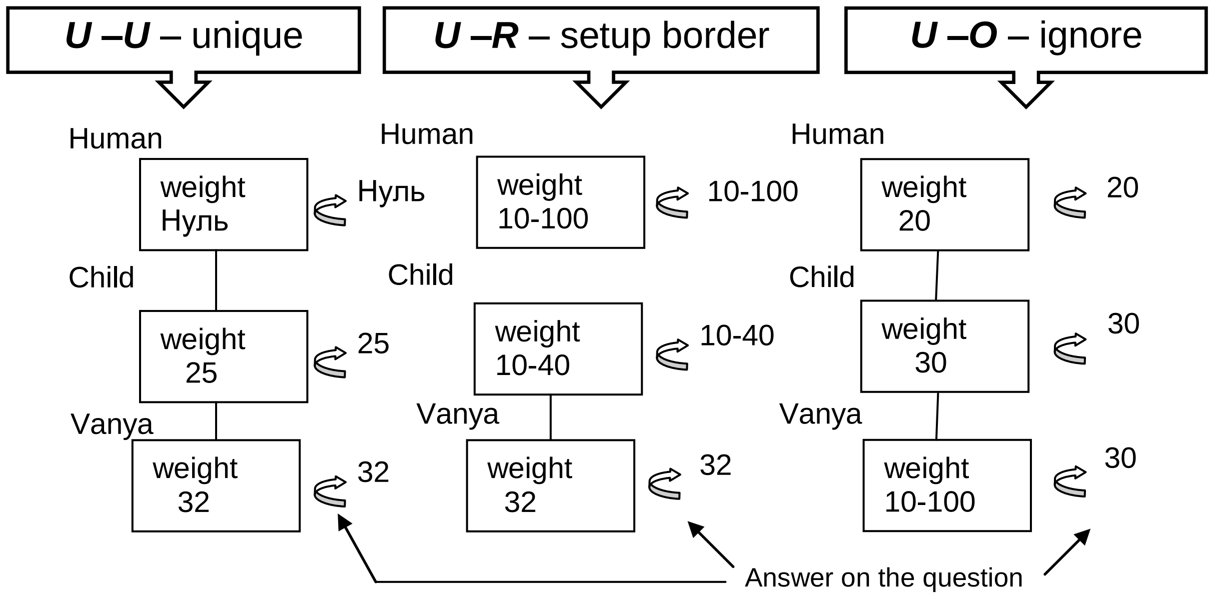


Figure 2.16. Indexes of inheritance (U, R, O) and answers to access to the value of the slot.

Thus, the frame-based languages satisfy four basic characteristics of knowledge - an interpreted, structured, connectivity and activity. Using frames in fundamental sciences enables the formation of a more rigorous conceptual apparatus and the integration of conventional models with frame formalisms. For descriptive sciences, frames are one of the few ways of effective formalization for the creation of a conceptual apparatus.

2.3. Logical models of knowledge.

Logical models of knowledge are the basis of human reasoning and inferences, which, in turn, can be described by suitable logicalcalculations[6]. Such calculus should be primarily attributed to Aristotle, as well as the application of the propositional calculus and predicate, which is axiomatic, and is used as a logical model of knowledge.

After more than 2000 years of unaltered state, the syllogistics of Aristotle received development and important practical application in works on artificial intelligence.

Logical calculations can be represented as formal systems in the form of a four:

6Logical calculations are the theory of formal logical computations. This theory is also called mathematical or formal logic. Historically, logical calculi were developed for the theoretical formalization of the process of proof in various theories.

$$\mathbf{M} = < \mathbf{T}, \mathbf{P}, \mathbf{A}, \mathbf{B} > \qquad (2.2)$$

Where: **T***is the set of base elements* of a different nature, for example, words from some limited vocabulary, letters of some alphabet, details of the child's constructor, included in some set, etc.It is important that for a set **T** there is some way of determining whether or not an element x belongs to this set.The procedure of a verification can be any, but for a finite number of steps it should give a positive or negative answer to the question whether x is an element of a given set **T**.We denote this procedure by **P (T)**.

The set **P** is the *set of syntactic rules* on the basis of which the correctly constructed formulas are constructed.For example, from the words of a limited dictionary syntactically correct phrases ($\mathbf{X_1}$, $\mathbf{X_2}$, ..., $\mathbf{X_N}$) or from the details of the child's constructor, new constructions ($\mathbf{X_1}$, $\mathbf{X_2}$, ..., $\mathbf{X_N}$) are assembled with the help of nuts and bolts.The existence of a procedure **P** (**P**) with the help of which in a finite number of steps it is possible to get an answer to the question of whether any of them, for example, the set $\mathbf{X_k}$, is syntactically correct.

A*is a set of correctly constructed formulas* (CCF), whose elements are called **axioms**.As for other components of the formal system, there should be a procedure ***P*** (**A**), with the help of which for any syntactically correct set it is possible to obtain an answer to the question of its belonging to the set **A**.

The set **B** is the **set of inference rules**, which from the set **A** make it possible to obtain new correctly constructed formulas– ***theorems***. The recent rules can again be applied to the latter from **B**.Thus is formed a plurality of output of the formal system in aggregates.If there is some procedure ***P***(**B**) by means of which it is possible to determine for any syntactically correct collection whether it is output, then the corresponding formal system is said to be ***solvable***.This shows that it is the inference rules that are the most complex part of the logical model.

For the knowledge entering into the knowledge base using the logical model, we can assume that the set **A** form all the information units that are entered into the knowledge base from the outside, and with the help of the derivation rules new ***derived*** knowledge is derived from them. In other words, the formal system is the generator of the ***generation of new knowledge***, which form the set of knowledge deduced in the given system knowledge. This property of logical models makes them attractive for use in knowledge bases. It allows you to store in the database only that knowledge that form a set of axioms **A**, and all other knowledge to obtain from them according to the rules of inference.

Examples of a formal system are the *propositional calculus* and *predicate calculus*, which are considered in the third and fourth sections.

2.4. Production models

The production model of knowledge– it is a model based on rules allowing present knowledge in the form of sentences like "If (condition) then (action)."It is a fragment of the semantic network and is based on temporary relationships between the states of objects.

The production model has the disadvantage that when a sufficiently large number (several hundred) of products is accumulated, they begin to contradict each other due to the irreversibility of disjunctions. In this case, the developers begin to complicate the system, including the modules of fuzzy inference or other means of conflict resolution, - priority rules, depth rules, heuristic exclusion, return mechanisms, etc.

Models of this type use some elements of the logical and network models described above. From the logical models the idea of the rules of inference is borrowed, which here are called ***products***, and from network models - the description of knowledge in the form of a ***semantic network***. As a result of applying the output rules to the fragments of the network description, the semantic network is transformed by changing its fragments, building up the network and excluding unnecessary fragments from it.Thus, in the production models, procedural information is explicitly identified and described by other means than the declarative information. Instead of logical inference, characteristic of logical models, in the production models there is ***a conclusion on knowledge***.

The products, on the one hand, are close to logical models, which allows them to organize effective withdrawal procedures on them, and on the other hand, allow more clearly reflect knowledge than classical logical models. They are no strict limitations of logical calculations, which makes it possible to change the interpretation of the elements of production.

In general, a ***product*** is an expression of the following type:

$$\mathbf{(i); Q; P; A \Rightarrow B; N} \qquad (2.3).$$

Here **i** is the name of the product, with which the given product stands out from the whole set of products.

A name can be a certain ***lexeme***, reflecting the essence of this product (for example, "buying a book", "locking code"), or the serial number of products in their set stored in the system's memory.

Element **Q** characterizes the scope of products. Such spheres are easily distinguished in the ***cognitive***[7]structures of humans. Our knowledge is sort of "laid out on the shelves". On one "shelf" are stored the knowledge of how to prepare food, on the other - how to get to work, etc. Separation of knowledge into specific areas allows you to save time on finding the right knowledge. The same division into spheres of the database system of intellectual knowledge is appropriate and when used for the representation of knowledge production models.

The main element of the product is its core: **A ⇨ B**.The interpretation of the product core can be different and depends on what stands to the left and to the right of the ***sequence*** sign ⇨. The usual reading of the product core looks like this: IF **A**, then **B.** More complex kernel constructions allow on the right side an alternative choice, for example, IF **A**, TO $\mathbf{B_1}$, OTHER $\mathbf{B}_2$.

8Cognitive - (Latin cognitio - perception, cognition) refers to cognition, to the functions of the brain that provide the formation of concepts, operating them and obtaining output knowledge. In psychology, the term "cognition" means the ability to acquire knowledge and its processing.

Sequence can be interpreted in the usual logical sense as a sign of logical follow-up of element **B** from true **A** (if **A** is not a true expression, then nothing can be said about **B**). Other interpretations of the product kernel are possible, for example, **A** describes some ***condition*** necessary for performing action **B**.

The element **P***is the condition for the applicability of the product kernel*. Usually **r** is a logical expression (typically a predicate). When **P** is true, the product core is activated.If **P** is false, then the product kernel cannot be used. For example, if the product: "the availability of money; If you want to buy a thing **X**, then pay its cost to the cashier and give it to the seller ", the condition of the kernel's applicability is false, i.e. If there is no money, then it is impossible to apply the core of the product.

Element **N** describes the ***post-condition of the product***. They are updated only if the core of the product is realized. Post-conditions for products describe the actions and procedures that must be performed after the kernel is implemented.

For example, after buying a certain item in the store, you need to reduce the number of items of this type per unit in the inventory of goods available in this store. Execution of the post-condition **N**cannot occur immediately after the realization of the product core.

If a certain set of products is stored in the system's memory, then they form a ***system of products***. In the production system, special procedures for product management must be set up, with the help of which the products are actualized and the choice for performing a particular product.

The basic structure of the production system consists of three main components [11]. The first of these is the set of rules used as a knowledge base, so it is also called ***rule base*** (Figure 2.17). The second component is working memory (or memory for short-term storage), which stores the prerequisites for specific tasks in the domain and the results of the conclusions derived from them. The third component is a ***logical inference mechanism*** that uses rules according to the contents of the ***working memory***.

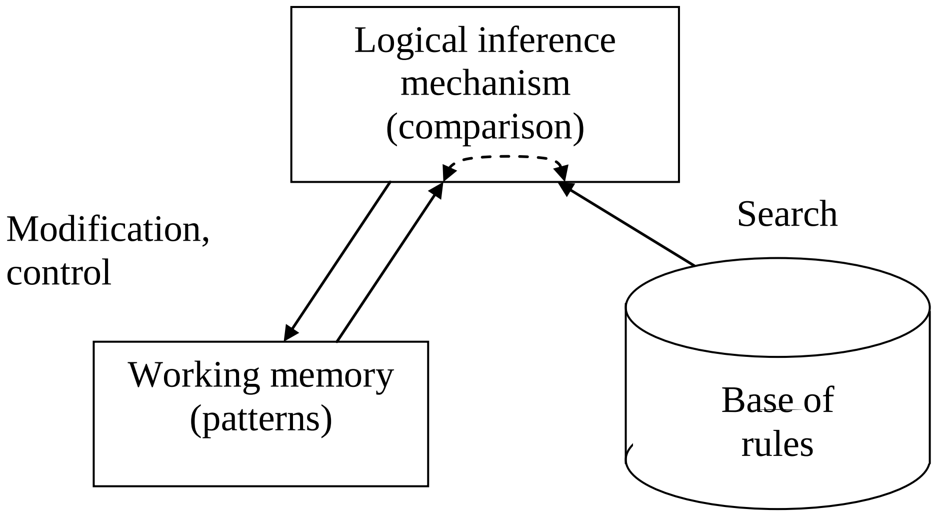

Logical inference mechanism (comparison)
Modification, control
Search
Working memory (patterns)
Base of rules

Figure 2.17.Structure of the production system

In order to show how these elements, interact, let us consider a simple example [8]. The data written to the working memory are samples in the form of a set of symbols, for example, "intention - rest", "resting place - mountains", etc. The rules written to the rules database reflect the contents of the working memory.

In the conventional part of the rules there are either single samples or several conditions connected by the preposition "AND", and in the final part - samples additionally registered in the working memory.

For example, the rules are as follows:

Rule 1. IF "intention is rest" AND
"Road bumpy" THAN "to use a jeep".

Rule 2. IF "resting place - mountains" THAN "road is bumpy".

After the samples "intention-rest" and "rest-place" are recorded in working memory, the possibility of applying these rules is considered. Two approaches are known for obtaining logical inference in the system - ***direct output*** and ***reverse output***.Consider the essence of these approaches.

With the ***direct output***, work is done to extract the pre-recorded contents of the working memory, apply the rules and supplement the data placed in the working memory. First, the output mechanism matches the samples in the conditional part with the samples stored in the working memory. If all the samples are in the working memory, then the conditional part is considered true, otherwise it is false. In this example, the pattern "intention-rest" exists in working memory, and the sample "bumpy road" is absent, so the conditional part of rule 1 is considered false. The conditional part of rule 2 is true. Since in this case there is only one rule with a true conditional part, the output mechanism immediately executes its final part and the sample "bumpy road" is entered into the working memory. If you try to apply these rules again, you can only apply rule 1, since rule 2 has already been applied and dropped out of the number of candidates. By this time, the contents of the working memory were supplemented with a new sample - the result of applying rule 2, so the conditional part of rule 1 becomes true, and the contents of working memory are replenished with a sample of its final part - "use a jeep". As a result, there are no rules that can be applied, and the system stops.

Reverse conclusion is a way of deducing knowledge, in which, on the basis of facts requiring confirmation, to act as a conclusion, the possibility of applying a rule suitable for confirmation is explored. Let's say that the ultimate goal is to "use a jeep," and first investigates the possibility of applying rule 1, which confirms this fact. Since the pattern "intention - rest" from the conditional part of rule 1 has already been entered in the working memory, then to achieve the goal it is enough to confirm the fact that "the road is bumpy". However, if you take a sample of "road bumpy" for a new goal, then a rule is needed to confirm this fact. Therefore, the possibility of applying rule 2 is investigated. The conditional part of this rule at the moment is true, therefore rule 2 can be changed immediately, working memory will be supplemented with a sample of "bumpy road", and as a result of the possibility of applying rule 1, the goal is "to use a jeep".

In the case of a reverse conclusion, the conditions for stopping the system are obvious: either the original goal is reached or the rules that are applicable to the achievement of the goal in the course of the withdrawal end. As for the direct conclusion, then, as was mentioned above, the absence of applicable rules is also a condition for stopping. However, the system also stops when a certain condition is fulfilled, which is satisfied by the contents of the working memory, for example, by checking the appearance of the sample "use a jeep". It is proved that for backward conclusions there is a tendency to exclude from consideration rules that do not directly relate to a given goal, which makes it possible to improve the efficiency of the withdrawal.

Production systems are easy to use and due to the well preparedness of development tools, a large number of systems based on this knowledge model have been created. Production systems, unlike frames and other systems, do not have such functions as, for example, the establishment of high-level relationships between frames, but, on the other hand, thanks to this, the design of the system and its creation are simplified. In the future, probably, not only purely production systems will find use, but also combinations of them, for example, with frame systems.

3. Basics of propositional logic.

3.1. Introduction to propositional logic

A ***statement*** is a sentence expressing a proposition. If the proposition (the result of the argument), make up the content (the meaning) of a ***sentence*** is true, and this ***statement*** saying that it is true

Reasoning is a chain of interconnected conclusions. Depending on the nature of the relationship, reasoning can be either ***inductive*** or ***deductive***. If the relationship of reasoning is built on induction, then the argument is ***inductive***. Consider these concepts.

Induction is a method of reasoning from particular facts and positions to general conclusions, that is, it is an inference made on the basis of concrete facts and leading to a certain hypothesis, i.e. to the ***approval*** of generalizing abstractions, typical for all together and each separately of the facts and providing for a simultaneous discussion of the criteria for justifying the reliability of the obtained hypothesis.

To understand the meaning of the induction use, let us consider an example [13].

We construct graphs of two equations with two variables, for example:

$$x\text{-}2y + 4 = 0 \text{ and } 2\,x + y\text{-}5 = 0.$$

Convinced that the graphs of these equations in the Cartesian coordinate system are straight lines, we can conclude by induction that the graphs of any equation of the form $ax + by + c = 0$ in the same coordinate system will be a straight line. This conclusion is correct. But in order to finally verify its reliability, it is necessary either to sort through all possible x and y coordinates in combination with all possible values of the coefficients, which is unrealistic, or to come up with some more ingenious and effective method of proof. The search for effective methods of proving the reliability of reasoning is an obligatory section in the structure of the theory of any mathematical logic.

In deductive reasoning, which is otherwise called ***calculus***, first, the relationship of reasoning is based on deduction, secondly, the links of the chain of reasoning are additionally connected by the relation of logical sequencing.

Deduction (***conclusion***) is a method of reasoning, in which the new position is derived purely logical way from general positions to particular conclusions. The beginning (premises) of deduction is a certain set of generalizing abstractions[8], and the end (conclusion) is the goal of reasoning, which is represented as a question-fact or a question-abstraction of a minimal scale. Here are a few examples.

Example 1. All people are mortal. Socrates is a man. Therefore, Socrates is mortal (conclusion).

Example 2. Every natural number whose sum of digits is divided by three is itself divisible by three. The sum of the digits of the number 4635 is divided by three. Consequently, the number 4635 is divided by three.

As in the case of inductive reasoning, here, also, in order to verify the validity of the reasoning, it is necessary to verify, that is, prove that the sum of the digits of the number 4635 is divided by three. This is a simple case of proof, however, it is necessary in any deductive reasoning. In this example, there is no doubt about the first premise, the truth of which is proved in number theory.

Example 3. The graph of an equation of the form $ax + by + c = 0$ in a rectangular Cartesian coordinate system is a straight line. The graph of the equation $by + c = 0$ is a straight line parallel to the x-axis. Therefore, the graph of a straight line parallel to the y axis is the equation $ax + c = 0$.

The essence of the relation of logical implication is that the truth of the conclusion follows only from the simultaneous truth of all and only all premises with all possible variants of the replacement of abstractions by facts. The number of parcels of N can be large (in any case, more than ten), and each of the premises is a general abstraction, the truth of which can be clarified only after its replacement by a concrete fact, the possible number (P) of which exceeds ten and sometimes reaches hundreds of thousands. It is practically impossible to ascertain the simultaneous truth of all premises with all possible variants of replacing parcels with facts. It is proved that this problem has the computational complexity of the NP-complete task[10] [5]. In this connection, deductive reasoning is always supplemented by formal so-called inference rules that allow one to simplify it step by step by step-by-step application of it to the deduction itself without disturbing the logical relation, ultimately leading to an obvious scheme, the truth of which can not be established. These rules are called tools for analyzing the validity of reasoning.

9Abstraction - the principle of ignoring minor aspects of the subject in order to highlight the principal. Abstract (in abstraction) identifies the essential characteristics of an object that distinguish it from all other kinds of objects and thus clearly defines its limits in terms of conceptual observer.

Human intellectual activity distinguishes between logical and analytical thinking. Logical thinking is the possession of inductive and deductive reasoning. Analytical thinking is the skill:

1) abstract the phenomena observed in the real world, processes, objects;
2) establish the cause-effect relationship between the objects, processes, phenomena, their properties and characteristics reflected;
3) structuring complex processes for the purpose of their knowledge on the basis of a systematic approach;
4) determine the criteria for comparing the reflected objects, processes, the phenomena of their properties and characteristics;
5) to compare objects, processes, phenomena, their properties and characteristics on the basis of given criteria;
6) to define and abstract the operating environment in which objects, phenomena, processes;
7) quantify the quality of the functioning of processes, objects and systems of objects;
8) mathematically simulate the reflected processes, phenomena, objects and their systems.

Each of the functions of analytic thinking can be easily realized by logical thinking, i.e. Logical thinking can be used as a model of analytical thinking.

By definition, artificial intelligence models logical and analytical thinking. Logical thinking, in turn, models analytical thinking. Consequently, logical thinking is a universal model of artificial intelligence.

3.2. Mathematical logic and its connection with logical thinking.

Mathematical logic is a functionally complete set of formal tools designed to model and implement inductive and deductive reasoning [7,12]. These tools include the following elements (see Figure 3.1):

1) ***formal languages of logic***, intended for statement of intellectual problems and knowledge representation (modeling) in axiomatic theories of any subject areas;
2) ***mathematical structures*** for representing (modeling) knowledge systems, equivalent transformations of knowledge systems, manipulating knowledge and formulating new knowledge;
3) ***formal rules*** of inference and algorithms as a tool for substantiating the reliability of reasoning, i.e. Proof of the truth of the conclusions in the argument.

Several logical systems are known in the foundations of classical mathematics, but one of them is justified as a foundation of mathematical logic: *first-order logic,* which also has other names – *the logic of predicates, the predicate calculus*. Therefore, for mastering the basics of artificial intelligence, it is necessary to thoroughly study the theoretical foundations of the logic of predicates, which are set forth in the next section of the manual.

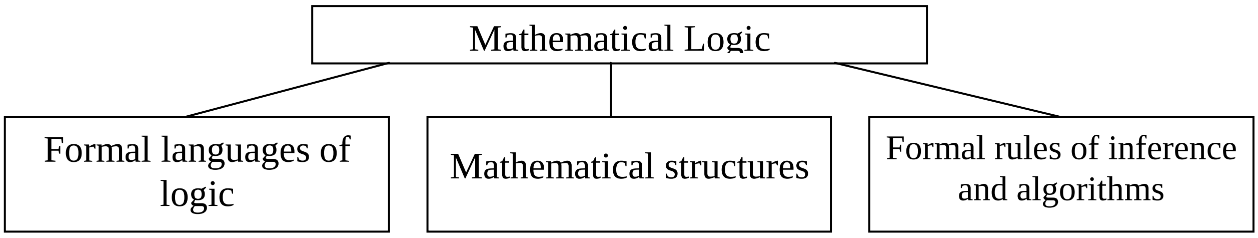

Mathematical Logic
Formal languages of logic
Mathematical structures
Formal rules of inference and algorithms

Figure 3.1. Components of mathematical logic.

The main advantage of using predicate logic for representation of knowledge is that a powerful output mechanism possessing well-understood mathematical properties can be directly programmed. In addition, you need to know what other logic systems complement the logic of predicates and in what situations you should resort to their help. You should also know about graphical representations of predicate logic – semantic networks and frame networks, in particular about their advantages and disadvantages in comparison with the logic of predicates.

There are six aspects that uniquely and functionally fully define calculus as a mathematical structure:

1) the alphabet of the calculus;
2) the rules of language formation in the alphabet – the syntax of the language;
3) the rules for assigning truth values to formulas – the semantics of the language;
4) rules of inference in calculus, i.e. Rules that determine the correct (preserving the relation of logical sequence) transition from one theorem to another, on the one hand, and formalizing certain standard methods of reasoning – on the other;
5) rules for the equivalent transformation of formulas of the calculus;
6) algorithms that effectively recognize the reliability of reasoning.

Historically, the predicate calculus has grown from the logic of utterances, which is an integral part of the logic of predicates. Therefore, at start, briefly consider the *logic of utterances* in these aspects, and then proceed to the *logic of predicates*.

