Non-procedural problem-solving models in next-generation intelligent computer systems

Maksim Orlov, Anastasia Vasilevskaya
Belarusian State University of Informatics and Radioelectronics
Minsk, Belarus
Email: orlovmassimo@gmail.com, vnastyap@gmail.com

***Abstract*—In the article, an approach to the design of problem solvers of intelligent systems based on non-procedural models is considered. The developed approach makes it possible to integrate any problem-solving models, including the principles of logical inference, to solve problems based on a general formal model.**

***Keywords*—Knowledge-driven systems; logical problem-solving models; logical graph languages; production problem-solving models; functional problem-solving models.**

I. Introduction

Currently, the usage of intelligent systems in a variety of fields is becoming increasingly relevant. Modern Artificial Intelligence technology is a whole family of various private technologies focused on the developing and maintening various types of components of intelligent computer systems that implement a variety of models for information representation and processing, different problem-solving models focused on the development of various classes of intelligent computer systems [1].

Modern intelligent computer systems consist of a knowledge base, a problem solver, and an intelligent interface. As the analysis of such systems shows, problem solvers do not have the proper level of semantic compatibility, are not able to fully coordinate their actions when solving complex problems and, in principle, solve problems in conditions when the problems are insufficiently formalized and the algorithm for solving them is unknown in advance [2]. In this article, an approach to the implementation of non-procedural problem-solving models in next-generation intelligent computer systems, based on ensuring compatibility between different models, is considered. The main attention is paid to logical problem-solving models, as an example of a procedural model, by analogy of which an approach to the design of any other models is implemented.

Logic solves the problems of proving the truth of propositions, argumentation of a proposition, the problem of generating and refuting hypotheses. When solving problems using logical models, it is possible to clearly trace the process of "reasoning" of the system, to obtain a protocol for solving the problem, which is an important knowledge. Obtaining new logic formulas based on existing ones is carried out by logical inference.

Frequently, in modern logical inference systems, components such as the rule base, working memory, and the logical inference mechanism are distinguished. These components have a strict boundary and are sometimes implemented in different programming languages and using different models of knowledge representation. This approach significantly limits the compatibility level for the subsystems of these computer systems and the level of compatibility of computer systems with each other as a whole.

Another problem of the current state of systems in which logical inference is implemented is that the semantics of the processed information is not taken into account. The system receives a certain set of logical rules, inference rules, and factographic statements as input and applies these rules on a working memory model (on a set of facts). Considering the semantics of the processed information allows not only to increase efficiency in solving problems using logical models but also to increase the level of negotiability and compatibility of computer systems.

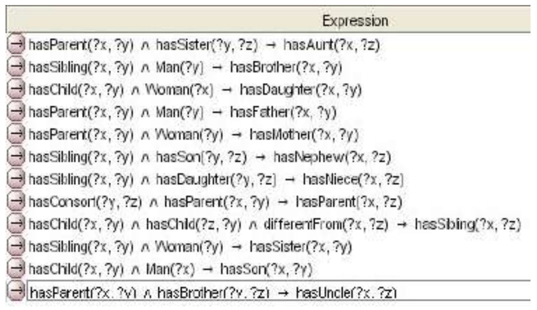
Likewise, no problem solvers have been developed that are able to combine different models for solving complex problems and ensure compatibility between them. Compatibility must be ensured not only within the same model, for example, compatibility of different logics, but also between different problem-solving models. When developing such problem solvers, it is important to notice not only the differences between different approaches, different logical models, but also their similarities.

The purpose of this work is not to develop a new problem-solving method or a new logic class, as well as to negate existing achievements in this field. The purpose of the work is to develop a model that allows integrating any problem-solving models and principles of logical inference for solving problems in intelligent systems based on a general formal model. In order to use any new or existing model, it is necessary to bring it to the formalism proposed in this article, which will allow integrating and synchronizing it with compatible components already available in the corresponding library.

II. Analysis of existing approaches to solving the problem

At the moment, many logical inference systems have been implemented [3], using the well-known rules of direct conclusion and resolution in various logic types, however, the problems of compatibility of the systems described above and collective problem solving using various problem-solving models remain relevant.

Each problem-solving model is defined by a language that provides a representation of a certain class of problem-solving methods in the memory of a cybernetic system and by an interpreter of these methods that defines the operational semantics of the specified language. It is necessary to consider the languages that can be used to set a logical problem-solving model. Such languages are Rule Interchange Format (RIF), Semantic Web Rule Language (SWRL), SHACL Rules, and Notation3 Rules, which are used in Semantic Web [4], [5]. In Figure 1, an example of rules in the SWRL language is represented.



Expression
hasParent(?x, ?y) ∧ hasSister(?y, ?z) → hasAunt(?x, ?z)
hasSibling(?x, ?y) ∧ Man(?y) → hasBrother(?x, ?y)
hasChild(?x, ?y) ∧ Woman(?x) → hasDaughter(?x, ?y)
hasParent(?x, ?y) ∧ Man(?y) → hasFather(?x, ?y)
hasParent(?x, ?y) ∧ Woman(?y) → hasMother(?x, ?y)
hasSibling(?x, ?y) ∧ hasSon(?y, ?z) → hasNephew(?x, ?z)
hasSibling(?x, ?y) ∧ hasDaughter(?y, ?z) → hasNiece(?x, ?z)
hasConsort(?y, ?z) ∧ hasParent(?x, ?y) → hasParent(?x, ?z)
hasChild(?x, ?y) ∧ hasChild(?z, ?y) ∧ differentFrom(?x, ?z) → hasSibling(?x, ?z)
hasSibling(?x, ?y) ∧ Woman(?y) → hasSister(?x, ?y)
hasChild(?x, ?y) ∧ Man(?x) → hasSon(?x, ?y)
hasParent(?x, ?y) ∧ hasBrother(?y, ?z) → hasUncle(?x, ?z)

Figure 1. Writing rules in the SWRL language

The described languages do not provide for the possibility of representing formulas in various logic types, so it is impossible to solve the described problems with them. Rule languages are specially built to infer conclusions. The syntax and semantics of ontology languages and rule languages are quite different, so the question arises how to combine them. There are several approaches, such as homogeneous and hybrid ones.

In a homogeneous approach, ontologies and rules are used on the same rights, i.e. a common language is created in which the same predicates are used both to express ontological statements and formulate rules (in particular, rules can be used to define classes and features of ontology). In this case, the problem of compatibility actually disappears, since the syntax and the interpretations become common – they only need to be extended to the rules, which is performed in a fairly standard way. The disadvantage of this approach is that combining different means in one language complicates its implementation greatly, and a homogeneous approach is often inapplicable, since ontologies and rule systems can be built independently by different specialists.