3.3. Propositional logic basics

3.3.1. Alphabet of propositional logic.

Elements (symbols) of the alphabet are:

- utterances denoting lower case letters;
- five logical connectives (Table 3.1)

An utterance is a narrative (affirmative) sentence, about which it is possible to say whether it is true or false (True or False, True or False).

Table 3.1.

Elements and symbols of propositional logic

Name of the bundle	Designation	Type	Other designations
Negation	⌉	Unary	$\overline{\ }$, ~ , ***not***
Conjunction	$\wedge$	Binary	& , , ***and***
Disjunction	$\vee$	Binary	/ , ***or***
Implication	$\supset$	Binary	$\Rightarrow$, $\rightarrow$, ⇨
Equivalence	$\equiv$	Binary	⇔ , $\leftrightarrow$

Here some examples of statements [13]:

1. It's raining. This sentence is a statement, and it can be replaced (identified), for example, by the letter *q*.
2. The road is wet. This is also a statement, and it can be identified by the letter *r*.

In the future, the process of identifying (naming) a particular statement with a particular lowercase letter will be denoted by the symbol

$\rightleftharpoons$ (equally by definition).

3. *p* $\rightleftharpoons$ Snow white. *H* $\rightleftharpoons$ The president resigns.
4. *e* $\rightleftharpoons$ The earth spins. *n* The $\rightleftharpoons$ length of the circle is equal to its diameter.

It should be noted that in the statement the relationship (relationship) of the reflected object with its properties or characteristics, or any actions of the object, or the interrelation of two or more reflected objects is represented. This is important for understanding the semantics of the term "predicate". For example:

5. *k* $\rightleftharpoons$ Side *a*, side *b*and side *c*are sides of an isosceles triangle *T*.

There are two agreements in the logic of the statement:

1) The conclusion about the truth (*T*) or falsity (*F*) of a particular utterance is given by the subject modeling the real world, when setting and solving a certain problem. The semantics (semantic meaning) of the reflected utterance and the idea of its truthfulness is completely determined by the model of the world of this subject. It is understandable that different subjects can have or form different models of the real world.

2) The basis of the logic of utterances is deductive reasoning, therefore it has another name – the "calculus of statements".

3.3.2. The Rules for the formation of the language in the alphabet (the syntax of the language).

To describe the rules, we introduce the notion of a ***metacharacter***. A metacharacter is a designation that does not belong to the language, which allows you to enter concepts and the properties of this language, as well as specify the order in which must apply with the rules of the language.

So introduce four metacharacters to illustrate the given rules with examples:

1) ***x*** **2)** ***y*** **3) (4))** (3.1) .

The metacharacters *x* and *y* will serve to denote formulas, and the brackets "(" and ")" indicate the order in which the rules are applied.

The rules for the formation of a language in the alphabet are as follows:

The basic rule: every utterance is a formula.The metacharacters x and y will serve to denote formulas, and the brackets "(" and ")" indicate the order in which the rules are applied.

The rules for the formation of a language in the alphabet are as follows:

The basis rule: every utterance is a formula.

The rule of the induction step: if *x* and *y* are formulas, then $\rceil x,\ (x \wedge y), (x \vee y), (x \supset y), (x \equiv y)$ are also formulas.

Rule of restriction: formulas can be formed only by the rules 1 and 2.

There are no other rules.

How do the metacharacters of parentheses indicate the order in which the rules are applied?

Let's consider an example.

Suppose that$x = (p \wedge (q \vee r))$.In constructing the formula x, the induction step rule was applied twice: the first time when the formula$(q \vee r)$was constructed from the formulas *q* and *r*, and the second time when constructing the final formula from the formulas*p*and $(q \vee r)$.These rules for the formation of a language in the alphabet are used to represent compound arbitrarily complex sentences.

Formulas of language are divided into *atoms* or atomic formulas and *formulas* (without epithets), to which all compound formulas refer, i.e. Formulas formed with the help of connectives, and atoms are indivisible (initial) statements.

3.3.3. The rules for assigning truth values to formulas (semantics of language)

By definition, an atom can have only two values: either "true" (T) or "false" (F). Each of these values is called a truth value. The rules for assigning truth values to the formulas of five bundles are presented in Table. 3.2.

Table 3.2.

Truth table

x	y	$\rceil$x	x ∧ y	x ∨ y	x ⊃ y	x ≡ y
T	T	F	T	T	T	T
T	F	F	F	T	F	F
F	T	T	F	T	T	F
F	F	T	F	F	T	T

We introduce a number of new concepts that will be needed in considering the following aspects of the propositional calculus.

1) "interpret the formula" – assign to it one of the two values of the truth of *T* or *F*.

2) "interpretation for a formula" is a set of truth values of all atoms entering into a formula intended for simultaneous replacement of the atoms themselves in this formula. A formula containing *K* of various propositions admits 2^k interpretations.

Let us illustrate these two concepts by examples of the interpretation of some formulas (Table 3.3).

Table 3.3.

Interpretation of some formulas

x y z	$y \supset z$	$x \supset (y \supset z)$	$x \wedge y$	$(x \wedge y) \supset z$	*w*	$\urcorner w$
TTT	F	T	T	T	T	F
T T F	F	F	T	F	T	F
T F T	T	T	F	T	T	F
T F F	T	T	F	T	T	F
F T T	T	T	F	T	T	F
F T F	F	T	F	T	T	F
F F T	T	T	F	T	T	F
F F F	T	T	F	T	T	F

The semantic elements of the table are as follows.

– first three columns of each line are one of the possible interpretations.

–"semantics of language" is a complete set of rules for interpreting formulas, i.e. This is the whole table 3.2.

–"the general validity of the formula" is the truth of the formula for all its possible interpretations; *w* in table. 3.3).

–"the inconsistency of the formula" (impracticability) is the falsity of the formula for all its possible interpretations - $\urcorner w$ in table 3.3

–"equivalence of formulas" - the formulas *x* and *y* are equivalent when the truth values *x* and *y* coincide with each common interpretation for *x* and *y*.

–"literal" is an *atom* or its negation; *atom* in mathematical logic is the simplest case of the formula, i.e. a formula that can not be decomposed into subformulas.

–"disjunction of formulas" is the formula *X*, formed from the initial formulas F1, F2, ..., Fn with the help of a ***disjunctive*** (and only) bunch:

$$X = F_1 \vee F_2 \vee ,... \vee F_n. \qquad (3.2)$$

–"conjunction of formulas" is a formula *Y* formed from the initial formulas F1, F2, ..., Fn with the help of a ***conjunctive*** (and only) bunch:

$$Y = F_1 \wedge F_2 \wedge ,... \wedge F_n. \qquad (3.3)$$

– "disjunct" is the formula *Z*, formed from the source letters (and only the letter) with the help of a ***disjunctive*** (and only) bundle:

$$Z = A_1 \vee A_2 \vee A_3 \vee ,... \vee An. \qquad (3.4)$$

The equivalent of a disjunct is the set of letters it contains:

$$Z = A_1 \vee A_2 \vee ,... \vee A_m \equiv \{A_1 , A_2 ,..., A_m\} \qquad (3.5)$$

–"R - literary disjunct" is a disjunct, in which R of letters.

–"single disjunct" is a disjunct with one letter.

–"empty disjunct" is a disjunct, in which there are no letters. Because It does not contain letters that could be true for some interpretation, it is always false.

–"scope of logical connectives" - for non-skid recording this area is ordered in descending order and corresponds to the sequence $\equiv , \supset , \wedge , \vee , \rceil$.

–"disjunctive normal form" is a formula:

$F= F_1 \vee F_2 \vee ,... \vee F_n$, где F_i – conjunction of letters.

–"conjunctive normal form" is a formula:

$F=F_1 \wedge F_2 \wedge ,... \wedge F_n$, где F_i – disjunction of letters.

–"satisfiable formula" -the formula is feasible if and only if there is at least one interpretation for which this formula is true. This interpretation is called the formula's model.

–"contrarian pair of formulas" is the set {A, A}

–"tautology" is a universally valid formula, true in all its interpretations. Disjunction, which contains a contrarian's of pair, is a tautology.

–"conjunctive normal form (CNF)" in boolean logic is a normal form in which the Boolean formula has the form of a conjunction of disjunctions of literals. A conjunctive normal form is convenient for the automatic proof of theorems.

–"reduced CNF" is the CNF, from which tautologies and repetitions of characters within the same disjunct have been removed.

3.3.4. The rules of inference in the propositional calculus (stereotypes of deductive reasoning).

Stereotypes have been developed over many decades, and some for many hundreds of years, and allow for the correct, i.e. Without disrupting the logical sequence relationship, transitions from one theorem to another, with the aim of bringing the structure of reasoning to the canonical form (to the reduced CNF). The essence of the canonical form will be disclosed in subparagraph **3.3.5**, and now, before discussing the rules of inference, we introduce a number of new concepts without which this discussion is impossible.

The notion of "relation of logical following".

Formula G is a logical consequence of the formulas F_1, F_2, ..., F_n if and only if for every interpretation I, wherein ($F_1 \wedge F_2 \wedge ,... \wedge F_n$.) – truthis also true. FormulasF_1, F_2, ..., F_n called **premises**, and G - finding with respect to logical inference.

Next stop on the concept of "necessary and sufficient conditions of logical consequence (investigation)."

The formula G is a logical consequence if and only if $F_1, F_2, ..., F_n$, when the formula $((F_1 \wedge F_2 \wedge ,... \wedge F_n.) \supset G)$ – is valid, or when formula $(F_1 \wedge F_2 \wedge ,... \wedge F_n \wedge \neg G)$ –) is contradictory

The concept of <u>"theorem of deductive reasoning."</u>

If a formula G is a logical consequence of the formulas$F_1, F_2, ..., F_n$, **the formula** $((F_1 \wedge F_2 \wedge ,... \wedge F_n\ ) \supset G)$ is called a theorem, and G is called the conclusion of the theorem.

The concept of <u>"evidence in deductive reasoning"</u> (ie, calculus).

The proof is a reasoned justification of how the conclusion in the theorem logically follows from its premises. The proof is presented in the form of an ordered sequence (trace) of inferences, as a result of which the truth value of the conclusion is established.

Now for the <u>rules of inference</u> in the propositional calculus.

To denote the logical sequence relationship, we introduce a new metacharacter —— (horisontal bar)

We will use this metacharacter to separate the theorems-premises and theorems-conclusions. Above the line we will write down a list of theorems-premises, below the line – a theorem-conclusion. This form of writing theorems will indicate that the conclusion theorem is a logical consequence of the premise theorems.

Metacharacter**;**(semicolon) will be used as a separator in the list of theorems..

Metacharacter, (comma) will be used as a separator of the premises within the theorem simulating a conjunctive bundle.

Metacharacter$\square$ (rectangle) will be used to denote contradictory formulas.

Consider the following example of using the logical sequence relation:

If the barometer falls, then there will be bad weather;

$$\frac{\text{The barometer falls}}{\text{There will be bad weather}}$$

The list of inference rules is as follows:

$$1)\ \frac{D\supset\Phi\ ;\ D\supset\Psi}{D\supset\Phi\wedge\Psi}\qquad 2)\ \frac{\supset\Phi\wedge\Psi}{D\supset\Phi}\qquad 3)\ \frac{D\supset\Phi\wedge\Psi}{D\supset\Psi}\qquad 4)\ \frac{D\supset\Phi}{D\supset\Psi\vee\Psi}$$

$$5)\ \frac{D\supset\Psi}{D\supset\Phi\vee\Psi}\qquad 6)\ \frac{D,\Phi\supset\Psi;\ D,X\supset\Psi;\ D\supset\Phi\vee X}{D\supset\Psi}\qquad 7)\ \frac{D,\Phi\supset\Psi}{D\supset\Psi\supset\Psi}$$

$$8)\ \frac{D\supset\Phi\ ;D\supset\Phi\supset\Psi}{D\supset\Psi}\qquad 9)\ \frac{D,\neg\Phi\supset\ \square}{D\supset\Phi}\qquad 10)\ \frac{D\supset\Phi;D\supset\neg\Phi}{D\supset\ \square}$$

$$11)\ \frac{D,\Phi,\Psi,\ D_1\supset X}{D,\ \Psi,\Phi,\ \ D_1\supset X}\qquad 12)\ \frac{D\supset\Phi}{D,\ \Psi\supset\Phi}$$

<u>Rules 1, 2 and 3,</u> by exploiting the essence of conjunctive bundles simplify the model of the theorem.

<u>Regulations 4 and 5,</u> exploiting the disjunctive nature of the bunch, allow conclusion of the theorem in the introduction of new additional formulas.

<u>Rule 6</u> forms a method of reasoning "analysis" of the two possible cases. When performing chip Γ valid Φ or X and Ψ is true if the conditions Γ and Φ, as

well as the conditions Γ and X, then Ψ is always true when the chip Γ that is set by considering two possible cases:

A) the conditions D and Φ are satisfied;

B) the conditions T and X are satisfied.

Rule 7 formalizes reception equivalent reformulation of the theorem, which allows one of the assumptions of Theorem placed in confinement in the form of parcels.

Rule 8 - inference rule (department), modus ponens, introduced more Aristotle. It indicates how one can be released from the parcel in custody.

Rule 9 - formalizes' argument to the contrary. "Let condition D and Φ can be made simultaneously. Coming to a contradiction, we conclude that the satisfiability of Γ always implies the validity of Φ.

Rule 10 - the rule "there is a conflict."

Rule 11 - a permutation of the premises does not affect the validity of the conclusion.

Rule 12 - adding an extra parcel, we do not violate the truth of the conclusion of the theorem.

Conclusions: The rules of inference 1 - 12 are functionally complete set of rules, applying that to the initial axioms and theorems are proved, we can finally get the proof of the true value of the conclusion of Theorem - purpose. However, the mandatory requirements of the construction of evidence by inference rules 1 - 12 are as follows:

1) The initial premises must be only axioms and proved theorems

2) Using inference rules 1 - 12 (and only These rules) can be build Compositions Axioms and Proven theorems, aiming at The end get Conclusion Theorem goals. These two Demands Are consistent Stereotype Classical Deductive reasoning, sometimes Called straight deduction, that is. e. Reasoning from True Abstract premises (axioms and Proven theorems) to True Specific conclusion. Consider Examples Non-compliance Of the requirements:

①a) All metals are elements;
б) Bronze – metal
Bronze – is an element

The conclusion is wrong: bronze is not an element. Here, the law of identity is violated, which prohibits in the process of this conclusion in the same concept to invest different content. In the premises of a) and b) The used metal is not in the same sense. The sending and) metal - is the chemical element, and in sending b) - metal - a substance used in the household, workplace.

②Bacon and Hobbeswere Egyptians
Bacon and Hobbes were idealists
Some idealists were Egyptians

The conclusion in the conclusion is correct, however, both premises are false: Bacon and Hobbes were English and materialists (and not idealists).

Practice has shown that the method of proving theorems by direct deduction is extremely inefficient for the following reason. The structure of the compositions at the intermediate steps of the transformations is accidental due to the fact that there are no criteria and control actions that would purposefully orient these compositions to a given goal. Therefore, the possibility of leaving aside the goal is not excluded, which is often the case in practice. This implies a global conclusion about the ineffectiveness of direct deduction in the proof of theorems. In this connection, rules 1-12 are most often used only to reduce the structure of the theorem to the canonical form.

3.3.5. Rules for the equivalent transformations of propositional calculus formulas.

Equivalent transformations of formulas are necessary in order to reduce the structure of the objective theorem to the canonical form (CF).CF is a reduced normal form with clauses instead of disjunctive formulas and with some other principal features that will be considered later.

If there are formulas in the premises and in the conclusion that do not meet the specified requirements, then they must be correctly replaced by equivalent formulas. This allows you to define rules for equivalent transformations (Table 3.4), where two new metacharacters are used.

$\boxtimes$ (envelope) - for Designations Universally valid Formulas

=(equality) - to indicate equality of formulas.

Table 3.4.

Equivalent transformation rules

№ rules	The rules of equivalent transformations of formulas Propositional calculus
1	$F \equiv G = (F \rightarrow G) \wedge (G \rightarrow F)$
2	$F \rightarrow G = \rceil F \vee G$
3	a. $F \vee G = G \vee F$; б. $F \wedge G = G \wedge F$
4	a. $(F \vee G) \vee H = F \vee (G \vee H)$; б. $(F \wedge G) \wedge H = F \wedge (G \wedge H)$
5	a. $F \vee (G \wedge H) = (F \vee G) \wedge (F \vee H)$; б. $F \wedge (G \vee H) = (F \wedge G) \vee (F \wedge H)$
6	a. $F \vee \square = F$; б. $F \wedge \boxtimes = F$
7	a. $F \vee \boxtimes = \boxtimes$; б. $F \wedge \square = \square$
8	a. $F \vee \rceil F = \boxtimes$; б. $F \wedge \rceil F = \square$
9	$\rceil(\rceil F) = F$
10	a. $\rceil(F \vee G) = \rceil F \wedge \rceil G$; б. $\rceil(F \wedge G) = \rceil F \vee \rceil G$

Example: Based on Table. 3.4 we obtain a disjunctive form for the formula $(P \vee \rceil Q) \rightarrow R$

$(P\vee \rceil Q) \rightarrow R = \rceil (P\vee \rceil Q)\vee R =$ (the rule 2)
$=(\rceil P\vee \rceil (\rceil Q))\vee R =$ (the rule 10a)
$=(\rceil P\wedge Q)\vee R$ (the rule 9).

There are no other rules for equivalent transformations.

3.3.6. Algorithms that effectively recognize the validity of reasoning.

We will not consider the more than a thousand-year history of searching for effective algorithms for proving theorems, and do a comparative analysis of all known algorithms. It suffices to mention only those algorithms that have found wide practical application in artificial intelligence.

The best known algorithm called the **resolution method (resolutive output).** He originated in 1930 (in the works of Erbran), development and formation was received in 1965 (in the works of Robinson), has been widely used since the beginning of the 1970s (in the works of group A. Colmrauer, who applied it on the basis of Prologue for proving theorems) And is used to the present.

There are many options for applying the resolution method, but it has the highest efficiency under the following conditions:

1) when the target theorem is transformed into the opposite theorem;

2) when the target theorem is represented in the structure of the reduced conjunctive normal form (PCNF) with the following principal feature: clauses in PCNF are Horn clauses.

The PCNF satisfying the second requirement is called the canonical form, having the form:

$(P_1 \vee P_2 \vee ...)\wedge(q_1 \vee q_2 \vee ...)\wedge...\wedge(r_1 \vee r_2 \vee ...)\wedge \rceil (C_1 \vee C_2 \vee ...)$, (3.6)

where, P_i, q_j, r_k – - letters parcels and C_t – letters conclusion (the concept of "opposite theorem" and "Horn clause" will be discussed later);

3) when in the proof of the objective theorem a deductive reasoning called reverse deduction is used, which differs radically from the direct (classical) deduction (the reverse deduction will be discussed later);

4) when the contradictoriness of the canonical form of the objective theorem is proved, using the advantage of the PCNF that it becomes contradictory if at least one of its clauses is false. This gives the right to consider the set of its clauses as equivalent to the CPNF, and the verification of the contradictory nature of the CPNF is reduced to verifying the truth of each of the clauses..

A)SUMMARY resolution method (rule 1).

The application of the method of resolutions is based on the fundamental method of deduction: the set of formulas of deductive reasoning is impossible if and only if the lie (A) is a logical consequence of its (this set).

The same property can be formulated differently:

Formula C is a logical consequence of the finite set of formulas E if and only if $E \wedge \rceil C$ is not feasible. This property of deductive reasoning, which follows from its definition, is widely known as the deduction principle.

In the formal representation it looks like this:

$\{ H_1 , H_2 ... ,H_n\} \models C \equiv \{ H_1 , H_2, ... ,H_n, \neg C \} \models F$, (3.7)

Where the metacharacter $\models$ – is the logical sequence relation.

$H_i(i= 1, n)$ – posting a deductive reasoning, C - Finally, in the argument.

On the basis of this property, the impossibility of the set E of disjunctions of the canonical form of the objective theorem of deductive reasoning can be verified, generating logical consequences from E until we obtain an empty clause.

For generating logical consequences we use the following scheme of reasoning. Let A, B and X be formulas.

Assume that the two formulas $(A\vee X)$ and $(B\vee \neg X)$ – true. If the formula X - is also true, $\neg X$ - F, and then we can conclude that B - true.

Conversely, if X is false, then we can conclude that A is false. Thus, in both cases (A □ B) - true.

Typically obtain $\{A\vee X , B\vee \neg X\} \models A\vee B$, which can also be written as:

$\{ \neg X\to A ,X\to B\}A\vee B. \models$ (3.8)

In the particular case where X is a statement, A and B are clauses, this rule is called the rule of resolutions.

B) <u>Summary of the conversion target to the opposite Theorem Theorem (rule 2).</u>

In classical mathematics, the concept of "the opposite theorem" means that the conclusion of the opposite theorem is a negation of the conclusion of the original theorem, and the premises of the opposite theorem are a negation of the premises of the original theorem. For example, A □ B - startes theorem, □ A □ □ B - opposite theorem. Since the negation of a universally valid formula is a contradictory formula, then it is necessary to prove the contradiction, and not the general validity of the opposite formula.

Thus, if the original target theorem has the form:

$(D_1 \wedge D_2 \wedge ... \wedge D_n) \to C$, (3.9)

where D_i - the Horn clause, C is the conclusion of the theorem, then, transforming formula (1) according to rule 2, we obtain a new formula without implication:

$(\neg (D_1\wedge D_2\wedge ... \wedge D_n) \vee C)$. (3.10)

Converting formula (2) into the opposite formula, we obtain the PCNF:

$(D_1 \wedge D_2\wedge ... \wedge D_n\wedge \neg C)$ (3.11)

PCNF (3) is contradictory if a disjunct D_i □ or C - is false.

From this follows a simple rule for checking the formula (3.11) for the contradictoriness: it is necessary to check each element of the set of disjuncts of the formula for its falsity. The highest effectiveness of such verification is provided by the method of resolutions.

Thus, the transformation of the objective theorem into the opposite theorem is necessary to obtain a pure PCNF, which needs to prove its inconsistency.

C) <u>Horn clause and their purpose.</u>

Horn's disjuncts are clauses that contain no more than one positive letter.

Clause ($\rceil p \vee \rceil q \vee \rceil r \vee S$) is equivalent to the implication ($p\vee q\vee r$) $\rightarrow S$.

A universally significant implication is a proved theorem. Therefore, Horn's clauses are used in the canonical form of goal theorems as the equivalents of premises, which are proven theorems. These parcels - theorems are called <u>rules.</u> The rules are widely used in the automatic proof of theorems.

Clause, which reduces to a single positive letters, is <u>a</u> certain <u>fact,</u> that A conclusion that does not depend on any premises. Facts are also widely used in the proof of theorems.

In addition to the proven properties of Horn disjuncts, we note two more of their qualities. The first is that when submitting packages exclusively by Horn's clauses, the method of resolutions necessarily completes the proof, which may not happen in the case of the non-Chernovsky representation of the premises, when the method may become obsessed.

The second is that the computational complexity of the proof of the desired theorem by the resolution method is minimal and does not exceed n2, where n is the number of characters in the canonical form with allowance for repetitions.

D) The essence of the reverse deduction.

The main drawback of direct deduction is that there are no criteria and control actions in it that would direct the trace of the proof to a given goal and would not allow the proof to move away from the goal. In reverse deduction, these shortcomings are eliminated.

Instead of starting from parcels, reverse deduction begins with a conclusion-goal and, applying the sending rules (proved theorems), replaces the current goals with new ones until these new goals turn out to be literary facts. Thus, the procedure of proof becomes the controlled goal of inference at each step of inferences, which will not allow it to go astray from the true path. The path itself turns into a critical one, i.e. In the shortest (examples are already in the predicate calculus, section 4).

E) Rules for proving theorems by the method of resolutions.

The initial object for the proof is the complete set of disjunctions belonging to the canonical form of the target theorem. Denote these sets by the symbol D. We order D by some criteria, thereby making it a numbered list, which we denote by S.

We prove the contradictoriness of the list S according to the following recursive scheme.

We take the target disjunct, standing in the head of the S list, and compare it with all subsequent, after it, clauses. At each step of matching the pairs of clauses, we watch the appearance of the contra-oriented letter pair {A, □ A}. If it does not exist, then proceed to the next clause without performing any actions. If a pair of characters appears in the matching pair of clauses, then by deleting it and generating the resolvent r, we write it to the end of the list S and assign it a number one more than the number of the last element in the original list S. The resolvent r is appended to the list, provided that It is not a tautology and is not absorbed by any clause of the S list. In these cases, it is simply ignored.

After the first headword of the list S is mapped to all subsequent clauses of the initial list S, and the calculated resolvents are put in the tail of the list S, the pair matching procedure is repeated now for the new clause. This new clause becomes the second number of the clause from the now expanded S_p list. Recursively, procedures are repeated for deleting the counter pairs, generating resolutions, assigning resolvents to the tail of the list by analogy with the previous cycle. But the first element in the list S_p does not participate in the comparison, and those resolvents that were generated in the first cycle are involved in this comparison. Recursion with subsequent head elements continues until the current list turns into an empty clause or a set of disparate clauses. If an empty clause appears, then the theorem is proved, otherwise the conclusion is false.

An empty clause is denoted by Λ and means a lie, or a formula that is identically zero.

The effectiveness of the proof depends on the criteria for ordering the list S. The criteria for ordering lists are as follows:

1) first in the list S written clauses of the conclusion in any order;

2) in the second order facts are recorded in any order;

3) in the third order, the rule-clauses are written, but in the order corresponding to the order of the facts, i.e. If the first place is the fact a_1, then the rule, including the fact a_1 in the rules sublist, should also be in the first place.

Let us consider examples of proofs of theorems by the resolution method.

<u>Example 1.</u>

Let us prove the validity of rule 6; Rules for the analysis of possible cases (see § 3.2.4). It is required to prove:

$\{ h, h\rightarrow (p\vee q) , p\rightarrow c , q\rightarrow c\}\rightarrow c$

The set of clauses, the unavailability of which should be established, is as follows:

$\{ h , \urcorner h\vee p \vee q , \urcorner p \vee c, \urcorner q \vee c , \urcorner c\}$

By the method of resolutions, an empty clause is obtained as follows:

1) $\urcorner c$ - denial of the conclusion
2) h - hypothesis
3) $\urcorner h \vee p \vee q$ - hypothesis
4) $\urcorner p \vee c$ - hypothesis
5) $\urcorner q \vee c$ is the hypothesis
6) $\urcorner p$ is the resolvent for 1 and 4
7) $\urcorner q$ is the resolvent for 1 and 5
8) $p \vee q$ is the resolvent for 2 and 3
9) q is the resolvent for 6 and 8
10) p is the resolvent for 7 and 8
11) Λ is the resolvent for 7 and 9.

<u>Example 2</u>

Let us prove the impossibility of the set.

$\{ p \vee q , p \vee r , \urcorner q \vee \urcorner r , \urcorner p \}$

1) $p\vee q$
2) $p\vee r$
3) $\rceil q \vee \rceil r$
4) $\rceil p$
5) $p \vee \rceil r$ is the resolvent for 1 and 3
6) q is the resolution for 1 and 4
7) $p\vee \rceil q$ is the resolvent for 2 and 3
8) r is the resolvent for 2 and 4
9) p is the resolvent for 2 and 5
10) $\rceil r$ is the resolvent for 3 and 6
11) $\rceil q$ is the resolvent for 3 and 8
12) Λ is the resolvent for 4 and 9.

4. The logic of predicates

4.1 Introduction to the logic of predicates

The ***predicate*** (Latin Praedicatuum - said) is the predicate of affirmation, that is, that which is expressed (affirmed or denied) in the judgment of the subject. For example, in the statement "the rocket reached the moon" predicate - "reached the moon." In the logic of utterances, an atom is viewed as an indivisible whole, whose structure and composition are not amenable to external analysis in the process of reasoning. However, there are many thoughts that can not be considered in a simple way. Let us give some examples.

Example 1. In the language of propositional logic, one can not express the fact that from the sentence "at least one student has solved all the control tasks" it follows the conclusion "at least one student has solved each control problem." Each of these statements is a statement, the implication of which is not a tautology.

Example 2. Let $\boldsymbol{a},\boldsymbol{b},\boldsymbol{c}-$the lines in the plane, $\|$ be the parallelism symbol of the lines, then the conclusion of $\boldsymbol{z}:\boldsymbol{a}\|\boldsymbol{c}$ follows from the premises $\boldsymbol{x}: \boldsymbol{a}\|\boldsymbol{b}$ and $\boldsymbol{y}:\boldsymbol{b}\|\boldsymbol{c}$. However, in the logic of statements the theorem $\boldsymbol{x}\wedge\boldsymbol{y}\to\boldsymbol{z}$ is not a universally valid formula.

Example 3. $\boldsymbol{P}$: Everyone is mortal
$\boldsymbol{R}$: Confucius is a man
$\boldsymbol{Q}$: Confucius is mortal.

The conclusion of the theorem R seems obvious, however, in the logic of statements the formula $(P\wedge Q)\to R$ is not a tautology.

Contradictions in the examples resulted from the lack of the possibility to penetrate into the internal structure of each of the utterances, to see in this structure the relationship between the objects in each of them and, most importantly, to display this relationship in the form of a formula.

To eliminate these shortcomings, the logic of predicates is intended, which is an extension of the logic of propositions, i.e. Together with the concepts of the logic of propositions, it contains a number of other concepts that enhance logical thinking.

In the logic of predicates, in addition to the symbols of the logic of the utterance, symbols of reflected objects, symbols of properties and characteristics of these objects, functional symbols, as well as symbols of relations between objects, their characteristics and properties are additionally introduced. In addition, generalizing and concretizing expressions "all" and "some" (the so-called quantifiers) are also introduced, allowing quantitatively to characterize the relationships of objects, properties, characteristics and relationships. Addition of the logic of statements with the specified symbols allows you to explicitly define and manage objects, relations, properties and characteristics of objects, i.e. To exercise external interference in the internal structure of utterances.

4.2. Alphabet of predicate logic.

The symbols (elements) of the alphabet are the predicate logic.

1) Constants (individual symbols). Usually these are object names, property names, characteristics, property values or characteristics;

2) Symbols of objective variables. Usually, these are small letters x, y, z, possibly with indices. The subject variable implies the presence of a domain of its definition, i.e. A finite set of values of a given variable;

3) Functional symbols. Usually, these are small letters f, q, h or meaningful words from lowercase letters, for example, plus, minus, multiply, father, mother, daughter;

4) Predicate symbols. Usually, these are capital letters P, Q, R, or meaningful words from capital letters such as MORE, LESS, LOVE, CONTAINS, DEATH, MAN.

5) Bundles of the propositional calculus ($\rceil \ , \wedge, \vee, \supset, \equiv$)

6) Quantifiers of generality $\forall$ (for all) and existence $\exists$(exists).

In the language of predicates, the language of utterances is also contained, since the predicate becomes a statement after the replacement of its objective variables by their concrete values.

4.3. Rules for the formation of the language in the alphabet (syntax)

The alphabet of the logic of predicates allows us to define new concepts necessary for the language of this logic.

The concept of "attitude". In the logic of predicates, the predicate expresses the mutual relationship of the reflected objects with their properties, characteristics, actions, the interrelation of objects, processes, actions. The number of these items in one respect is not limited. The ratio can be two-seater (i.e., between two items), triple, and generally n-local. A true saying is an attitude. Formally, the most common double relation (binary) is considered as the set of pairs of elements and is denoted by the vector <x, y>, where x is called the first element, and y is the second element of the coordinate of the pair. The vector representation of n - dimensional relations is also widespread.

The term "term" is any constant, an objective variable and a function.

The term "function" symbolizes an action that assigns one defined constant to the list of constants. Every function has a certain number of arguments. If the function symbol f has n-arguments, then f - is called an n-local function symbol. A constant can be treated as a function symbol without arguments.

The designation of the function is f (t_1, t_2, ... t_n), where t_i are terms; For n = 0, the function is simply denoted by f.

Examples:

1) plus (t_1, t_2, t_3) is $t_1 + t_2 + t_3 = t_\Sigma$

2) plus (plus (t_1, t_2, t_3, t_4), minus (t_5, $_tt_6$), multiply (t_1, ..., t_n))

3) father (father (Ivan)), which defines Ivan's grandfather.

A predicate is a generalization of the concept "utterance", consisting of:

1) in abstracting the objects involved in the interrelations of this statement;

2) in ensuring the possibility of external management of the internal interrelations of this utterance with the help of quantifier symbols.

Every predicate has a certain number of arguments. If the predicate symbol P has n - arguments, then it is called an n - local predicate symbol.

The predicate notation is P (t_1, ..., t_n), where t_i are terms; For n = 0 the predicate is denoted by P instead of P ().

The predicate symbolizes an action that assigns one of the truth values of I or A to the correspondence of the list of constants. This means that the predicate (as a propositional form) after replacing the arguments in it - subject variables with concrete constants - turns into a statement that can be either true (I), Or false (L). A true predicate is called an attitude.

Just like in the utterance, the predicate expresses the interrelationships between different objects: between the reflected object and its properties, characteristics and actions, between pairs or a large number of objects, processes, phenomena.

A predicative symbol, as a rule, represents a predicate in a narrative sentence about which one can say whether it is true or false. But very often the predicate symbol reflects the semantics of the relationship of the objects to be reflected.

Examples:

1) LOVES (Ivan, Maria)

2) LOVES (father (Ivan), Ivan)

3) MORE (x, 3), where $x \in R$ is the set of natural numbers.

4) MORE (plus (x, 1), x)

5) CAPITAL (x, y)

Where x {Moscow, Kiev, Minsk},

Y $\in$ {Russia, Ukraine, Belarus}

6) CONTAIN (x, y, v)

Where x $\in$ {carbon, sulfur},

Y $\in$ {steel, silicon},

V $\in$ {set of real numbers}

7) DELIVERY (x, y, z, i, j, k, l, m),

Where x | what? | $\in${List of accessories},

Y | to whom? | $\in$ {List of recipients},

Z | who? | $\in${List of suppliers},

I | where? | $\in${ List of sources},

J | where? | $\in${List of receivers},

K | how many? | $\in${List of parts},

L | for the price? | $\in${List of prices}?

M | transportation costs? | $\in$ {List of transportation unit costs}

8) PROPERTIES AND CHARACTERISTICS (z, x_1, x_2, x_3, y_1, y_2, y_3),

Where is the object to be reflected? {List of objects to be reflected},

X1 | color? | $\in$ {List of colors},

X2 | smell? | $\in${List of smells},

X3 | taste? | $\in${List of flavors},

Y1 | geometric form? | $\in$ {List of geometric forms},

Y2 | hardness? | $\in$ {List of hardness indicators},

Y3 | specific gravity? | $\in${List of weights}.

The term "atom" is a predicate represented in the form of the formula P (t_1, t_2, ..., t_n).

The syntax of the predicate logic language is determined recursively by the following rules.

1) The atom has a formula.

2) If F and G are formulas, then ($\rceil$F), (F $\vee$G), (F$\wedge$ G), (F$\rightarrow$ G) and (F $\equiv$G) are formulas.

3) If F is a formula, and x is a subject variable of F, then ($\forall x$) F and ($\exists x$) F are formulas.

4) formulas are generated only by the final application of rules 1-3.

The scope of the quantifier is the formula to which it is applied.

For example, the scope of the existence quantifier in the formula $(\forall x)(\exists y)$ LESS (x,y) is the formula LESS (x,y), and the scope of the universal quantifier is the formula $(\exists y)$ LESS (x,y).

In the formula $(\forall x)(Q(x)\rightarrow R(x))$, the scope of the universal quantifier is a formula $(Q(x)\rightarrow R(x))$.

The occurrence of a variable in a formula is said to be connected if and only if it coincides with the entry into the quantifier complex or, or is in the range of action of such a complex.

The occurrence of a variable in a formula is free if and only if it is not bound.

For example, in the formula $(\forall x)P(x,y)$ the variable is connected, because Both occurrences are connected. However, the variable is free, because The only entry is free.

The variable in the formula can be free and bound, for example, a variable and free, and is related in the formula $(\forall x)P(x,y)\wedge(\forall y)Q(y)$.

The concept of "the semantics of the quantifier of universality $\forall$." The quantifier of universality $\forall$ establishes the rule for interpreting the formula F to which it is applied. For each concrete value of the variable x (which the quantifier $\forall$ binds in the formula F) belonging to the domain of the variable x definition, the formula F, as a function of x, can take either only true values, or only false ones, but the combination of and is unacceptable throughout the domain of definition.

The concept of "semantics of the quantifier of existence $\exists$."

The existence quantifier $\exists$ establishes the rule for interpreting the formula F to which it is applied. At least for one particular value of the variable x (which $\exists$ binds in the formula F) belonging to the domain of definition x, the formula F, as a function of x, necessarily takes a true (or false) value.

An example of reasoning, written in the language of predicate logic.

1) each person is mortal.

2) Socrates is a person, therefore Socrates is mortal.

" x is a person" denoted by a PERSON (x),

" x is mortal," through MORTAL (x).

The first assertion is represented by the formula:

$(\forall x)($ PERSON $(X)\rightarrow$ MORTAL $(x))$.

The statement "Socrates is a man" is the formula of the PERSON (Socrates), and the statement "Socrates is mortal" is the formula MORTEN (Socrates).

In general, the argument is represented by the formula:

$(\forall x)($ PERSON $(X)\rightarrow$ MORTAL $(x))\wedge$ PERSON(Socrates)

$\rightarrow$MORTAL(Socrates).

4.4. Rules for assigning truth values to formulas (semantics of language).

To determine the interpretation for the predicate logic formula, you must specify:

• the general domain set D for all object variable, including in the formula;

• the values of constants;

• truth-values of predicates, including in the formula;

• the values of functions, including in the formula.

Regard to the above mentioned :

1) to each constant is assigned an element from D;

2) to each n-place functional symbol is assigned the displaying from D^n to D (note that $D^n=\{(x_1,\ldots,x_n)/x_1\in D,\ldots,x_n\in D\}$

3) to each n-place predicate symbol is assigned the displaying D^n into the set of {T,F}.

For each interpretation of the formula from the domain D, the formula can be equal to the truth-value T or F according to the following rules:

1) If the values of formulas G and H are specified, then the truth-values of formulas $\rceil$G,($(G \wedge H),(G \vee H),(G \rightarrow H),(G \equiv H)$) are obtained according to the table 2 of propositional logic.
2) $(\forall x)$G obtains the meaning of T, if G obtains the value T for every x of D; otherwise it gets the value of F.
3) $(\exists x)G$gets the value T, if G gets the value T if only for one x from D; otherwise it obtains the value T.
4) A formula, containing free variables cannot obtain a truth-value. In predicate logic the following convention operates: the formula either doesn't contain free variables, or free variables are considered as constants.

Example: let's estimate the formulas

a) $(\exists x)(P(f(x)) \wedge Q(x, f(a)))$

b) $(\exists x)(P(x) \wedge Q(x,a))$

c) $(\forall x)(\exists y)(P(x) \wedge Q(x,y))$

In the next interpretation I:

$D = \{1,2\}, a = 1, f(1) = 2, f(2) = 1, P(1) = F,\ P(2) = \text{T},\ Q(1,1) = \text{T},$
$Q(1,2) = T,\ Q(2,1) = F,\ Q(2,2) = T.$

For formula a) : if $x = 1$, then

$P(f(x) \wedge Q(x, f(a)) = P(f(1)) \wedge Q(1, f(a)) = P(2) \wedge Q(1, f(1)) =$
$P(2) \wedge Q(1,2) = T \wedge T = T$;

If $x = 2$, then

$P(f(x) \wedge Q(x, f(a)) = P(f(2) \wedge Q(2, f(1)) = P(1) \wedge Q(2,1) =$
$F \wedge F = F.$

Because of in the domain there is a member, namely x=1, such that $P(f(x) \wedge Q(x, f(a))$ is true, then a) is true in the interpretation of I.

For formula b):

if $x = 1$, then

$P(x) \wedge Q(x,a) = P(1) \wedge Q(1,1) = \text{F} \wedge \text{T} = \text{F}$;

if x=2, then

$P(x) \wedge Q(x,a) = P(2) \wedge Q(2,1) = \text{T} \wedge \text{F} = \text{F}.$

i.e. in the domain D doesn't exist such member, that $P(x) \wedge Q(x,a)$ is true, that formula b) will be false in the interpretation of I.

For formula c):

if $x = 1$, then $P(x) = P(1) =$ F. Consequently, $P(x) \wedge Q(x, y) =$ F for $y = 1$ and for $y = 2$. Because of x exists, namely $x = 1$, such that $(\exists\, y)\ (P(x) \wedge Q(x, y)$ is false , then formula c) is false in interpretation of I , i.e. this formula is contradicted be interpretation of I.

Let’s define a number of the most important concepts for predicate logic.

Consistent (satisfiable) formula - the formula G is satisfiable (consistent) if and only if exists such interpretation of I, that G has value T in interpretation of I. If the formula G is И in interpretation of I, then there is model of the formula G, and I satisfies to the formula G.

Inconsistent formula is when there is no interpretation, which satisfies G.

Valid formula is when there is no interpretation I, which does not satisfy G.

Logical corollary – formula G is a logical consequence of formulas $F_1, F_2, \ldots\ldots, F_n$ if and only if for each interpretation of I, if $F_1 \wedge F_2 \wedge \ldots\ldots \wedge F_n$ is true, then G is also true in the interpretation of I.

Notice: in that amount of domains of object variables isn’t limited in predicate logic, i.e. it can be an infinite number, so there is an infinite number of interpretations of the formula. Consequently, in contrast to propositional logic, there is no possibility to prove validity or inconsistency of formula by determining its truth-values with all possible interpretations, even in the simplest cases. That’s why the proof of theorems (truth of reasoning) in the logic of predicates is carried out only by the method of resolutions, but it has its own special aspects.

4.5. The rules of inference in predicate calculus.

Predicate logic is a section of ***symbolic logic*** that studies reasoning and other ***language contexts***, by taking into account the internal structure of the simple ***proposition statements*** included in them, while the language expressions are interpreted functionally, i.e. as ***signs*** of some ***functions*** or as ***argument signs*** of these functions [4,5].

Predicate logic, as a propositional logic, is based on the deductive argument. Therefore, the inference rules of propositional logic, given in paragraph 3.3.4., equally work on predicate logic.

However, they are supplemented by the rules of introduction and reducing of the quantifiers of universality and existence. Let E is the set of formulas.

The rules for introducing quantifiers are following:

For quantifier $\forall$ $\dfrac{E \Rightarrow A(x)}{E \Rightarrow (\forall x)A(x)}$,

For quantifier $\exists$ $\dfrac{E \Rightarrow A(x)}{E \Rightarrow (\exists x)A(x)}$.

The r**emoval rules of quantifiers** are next:

for quantifier $\forall$ $\dfrac{E \Rightarrow (\forall x)A(x)}{E \Rightarrow A(x)}$,

for quantifier $\exists$ $\dfrac{E \Rightarrow (\exists x)A(x)}{E \Rightarrow A(x)}$.

As well as in propositional logic, in predicate logic the inference rules are applied rarely. The reasons are the same and therefore they are not repeated here.

Basically, the inference rules are used for equivalent transformations of the analytical representation of the theorems with purpose to bring them to the canonical form.

4.6. Rules of the formulas' equivalence transformations of the predicate calculus.

In first order logic (predicate logic) the conditions for the effective application of the resolution method for proving theorems are the same as in the propositional logic. We recall, that one of these conditions is representing of the theorems in RCNF. Rules of the formulas' equivalence transformations, introduced in the logic of propositions are equivalent for first-order logic also. However, existence in the formulas of universal and existential quantifiers makes it difficult to apply the theorems to RCNF.

In this regard the set of rules allowing to exclude the specified quantifiers from the formulas is additionally introduced. These rules are divided up into two groups:

1) Rules of formation of prenex normal forms (PNF);
2) Rules of formation of the skolem standard forms (SSF).

Let's consider these forms and rules of their formation.

The formula F is in the prenex normal form (PNF), if and only if it's of the form of :

$(Q_1\ x_1)(Q_2 x_2)\ldots.(Q_n x_n)\,(M),$ where each $(Q_i x_i), i=\overline{1,n}$ is or $(\forall x_{i)})$, or $(\exists x_i)$, and $(M)$is formula, which doesn't contain quantifiers. $(Q_1\ x_1)(Q_2 x_2)\ldots.(Q_n x_n)$ is called prefix, and (M) is called matrix of formula F .

For instance, in the formula$(\forall x)(\forall y)(\exists z)(Q(x,y)\supset R(z))$ the prefix $(\forall x)(\forall y)(\exists z)$ precedes the matrix $(Q(x,y)\supset R(z)).$

Let's consider the rules of formulas' equivalence transformations, which contain quantifiers.

Let F is a formula, containing free variable *x*. We will designate this formula $F[x]$. Let G is a formula, which doesn't contain variable *x*. Let Q is a quantifier $\forall$ or quantifier $\exists$. In this case the rules are following:

1a) $(Qx)F[x]\vee G\equiv(Qx)(F[x]\vee G);$
1b) $(Qx)F[x]\wedge G\equiv(Qx)(F[x]\wedge G);$
2a) $\rceil((\forall x)P[x])\equiv(\exists x)(\rceil F[x]);$
2b) $\rceil((\exists x)F[x])\equiv(\forall x)(\rceil F[x]);$
3a) $(\forall x)(F[x]\wedge(\forall x)H[x]\equiv(\forall x)(F[x]\wedge H[x]);$

3b) $(\forall x)(F[x] \vee (\forall x)H[x] \equiv (\forall x)(F[x] \vee H[x]);$

4a) $(Q_1 x)F[x] \vee (Q_2 x)H[x] \equiv (Q_1 x)(Q_2 z)(F[x] \vee H[z]);$

4b) $(Q_3 x)F[x] \wedge (Q_4 x)H[x] \equiv (Q_3 x)(Q_4 z)(F[x] \wedge H[z]);$

It is always possible to transform any formula to PNF by using the rules of the formulas' equivalence transformations of propositional logic and the specified eight rules. Let's consider an example:

Let's reduce the formula $(\forall x)P(x) \supset (\exists x)Q(x)$ to PNF. Using the elimination rule of implication linking we will receive:

$(\forall x)P(x) \supset (\exists x)Q(x) \equiv \rceil((\forall x)P(x) \vee (\exists x)Q(x)$.