In a hybrid approach, the usual predicates, which are defined by rules (they can participate both in the conditions of rules and in their conclusions), and the predicates of ontologies, which are used as constraints in the conditions of rules, are strictly distinguished. The inference occurs through the interaction of individually implemented (existing) inference programs for rules and ontologies. The hybrid approach separates the builders of ontologies and rule systems from each other but also requires additional constraints to guarantee the solvability of the main problems for combinations of ontologies and rule systems (with solvable problems).

Semantic networks are convenient for representing knowledge of any kind, including logical formulas. The usage of semantic networks for deductive inference was researched by Quillian in 1966 [6]. He formally represented the semantics of natural language words and gave several examples of the inference technique. The deductive capabilities of Quillian were actually determined by the concept of “subclass” and the “modification” relation. The concept can be defined in terms of a more general concept and with the help of a modifying property, which is an “attribute – attribute value” combination.

An important technique used in semantic networks is a hierarchy, or classification system. In accordance with this technique, objects related to the subject domain are classified into a number of categories or classes based on their common properties. Using a hierarchical system in an extensive knowledge base of an intelligent system, it is especially convenient to use logical inference, since the inference that is valid for general concepts will be valid for particular concepts in relation to this general one. Despite the local success of such work, the systems remained static, non-extensible, and unable to be compatible.

Another important technique used in logical inference on semantic networks is knowledge localization [7]. The essence of localization is the possibility to identify an area of the semantic network in which subject knowledge are located (for example, constants, instances of classes), suitable for usage in logical inference premises. Taking into account the hierarchy of the knowledge base, it becomes most convenient to allocate a universe of reasoning, exceeding the scope of which is not advisable. Thus, the range of values of variables contained in the premises of logical formulas is limited, which allows reducing significantly the cost of searching in large knowledge bases.

Fuzzy inference systems [8], [9] are quite popular at the moment, whose semantic compatibility was also not considered.

III. Proposed approach

As part of this work, it is proposed to use an *OSTIS Technology* [10] as a basis, the principles of which make it possible to implement not just logical, production, functional, and other problem-solving models but also to ensure their compatibility, to implement a problem solver capable of combining various problem-solving models, including various logic types, to lay the foundation for creating interoperable computer systems.

The systems developed on the basis of the OSTIS Technology are called ostis-systems. The OSTIS Technology is based on a universal way of semantic representation of information in the memory of intelligent computer systems, called an *SC-code*. SC-code texts are unified semantic networks with a basic set-theoretic interpretation. The elements of such semantic networks are called *sc-elements* (*sc-nodes* and *sc-connectors*, which, in turn, depending on orientation, can be *sc-arcs* or *sc-edges*). The *Alphabet of the SC-code* consists of five main elements, on the basis of which SC-code constructions of any complexity are built, including more specific types of sc-elements (for example, new concepts). The memory that stores SC-code constructions is called semantic memory, or *sc-memory*.

The main advantage of using the SC-code for formalization and processing of logical formulas is that it provides compatibility between different problem-solving models. Any ostis-system has a problem solver, and there are problems for which the algorithm for solving them is unknown in advance and for which there is no ready-made method. The system must think and determine which agents can be involved in solving a particular problem.

The SC-code allows describing the relations between concepts of any form and complexity, which makes it a suitable option for using logical inference in next-generation intelligent computer systems, as well as using the hierarchy technique due to the ontological approach underlying the ostis-systems knowledge bases.

Within the technology, several universal variants of visualization of *SC-code* constructions are proposed, such as *SCg-code* (graphic variant), *SCn-code* (nonlinear hypertext variant), *SCs-code* (linear string variant).

Within this article, fragments of structured texts in the SCn and SCg codes [11] will often be used, which are simultaneously fragments of the source texts of the knowledge base, understandable to both human and machine. This allows making the text more structured and formalized, while maintaining its readability.

The basis of the knowledge base within the OSTIS Technology is a hierarchical system of subject domains and ontologies. Based on this, in order to solve the above problems, it is proposed to implement the following hierarchy of integrated subject domains:

Subject domain of logical formulas, propositions, and formal theories
$\Rightarrow$ *private subject domain**:
- *Subject domain of logical languages*
- *Subject domain of logical inference*

Subject domain of logical languages
$\Rightarrow$ *private subject domain**:
Subject domain of the propositional logic language

Subject domain of the propositional logic language
$\Rightarrow$ *private subject domain**:
Subject domain of the predicate logic language

Subject domain of logical problem-solving models
$\Leftarrow$ *private subject domain**:
- *Subject domain of logical languages*
- *Subject domain of logical inference*

Inheritance of subject domains allows using the described logics and their components in the description of any logics. The basic concepts allow developers of an intelligent system to add new logics. To implement a specific logical problem-solving model, it is necessary to create a subject domain that will be private in relation to the *Subject domain of logical problem-solving models* and the subject domain of some *logical language*, for example, the propositional logic language, the predicate logic language, the language of fuzzy logic, and others.

The *Subject domain of logical formulas, propositions, and formal theories* defines the denotational semantics of logical formulas, propositions, and formal theories and contains a formal specification of concepts necessary for the formation of logical formulas and propositions of any logics, including traditional, fuzzy, plausible, temporal, default logics, and any others. Logical formulas and propositions are interpreted using the concepts described in the *Subject domain of logical problem-solving models*, which includes a model and implementation of abstract agents necessary for solving logical problems. This subject domain includes the specification of concepts such as logical inference, inference rules, equivalent transformations, and axiom schemes.

Next, we will consider in more detail the fragments of sc-models of these subject domains and ontologies.

IV. Logical graph SCL language

Modern logic studies formal languages that serve to express logical reasoning. A logical language is a formal language intended to reproduce logical forms of natural language contexts, as well as to express logical laws and ways of correct reasoning in logical theories constructed in a given language. Logic does not study how knowledge was obtained – it allows representing knowledge, as well as deducing new knowledge from existing one (that is, deducing new formulas of the same logic from existing logic formulas), and establishing the accuracy of reasoning.

The **SCL Language** is a sublanguage of the SC-code for writing logical statements [12]. The SCL Language is a graph-type logical language used by ostis-systems. The texts of the SCL language are homogeneous semantic networks that are texts of the SC language. The alphabet of the SCL language is not allocated separately, since the alphabet of the SC-code is used, in which any statements, phenomena, regularities, programs, and any other knowledge can be described. The SCL language allows writing the texts of the propositional logic language, predicate logic language, and any other logical languages. The SC-code is a metalanguage for both the SCL language and for itself, that is, it allows describing the meaning of formulas written in SCL. Many formal languages, unlike SC, are not extensive enough to be a metalanguage for themselves. The specificity of the SCL language allocation is that the texts of this language can be processed in a special way. Logical inference inference can be made over propositions of the SL language.