According to the rule 2a we have:

$\rceil((\forall x)P(x) \vee (\exists x)Q(x) \equiv (\exists x)(\rceil P(x)) \vee (\exists x)Q(x)$.

Finally, using the rule 3b we will receive:

$(\exists x)(\rceil P(x)) \vee (\exists x)Q(x) \equiv (\exists x)(\rceil P(x) \vee Q(x))$.

The formula on the right part of the last ratio is presented in the prenex normal form (PNF).

The skolem standard form – is PNF, in the prefix of which, the existential quantifiers $\exists$ are missed, and matrix M is RCNF. From this definition becomes apparent, that skolem formula manipulations directed to excluding of the existential quantifiers from the prenex normal forms. Let's consider these transformation rules.

Suppose the formula F is in the prenex normal form, $(Q_1\ x_1)(Q_2 x_2)....(Q_n x_n)\ (M)$, where M is RCNF. Let's assume, that Q_r is a existential quantifier in the prefix $(Q_1\ x_1)(Q_2 x_2)....(Q_n x_n)$, $1 \le r \le n$.

If any universal quantifier doesn't stand in the prefix to the left of Q_r, then we choose the new constant c, which is different from other constants included in M, we replace all x_r, which are taken place in M, with a constant c and eliminate $(Q_r x_r)$ from a prefix. If Q_{s1}, Q_{s2} ,...., Q_{sm} is a list of all universal quantifiers, which are faced to the left of Q_r, $1 \le s1$, then we choose new m-placed function symbol f , which is different from other function symbols, replace all x_r in M with $f(x_{s1}, x_{s2} ,...., x_{sm})$ and eliminate $(Q_r x_r)$ from a prefix.

Then we apply this process to all existential quantifiers in the prefix; the last of the received formulas is SSF, skolem standard form. Constants and functions used for variable change of existential quantifier are called skolem constants and functions. Let's consider an example.

We obtain SSF for formula:

$(\exists x)(\forall y)(\forall z)(\exists u)(\forall v)(\exists w)P(x, y, z, u, v, w).$

Here to the left of $(\exists x)$ there are no universal quantifiers, to the left of $(\exists u)$ there are $(\forall y)$ and $(\forall z)$, and to the left of $(\exists w)$ there are $(\forall y)$, $(\forall z)$ and $(\forall v)$.

In this way, we replace variable x with constant a, variable u with binary function $f(y,z)$, variable w with triadic function $g(y,z,v)$. By so doing, after the specified replacements and elimination of existential quantifierswe receive the following skolem standard form for the assumption formula written above:

$(\forall y)(\forall z)(\forall v)P(a,y,z,f(y,z),v,g(y,z,v)).$

The considered rules of equivalence transformations make it possible to provide any theorem of predicate logic in the skolem standard form. Because of the prefix in this form contains only universal quantifiers, it means, for example, for $(\forall x)G,$ that the form receives the value of T, if G is true for each of x from the domain D (and otherwise it gets the value of F), then it gets a right to consider G as a simple expression and a universal quantifier, connecting x, just to eliminate from the prefix. Equally, this conclusion also applies to the universal quantifiers , connecting other variables. Therefore for the theorem proving in p connecting other variables. That's why, for proving the theorems in predicate logic we can use only matrixes, that there are in the RCNF.

In predicate logic the following theorem is also proved.

Let S is a set of disjuncts which represent the RCNF in the skolem standard form of a formula (theorem). Then the formula F is contradictory if and only if the set S is contradictory.

The mechanism of application of a resolution method, which was used for the theorem proving in propositional logic, can be also applied in predicate logic. However, in this case, three vital question arises:

1) how to find the complementary pairs for disjuncts, containing variables?
2) how to calculate the resolvent (resolution) of disjuncts, containing variables?
3) how to make the most of the reverse deduction in order to improve the efficiency of the resolution method?
4) Responses to these questions introduce particular specificity to the algorithm of a resolution method.

4.7. Features of the resolution method in proving theorems in predicate logic.

John Alan Robinson's idea (1965) – of the author of a resolution method [4], – is that in predicate logic it would possible to work, directly with the disjuncts, containing variables without resort to preliminary replacement of variables by constants from the domain D.

This idea is provided with usage of substitution operation and unification. In order to define a main point of these operations, we will introduce several new concepts and definitions.

The term “expression”. By the expression is meant term, a set of terms, a set of atoms, literal, disjunct, a set of disjuncts. When in the expression there are no varieties, it’s called ground expression: ground atom, ground literal, ground disjunct, ground term, which indicates the absence of variables in them.

The term “substitution”. Let’s preliminary, as an example, clarify what’s the problem of searching of complementary pairs in predicate logic. Let’s consider the following disjuncts.

$C_1: \ P(x) \vee Q(x)$

$C_2: \ \neg P(f(x)) \vee R(x)$

There isn’t any literal in disjunction C_1 yet, that is contrary to the some kind of literal in disjunct C_2. However, substituting to C_1 the function $f(x)$ instead of variable x, we will get:

$$C_1^*: \quad P(f(x)) \vee Q(f(x)).$$

Now the literal $P(f(x))$ in the C_1^* is contrary to literal $\neg P(f(x))$ in the C_2.

Consequently, it’s possible to get a resolvent (resolution) from C_1^* and C_2 :

$$C_3: \quad Q(f(x)) \vee R(x).$$

Definition 1: substitution is a finite set of the form $\{t_1 / v_1, ..., t_n / v_n\}$, where each v_i is a variable, each t_i is a term, which is different from v_i, and all v_i - are different.

Definition 2: Let $\Theta = \{t_1 / v_1, ..., t_n / v_n\}$ is substitution, and E is an expression. Then $E\Theta$ is the expression, derived from E be replacing simultaneously of all instances of variable $v_i \ (i = \overline{1,n})$ in expression E with the term t_i. $E\Theta$, is called as example *E*.

For example, let $\Theta = \{a/x, f(b)/y, c/z\}$ and $E = P(x,y,z)$. in such case substitution gets $E\Theta = P(a, f(b), c)$.

The term “composition of substitutions”. Let the substitutions $\Theta = \{t_1/x_1, ..., t_n/x_n\}$ and $\lambda = \{u_1/y_1, ..., u_m/y_m\}$.are given. Then the composition Θ and λ is the substitution of $\Theta \circ \lambda$, that is obtained from the set of $\{t_1\lambda/x_1, ..., t_n\lambda/x_n, u_1/y_1, ..., u_m/y_m\}$ by the elimination of all the members $t_j\lambda/x_j$, for which $t_j\lambda = x_j$ and of all members u_i/y_i such that $y_i \in \{x_1, x_2, ..., x_n\}$

Let’s consider the example of identifying the composition of substitutions $\Theta = \{f(y)/x, z/y\}$ and $\lambda = \{a/x, b/y, y/z\}$.

In this case, $\Theta \circ \lambda = \{f(b)/x, y/y, a/x, b/y, y/z\}$. However, by the definition, y/y must be eliminated; a/x and b/y also must be eliminated, because x and y are contained among the varieties of substitution Θ. As a result we obtain $\Theta \circ \lambda = \{f(x)/x, y/z\}$.

The term “unifier for a set of expressions”. Substitution Θ is called as a unifier for a set of expressions $\{E_1, E_2, ..., E_n\}$ if and only if

$E_1\Theta \equiv E_2\Theta \equiv ... \equiv E_n\Theta$. It’s considered, that a set of expressions is unifiable, if there is an unifier for it.

Unifier σ for the set of expression $\{E_1, E_2,..., E_n\}$ is called as the <u>most general unifier</u> (MGU) if and only if for each unifier Θ for this set there is such substitution λ, that $\Theta = \sigma \circ \lambda$. Let, for example, there is a set of expressions. in this case $\sigma = \{z/x, a/y, g(a)/u\}$ is the most general unifier for W, and $\Theta = \{b/x, a/y, b/z, g(a)/u\}$ is the most general unifier.

Let’s describe step-by-step the unification algorithm, which finds MGU, if the set $E = \{E_1, E_2,..., E_k\}$ is unifiable and reports about a failure, and if this is not the case [7]:

1) to set up $k = 0, E_k = E$ and $\sigma_k = \varepsilon$, where ε is an empty substitution , which not contains the members. Pass on to the step 2.
2) If E_k is not a singleton set, then go to step 3. Otherwise, put $\sigma = \sigma_k$ and finish the work.
3) Each of the letters in E_k is treated as a string of characters and the first subexpressions of letters, not being the same for all elements E_k, i.e. a so-called set of mismatches of the type $\{x_k, t_k\}$. If in this set x_k – variable, and t_k –term, different from x_k, then go to step 4. Otherwise, end the work with the conclusion: E do not unify.
4) Let $\sigma_{k+1} = \sigma_k \circ \{t_k / x_k\}$ и $E_{k+1} = E_k\{t_k / x_k\}$.
5) Install $k := k+1$ and go to step 2.

Consider the operation of the algorithm for finding LEU for the set $E = \{P(y, g(z), f(x)), P(a, x, f(d(y)))\}$. Steps are as follows:

1) $\sigma_0 = \varepsilon$ и $E_0 = E$.
2) Since E_0 is not a singleton set, then go to step 3.
3) Multiple disagreements $\{y, a),$ i.e. substitution $\{a/y\}$.
4) $\sigma_1 = \sigma_0 \circ \{a/y\} = \varepsilon \circ \{a/y\} = \{a/y\}$.
   $E_1 = E_0\{a/y\} = \{P(a, g(z), f(x)), P(a, x, f(g(a)))\}$
5) Since the set E_1 again not singleton, the set of discrepancies will be $\{g(z), x\}$, i.e. substitution $\{g(z)/x\}$.
6) $\sigma_2 = \sigma_1 \circ \{g(z)/x\} = \{a/y, g(z)/x\}$.
   $E_2 = E_1\{g(z)/x\} = \{P(a, g(z), f(g(z))), P(a, g(z), f(g(a)))\}$.
7) Wehave $\{z, a\}$, i.e. $\{a/z\}$.
   $\sigma_3 = \sigma_2 \circ \{a/z\} = \{a/y, g(a)/x, a/z\}$
8) $E_3 = E_2\{a/z\} = \{P(a, g(a), f(g(a))), P(a, g(a), f(g(a)))\} =$
   $= \{P(a, g(a), f(g(a)))\}$.

Since as ingleton set received, then the desired most common unifier $\sigma = \sigma_3 = \{a/y, g(a)/x, a/z\}$.

It can be shown that the unification of algorithm always ends in step 2, if the set E unifiable, and in step 3 –otherwise.

The notion of «splicing disjunct C». If two or more characters (with the same sign) disjunct C have the most common unifier σ, then $C\sigma$ is called gluing C. If $C\sigma$ –single clause, then the gluing called a single gluing.

Example: Let $C = P(x) \vee P(f(y)) \vee \rceil Q(f(y))$.

Then the first and second letters have the most common unifier $\sigma = \{f(y)/x\}$. Consequently, $C\sigma = P(f(y)) \vee \rceil Q(F(y))$ is a gluing of C.

The concept of a «binary resolution». Let C_1 and C_2 - two disjunct (called disjuncts-sentences), which have no common variables. Suppose also that l_1 and l_2 – two letters respectively in C_1 and C_2. If l_1 and $\rceil l_2$ have the most common unifier σ, then disjunct $(C_1 - l_1\sigma) \vee (C_2 - l_2\sigma)$ is called a binary resolution C_1 and C_2, and letters l_1 and l_2 are called cut letters.

Example: Let $C_1 = P(x) \vee Q(x)$ and $\rceil C_2 = P(a) \vee R(x)$. Since the variable x is included in C_1 and C_2, then we replace the variable x in C_2 on y, i.e. $\rceil C_2 = P(a) \vee R(y)$. Select $l_1 = P(x)$ and $l_2 = \rceil P(a)$. As $l_2 = P(a)$, $\rceil$then l_1 and l_2 have the most common unifier $\sigma = \{a/x\}$.

Consequently,

$$(C_1\sigma - l_1\sigma) \vee (C_2\sigma - l_2\sigma) \equiv (\{P(a), Q(a)\} - \{P(a)\}) \vee (\{\rceil P(a), R(y)\} -$$

$$\{\rceil P(a)\}) \equiv \{Q(a)\} \vee \{R(y)\} = \{Q(a), R(y)\} = Q(a) \vee R(y).$$

In this way, $Q(a) \vee R(y)$–binary resolution C_1 and C_2, and $P(x)$ and $\rceil P(a)$– are cut letters.

The concept of «first-order resolution of logic» Resolver of disjunct-sentence C_1 and C_2 is one of the following resolutions:

1) binary resolution C_1 and C_2;
2) binary resolution C_1 and gluing C_2;
3) binary resolution C_2 and gluing C_1;
4) binary resolution of gluing C_1 and gluing C_2.

Example: Let $C_1 = P(f(g(\rceil a))) \vee R(b)$,

$C_2 = P(x) \vee \rceil P(f(y)) \vee \rceil Q(y)$.

Then gluing of disjunct C_2 is $C_2\sigma = C_2' = \rceil P(f(y)) \vee Q(y)$ and resolution for C_1 and C_2' is $C = \rceil R(b) \vee Q(g(a))$.

4.8. Proof theory methods in predicate logic resolutions

There are no clear rules and recommendations, how to present in the form of the formula of the logic of predicates this or that reasoning. All this is done intuitively. In addition, intuition is in the proportion to the mastery in the proof theorems by method of resolutions. Therefore, only through practice and skills can you acquire the necessary intuition for modeling reasoning.

To some extent, the starting intuition can be obtained by reading the examples given below, considered in various works [5,7,8,12,13]. In the first two examples, detailed explanations are given, in the remaining ones, only as needed.

Example 1. Some patients like their doctors. No patient likes a witch doctor. Consequently, no doctor is a witch doctor.

We introduce the following notation for predicate symbols: P –patient, D– doctor, Z–witch doctor, L–likes.

Then the predicates listed below will denote:

$P(x)$ — x is patient;

$D(x)$ — x is doctor;

$Z(x)$ — x is witch doctor;

$L(x,y)$ — x likes y.

The facts and conclusion given in the argument can be represented by the following formulas:

Fact 1 F_1 : $(\exists x)(P(x) \wedge (\forall y)(D(y) \supset L(x,y)))$.

Fact 2 F_2 : $(\forall x)(P(x) \supset (\forall y)(Z(y) \supset \urcorner L(x,y)))$.

Conclusion G : $(\forall x)(D(x) \supset \urcorner Z(x))$.

In accordance with the conditions for the effective proof of theorems by the method of resolutions, we transform the facts F_1, F_2 and denying the conclusion $\urcorner G$ By the rules of equivalent transformations in the following disjuncts:

$$\left.\begin{array}{ll}(1) & P(a)\\ (2) & D(y) \vee \urcorner L(a,y)\end{array}\right\} \text{from } F_1$$

Here is a rule of the existential quantifier elimination and exclusion rule implication ligament.

(3) $\urcorner P(x) \vee \urcorner Z(y) \vee \urcorner L(x,y)$ from F_2

$$\left.\begin{array}{ll}(4) & D(b)\\ (5) & Z(b)\end{array}\right\} \text{from } \urcorner G$$

Here there is a rule 2afrom 2.3.6.1 , The rule of exclusion of the existence quantifier and the rule of exclusion of the implication bundle.

Performing the unification and gluing, we get:

(6) $L(a,b)$ resolver (2) и (4).

(7) $\urcorner Z(y) \vee \urcorner L(a,y)$ resolver (1) и (3).

(8) $\neg\, L(a,b)$ resolver (5) и (7).

(9) $\square$ resolver (6) и (8).

The theorem is proved.

Example 2. All people are animals. Therefore, the human head is the animal's head.

Suppose we have the following predicates:

$L(x)$ — x There is a person;

$A(x)$ — x There is an animal;

$G(x,y)$ — x It is a head y.

It is necessary to prove the theorem:

$$\frac{(\forall y)(L(y)\supset A(y))}{(\forall x)((\exists y)(L(y)\wedge G(x,y))\supset(\exists z)(A(z)\wedge G(x,z)))}$$

The transformation of the numerator (the premise theorem) gives a clause:

(1) $L(y)\vee A(y)$.

To obtain the remaining clauses, we transform the negation of the denominator (theorem-conclusion) as follows:

$$(\neg\forall x)((\exists y)(L(y)\wedge G(x,y))\supset(\exists z)(A(z)\wedge G(x,z)))\equiv$$
$$\equiv\neg(\forall x)((\exists y)(L(y)\wedge G(x,y))\vee(\exists z)(A(z)\wedge G(x,z)))\equiv$$
$$\equiv\neg(\forall x)((\forall y)(\quad L\neg(y)\vee\quad G(x,y))\vee(\exists z)(A(z)\wedge G(x,z)))\equiv$$
$$\equiv\neg\,(\forall x)(\forall y)(\exists y)(\neg\, L(y)\vee\ \neg G(x,y)\vee A(z)\wedge G(x,z))\equiv$$
$$\equiv(\exists x)(\exists y)(\forall z)(L(y)\wedge G(x,y)\wedge\ \neg A(z)\vee\ \neg G(x,y)).$$

Thus, (2) $L(b)$.

(3) $G(a,b)$.

(4) $A(z)\vee\quad G(a,z)$.

Applying the method of resolutions, we get:

(5) $L(b)$ from (1) and (2).

(6) $G(a,b)$ from (4) and (5).

(7) $\square$ from (3) and (6).

The theorem is proved.

Example 3. Parcels: Customs officials search everyone who enters the country, except for high-ranking officials. Some people who facilitate drug tracking enter the country and are searched only by people who also facilitate the transport of drugs. None of the dignitaries promoted the passage of drugs.
Conclusion: some of the customs officers facilitated the transport of drugs.
We introduce the following notation for predicates:

$E(x)$: x entered the country;

$V(x)$: x was a high-ranking official;

$S(x,y)$: y searched x ;

$C(x)$: x was one of the custom's representative;

$P(x)$: x facilitated the transport of drugs.

Parcels are represented by the following formulas:

$(\forall x)(E(x) \wedge \neg V(x) \supset (\exists y)(S(x,y) \wedge C(y)))$,

$(\exists x)(P(x) \wedge E(x) \wedge (\forall y)(S(x,y) \supset P(y))$,

$(\forall x)(P(x) \supset \neg V(x)$,

And the conclusion of the theorem by the formula:

$(\exists x)(P(x) \wedge C(x)$.

Converting parcels into clauses, we get:

(1) $\neg E(x) \vee V(x) \vee S(x, f(x))$.

(2) $\neg E(x) \vee V(x) \vee C(f(x))$.

(3) $P(a)$.

(4) $E(a)$.

(5) $\neg S(a,y) \vee P(y)$.

(6) $\neg P(x) \vee \quad V(x)$.

Negation of the conclusion:

(7) $\neg P(x) \vee \neg C(x)$.

The proof by the resolution method is presented as follows:

(8) $\neg V(a)$ from (3) and (6).

(9) $V(a) \vee C(f(a))$ from (2) and (4).

(10) $C(f(a))$ from (8) and (9).

(11) $V(a) \vee S(a, f(a))$ from (1) and (4).

(12) $S(a, f(a))$ from (8) and (11).

(13) $P(f(a))$ from (12) and (5).

(14) $\neg C(f(a))$ from (7) and (13).

(15) $\Box$ from (10) and (14).

The conclusion is proved.

Example 4. There are students who love all teachers. None of the students love ignorant people. Therefore, none of the teachers is ignorant.

Denote by:

$C(x)$ — x there is a student;

$P(x)$ — x there is a teacher;

$H(x)$ — x there is an ignorant person;

$L(x,y)$ — x loves y .

On the language of predicate logic after reduction to standard form it is written as follows:

$(\exists x)(C(x) \wedge (\forall y)(P(y) \supset L(x,y)))$

$(\forall x)(C(x) \supset (\forall y)(H(y) \supset \neg L(x,y)))$

$\overline{(\forall y)(P(y) \supset \quad H(y\neg))}$

Converting two theorems-parcel of the numerator gives the following clauses:

(1) $C(a)$.

(2) $\neg P(y) \vee L(a, y)$.

(3) $\neg C(x) \vee \neg H(y) \vee \neg L(x, y)$.

After transforming the negation of the conclusion from the denominator, we get:

$\neg(\forall y)(\neg P(y) \vee \neg H(y) \equiv (\exists y)(P(y) \wedge H(y))$, that gives clauses:

(4) $P(b)$.

(5) $H(b)$.

By unification and gluing, we get:

(6) $L(a,b)$	from (2) and (4).
(7) $\neg H(y) \vee \neg L(a, y)$	from (1) and (3).
(8) $\neg L(a,b)$	from (5) and (7).
(9) $\square$	from (6) and (8).

The theorem is proved.

Example 5. The problem of the monkey and banana.

The monkey wants to eat a banana, suspended from the ceiling of the room. The growth of the monkey is not enough to reach the banana. However, it can walk around the room, move a chair in the same room, climb onto a chair and get a banana. Show the order of the monkey's actions, in which it will get a banana.

Here are predicates:

$P(x, y, z, s)$ means that in the state of s the monkey is at the point - x, the banana is at the point - y, and the chair is at the point – z;

$R(s)$ means that in the state of s, the monkey can get a banana.

The functions, involved in the task description, are given as follows:

To walk (y, z, s) - is a condition that is obtained if firstly, the monkey was in a state of **s** and moved from point **x** to point **z**;

To wear (y, z, s) - is a condition that is obtained if firstly, the monkey was in a state **s** and moved from point **y** to point **z**, carrying a chair;

To climb (s) - is a condition, that is obtained if the monkey was in a state **s** and climbed onto a chair.

We assume that initially the monkey was at the point **a**, the banana -at the point **b**, the chair- at the point **c** and the monkey was in a state of s_1.

Thus, we have the following axioms:

(1) $\neg P(x, y, z, s) \vee P(z, y, z, \textbf{to go}(x, z, s))$.

(2) $\neg P(x, y, x, s) \vee P(y, y, y, \textbf{wear}(x, y, s))$.

(3) $\neg P(b, b, b, s) \vee R(\textbf{climb}(s))$.

(4) $P(a, b, c, s_1)$.

Here disjunction (1) means that in any state the monkey can go from point x to point z.

Disjunction (2) means. What if the monkey is near a chair that stands at a point x then it can move it to any point y.

Disjunct (3) means that if the chair and monkey are under a banana, then the monkey can climb on a chair and get a banana.

Disjunction 4) describes the initial situation.

The conclusion of the theorem corresponds to a disjunct

(5) $\urcorner R(s) \vee \mathbf{answer}(s)$.

In this clause the predicate response requires setting the order of the monkey's actions, corresponding to the state of the monkey with the banana.

Using clauses (1) - (5), we derive the following resolutions:

(6) $\urcorner P(b,b,b,s) \vee$ **answer** (**climb** (s)) from (5) and (3).

(7) $\urcorner P(x,b,x,s) \vee$ **answer** (**climb** (**wear** (x,b,s)) from (6) and (2).

(8) $\urcorner P(x,b,z,s) \vee$ **answer** (**climb** (**wear** $z,b,$**walk**$(x,z,s)))))$ from (7) and (1).

(9) **answer** (**climb** (**wear**$(c,b,$**walk**$(a,c,s_1))))$ from (8) and (4).

Disjunct (9) gives the answer. It can be interpreted as performing the following actions (starting with the innermost function in the clause (9) and moving outward):

1. The monkey comes from the point a to point c.
2. The monkey comes from the point c to point b, carrying a chair.
3. The monkey climbs into a chair.

After these actions, the monkey takes out a banana.

4.9. Implementation of the resolution method in Prologue

The highest degree of understanding of the algorithm for proving a theorem can be achieved when the description of each step of the algorithm is illustrated by the corresponding actions on a concrete example, the completeness of which allows us to reflect all its features realized in some formal language. To such languages is the declarative language Prolog [10].

Let us first consider the algorithm for the proof of theorems by the method of resolutions using the example of the known problem of Socrates mortality [14].

As a machine implementation, we use one of the versions of Prologue [11, 15, 16], in which predicates can be written in Russian.

The text of the prologue program is as follows:

```
domains% Domain name
name=symbol
predicates% Predicate section
    mortal(name)
clauses% Section of facts and rules
    human(Socrat).              %Fact about a man named Socrates
```

% Contract is a constant
mortal(X):— human(X). % The rule "every person is mortal"

The program writes a comment after the% sign to the end of the line. Pay attention to the mandatory presence of a point at the end of each fact and rule. Finally, about the rules in the prolog programs. Each rule consists of the head (the left half of the rule) in which the predicate to be determined is placed, and the body (the right half of the rule) in which the conditional part of the rule is placed.

The head and body rules in the program are related by the sign (-), which determines the truth of the statement when the condition is met. This sign in the program can be replaced by the English word if (if).

In the rules, the comma (,) matches the sign of the logical AND operation, and the semicolon (;) is the OR sign. In the Prologue language, only Horn's expressions are limited, and only the method of resolutions is applied to them. Recall that the expression Horn looks like:

$$\rceil D_1 \vee \rceil D_2 \vee ... \vee \rceil D_n \vee D_m ,$$

where D_i - literals, D_m - Conclusion of the theorem

Using a bunch of implication $\supset$, The theorem can be rewritten as follows:

$$D_1 \wedge D_2 \wedge ... \wedge D_n \supset D_m ,$$

That in the syntax of the system Prolog is written in the form:

$$D_m :- D_1 , D_2 ,...., D_n .$$

In the above Prolog program, only parcels (parcels-facts and parcels-rules) are presented and so far there are no conclusions of the theorem. The conclusions of the theorem are formulated by indicating a certain goal.

After entering the program in the prolog system, the latter can be asked questions about the relationships described in the program. The question posed to the system is for her the goal, for which the system uses the information of the clauses section. Usually the system suggests entering the target from the keyboard right after the invitation, for example, in the form? - (or Purpose:). In some prolog systems this is an example of the so-called external goal.

The question of whether Socrates is mortal is introduced as the conclusion of the theorem:

? - **Mortal (Socrat).**

The prolog-system will answer yes to this question.

In prolog systems with an internal goal in the program, it suffices to write

The goal is mortal (Socrates).

Consider how the system receives this response. If the conclusion of the theorem can not use only the premisses-facts, then the algorithm is sent to the premise-rules (Horn's clauses) and replaces the current goals with new ones until these new goals turn out to be simple premises-facts. Since the fact is mortal (Socrates) is absent, the first (and only) suitable rule is

Mortal (X): - human (X).

This rule implements the attitude of mortal. The predicative rule symbol is comparable with the predicate symbol of the conclusion of the theorem.

Since the conclusion is mortal (acrit), the value of the variable X should be assigned as follows:

X = contract

There was a unification of the predicate mortal (X) of the rule and mortal (acrit) of the conclusion of the theorem. Then the original goal is mortal (Socrat) replaced by the new goal man (Socrates), since this predicate is the resolvent of the original conclusion of the theorem and the head of the rule after their unification.

This goal is immediately achieved, since it occurs in the premise of the theorem as a fact. This step completes the calculation, since a new resolvent is obtained-an empty clause-gluing the negation of the conclusion of the theorem (as required by the resolution method) and the current goal.

All steps to achieve the goal of death (Socrates) are presented in Figure 4.1 as a tree of logical inference. The tree tops correspond to the goals or lists of goals that you want to achieve.

The arcs between the vertices correspond to the use of alternative program sentences that convert a target corresponding to one vertex to a target corresponding to another vertex. The root (upper) goal is reached when there is a path from the root of the tree (the top vertex) to the leaf marked with the "yes" mark. The leaf is marked with a "yes" mark if it is a fact-parcel.

In the second example, we consider the problem of family relations [8, 9]. Figure 4.2 shows the relationship of parents and children in some family, and for the reason that in the Prolog language the constants begin with a lowercase letter, all the names in the figure are represented.

The fact that Tom is Bob's parent can be written on Prolog like this:

the **parent (tom, bob).**

Here we have chosen the **parent** as a relation (predicate), and the **tom** and the **bob** are the arguments of this relation.

In the Prolog-program in the **domains** section, arguments of the **parent** predicate and described below **ancestor** predicate are described as atoms of a symbolic type (**symbol**).

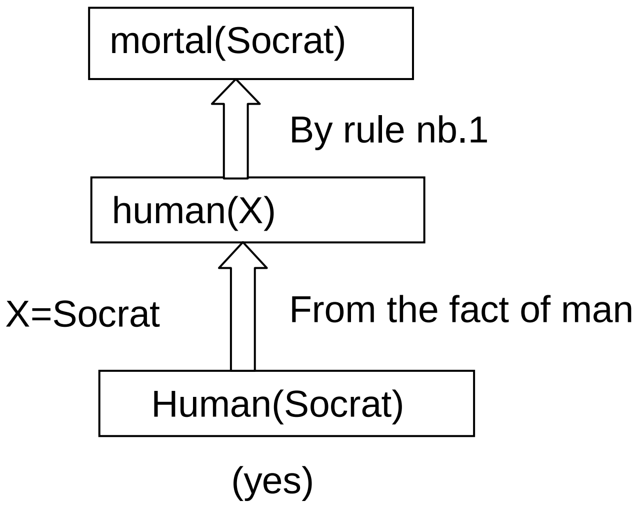


Figure 4.1. Logical output tree

In the clauses section of the program, all available facts are written directly on the tree of family relations. Here are two rules for determining the predicate **ancestor**, the explanations of which will be given later.

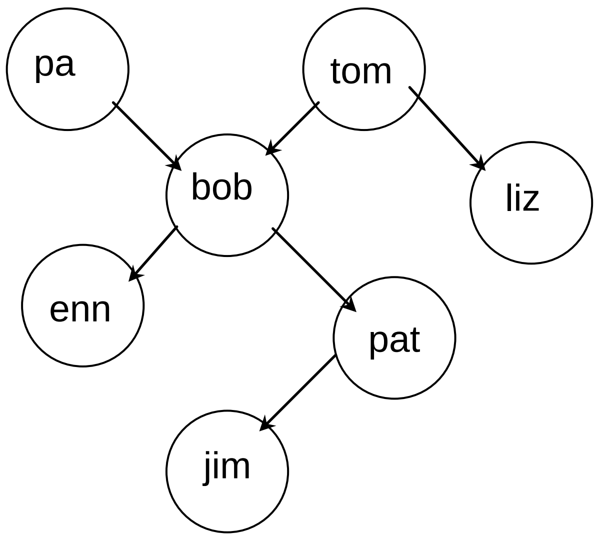


Figure 4.2. Tree of the family relationships.

Prolog program text:

```
domains                                % Domain Name
        name = symbol
predicates                             % Predicate section
        parent (name, name)
        ancestor (name, name)
clauses                                % Section of facts and rules
        parent (pam, bob).
        parent (tom, bob).
        parent (tom, liz).
        parent (bob, enn).
        parent (bob, pat).
        parent (pat, jim).
        ancestor (X, Z): -       % Rule pr1: X - ancestor of the descendant Z
                parent (X, Z).
        ancestor (X, Z): -     % Rule pr2: X - ancestor of the descendant Z
                parent (X, Y),
                ancestor (Y, Z).
```

If in the problem some relations can be defined in several ways, then there will be several rules. So, in this program the ancestor relation is defined by two rules. The first rule determines the immediate (immediate) ancestors, and the second - the remote ones. Some X is the ancestor of some Z in the case when between X and Z there is a chain of people related to each other by the parent-child relationship, as shown in Fig. 4.3. In our example in Fig. 4.2 Tom is the closest ancestor of Liz and the distant ancestor of Jim.

The first rule is easily constructed from Fig.4.3a:

For all X and Z

X is the ancestor of Z, if

X is the parent of Z,

Which corresponds to the recording program:

ancestor (X, Z): - parent (X, Z).

The second rule for the **ancestor** predicate is more complicated, since building a chain of **parent** relations (see figure 4.3b) can cause some difficulties. If the length of the chain is strictly fixed, then you can build a rule by writing in the body through commas the entire sequence of relations, for example, like this:

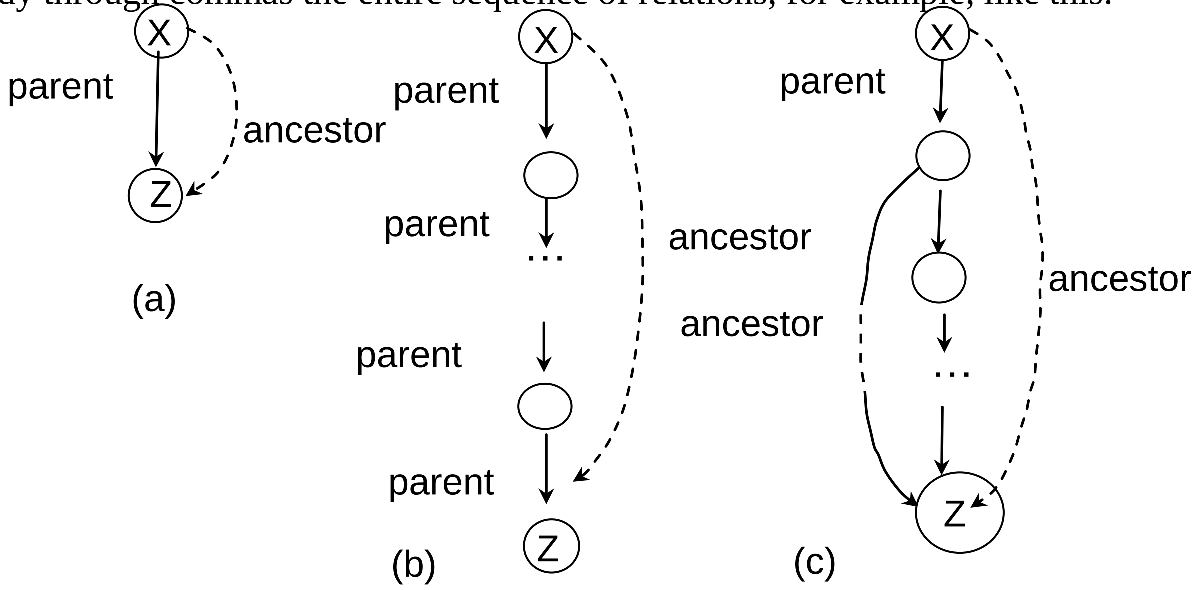


Figure 4.3. Example of **ancestor** relationship:

(a) X is the nearest ancestor of Z; (b) X is a remote ancestor of Z;
(c) the recursive formulation of the **ancestor** relationship.

```
ancestor (X, Z): -   parent (X, Y1),
                     parent (Y1, Y2),
                     parent (Y2, Y3),
                     parent (Y3, Z).
```

This rule can be used only for a given chain and no more. There is, however, a correct and elegant formulation of the **ancestor** relationship that will work for the ancestors of an arbitrary remoteness. The key idea here is to determine the ancestor relationship through itself. Such a definition is called **recursive**. Fig.4.3c illustrates this idea.

```
For all X and Y,
    X is the ancestor of Z, if
    There exists a Y such that
        (1) X is the parent of Y and
        (2) Y is the ancestor of Z.
```

Prolog sentence, having the same meaning, is written like this:

```
ancestor (X, Z): -
    parent (X, Y),
    ancestor (Y, Z).
```

The question of whether Bob is a parent of the Pat should be introduced like this:

? - **parent (bob, pat).**

Having found the corresponding fact in the program, the system will respond

Yes

For the question

? **- the parent (liz, pat)** the system will respond

No ,

Because the program nothing says about whether Liz is Pat's parent.

As already indicated, the goal for the program can be internal. Then it is indicated in the **goal** section, which is written before (or after) the section **clauses** and necessarily ends with a dot. For example, the last question for a system with an internal purpose will be written in the program as follows:

goal

parent (liz, pat).

The system is able to answer even more interesting questions. For example, "who is the parent of Liz?":

? - **parent (X, liz).**

The system will not simply answer "yes" or "no" to this question. She will tell us what the value of X (previously unknown) should be, so that the above statement is true, and also indicate the number of solutions. Therefore, we get the answer:

X = tom 1 is resolved.

The question "who is Bob's children?" you can pass system in this form:

? —**parent (Bob, X).**

In this case, the response is received:

X = Ann

X = Pat 2 solved.

The program can set and more complex questions, such as Who is the parent of the parent Jim? ". Because there is no type predicate roditel'_roditelja, the question can be composed of two simple questions using a bundle, like so:

? —**parent (X, Y), parent (Y, Jim).**

The answer is: **X = Bob**

Y = Pat 1 resolved.

This is an example of a compound request.

Let's consider job of the algorithm theorems to be proved in the prologue, for example, if you want to try to achieve the goal:

?—**ancestor (Tom, Pat).**

First, the algorithm tries to find a parcel in the theorem from which should be immediately referred to the conclusion. Obviously, the only suitable assumptions for this rules are rules of WP1 and WP2. These two rules implement relevant **ancestor**. Predicate symbols these rules compatible with the predicate symbol of the conclusion of the theorem. First, the algorithm tries sending rule WP1, standing in the program first:

ancestor (X, Z):—parent (X, Z). % WP1

Since the conclusion of the–**ancestor (Tom, Pat),** the variables must be attributed as follows:

X = Tom, **Z** = Pat

The unification took place {Tom /X, Pat/Y} predicates **ancestor(X, Z)** rules of WP1 and **ancestor (Tom, Pat)** conclusion of the theorem. Then the original purpose of **ancestor (Tom, Pat)** is replaced with a new order to parent (Tom, Pat), because the predicate is rezol'ventoj the original conclusion of theorem and head WP1 rules after their unification.

The assumptions of the theorem there is no rule predicate symbol which was commensurate with the new goal of **parent (Tom, Pat)**, so this goal proves to be unsuccessful. This suggests that resolvent –empty disjunct((□) (withdraw from the source of a multitude of disjuncts cannot be the reasoning for this option. Now the algorithm makes a return to the original purpose, i.e. to the conclusion of the theorem to try the second alternative top-level target output **ancestor (Tom, Pat)**. Thus, try rule WP2

ancestor (X, Z):—

parent (X, Y),

ancestor (Y, Z).

As before, the variables have been attributed to the value **X** = Tom, **Z** = Pat. At this point, the variable **Y** is not yet attributed to any value. The original purpose of **ancestor (Tom, Pat)** is replaced by the two goals of a **parent (Tom, Y)** and the

ancestor of (Y, Pat), which are rezol'ventoj WP2 and original purpose–the conclusion of the theorem, received after the unification of comparable predicates.

Having now two goals, achieve them algorithm is trying to, in the order in which they are written. To achieve the first of these is easy, because it corresponds to the fact of parcels. Having completed the unification of predicate **parent (Tom, Y)** and predicate **parent (Tom, Bob)**, which stands in the premises first, the algorithm of variable **Y** assigns the value **Bob**. Thus the first goal and the second turns to the purpose of **ancestor (Bob, Pat).**

To achieve this goal, once again the rule is applied WP1. Note that this (second) application of rule WP1 has nothing to do with its first use. Therefore, the algorithm uses a new set of variables rules whenever it applies. To specify this, we rename the variables for the new WP1 rule its application as follows:

ancestor (X',Z’): —parent (X’,Z’).

The head of this rule matches the predicate symbol of current target **ancestor (Bob, Pat**), so the unification {Bob/**X’** Pat/**Z’**). The current target is replaced with a view to the **parent (Bob, Pat)-**new resolvent. Such an objective is achieved immediately, as occurs in the assumptions of the theorem as a fact. This step completes the calculation, because new clause-empty resolvent is gluing the negation of the conclusion of the theorem (as required by resolutions method) and the current target.

All the steps to achieve the goal of **ancestor (Tom, Pat)** are presented in Figure 4.4 as tree inference. As already pointed out, when considering Figure 4.1, tops the tree correspond to the goals or purposes of lists that you want to achieve.

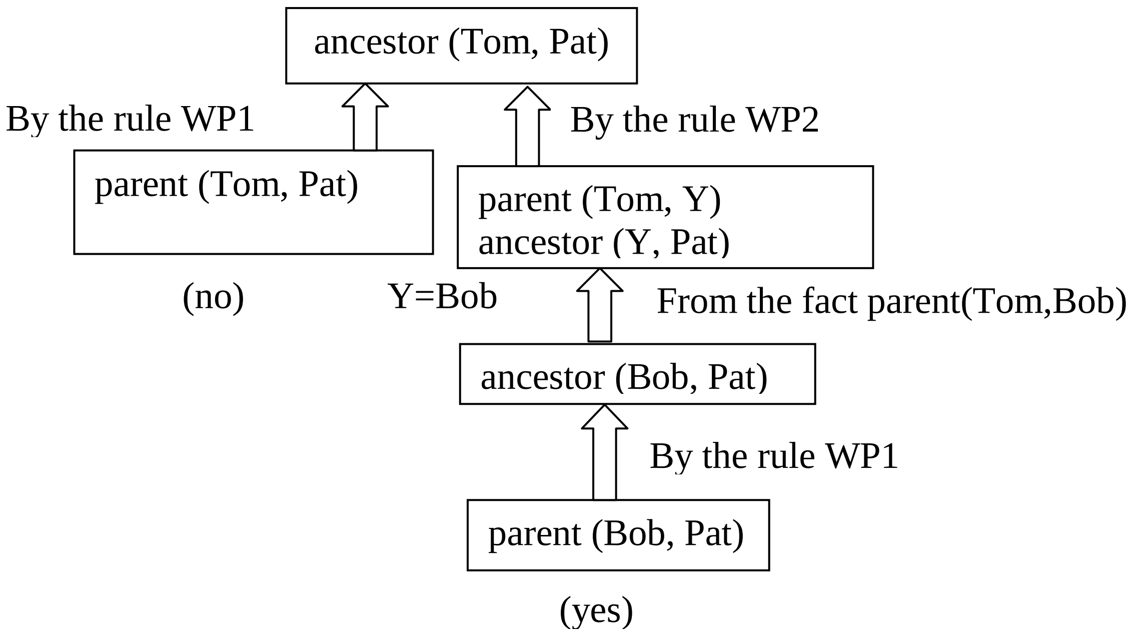


Figure 4.4. Tree inference in query ancestor (Tom, Pat).

5. Fuzzy knowledge

When attempting to formalize human knowledge researchers soon encountered a problem which has curtailed the use of traditional mathematical device to describe them. There is a whole class descriptions, operating quality characteristics of objects (a lot, a little, strong, very strong, etc.). These characteristics are usually blurred and cannot be uniquely interpreted, but contain sensitive information (for example, "one of the possible signs of flu is high temperature ").

In addition, the tasks performed by intelligent systems, often have to use implicit knowledge, which cannot be interpreted as completely true or False (Boolean true/false or 1/0). There are knowledge, the correctness of which is expressed in some intermediate figure such as 0.4.

How, without destroying the property of fuzziness and imprecision, submit such knowledge formally? To resolve such issues, American mathematician Lotfi Zadeh proposed formal apparatus of fuzzy (fuzzy) algebra and fuzzy logic [17]. Later this area is widely [18, 19] and marked the beginning of one of the branches of S under the name Soft Computing (soft computer).

L. Zadeh has introduced one of the major concepts in fuzzy logic-the concept of a **linguistic variable**.

Linguistic variable (LV) is a variable whose value is determined by a set of verbal (i.e. verbal) characteristics of some property. For example, the LP "growth" is defined through a set of {dwarf, low, medium, high, very high}.

Fuzzy system based on product type rules, but as a premise and conclusion rule used linguistic variables, thus avoiding the limitations inherent in the classic produkcionnym rules.

5.1. The basic concepts of the theory of fuzzy sets

The exact values of the input variables are converted to the value of linguistic variables through the application of certain provisions of the theory of fuzzy sets, namely, using certain membership functions.

Values of linguistic variable defines as fuzzy sets (FS), which in turn determine on a certain basis a basic set or a set of a numerical scale having a dimension. Each LV value is defined as a fuzzy set (for example, FS "low growth").

Specific definition of the degree of belonging is possible only when working with experts. When discussing the issue of FS in a linguistic strategy, it is interesting to estimate how much all FS is to some extent necessary for a sufficiently accurate representation of the physical quantity. At the present time, there is an opinion that for many applications, 3-7 FS per variable is sufficient. The minimum value of the number of FS is completely justified. This definition contains two extremes (minimum and maximum) and mean. For most applications, this is quite enough. With regard to the number of FS, it is not limited and does not depend on the applications and the required degree of description of the system. The number 7 is due to the capacity of short-term human memory, in which, according to modern ideas, up to seven pieces of information can be stored.

The fuzzy set is defined through some base scale B and against FS - μ (xi), xi B, which takes values at the level [0 ... 1]. Thus, a fuzzy set B is a collection of pairs of the form (xi, μ (xi)), where xi є B. Often there is such a record:

$$B = \sum_{i=1}^{n} \frac{x_i}{\mu(x_i)}$$

Where x_i.- i-e value of the base scale.

The form of the membership function can be absolutely arbitrary. Now the notion of the so-called basic membership functions has been formed (see Figure 5.1).