One of the important features of SCL is its ability to represent predicate logic language texts taking into account the semantics of these texts (propositions). The SCL language is naturally oriented to work in the formal system of the predicate logic language. The SC language allows writing any relations and correspondences in a graph representation. The predicate value from a certain set of sc-variables corresponds to the result of a search operation on the template of some sc-construction (found or not found), which includes sc-constants and/or sc-variables with the corresponding configuration of relations between them. An approach based on the SCL language for the representation of formulas provides an opportunity to write generality and existence quantifiers not explicitly (this is not prohibited but is superfluous). The existence quantifier is an "embedded" concept in the sense that if some sc-element is included in some sc-structure, then the corresponding concept exists in this sc-structure. Thus, the existence quantifier is imposed automatically (unless another quantifier is explicitly imposed) on those sc-variables that are included in atomic logical formulas. The generality quantifier is imposed by default (unless another quantifier is explicitly imposed) on variables included in the equivalence and implication connectives in accordance with the denotational semantics of logical languages.

Such features simplify logical inference in the predicate logic in the SCL language, since this eliminates the need to bring the proposition into the Skolem normal form due to built-in quantifiers and the need for unification procedures conditioned by the search operation of the sc-construction by template, in which the necessary substitutions of variables occur.

V. Examples of formalizing statements in the SCL language

A proposition is understood as a certain structure (which includes sc-constants from some subject domain and/or sc-variables) or a logical connective that can be interpreted as true or false within any subject domain.

proposition
⇒ *subdividing**:
{• *atomic proposition*
• *non-atomic proposition*
}
⇒ *subdividing**:
{• *factographic proposition*
• *logical formula*
}

logical formula
⇒ *subdividing**:
{• *atomic logical formula*
• *non-atomic logical formula*
}

The truth of a proposition is set by indicating whether the sign of this proposition belongs to a formal theory corresponding to a given subject domain. The falsity of a proposition is set by specifying the belonging of the negation sign of this proposition to this formal theory.

In Figure 2, an example of a logical formula that is true within one formal theory and false within another is represented.

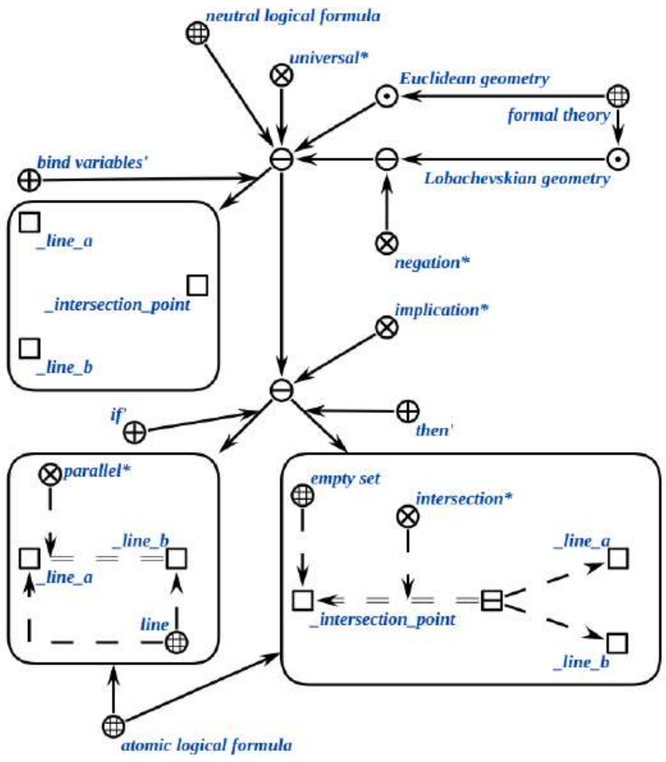


Figure 2. An example of a logical formula that is true within one formal theory and false within another

An *atomic logical formula* of the SCL language is interpreted as the set of all characters of some sc-text (sc-structure) containing at least one variable sc-element. Variables are free and bind subject variables that are intensional objects and are associated (have a value) with some constant element from the knowledge base. Figure 3 shows an example of an atomic logical formula that contains information about a triangle whose sine of the inner angle is equal to one.

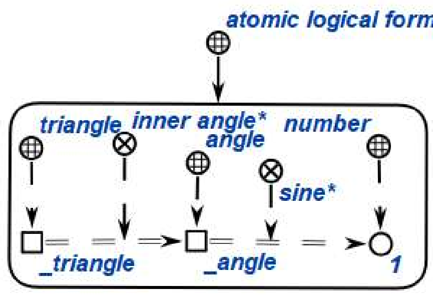


Figure 3. An example of formilizing an atomic logical formula

Each *non-atomic logical formula* of the SCL language is interpreted as a connective belonging to a relation corresponding to the type of non-atomic formula (conjunction, disjunction, negation, implication, equivalence, existence, universality) and linking the signs of the formulas included in the specified non-atomic formula. An example of a non-atomic logical formula is shown in Figure 4. This formula contains information that any triangle is either an acute triangle, or an obtuse triangle, or a right triangle.

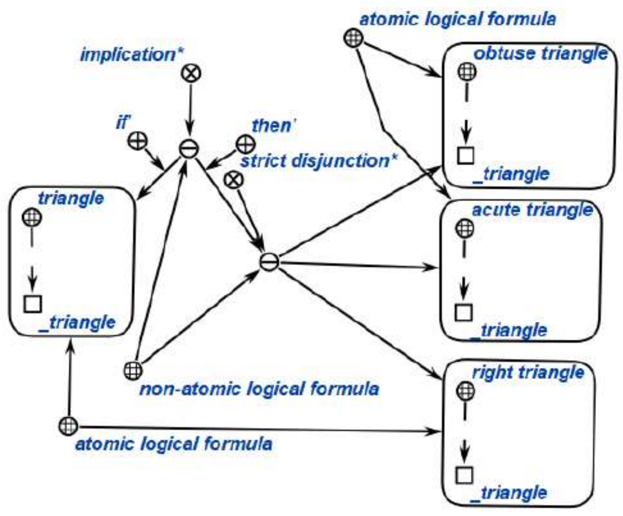


Figure 4. An example of formilizing a non-atomic logical formula

A statement is a semantic neighborhood of some logical formula, which includes the full text of this logical formula, as well as the fact that this logical formula belongs to some formal theory. The sign of a logical formula, the semantic neighborhood of which is a statement, is the main key sc-element within this statement. The signs of the concepts of the corresponding subject domain, which are part of any subformula of the specified logical formula, will be the key sc-elements within this statement.

The full text of some logical formula includes:

- the sign of this logical formula;
- signs of all its subformulas;
- elements of all logical formulas whose signs are included in this structure;
- all pairs of belonging that connect logical formulas whose signs are included in this structure with their components.

In Figure 5, there is an example of a statement that shows that the corresponding angles at the intersection of parallel lines of the secant are equal within the formal theory of Euclidean geometry.