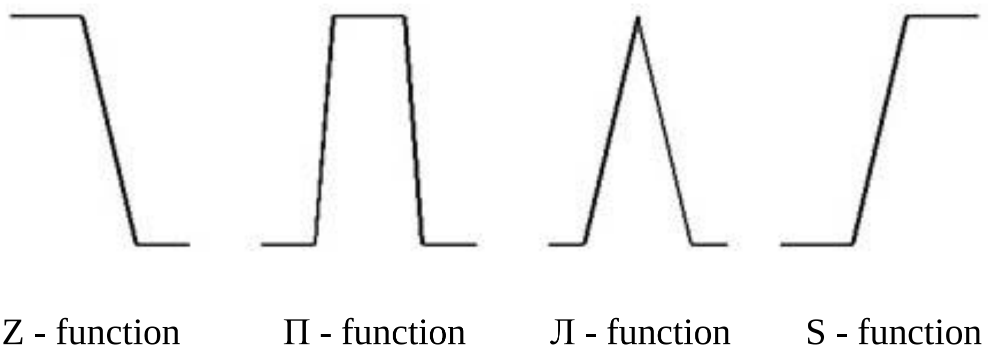


Figure 5.1. Standard membership functions

You can choose the basic rules for defining functions and give them some kind of algorithm for formalizing tasks in terms of fuzzy logic.

Step 1. For each term taken in linguistic periodicals, it is necessary to find a numerical value or a range of values that best describe the term. Since this value or value is the "prototype" of our term, a single value of the membership function is selected for them.

Step 2. After determining the indicators with a single identity, you need to determine the value of the parameter with the membership "0" to this term. This value can be selected as a value with membership "1" to another term from the number defined earlier.

Step 3. After determining the extreme values, you need to determine the intermediate values. For them, n- or h-functions are chosen from the number of standard functions.

Step 4. For the exponents corresponding to the extreme values of the parameter, the S- or Z-function accessories are selected.

Standard membership functions are easily applicable to most problems. However, if you want to solve a specific problem, you can choose a more suitable form.

Recognition that this particular value depends on the importance of FS. This function should not be confused with a probability that is objective and subordinate to other mathematical dependencies. For example, for a two-way fuzzy set, the "high" LV for "car price" in conditional units can differ materially from their social and financial situation.

"High_car_price_1" = {50000/1 + 25000 / 0,8 + 10,000 / 0,6 + 5000 / 0,4}.

"High_car_price_2" = {25000/1 + 10000 / 0,8 + 5000 / 0,7 + 3000 / 0,4}

Let's consider an example.

Let us face the task of interpreting the values of LV "age", such as "young" age, "advanced" age or "transitional" age. We define "age" as LV (Fig. 5.2). Then the "young," "advanced," "transitional" will be the values of this linguistic variable. More fully, the basic set of LV values "age" is as follows:
B = {infant, child, young, young, mature, advanced, senile}.

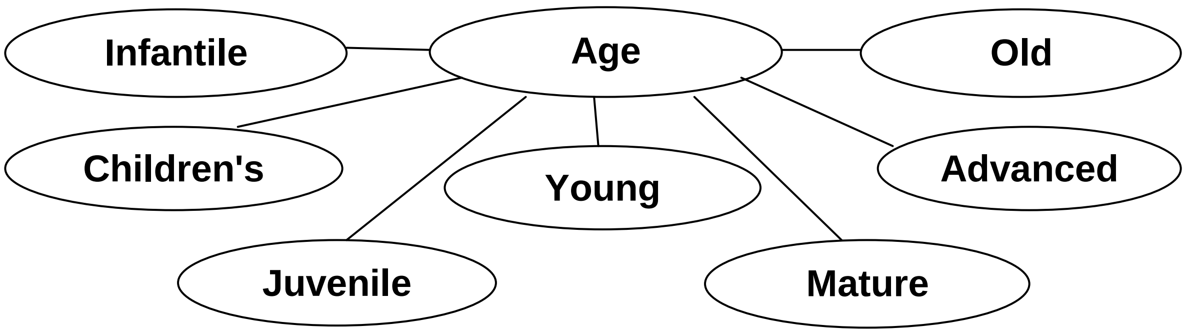


Figure 5.2. The linguistic variable "age" and fuzzy sets that determine its value

For LV "age" the basic scale is a numerical scale from 0 to 120, indicating the number of lived years, namely the function determines how much we are confident that this number of years can be attributed to this age category. In Fig. 5.3 reflects how the same values can scale can participate in the definition of different FS.

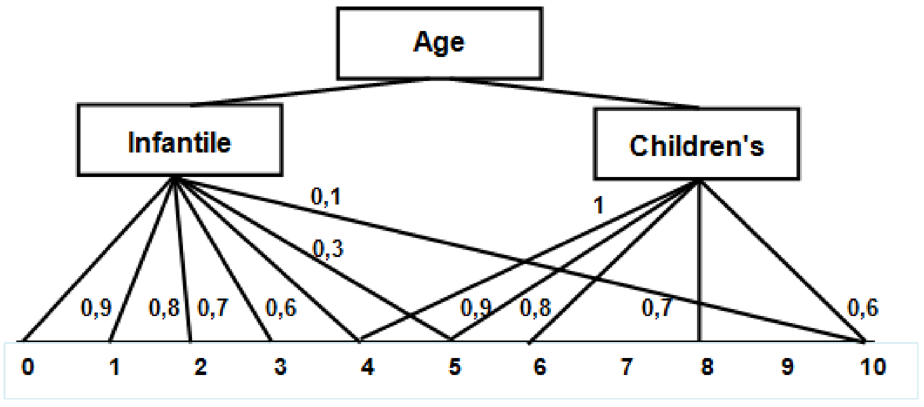


Figure 5.3. Formation of fuzzy sets for the basic set of LV values "age"

For example, to determine the value of FS "infant age" can be as follows:

$$' \quad \textit{Infantile} = \left\{\frac{0.5}{1} + \frac{1}{0.9} + \frac{2}{0.8} + \frac{3}{0.7} + \frac{4}{0.6} + \frac{5}{0.3} + \frac{10}{0.1}\right\}$$

Figure 5.4 illustrates the estimate of FS by some average expert who performs up to six months with a high degree of reliance on infants ($\mu = 1$). Children under four years of age are considered the same, but with a level of confidence ($0.5 < \mu < 0.9$), and within 10 years the child can be considered an infant.

Thus, fuzzy sets allow to take into account subjective opinions of separate individuals (experts) when defining concepts.

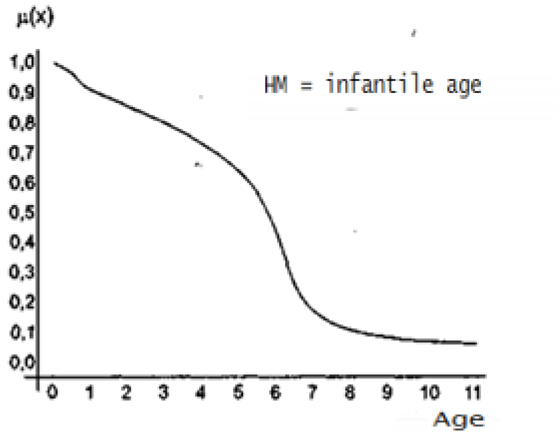


Figure 5.4 Chart of belonging function
to the fuzzy set "infantile age"

5. 2. Operations with fuzzy knowledge

After the identification of all NMs and their distribution functions, they begin to develop a base of knowledge based on fuzzy rules. Most fuzzy systems use production rules to describe the dependencies between linguistic variables. A typical production rule consists of antecedent9 (part IF ...) and consequent (part THEN ...). An antecedent may contain more than one parcel. In this case, LPs are combined by logical connectives AND or OR.

There are many different ways for operations with fuzzy knowledge, expressed by linguistic variables. These methods are basically heuristics. When presenting knowledge, experts face the uncertainty problem of some characteristics. Special methods are developed for the account of these uncertainties:

9Antecedent (from Latin antecedens - preceding) - in the conditional statement "If A, then B" saying "A"; Saying "B" is called the consequent. For example, in the conditional statement "If it's day now, it's light", the antecedent is the saying "Now is the day". Sometimes the term "A." is used in the broad sense to refer to a premise, reason, cause, condition, etc.

5.2.1. The method of «fuzzy logic»

We will specify for the example rules of several operations (Zade's fuzzy logic)

1) Two fuzzy sets A and B are equal if $\mu_A(u_i) = \mu_B(u_i), \quad u_i \in u.$

2) A fuzzy set A contains in B (A⊂B),if $\mu_A(u_i) \le \mu_B(u_i), u_i \in u.$

3) Addition of fuzzy sets A – $\overline{A}$:

$$\overline{A} = \{\mu_{\overline{A}}(u_i) = 1 - \mu_A(u_i)\}, u_i \in U.$$

1) Integration of A and B- $A \cup B$(operation OR) where the result is:

$$C = \sum_{i=1}^{n} \{\max\{\mu_A(u_i), \mu_B(u_i)\} / u_i\}$$

(5) Intersection $A \cap B = C$ (operation AND) where

$$C = \{\mu_C(u_i) = \min(\mu_A(u_i), \mu_B(u_i))\}$$

(6) Algebraic product:

$$A * B = C = \{\mu_C(u_i) = \mu_A(u_i) * \mu_B(u_i)\}$$

.

It should be noted that the classical probabilistic approach is different in defining the aggregation operation:

$$\mu(x) = \mu 1(x) + \mu 2(x) - \mu 1(x) * \mu 2(x)$$

The strengthening or weakening of linguistic concepts is achieved through the introduction of special quantifiers. For example, if the term "senile age" *A* is defined as:

$$A = \left\{ \frac{60}{0.6} + \frac{70}{0.8} + \frac{80}{0.9} + \frac{90}{1} \right\}$$

then the notion of "*very* senile age" will be defined as

$$con(A) = A^2 = \sum \frac{x_i}{\mu_i^2}$$

that is, "*very* senile age" equals:

$$A= \left\{ \frac{60}{0.36} + \frac{70}{0.64} + \frac{80}{0.81} + \frac{90}{1} \right\}$$

5.2.2. Certainty factor Method

The "certainty factor" method was developed by one of the first author of Shertliff. He introduced the notion of a certainty factor to measure the degree of approximation to some conclusion h, be the result of the e-evidences at that time.

CF[h:e] = B[h:e] - MOD[h:e]

Where, CF - a certainty factor, B is a belief, h is a hypothesis, e is an evidence, MOD is a measure of distrust.

The value of CF = [-1; 1], where

-1-absolute lie

+ 1 absolute truth

0-Absolute ignorance.

Again, that B and MOD-[0; 1] are not probabilistic characteristics, but the opinion of the expert. For hypothesis h, with two evidences e1 and e2:

B [h:e1; e2] = B [h:e1] + B [h:e1] (1-B [h:e2])

The efficiency of the second evidence on hypothesis h, when setting the evidence e1, results in the removal of the belief to a full certainty of the distance dependent on e2. This formula has two properties:

1. Symmetrical as respects to e1 and e2.
2. As evidences accumulate, belief moves towards certainty.

Rules can be supplied with a divider ratio of 0-1 that indicates the reliability of that rule.

The main drawback of the method is that it is very difficult to distinguish the case of conflicting evidences from a case of inadequate information.

5.2.3. "Bayesian Approach" (another name is the evaluation of competing hypotheses)

The probability of the implementation of some hypothesis H with certain supporting evidences E is calculated on the basis of the priori probability of the hypothesis without corroborating evidences and the probability of the evidences being implemented, in the circumstances that the hypothesis is correct or incorrect under the known Bayes formula.

Example: P (h) is a prior probability that the patient is suffering from influenza. It is important to know that the patient is suffering from influenza. It is known that he has a high fever. P (h:e) is the probability that if a patient has the influenza, he has a high fever.

Bayes ' formula makes it possible to adjust the posterior conditional probability of hypothesis H if evidence is available E.

$$P(H:E) = \frac{P(H \& E)}{P(E)}$$

With a simple transformation, this formula can be put in a more convenient form:

$$P(H:E)=\frac{p^{+}p}{p^{+}p+p^{-}(1-p)} \qquad (1)$$

Here on the right are easily identifiable assumed prior probabilities: $p=P(H),\ p^{+}=P(E:H),\ p^{-}=P(E:не\,H)$.

Another formula is required to operate the system, which recalculates the conditional probability in the absence of evidence (the user's negative response), because in some cases this may result in an increase in the conditional (posteriori) probability of the hypothesis

$$P(H:не\,E)=\frac{(1-p^{+})p}{1-p^{+}p+p^{-}(1-p)} \qquad (2)$$

It is assumed that the evidences is statistically independent, therefore when considering the current evidence as prior probability, it is accepted as the priori probability of the hypothesis.

Since the KB system of fuzzy logic is probabilistic, the user is given the opportunity to determine the existence of the evidence e also with some uncertainty by expressing its response q, for example in the ten-point system: -5 (firm NO), 0 (don't know), 5 (firm YES). It is assumed that the dependence of the conditional probability of hypothesis on the magnitude q has piecewise character(see fig. 5.5).

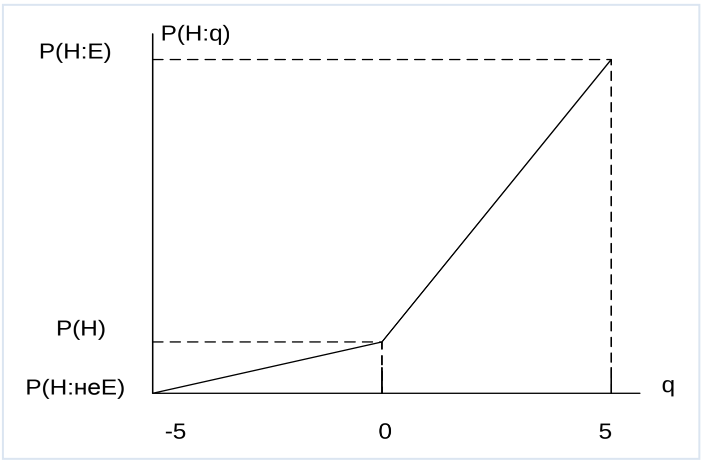


Figure 5.5. The dependence of the conditional probability of hypothesis on the magnitude q

Here $P(H:E), P(H:не\,E)$ are defined by formulas (1), (2), in which $p=pt\equiv P(H)$ is the current prior probability of a hypothesis with reference to the remaining evidences.

According to this plot, after the user responds, the unknown conditional (new current posterior) probability of the hypothesis is adjusted by the following formulas

$$P(H:q) = P(H) + (P(H:E) - P(H))\frac{q}{5}, \quad 0 \le q \le 5$$

$$P(H:q) = P(H:\text{не}\,E) + (P(H) - P(H:\text{не}\,E))\frac{(5+q)}{5}, \quad -5 \le q < 0$$

From a programmer's point of view, it is sometimes easier to use a characteristic called a "chance".

Transformation of probability to "chance": O = P/(1-P)

For example, if the probability of recovery is P = 0.3, then the chance to be healed O= 0.3/(1-0.3) = 0.43.

The opposite calculation: P = O/(1 + O).

In some systems, the so-called likelihood ratio is used: LR(h):

$$LR(h) = P(h:e)/P(h:\neg e)$$

$$LR(e) = P(e:h)/P(e:\neg h)$$

For example, it is the probability of getting a positive result of the diagnostic test from the patients to the probability of getting a positive test result from the healthy people.

Then the chance after experiment:

O'(h)=O(h)*LR(h:e)

Likelihood ratio LR>0 always, moreover the LR>1 is a testimonial of a hypothesis, but LR<1 is against hypothesis.

The value of LR = 1 indicates that the evidences does not affect on the LR hypothesis. Sometimes can be found LR = 0 or ∞. Then you must match the data to avoid these values.

LR shows how much more likely the hypothesis becomes when evidences are available than in the absence of evidences. If the evidences is unreliable, then:

LR’ = LR* WE + (1-WE)

Here, WE - the weight of the evidence is the probability that the evidence is reliable.

Thus, the main benefits of the Bayesian approach are the following opportunities:

1) multiple independent data sources can be combined
2) the formula is easily corrected for calculation after the evidence has been shown.

For output on fuzzy sets the above special relations and operations are used over them. One of the first uses of the FS theory was the use of certainty factors for output recommendations of the medical expert system MYCIN [20].

This method uses several heuristic devices. It became an example of the processing of fuzzy knowledge that had affected the development of successor systems. Fuzzy expert systems, in addition to their main advantage — the best adaptedness to the realities of the real world — have two advantages over traditional ones. First of all, they are free from what is called <spinlocks>in the building of conclusions. Secondly, the different bases of fuzzy rules can easily be combined, which is rarely possible in common expert systems. At present time, most of the tools for developing knowledge-based systems include elements of working with FS, in addition, special software tools for implementing so-called fuzzy output, for example the "shell" FuzzyCLIPS, have been developed.

In a broader sense, fuzzy control is a methodology for creating management systems in which the mapping between real input and output parameters is represented by fuzzy rules. Fuzzy control proved to be very successful in commercial products such as automatic gearboxes, camcorders and electric razors. Critics of this approach affirm that such applications have been successful because they use small rules bases, logical conclusions do not form chains, and parameters can be adjusted to improve system performance. Indeed, the fact that the rules of operation of these systems are implemented by fuzzy operators may not be a key factor in their success; the secret is to use a concise and intuitive way to set a smoothly interpolated function with real values.

6. Inference control strategy

Among the known models of knowledge representation discussed in section 2, the production model has become most widespread. When using the production model, the knowledge base of the artificial intelligence system consists of a set of rules (product cores). The program that manages the search of rules is called the output machine [1]. Let's take a closer look at the tasks and work of the production output machine.

Any expert system of the production type must contain three main components: rule base, working memory, and output mechanism [8].

The rule base (RB)-formalized by the rules of the products knowledge of a specific subject area.

Working memory (WM) is an area of memory that stores a lot of facts describing the current situation and all the attribute-value pairs that have been set at a particular time. The contents of WM in the process of solving a task change normally, increasing in volume as rules are applied. In other words, WM is a dynamic part of the knowledge base, the contents of which depend on the environment of the task to be solved. In the simplest ES, the facts stored in WM do not change while the task is being solved, but there are systems where you can modify and delete facts from WM. These are systems that work in a situation of incomplete information.

The output mechanism performs two main functions:

• review of existing in working memory facts and rules from the RB, as well as adding new facts to the WM;

• determination of the order of viewing and applying rules. The order can be direct or reverse.

Direct order-from facts to conclusions. In expert systems with direct outputs on known facts, a conclusion, which follows from these facts is sought. If this conclusion is found, it is placed in the working memory. Direct outputs are often used in diagnostic systems, and are called data-driven outputs.

The backward chaining order-from the conclusions to the facts. In the systems with backward chaining, there is a hypothesis of the final judgment first, and then the output mechanism tries to find facts in the working memory that could confirm or refute the hypothesis. The process of finding the necessary facts may include a fairly large number of steps, and new hypotheses (objectives) can be put forward. Backward chaining are controlled by the goals.

In the vast majority of knowledge-based systems, the output mechanism is a small program and consists of two components — one that implements the output itself (output component), and the other manages the process (the control component). The output component action is based on the application of the rule called the **modus ponens**.

The rule of the **modus ponens** (used in the predicate logic and it is essentially one of the methods of the output rules), from two expressions *A*, and $A \rightarrow B$ displays the new expression *B*.

In other words, if it is known that the truly assertion *A* and expression *"A"* brings (imply) expression *"B"* (i.e. there is "if A,then B") rule, then the expression *B* is also true.

Let's look at an example. If the "barometer falls" statement is true, and there is a rule "if the barometer falls, the rain is possible". Then the "rain possible" statement is also true.

The rules are working when there are facts that satisfy their left side: if the premise is true, then the conclusion must also be true.

Let's look at an example of the output of a solution in a logical model based on an output rule, a **modus ponens**.

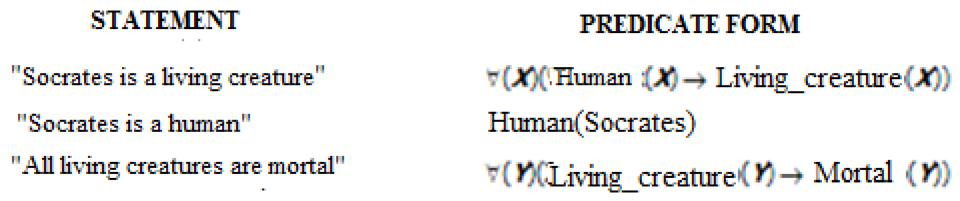
Statements are given:

- "Socrates is a human";
- "Human is a living creature";
- "All living creatures are mortal".

It is required to establish the assertion **"Socrates is mortal"**.

Solution:

Step 1. Represent the statements in the predicate form:



STATEMENT	PREDICATE FORM
"Socrates is a living creature"	$\forall(X)(\text{Human}(X) \rightarrow \text{Living\_creature}(X))$
"Socrates is a human"	Human(Socrates)
"All living creatures are mortal"	$\forall(Y)(\text{Living\_creature}(Y) \rightarrow \text{Mortal}(Y))$

Step 2. Based on the output rule (the modus ponens) and substitution (Socrates/X) in the first predicate, we obtain the assertion:

"Socrates is a living creature."

Step 3. Based on the output rule (the modus ponens) and substitution (Socrates/Y) in the third predicate, we obtain the assertion:

"Socrates - mortal"

The output component should function even if there is lack of information.

The resulting solution may not be accurate, but the system should not stop due to the fact that any part of the input information is missing.

Therefore, the control component determines how the rules are applied and performs four functions.

1. Correlation -the sample (antecedent) rules are mapped to the existing in WM facts.
2. Conflict set resolution-selection of the one of several rules providing that they can be applied simultaneously
3. Selection- if in a particular situation several rules can be applied at once, then one of them is chosen, the one most suitable for the given criterion (conflict resolution).
4. Rule triggering - if the rule sample when matching coincides with any facts from the working memory, then the rule is triggered and it is marked in the RB.
5. Manipulation-the working memory is modified by adding to the conclusion the triggered rule. If the right side of the rule contains a reference to an action, it is executed (as in the information security systems, for example).

Rule interpreter works cyclically. In each cycle, it looks through all the rules to identify those whose premises coincide with the known facts from the working memory. After the selection, the rule fires, its conclusion is stored in the working memory, and then the cycle is repeated again.

Only one rule can work in a single cycle. If several rules are successfully mapped to the facts, then the interpreter makes a choice according to a certain criterion of the single rule that is triggered in this cycle. The cycle of the interpreter's operation is schematically shown in fig. 6.1.

Information from working memory is matched sequentially with the premises of the rules to identify successful matching. The totality of the selected rules is the so-called conflict set. To resolve a conflict, the interpreter has the criteria by which it selects a single rule, and then it fires. This is expressed in the recording of the facts that form the conclusion of the rule, in working memory or in changing the criterion for choosing conflicting rules. If the conclusion of a rule includes the name of an action, it is executed. The work of the output machine depends only on the state of the working memory and on the composition of the base knowledge.

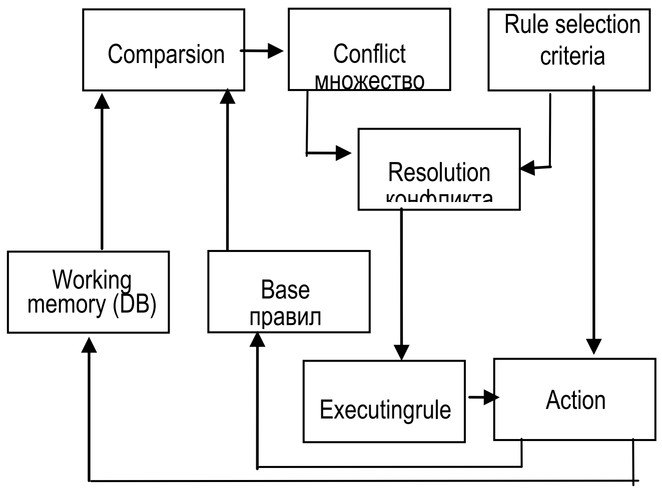


Figure 6.1 The cycle of the output rule interpreter

In practice, the work history is usually taken into account, that is, the behavior of the output mechanism in previous cycles. Information about the behavior of the output mechanism is stored in the status memory (Figure 6.2). Usually, the status memory contains a log of the system.

From the selected search method, that is, the output strategy, the order of application and operation of the rules will depend. The selection procedure is reduced to determining the direction of the search and the method of its implementation. The procedures that implement the search are usually "wired up" to the output mechanism, so in most intelligent systems, knowledge engineers do not have access to them and, therefore, can not change anything in them at will.In practice, the work history is usually taken into account, that is, the behavior of the output mechanism in previous cycles. Information about the behavior of the output mechanism is stored in the status memory (Figure 6.2). Usually, the status memory contains a log of the system.

From the selected search method, that is, the output strategy, the order of application and operation of the rules will depend. The selection procedure is reduced to determining the direction of the search and the method of its implementation. The procedures that implement the search are usually "wired up" to the output mechanism, so in most intelligent systems, knowledge engineers do not have access to them and, therefore, can not change anything in them at will.

When developing a management strategy for the conclusion, it is important to identify two issues:

1. Which point in the state space should be taken as the starting point? From the choice of this point depends on the method of implementing the search - in the forward or reverse direction.

2. What methods can improve the search efficiency of the solution? These methods are determined by the chosen strategy of enumeration - in depth, in width, in subtasks or in some other way.

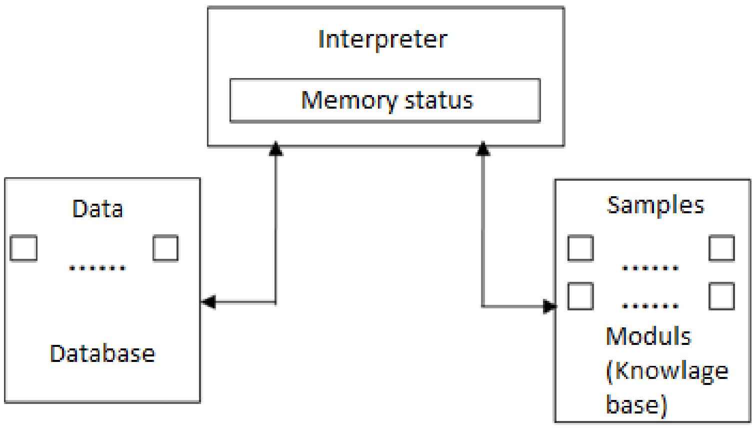


Figure 6.2. Schema of the rules interpreter

The state space is a graph whose vertices correspond to the situations encountered in the problem ("problem situations"), and the solution of the problem is to find the path in this graph.

6.1. Direct and reverse output

In the reverse order of the derivation, a hypothesis is first put forward, and then the withdrawal mechanism goes back to facts, trying to find those that support the hypothesis (Figure 6.3, left side).

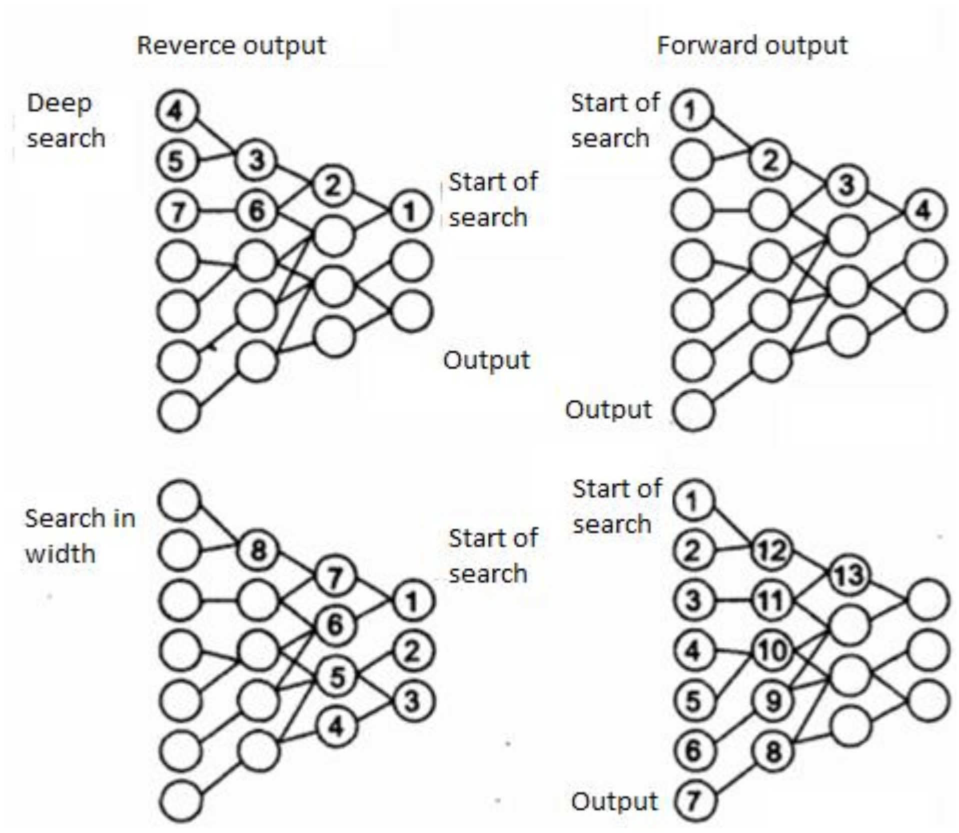


Figure 6.3. Output strategy

If the hypothesis turns out to be correct, then the following hypothesis is chosen, which details the first and is a sub-goal in relation to it. Further, facts are found that confirm the truth of the subordinate hypothesis. Conclusion, in which the search for evidence begins with a targeted statement. The conditions under which the target statement is deducible are clarified. These conditions are taken for new target statements and the search process continues. IN. (Conclusion Reverse) ends when all the next conditions turn out to be axioms or the process conditions terminates without leading to axioms. A derivation of this type is called a managed goal, or a managed sequential. Reverse search is used in cases where goals are known and there are relatively few of them.

In systems with direct derivation from known facts, a conclusion is found which follows from these facts (see Figure 6.3, right-hand side). If this conclusion can be found, it is stored in the working memory. The derivation leading from the original axioms to the target expression. With C.D. (Conclusion Direct) because of the ambiguous choice of the applicable axioms and the rules of inference, a decision tree is formed and the process of finding the chain leading from the original axioms to the target expression is exhaustive. The standard procedure used to traverse the decision tree is the return-backtracking procedure. Direct output is often called a data-driven output, or output controlled by antecedents.

There are systems in which the conclusion is based on a combination of the above methods - the inverse and the limited direct. Such a combined method is called cyclic.

Example 1:

There is a fragment of the knowledge base of the production system, which has two rules:

R1. If "rest is in summer" and "person is active", then "go to the mountains".

R2. If "he loves the sun," then "rest in the summer."

Suppose the system received the facts - "an active person" and "love the sun."

DIRECT OUTPUT - based on actual data, get a recommendation.

1st passage.

Step 1. We try R1, it does not work (there is not enough data for "rest in summer"). Step 2. We try R2, it works, the fact "rest - in the summer" comes to the base.

2nd passage.

Step 3. We try R1, it works, the goal "to go to the mountains" is activated, which acts as the advice given by the expert system.

REVERSE CONCLUSION - confirm the selected target using the available rules and data.

1st passage.

Step 1. The goal is to "go to the mountains": try R1 - there is no "rest in summer", they become a new target and a rule is sought where the goal is on the left.

Step 2. The goal of "rest in summer": rule R2 confirms the goal and activates it.

2nd passage.

Step 3. We try R1, the desired goal is confirmed.

6.2. The essence of the basic control strategies of the output

In intelligent decision-making systems with a knowledge base of hundreds of rules, it is desirable to use an output management strategy that minimizes the time for finding a solution and thereby improves the efficiency of the output. The number of such strategies are [1-4]:

1. depth search,
2. Search in width,
3. subdivision into subtasks,
4. alpha-beta algorithm,
5. strategy of simplicity / complexity,
6. LEX-strategy,
7. MEA-strategy.

Consider the simplest example of problems - to build a meaningful word from a certain set of letters (Cyrillic letters – “к”, “о”, “т”). At each level, add the letter (Figure 6.4).

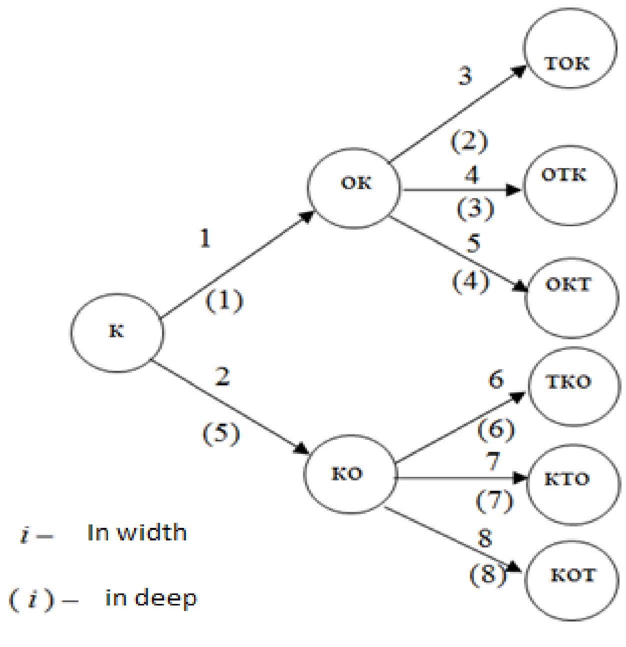


Figure 6.4. Search strategies in width and depth

As the number of levels on the graph increases (here - the inclusion of other letters in the set), an exponential growth of the number of nodes is observed, the so-called combinatorial explosion. Therefore, the actual task is to choose an appropriate search strategy. Consider them.

When searching in depth, the next sub-goal is chosen as the one that corresponds to the next, more detailed level of the task description. The rules selected in the list of applications based on data that were included in the working memory relatively recently are located in this list earlier than the rules, in the selection of which the older data is used. For example, the diagnostic system, making known on the basis of known symptoms the presence of a certain disease, will continue to request clarifying signs and symptoms of this disease until it fully confirms or refutes the hypothesis put forward. On the graph this corresponds to the construction of the descendants of the node, and then returns to the neighboring evil at the same level of the graph.

The depth search algorithm can quickly find a solution, especially if it uses heuristics to select the next branch (you need 4 steps to search for the Oct combination). But this algorithm can never end if the state space is infinite.

When looking at the width, the rules selected in the list of orders based on data that was included in the working memory for a relatively long time are located in this list earlier than the rules, in the selection of which the more recent data was used. For example, the diagnostic system will first analyze all the symptoms that are on the same level of the state space, even if they are related to different diseases, and only then go to the symptoms of the next level of detail. On the graph this corresponds to the construction of all the neighbors of the node at the same level, and then its descendants are built. A search algorithm in width searches for a solution whose path to the graph is the shortest path, if it exists, that is, it finds the shortest path between the initial state and the solution. Algorithms that have this property are called solvable.

Subtargeting - involves allocating subtasks, the solution of which is seen as the achievement of intermediate goals towards the ultimate goal. An example confirming the effectiveness of the strategy of subdivision into subproblems is the search for malfunctions in the computer - first the failed subsystem (power, memory, etc.) is detected, which greatly narrows the search space. If you can correctly understand the essence of the problem and optimally divide it into a system of hierarchically connected goals-sub-goals, then you can achieve that the path to its solution in the search space will be minimal.

Alpha-beta algorithm allows you to reduce the state space by removing branches that are unpromising for successful searching. Therefore, only those vertices that can be accessed as a result of the next step are viewed, after which unpromising directions are excluded. The alpha-beta algorithm has found wide application mainly in systems focused on various games, for example, in chess programs.

The simplicity/complexity strategy is determined by the number of verification operations that need to be performed when analyzing the conditions of this rule. Preference is given to simpler or vice versa, more complex rules.

LEX-strategy assumes at first removal from the list of applications of all rules which already were earlier used. The remaining rules with an equal convexity value are then sorted by the "newness" of the data used. If it turns out that the two rules use the same "freshness" data, then preference is given to the rule that involves more data in the prerequisite analysis.

The MEA-strategy is in many respects similar to the previous one, but when analyzing the novelty, only the first conditions in the premise of the rules are taken into account. If it turns out that there were two candidates with equal indicators in the list of applications, then the mechanism of the LEX-strategy is used to choose between them. MEA is an abbreviation of one of the first methods for solving artificial intelligence problems by constructing an inverse chain of reasoning. Mean-Ends Analysis (means-analysis of the result).

In modern FIS, LEX and MEA strategies are used more often, and the LEX has proved to be a good general strategy, while the MEA is an effective strategy for solving more specific tasks such as planning.

In addition to those discussed above, the so-called **heuristic search strategy** is also known, in which additional knowledge about the problem is connected to the search algorithm. A simple form of heuristic search is "climbing a mountain." Here, in the search process, some evaluation function is used, with the help of which it is possible to roughly estimate how "good" the current state is. Here is the algorithm:

1) Being at a given point in the state space, apply the rules for generating a new set of possible solutions, for example, the set of moves of chess pieces allowed in a given position.

2) If one of the new states is the solution to the problem, then stop the process. Otherwise, go to the state that is characterized by the highest value of the evaluation function and return to step 1.

To difficulties in the implementation of the algorithm of climbing, one can include such.

- How to set an evaluation function?
- How to act when all possible moves are equally good or bad (when climbing - "exit to the plateau")?
- And if there are local maxima, of which only descent is possible, that is, "deterioration" of the state (take the queen and then lose)?

Another form of the heuristic method is "at first the best." It compares not only those states in which a transition from the current one is possible, but all that can be "gotten to" (look around as much as possible of the state space and be ready, if necessary, go back to where we already were and go Another way). A variation of this form is "the first best", at which the function **f (n) = g (n) + h (n)** is minimized, where

g(n) is the distance on the graph from node n to the initial state of the search space;

h(n) is the distance on the graph from node n to the final state of the search space.

Conclusions on the management strategies of the conclusion:

1) The problem of any complexity, in principle, can be reduced to the problem of search in the state space, if only it can be formalized in terms of the initial state, the final state and the transition operations in the state space.

2) The search in the state space should be guided in a certain way by the knowledge presented about a particular subject area.

The approach based on strategies for finding solutions in production ES is known for a long time. Very popular in the early 90's, ES GURU (INTER-EXPERT) also used similar mechanisms for managing search strategies. The ability to change strategies in the course of solving problems in a programmatic manner and accumulating experience, which strategies give the best results for certain classes of tasks, allows us to obtain effective mechanisms for finding solutions in SDR on the basis of products.

In concluding this lecture, it should be noted that there are various methods for finding solutions in semantic networks, for example, the method of traversing the semantic network is multiparasing. This method is original in that it allows you to "parallel" several markers in parallel and, thus, parallelize the process of searching information in a semantic network, which increases the speed of searching. These methods are used, as a rule, when presenting the text in the form of an object-oriented semantic network and are not considered here.

Frame-based search, Case-based Reasoning (CBR), plausible reasoning, fuzzy search methods, and other methods for finding AI solutions are also not considered here. They are recommended to be studied independently.

7.Agent technologies in distributed data analysis

7.1. The concept of software agent, tasks, structure, properties

One of the consequences of the development of the idea of organizing distributed computing through the transfer of executable code has become an increasing interest in so-called software agents and technologies for their use.

E. Tanenbaum proposed the following definition [1]. ***A software agent*** is an autonomous process capable of responding to the execution environment and causing changes in the execution environment, possibly in cooperation with users or other agents. At the same time the agent is affected by himself from the environment [21 – 23].

E. Tanenbaum also introduced the classification of agents, where the following main types are distinguished.

Stationary and mobile agents. Mobile agents, in contrast to stationary agents, are able to move from one node of the computing environment (VS) to another.

Cooperative and competing. Cooperative agent is able to unite with other agents to solve a common problem. Competing agent is able to compete with other agents in order to protect the interests of its owner (for example, trading agents on the stock exchange).

Agents can be used to solve the following tasks:

– mobile computing: agent migration can be supported not only between permanently connected nodes to the network, but also between mobile platforms connected to the permanent network at certain periods of time and possibly through low-speed channels. The client connects to the permanent network for a short period of time from the mobile platform, sends the agent to executethe task and disconnects. Then the client connects to another point on the network and takes the agent's results. The second option is the server where the agent must move, connects to the network, and then disconnects. In this case, the agent must be able to move to such a temporarily connected server and return to the permanent network.

– information management tasks;

– search for the information (a lot of information - one person is not able to find the information he needs and analyze it – the use of the agent, who travels the net for searchingthe information, best suited to the needs of the person). Search agents contain information about various information sources (including the type of information, the accessmethod to it, and such information source characteristics as reliability and accuracy of data);

– selection (processing) of information. Of all the data that coming to the client, select only those data that may be interestingfor the client. Used in combination with searching agents (at first – search, then – selection);

–data monitoring. Notify the user of changes in various data sources in real time (for example, the mobile agent moves to the compute node where the data source is located, this is more effective than using a static agent sending requests to the data source);

–universal access to data. Agents are intermediaries for working with various data sources, and have mechanisms for interaction with each other (for example, the agent creates several agents, each of those works with its data source).

In 1996 S. Franklin and A. Grasser proposed the following general definition of the agent:

Autonomous agent is a system that is inside the environment and is part of it, perceives this environment (its signals) and affects the environment to execute its own program of actions.

The following main components of an autonomous agent can be distinguished (Figure 7.1):

1. Sensors: agent blocks that provide information about the environment and other agents;

2. Actuators: blocks of the agent, providing an impact on the environment.

During work, a simple automation agent is guided by a standard set of rules"If-then" (Figure 7.2)

An autonomous agent must have the following properties:

- reactivity;
- autonomy;
- purposefulness;
- communicativeness.

Different authors do not interpret these properties in the same way. We will make an effort to explain them more detailed.

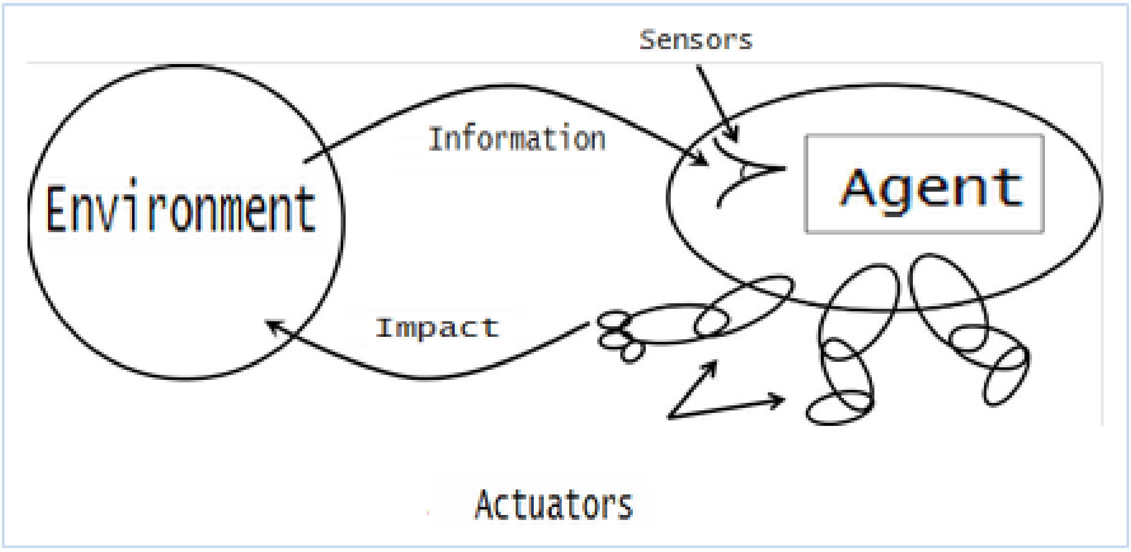


Figure 7.1. Autonomous agent.

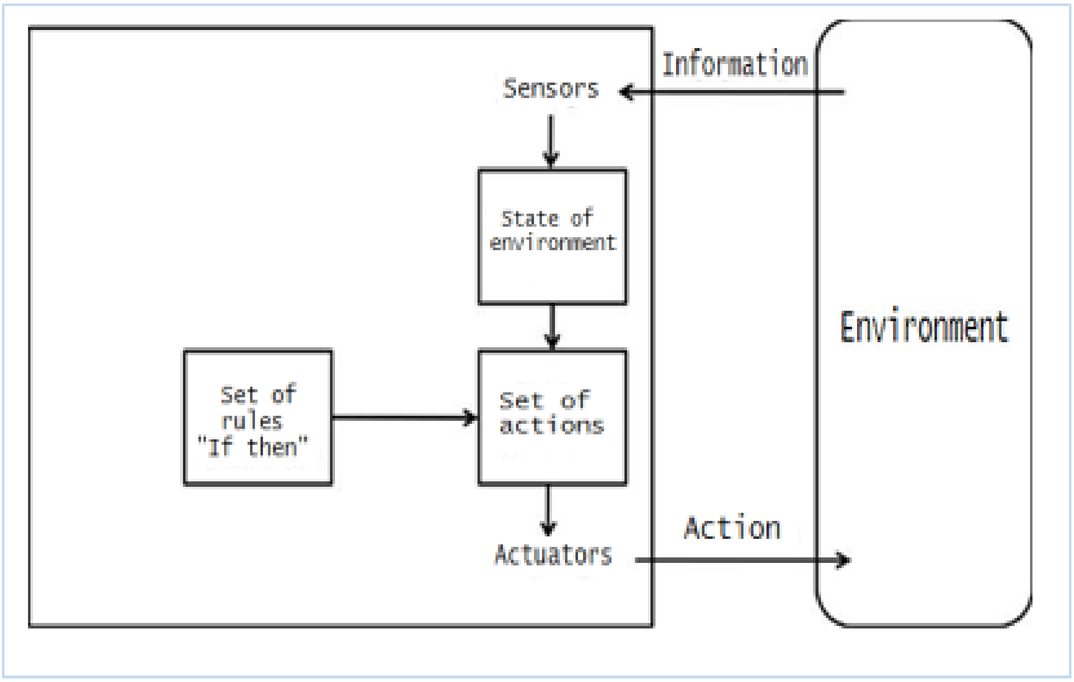


Figure 7.2. The structure of an autonomous agent.