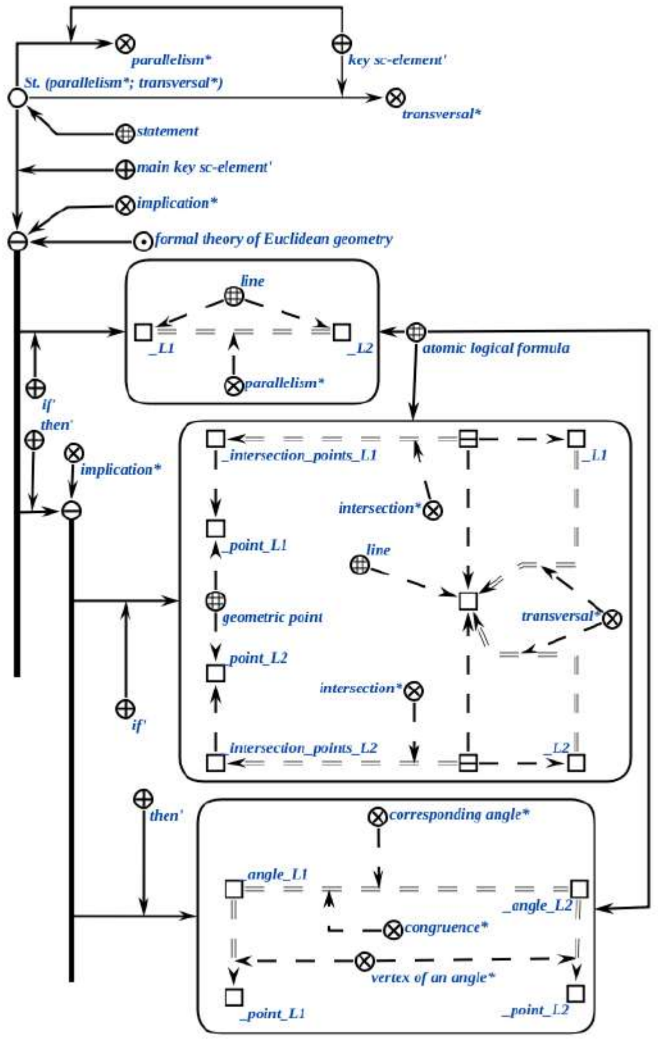


Figure 5. An example of the statement

A definition is a statement, the main key sc-element of which is a connective of equivalence that uniquely defines some concept based on other concepts. For the same concept within one formal theory, there may be several equivalence statements* that uniquely define some concept based on others, however, only one such statement within this formal theory can be marked as a definition. The remaining equivalence statements* can be interpreted as explanations of this concept.

In Figure 6, an example of a definition is given, which shows that a rhombus is a quadrilateral with all sides equal within the formal theory of Euclidean geometry.

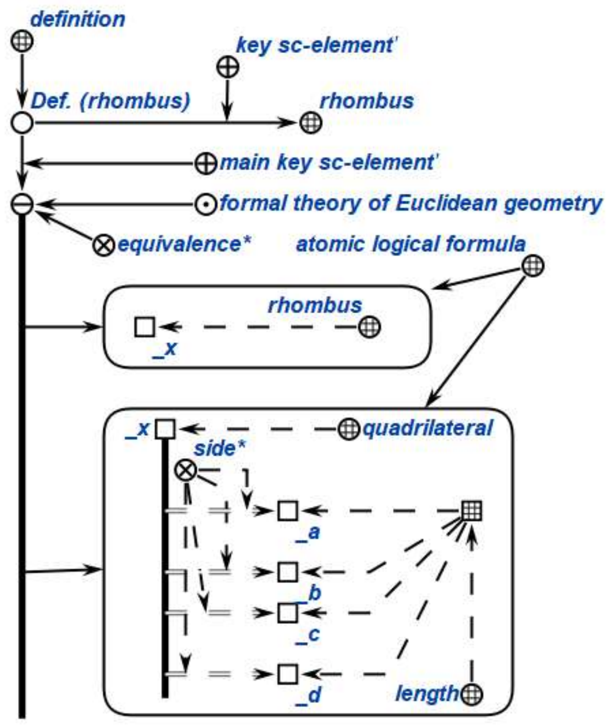


Figure 6. An example of a definition

VI. Machine of the SCL logical inference

An inference in a formal system is any sequence of formulas, so that any formula is either an axiom of this formal system, or a direct conclusion of any previous formulas according to one of the inference rules. The idea of deducibility is central to logic: in any formal axiomatic theory, a 'theorem' is a formula that is deduced from axioms. The correctness of conclusions is introduced and verified completely formally, without any connection with the truth of the premises included in it, i.e. exclusively from the point of view of the reasoning structure. From a practical point of view, the most important property of such formal correctness of reasoning is as follows: if we have managed to prove, using the methods of formal logic, the accuracy of the reasoning, and we know from experience that all the premises used are true, then we can be sure of the truth of the conclusion [13]. The truth of the premises used is set by the state of the knowledge base.

Various logical approaches allow designing problem solvers for intelligent systems in different subject domains, taking into account their specifics. The *Knowledge processing machine* for each specific system largely depends on the purpose of this system, the set of problems to be solved, the subject domain, and other factors. Some operations required in one subject domain will be redundant in another. For example, in a system that solves problems in geometry, chemistry, and other natural sciences, the usage of deductive inference methods will be reasonable, since the solution of problems in such subject domains is based only on reliable rules. In systems of medical diagnostics, for example, a situation constantly arises when a diagnosis can only be made with a certain degree of confidence and there can be no absolutely reliable answer to the question posed. In this regard, there is a need to use different knowledge processing machines in different systems, while the composition and capabilities of the knowledge processing machine in a particular system is determined not only directly by the developer but requires consultations with experts in this subject domain. Nevertheless, the basis for all logic types is classical logic, and its most general methods extend to other logics with some modifications, clarifications, and limitations.

Let us give a brief classification of existing logical problem-solving methods:

- **Classical deductive inference.** Classical deductive inference is the most popular in the building of automatic problem solvers, since it always gives a reliable result. Deductive inference includes direct, reverse, and logical inference (the resolution principle, the Erbran procedure, etc.) [13], all kinds of syllogisms [14], etc. The main problem of deductive inference is the impossibility of its usage in a number of cases when there is no reliable knowledge.
- **Inductive inference.** Inductive inference provides an opportunity to use various assumptions in the decision process, which makes it convenient for usage in poorly and difficultly formalizable subject domains, for example, in the building of medical diagnostic systems. The principles of inductive inference are discussed in detail in [15], [16].
- **Abductive inference.** In artificial intelligence, an abductive inference is usually understood as the inference of the best abductive explanation, i.e. the explanation of some event that has become unexpected for the system. Moreover, the "best" explanation is such one that satisfies special criteria determined depending on the problem being solved and the formalization used. The abductive inference is discussed in detail in [17], [18].
- **Fuzzy logic.** The theory of fuzzy sets and, accordingly, fuzzy logic is also used in systems related to difficultly formalizable subject domains [19], [20].