1. The property of reactivity means that the agent responds to changes in the environment from time to time. The agent has sensors, through which he receives information from the environment. The sensors can be very different. These can be microphones that perceive acoustic signals and convert them to electrical, video capture cards, computer keyboards or a common memory area into which the environment puts data and from which the software agent takes data for calculations. Not all changes in the environment become known (available) to the agent's sensors. This is quite natural. After all human does not perceive sounds with a frequency exceeding 30 kHz, radio waves, etc. So the environment is not completely observable for the agent. Similarly the agent affects the environment through a variety of executive mechanisms including shared memory. Of course the power of impact as well as the power of perception is limited. An agent can translate an environment from some state to some other, but not from any to any.

2. The property of autonomy means that the agent is self-governing, he controls his actions by himself. The software agent on some server has the ability to "self-launch". It does not require any special actions from the user to ensure its start (just as we do "double-click" on a certain file icon).

3. The purposefulness property means that the agent has a specific goal and his behavior (impact on the environment) is subordinated to this goal, and it is not a simple response to signals from the environment. In other words, the agent is a control system and not a managed object.

4. The property of communicativeness means that the agent communicates with other agents (including people) using some language. This is not necessarily a single language for all agents. It is enough when a pair of communicating agents has a common language. Language can be complicated, like, for example, natural language. But it can also be primitive: the exchange of numbers or short words. If the verbose phrases of a complex language usually carry all the information in themselves, then the words of a simple language assume a "silence": both sides of the dialogue "know" what is being talked about (as in the well-known anecdote about numbered anecdotes).

5. Autonomous agents possessing the learning property are singled out as a separate category of intelligent agents. The learnability property means that the agent can adjust his behavior based on previous experience. It is not just the accumulation of environmental parameters in memory, i.e. using of historical data, but the comparison of the history of their own actions with the history of their influence on the environment, and the change their program of action in this regard.

6. One of the most important features of an agent is intelligence. The intellectual agent has certain knowledge about himself and the environment, and he is able to determine his behavior on the basis of this knowledge (Figure 7.3).

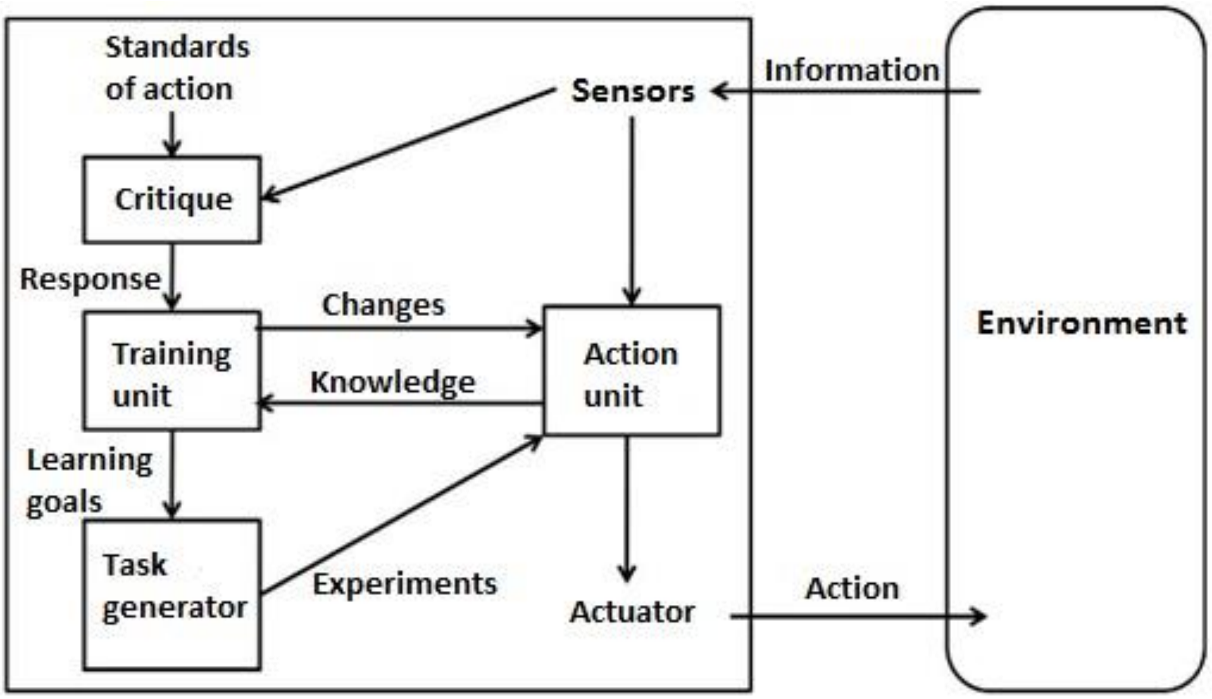


Figure 7.3. Structure of the intellectual agent.

Intellectual agents are the main field of interest of agent technology. An agent's environment is also important: it can be a real world or a virtual one, which becomes important due to the wide spread of the Internet.

It is required the ability to learn and even self-learning from agents [21, 22]. The ability to plan their actions divides agents into regulatory and planning.
Planning properties. If the ability to plan is not provided (the regulatory type), the agent will constantly reassess the situation and renew its impact on the environment. The planning agent can schedule several actions at different time intervals. In this case the agent can simulate the development of the situation, which makes it possible to respond more adequately to current situations. In this case the agent must take into account not only its actions and reaction to them, but also must preserve the models of objects and agents of the environment to predict their possible actions and reactions.

7.2. Multiagent systems

The aggregate of several agents working together and therefore having the property of sociability is called the multiagent system.

Multiagent systems can be used to solve problems that are difficult or impossible to solve with a single agent or a monolithic system.

It is not necessary that all agents interact (communicate) with each other in a multiagent system. In an extreme case there is no communication at all. Such systems we will be called discrete multiagent systems. The second extreme case that each agent communicates with everyone else. Such system is called a fully connected multiagent system.

A multiagent system acting as a single agent should be characterized by some common goal for all subagents and coordination of actions to achieve this goal. Since there are other situations where agents are not so closely connected, and such systems can be called agent societies. However the absence of a unified goal does not deny the possible group behavior of agents. But it is rather episodic than systematic.

An important difference of a multiagent system from a program or one agent is that the software agents (at least some of them) that were included in the system were not designed specifically for this system. Maybe it's reusable agents or agents designed to solve more universal tasks. In these cases agents have their own goals that do not coincide completely with the goals of the system (organization), but are compatible with them. Nevertheless, they can be useful to each other to solve the problems facing them and, therefore, from this point of view, the property of communicativeness is very important for them.

From the organizational point of view, there are common goals of the whole community, and these common goals are expressed primarily in roles (which agents play) and the norms of interaction.

Researches in the field of decision support systems in recent years have increasingly shifted from creating systems in the form of a traditional "toolbox" to the paradigm of collaboration and the integration of independent applications. The rapidly growing field of research of intelligent agents and multi-agent systems offers the possibility of creating more efficient systems based on a unified approach.

7.3. Agent platforms

The agent platform is a software shell that can create, interpret, launch, move and destroy agents [22]. As "air" for the agent, the agent platform provides him with an environment for execution. Just like the agent, the agent platform is associated with the authority that determines the organization or person and the system works on their behalf.

The agent system is uniquely identified by the name and address (Figure 7.4).

The platform includes at least one location and a connection interface - the ***communication infrastructure*** (CI). The location provides an agent runtime environment on the computer. It can contain several agents at the same time. CI implements a communication service, a name service and a security service.

Several agent systems can be located on one machine. The type of agent system describes the aggregate of agent parameters. For example, if the type of agent system is "aglets", it means that the agent system is created by IBM, supports the Java language as the agent implementation language, and uses Java Object Serialization to turn agents into serial form. The network region administrator determines communication services for intraregional and interregional interactions (Figure 7.5).

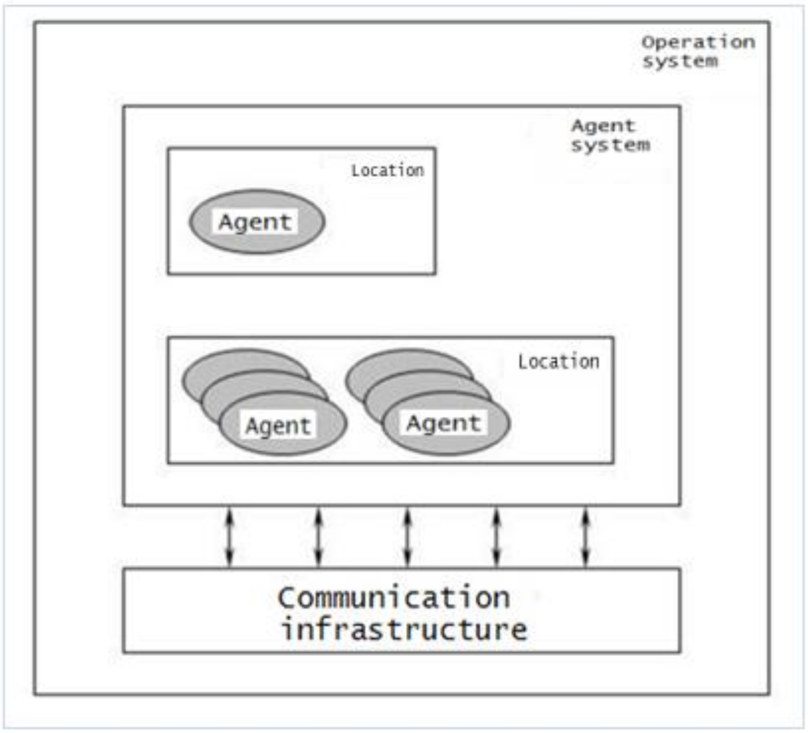


Figure 7.4 The agent system

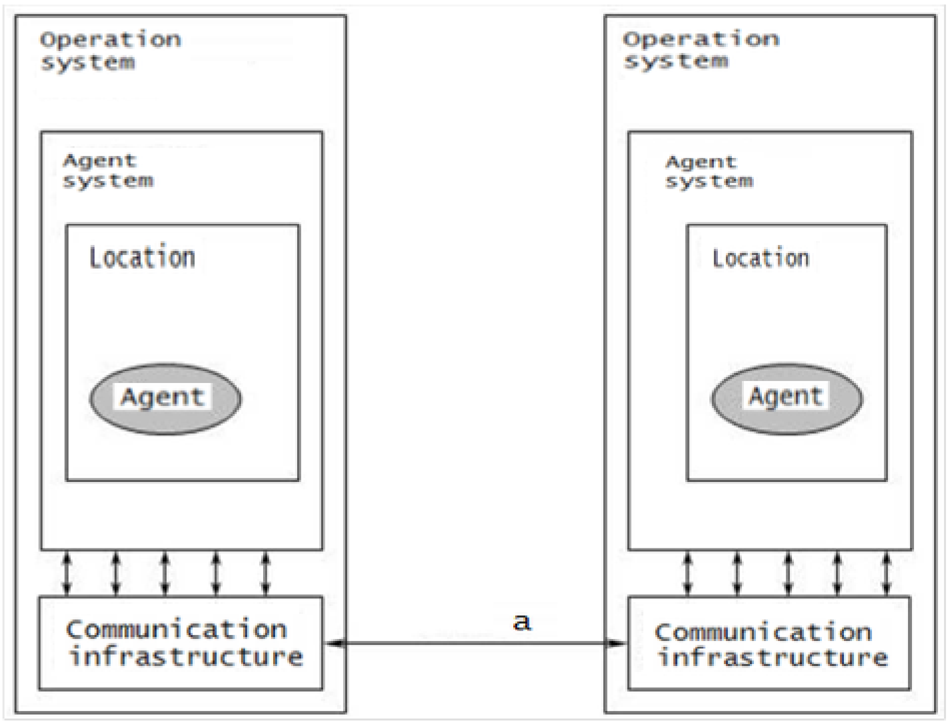


Figure 7.5. Relations between agent systems.

The communication infrastructure provides transport communication services (for example, TCP / IP), a name service and a security service for agent systems.

Currently, there are several dozen MAC. Many of them (Gypsy, JADE, Ajanta, JATLite etc.) have been developed in universities to study this technology.

Some systems (such as ASDK, Xelopes, JAFMAS, etc.) exist at the library level, providing to the programmer only base classes for implementing the main components: agents, platforms, mechanisms of interaction and security. On their basis, independent systems are developed, for example, MagNet, E-Commercia. Recently, commercial systems have appeared, such as Gossib of Tryllian company, Bee-gent and Plangeny of Toshiba Corporation. Unfortunately, the documentation for them is not available.

7.4. Security in mobile agent systems

Agent distributed computing systems have found a certain niche in the field of military communications, but principle of their workhampertheir widespread distribution: it is unlikely that a sensible administrator of the computer network will allow users to travel to autonomous intelligent, self-learning software systems that can migrate and be executed on any computing node. In this regard, there are a number of serious problems related to the security of agent platforms.

On the one hand, agents can carry personal information to ensure their own work. For example, an agent in an e-commerce system may contain a credit card number and a user's passport data in order to make transactions on his behalf. Accordingly, it is necessary that the environment provide an agent-safe execution environment.

On the other hand, a third-party agent may attempt to attack the underlying environment and retrieve data or acquire other resources. In this case, the agent platform should be sufficiently well protected and be able to confront such attacks.

Distinguish the following possible security problems in the operation of agent platforms:

1. Agent attacks Host: Agent can steal or modify host data.
2. The host attacks the Agent: The host can steal or modify the Agent's data, change its status or code
3. A malicious agent attacks another agent;
4. Attack by other elements.

The attack variant "Agent attacks the Host" is a standard attack where code obtained from an unreliable source tries to gain full access to the system or to interfere with the normal performance of tasks by increasing its enforcement powers. In this case, traditional security measures help with the attack, such as: access level control; sanders; authentication; cryptography.

Also, there are agent-specific security methods, for example, the analysis of the agent's traffic history. In this case, the platform can learn about the history of the movement of the agent from the log and draw a conclusion about its quality on the basis of information on which platforms he visited before.

The attack option "Host attacks agent" is more complicated in providing security. In such situations traditional facilities do not work, because host must have complete information about the agent code for its execution. In this case, the following tools can be used:

1) Mobile cryptography: functions and data of the agent are encrypted in such way that the host cannot understand how the functions work and extract the code. The disadvantage of this method is that we need to search for an encryption scheme for arbitrary functions, as well as we need to transfer the cropping key.

2) Safe movement: migration only to certain (trusted) hosts.

3) Use of fictitious data: in database, the system analyzing the work of agents, a fictitious data set is stored, which do not change during the normal operation of the agent.

4) Use of trusted hardware: these can be smart cards, integrated circuits, etc.

Thus, the basic idea of using agent technology in the field of data analysis is to encapsulate in the agent a cycle of extracting knowledge from the data. In this case, the agent acts as an analyst. To solve the problems of data mining, various types of mobile agents that use various analysis algorithms are implemented and successfully applied.

Literature

Basic

1. Базы знаний интеллектуальных систем / Т.А. Гаврилова, В.Ф. Хорошевский — СПб: Питер, 2000. – 384 с.

2. Джексон П. Введение в экспертные системы / Питер Джексон; Пер. с англ. и ред. В.Т. Тертышного . – 3-е изд. – М. : Вильямс, 2001 . – 622 с.

3. Попов Э.В. Экспертные системы: Решение неформализованных задач в диалоге с ЭВМ. – М.: Наука, 1987. – 288 с.

4. Смолин Д.В. Введение в искусственный интеллект: конспект лекций. – 2-е изд., перераб. – М.: ФИЗМАТЛИТ, 2007. – 264 с.

5. Нильсон Н. Принципы искусственного интеллекта.– М.: Радио и связь, 1985. – 376 с.

6. Вагин В.Н. Дедукция и обобщение в системах принятия решений. – М.: Наука, 1988. – 384 с.

7. Поспелов Д.А. Структурное управление: теория и практика. – М.: Наука, 1986. – 288 с.

8. Уэно Х. и др. Представление и использование знаний. Пер. с япон./ Под ред. Х.Уэно, М. Исидзука. – М.: Мир, 1989. – 220 с.

9. Доорс Дж. и др. Пролог – язык программирования будущего. Пер с англ. ; Предисловие А.Н. Волкова. – М.: Финансы и статистика, 1990. – 144 с.

10. Ин Ц., Соломон Д. Использование Турбо-Пролога. – М.: Мир, 1993. – 608 с.

11. Марселлус Д. Программирование экспертных систем на Турбо Прологе. – М.: Финансы и статистика, 1994. – 256 с.

12. Лорьер Ж.Л. Системы искусственного интеллекта. Пер. с франц. – М.: Мир, 1991. – 568 с.

13. Люгер Дж. Ф. Искусственный интеллект: стратегии и методы решения сложных проблем = Artificial Intelligence: Structures and Strategies for Complex Problem Solving / Под ред. Н. Н. Куссуль. – 4-е изд. – М.: Вильямс, 2005. – 864 с.

14. Искусственный интеллект. Кн.2. Модели и методы: Справ. / Под ред. Д.А. Поспелова. – М.: Радио и связь, 1990. – 304 с.

15. Методические указания к изучению курса «Основы искусственного интеллекта» для студентов всех форм обучения специальности 22.03

«Системы автоматизации проектирования» / Сост. А.И. Кондратенко. – К.: КПИ, 1992. – 60 с.

16. Братко И. Алгоритмы искусственного интеллекта на языке Prolog – М.: Издательский дом «Вильямс», 2004. – 640 с.

17. Заде Л. Понятие лингвистической переменной и его применение к принятию приближенных решений. – М.: Мир, 1976. – 166 с.

18. Тэрано, Т., Асаи, К., Сугэно, М. Прикладные нёчеткие системы. - М.: Мир, 1993. – 368 с.

19. Рутковская Д., Пилиньский М., Рутковский Л. Нейронные сети, генетические алгоритмы и нечеткие системы: Пер. с польского И. Д. Рудинского. – М.: Горячая линия – Телеком, 2004. – 452 с.

20. Рассел С., Норвиг П. Искусственный интеллект: современный подход = Artificial Intelligence: a Modern Approach / Пер. с англ. и ред. К. А. Птицына. – 2-е изд. – М.: Вильямс, 2006. – 1408 с.

21. Радченко И.А. Интеллектуальные мультиагентные системы: учебное пособие.– СПб: Балт. гос. техн. ун-т, 2006. – 88 с.

22. Ahmed S., Karsiti M.N. (eds.) Multiagent Systems. InTech, 2009, – 434 p.

23. Alkhateeb F., Al Maghayreh E., Abu Doush I. (eds.) Multi-Agent Systems - Modeling, Interactions, Simulations and Case Studies. InTech, 2011, – 512 p.

Additional

24. Осуга С. Обработка знаний – М.: Мир, 1989. – 293 с.

25. Приобретение знаний / Под ред. Осуги С., Саэки Ю. – М.: Мир, 1990. – 304 с.

26. Трусов В.А. Теоретические основы систем искусственного интеллекта (конспект лекций по учебной дисциплине «Искусственный интеллект» для студентов специальностей 7.080401 и 7.080403. – Д.: НГА Украины, 1999. –126 с.

27. Макаров И.М., Лохин В.М., Манько С.В., Романов М.П. Искусственный интеллект и интеллектуальные системы управления. – М.: Наука, 2006. – 336 с.

28. Шиханович Ю. А. Логические и математические исчисления. – М.: Научный мир, 2011. – 256 с.

29. Дубровский Д.И. Искусственный интеллект: междисциплинарный подход. – М.: ИИнтеЛЛ, 2006. – 446 с.

30. Девятков В. В. Системы искусственного интеллекта / Гл. ред. И. Б. Фёдоров. – М.: Изд-во МГТУ им. Н. Э. Баумана, 2001. – 352 с.

31. Новак В., Перфильева И., Мочкрож И. Математические принципы нечёткой логики = Mathematical Principles of Fuzzy Logic. – М.: Физматлит, 2006. – 352 с.

32. Рутковский Лешек. Искусственные нейронные сети. Теория и практика. – М.: Горячая линия - Телеком, 2010. – 520 с.

33. Усков А. А., Кузьмин А. В. Интеллектуальные технологии управления. Искусственные нейронные сети и нечеткая логика. – М.: Горячая Линия – Телеком, 2004. – 143 с.

34. Круглов В. В. Дли М. И. Голунов Р.Ю. Нечёткая логика и искусственные нейронные сети. – М.: Физматлит, 2001. – 221с.

35. Дьяконов В.П., Круглов В.В. MATLAB. Математические пакеты расширения. Специальный справочник. – СПб.: Питер, 2001. – 480с (имеются главы по нечёткой логике и нейронным сетям).

36. Дьяконов В. П., Абраменкова И. В., Круглов В. В. MATLAB 5 с пакетами расширений. Под редакцией проф. В. П. Дьяконова. – М.: Нолидж, 2001. – 880с (имеются главы по нечёткой логике и нейронным сетям).

37. Дьяконов В. П., Круглов В. В. MATLAB 6.5 SP1/7/7 SP1/7 SP2+Simulink 5/6. Инструменты искусственного интеллекта и биоинформатики. – М.: СОЛОН-Пресс, 2006. – 456с.

38. Штовба С. Д. Проектирование нечетких систем средствами MATLAB. – М.: Горячая линия — Телеком.- 2007. – 288 с.

39. Uziel Sandler, Lev Tsitolovsky Neural Cell Behavior and Fuzzy Logic. Springer, 2008. – 478 с.

40. Искусственный интеллект: Применение в интегрированных производственных системах. Пер. с англ./ Под ред. Э. Кьюсиака. – М.: Машиностроение, 1991. – 544с

41. Братко И. Программирование на языке Пролог для искусственного интеллекта. – М.: Мир, 1990. – 560 с.

42. Гаврилов А.В. Системы искусственного интеллекта: Учеб. пособие: в 2-х ч. – Новосибирск: Изд-во НГТУ, 2001. – Ч.1. – 67 с.

43. Адаменко А.Н., Кучуков А.М. Логическое программирование и Visual Prolog. – СПб.: БХВ–Петербург, 2003. – 992 с.

44. Котов Ю.В. Как рисует машина. – М : Наука, 1988. – 224 с.

45. Роджерс Д., Адамс Дж. Математические основы машинной графики. – М.: Мир, 2001. – 604 с.

46. Кейтер Дж. Компьютеры- синтезаторы речи. –М.: Мир, 1985. – 237 с.

47. Частиков А. П., Гаврилова Т.А., Белов Д.Л. Разработка экспертных систем. Среда CLIPS. – СПб.: БХВ–Петербург, 2003. – 608 с.

Authors:
Udovik I.M., Korotenko G.M., Korotenko L.M., Trusov V.A., Kharj A.T.

Methods and systems of artificial intelligence

Tutorial

for students of
122 "Computer Science and Information Technologies"
(Discipline 12 Information Technologies)

Editorial and publishing complex

Signed in print _____.Format 30x42 / 4.
Offset paper. Ryzohrafiya.Md. printing.f. .
Obl.-view. sh .Circulation approx. Zam. №.

State Higher Educational Institution
"National Mining University"
49005, Dnipro, pr. D. Javornycjkyj, 19