The theory of fuzzy logic is discussed in more detail in [9] and other publications.

- **Default logic.** The default logic is used, among other things, in order to optimize the reasoning process, supplementing the process of reliable inference with probabilistic assumptions in cases where the probability of error is extremely small. The default logic is discussed in more detail in the articles [21], [22].
- **Temporal logic.** The usage of temporal logic is very relevant for non-static subject domains in which the truth of a statement changes over time, which significantly affects the course of solving a problem [23], [24]. It should be noted that the knowledge representation language used in this work provides all the necessary capabilities for describing such dynamic subject domains.

A formal clarification of various information processing models in graphodynamic associative memory is **abstract graphodynamic associative machines**. The models of information processing, in particular, include models of parallel processing of knowledge corresponding to different logics and strategies for solving problems [25].

The advantage of using graphodynamic associative machines as a tool for creating next-generation intelligent computer systems is conditioned by the following aspects:

- the associative method of access to processed information is implemented in a fundamentally simpler way;
- it is much easier to maintain the open character of both the machines themselves and the formal models implemented on them;
- they are a convenient basis for the integration of various information processing models.

The other advantages of graphodynamic associative machines are conditioned by the advantages of graph texts and graph languages.

An **abstract scl-machine** is a logical inference machine, which belongs to the class of abstract sc-machines [12]. The internal language of the scl-machine is the above-mentioned SCL logical graph language, its operations correspond to the rules of logical inference. The family of specialized abstract graphodynamic knowledge processing machines is a formal clarification of the operational semantics of the above-mentioned specialized graph knowledge representation languages, each of which corresponds to one or more abstract machines.

These abstract machines correspond to different problem-solving models, different logics, different models of plausible reasoning. An agent from a family of logical inference agents can represent any inference rule that can be applied to solve a logical problem. In addition, agents are needed to perform equivalent transformations of a logical formula (for example, to write an equivalence formula as a conjunction of two disjunctions) and other agents that help apply inference rules on a set of logic language formulas.

Abstract scl-machine
$\Rightarrow$ *decomposition of an abstract sc-agent**:
{
- *Abstract sc-agent for applying the inference rule*
- *Abstract sc-agent of equivalent transformations of a logical formula*
- *Abstract sc-agent of direct logical inference*
- *Abstract sc-agent of reverse logical inference*

}

The purpose of an Abstract sc-agent for applying the inference rule is to apply a given inference rule with given logical formulas. This sc-agent is activated when an initiated action belonging to the class *action of applying the inference rule* appears in the sc-memory. After the sc-agent checks the initiation condition, the process of applying the inference rule is performed, which consists in checking whether there are structures in the sc-memory that correspond to the condition for applying this rule and generating sc-constructions in accordance with the applied rule. The Agent of applying the inference rule is often used in the operation of direct inference and reverse inference agents, as well as others. An example of an inference rule can be the Modus ponens rule shown in Figure 7.

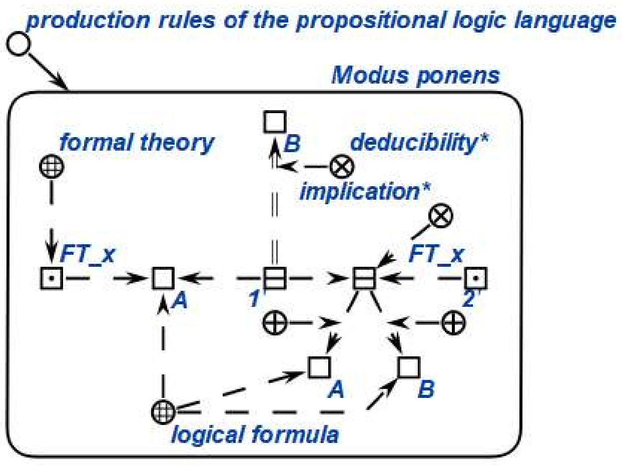


Figure 7. Formalization of the Modus ponens inference rule

The purpose of an Abstract sc-agent of equivalent transformations of a logical formula is to apply certain rules that bring the logical formula into a certain form. This sc-agent is activated when an initiated action belonging to the class *action of equivalent transformation of a logical formula* appears in the sc-memory. After the sc-agent checks the initiation condition, the process of converting the formula from one form to another is performed, while no new knowledge is generated in the sc-memory from the point of view of the subject domain under consideration. The response of this agent is a set of formulas that are equivalent in meaning but different in form of representation. As such forms, for example, conjunctive normal form or disjunctive normal form can serve. The Agent of equivalent transformation is often called during the operation of the agent for applying the inference rule, since logical formulas are not always in the form that is available for applying a particular inference rule but can be brought to the required form.

The purpose of an Abstract sc-agent of direct logical inference is to generate new knowledge based on some logical statements. This sc-agent is activated when an initiated action belonging to the class *direct logical inference action* appears in sc-memory. After the sc-agent checks the initiation condition, the process of direct logical inference is performed, which consists of cyclic operations of applying inference rules, generating new knowledge in sc-memory, and checking some condition, for example, the appearance of sc-elements from the target sc-structure in memory [26]. The input arguments of such an agent are the target structure, a set of formulas that are used during the inference by the agent of applying the inference rules, a set of inference rules, an input structure, and an output structure. As a result of performing the action by the agent of logical inference, an sc-structure is formed in the sc-memory, which is a decision tree. This tree consists of a sequence of nodes representing the applied rules that led to the appearance of the required knowledge in the sc-memory. Such a tree may be empty if the required structure could not be generated during logical inference. Figure 8 shows an example of the specification of the agent of direct logical inference.

The purpose of an Abstract sc-agent of reverse logical inference is to test hypotheses. Some hypotheses can be refuted, but by extracting the reasons why the hypothesis is refuted, it is possible to change the premise of the hypothesis so as to create a new hypothesis that can later become a useful theorem. This sc-agent is activated when an initiated action belonging to the class *reverse logical inference action* appears in sc-memory. After the sc-agent checks the initiation condition, the process of reverse logical inference is performed, which is similar to the process of direct logical inference, except that the search for rules is based not on the premises of formulas but on their conclusions [26]. The response of this agent will also be an inference tree showing which rules can be used to prove or refute the hypothesis put forward.

Abstract sc-agent of equivalent transformations of a logical formula

⇒ *decomposition of an abstract sc-agent**:

{

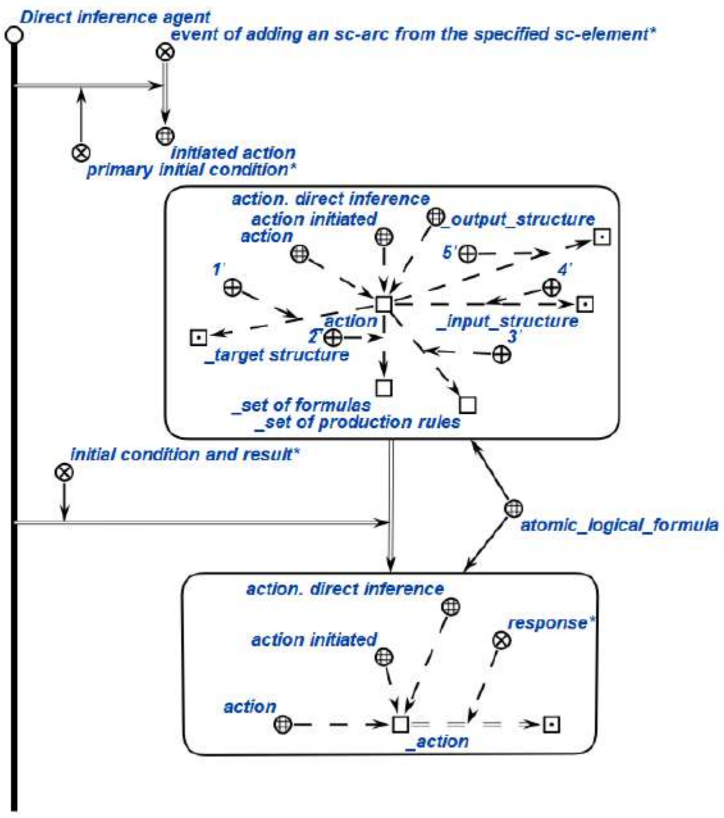


Figure 8. The specification of the agent of direct logical inference

- *Abstract sc-agent of transforming a formula into a conjunctive normal form*
- *Abstract sc-agent of transforming a formula into a disjunctive normal form*
- *Abstract sc-agent for applying de Morgan's laws*
- *Abstract sc-agent of equivalent transformations of a logical formula by definition*
- *Abstract sc-agent of applying the negation properties of logical formulas*
- *Abstract sc-agent of applying the law of idempotence of logical formulas*
- *Abstract sc-agent of applying the law of commutativity of logical formulas*
- *Abstract sc-agent of applying the law of associativity of logical formulas*
- *Abstract sc-agent of applying the law of absorption of logical formulas*
- *Abstract sc-agent of applying the law of contradiction of logical formulas*
- *Abstract sc-agent of applying the law of double negation of logical formulas*
- *Abstract sc-agent of applying the law of splitting logical formulas*

}

VII. Example of formal inference in the SCL language

With the help of *resolution rules*, it is possible to effectively prove the formulas of the propositional logic language. Any formula is equivalent to some formula in conjunctive normal form, and therefore it is sometimes convenient to apply the resolution rule. Using equivalent transformations, it is also possible to obtain formulas suitable for using the resolution rule. Figure 9 shows the formalization of the resolution rule.

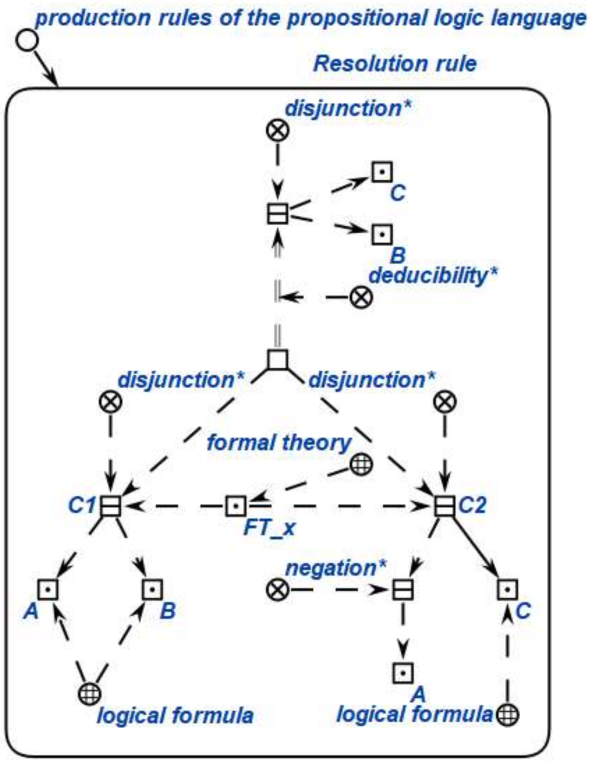


Figure 9. Formalization of the resolution rule

If any two disjuncts C_1 and C_2 have a pair of formulas A and $\neg A$, then a new disjunct can be formed from the remaining parts of the original disjuncts.

Let us give an example of the inference of a formula from a set of premises using the resolution principle [13].

If team A wins a football game, then city A' triumphs, and if team B wins, then city B' will triumph. Either only city A' or only city B' can win. However, if team A wins, then city B' does not triumph, and if team B wins, then city A' does not triumph. Consequently, city B' triumphs if and only if city A' does not triumph. The goal is to make sure that city B' triumphs if and only if city A' does not triumph.

Proving the inference of a formula is equivalent to proving the inconsistency of the inference of the negation for this formula. When using the resolution rule, this is especially convenient to use.

The formalization of logical formulas corresponding to the example is shown in Figure 10. Each non-atomic formula in the figure belongs to some formal theory, that is, is considered true.

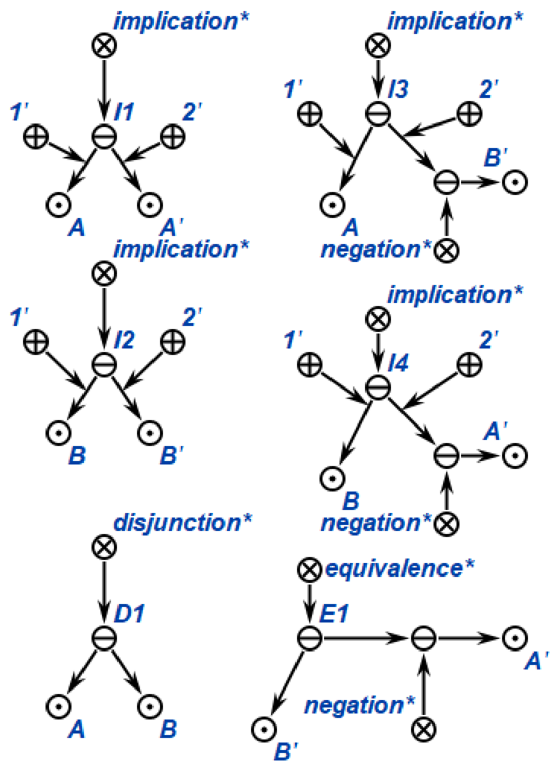


Figure 10. Formalization of rules for applying the resolution rule

Structure A is an atomic logical formula that contains the information "team A won", structure A' represents the formula denoting the triumph of city A'. Accordingly, the same is true for structures B and B'. First of all, it is necessary to bring the implication into conjunctive normal form according to the formula shown in Figure 11 and the equivalence by definition.

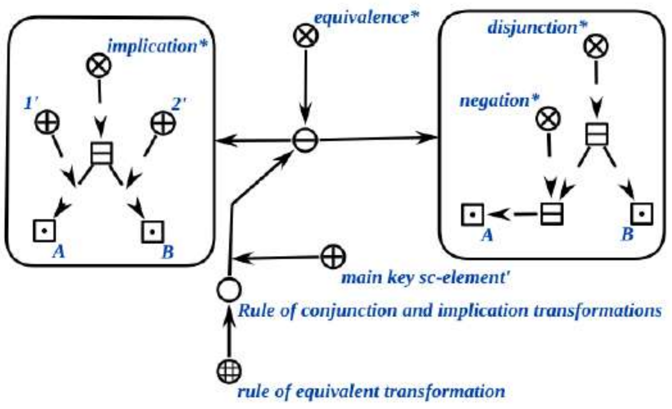

implication*
1'
2'
A
B
equivalence*
disjunction*
negation*
A
B
main key sc-element'
Rule of conjunction and implication transformations
rule of equivalent transformation

Figure 11. Formalization of rules for applying the resolution rule

Let us also apply negation to the formula that needs to be derived (equivalence). As a result, we obtain the following formulas (Fig. 12).

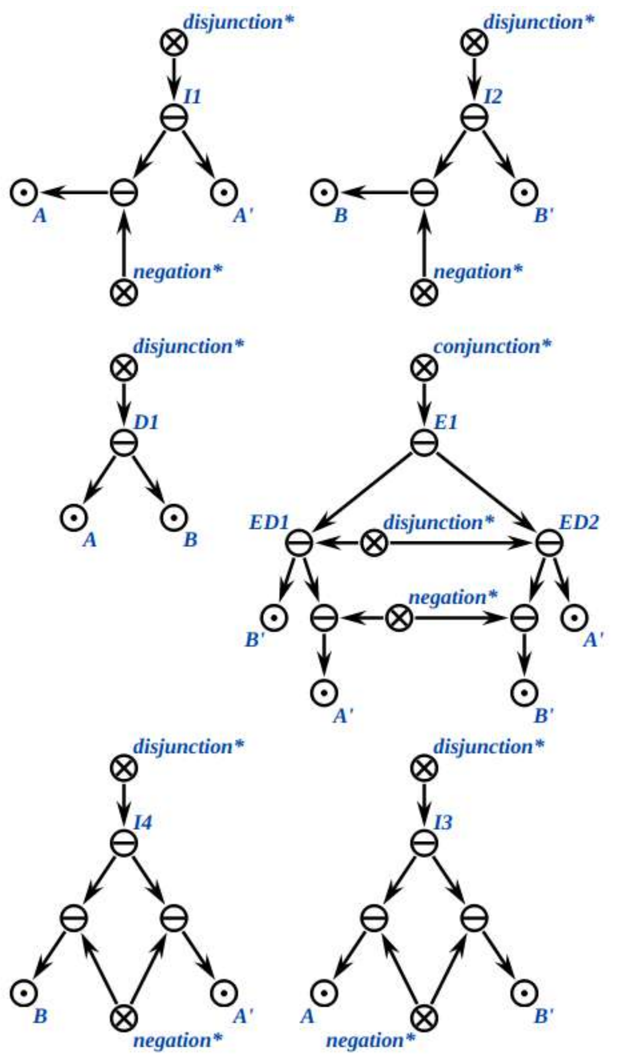


Figure 12. Formalization of rules for applying the resolution rule after conversion to conjunctive normal form

Further, applying the resolution rule for transformed formulas, we obtain an empty disjunction, which indicates the inconsistency of the set of formulas and proves the equivalence formula that city B' triumphs if and only if city A' does not triumph (Fig. 13 and 14).

VIII. Integration of Production and Functional Problem-Solving Models

Frequently, all the knowledge that a human operates with and that can be stored in the memory of an intelligent system can be divided into declarative and procedural. Declarative knowledge contains information about some objects, their features, properties, characteristics, the inclusion of objects among themselves in certain relationships, situations in which objects participate, the phenomena

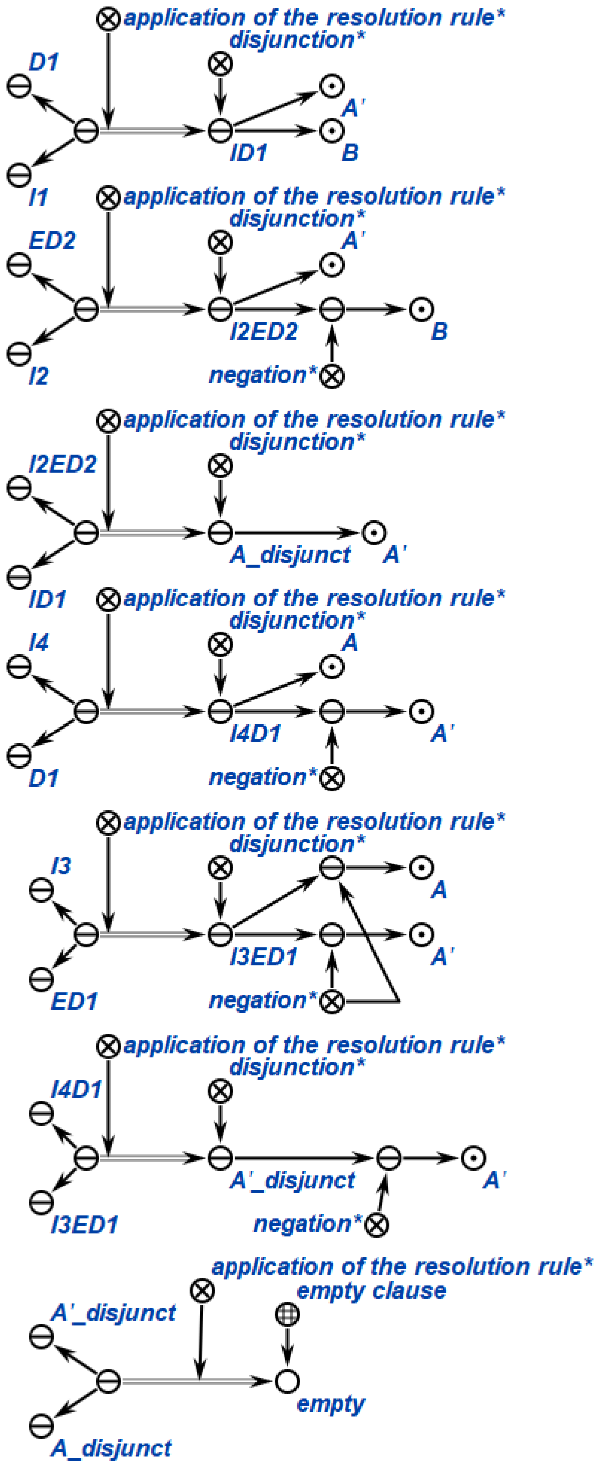

application of the resolution rule*
disjunction*
D1
I1
ID1
A'
B
application of the resolution rule*
disjunction*
ED2
I2
I2ED2
A'
B
negation*
application of the resolution rule*
disjunction*
I2ED2
ID1
A_disjunct
A'
application of the resolution rule*
disjunction*
I4
D1
I4D1
A
A'
negation*
application of the resolution rule*
disjunction*
I3
ED1
I3ED1
A
A'
negation*
application of the resolution rule*
disjunction*
I4D1
I3ED1
A'_disjunct
A'
negation*
application of the resolution rule*
empty clause
A'_disjunct
A_disjunct
empty

Figure 13. Application of the resolution rule

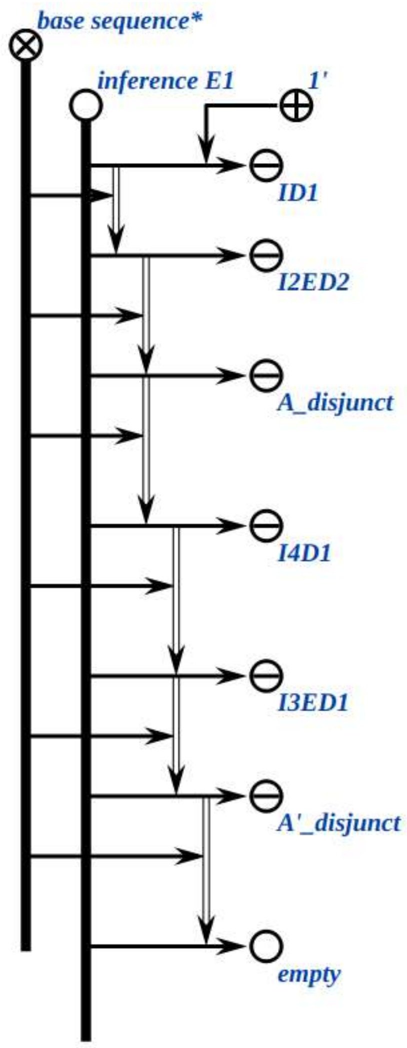


Figure 14. The result of applying the resolution principle

of reality, and its basic laws. Procedural knowledge allows the system to learn how to use certain declarative knowledge.

One of the ways to represent knowledge is a logical approach. Generalized knowledge about reality can be represented in the form of a formula of some calculus. However, even the simplest statements in natural language are not so easy to translate into the logic language, preserving the entire content of the text. Logical calculus is not suitable for displaying the totality of knowledge in intelligent systems.

Another way to describe knowledge is to use relational-type models. In such models, information units corresponding to objects, phenomena, facts, or processes are explicitly allocated.

The third way to describe knowledge is to use mixed-type models in which declarative and productive components are simultaneously present. Traditionally, this type of model includes frames and productions working on semantic networks. The production model is a development of the logical model. Production systems can be shown as transition graphs, which allows them to be represented in a natural way on the SC-code. Production systems often use an approach based on a “bulletin board”, which is implemented within the principles of the OSTIS Technology.

Production systems have the following advantages [27]:

- productions describe a variety of knowledge in simple structures with a high degree of standardization;
- production systems satisfy the modularity principle to a high degree. Any production with software implementation can be considered as an independent module, the addition of which to the production system and its withdrawal from it occurs without additional costs;
- production systems simplify the organization of parallel processes in which all productions included in the scope of ready-made ones can be performed independently of each other.

Functional problem-solving models are based on the concept of a function as a fairly general mechanism for representing and analyzing problem solving. In this case, the calculation model is implemented without states. A functional program cannot change the data it already contains but can only generate new ones. A neural network problem-solving model is a particular case of a functional problem-solving model. The advantages of functional methods are:

- high reliability due to clear structuring of data and functions;
- great capabilities for parallel computing.

The representation of functional models is also unified using the OSTIS Technology, and such models can be integrated with any other models when solving complex problems.

IX. Conclusion

In the article, the implementation of non-procedural problem-solving models of intelligent systems based on the OSTIS Technology is proposed, which makes it possible to realize compatibility between different problem-solving models and allow intelligent systems to solve complex problems. The hierarchy of complex subject domains necessary to achieve the set goals is designed.

The operational semantics of logical languages has been clarified in the form of a specification of the corresponding abstract sc-agents.

An example of the formalization of logical formulas, as well as the process of logical inference using semantic networks, is given.

The results obtained will allow structuring existing logics and using various approaches of non-procedural models in solving complex problems.

References

[1] Akshita Rastogi, Shivam, Rekha Jain, “Risk and challenges in intelligent systems,” *Proceedings of the Third International Conference on Information Management and Machine Intelligence, ICIMMI 2021*, 2022.

[2] Martin Molina, “What is an intelligent system?” 2022.

[3] Peter Flach, Kacper Sokol, “Simply logical – intelligent reasoning by example (fully interactive online edition),” 2022.

[4] J. M. Giménez-García, A. Zimmermann, and P. Maret, “Ndfluents: An ontology for annotated statements with inference preservation,” 2017.

[5] Abdur Rakib, Abba Lawan, “The Semantic Web rule language expressiveness extensions – a survey,” *Ontology-driven CropBase knowledge system*, 2019.

[6] Apatova N., Gaponov A., Smirnova O., “The possibilities of Artificial Intelligence in teaching higher mathematics,” 2021.

[7] Vadim Moshkin, Nadejda Yarushkina, “Modified knowledge inference method based on fuzzy ontology and base of cases,” *Creativity in Intelligent Technologies and Data Science*, 2019.

[8] Stefania Tomasiello, Witold Pedrycz, Vincenzo Loia, “Fuzzy inference systems,” *Contemporary Fuzzy Logic, A Perspective of Fuzzy Logic with Scilab*, 2022.

[9] Uehara, Kiyohiko and Hirota, Kaoru, “Fuzzy inference: Its past and prospects,” *Journal of Advanced Computational Intelligence and Intelligent Informatics*, vol. 21, pp. 13–19, 01 2017.

[10] Golenkov Vladimir and Guliakina Natalia and Shunkevich Daniil, *Open technology of ontological design, production and operation of semantically compatible hybrid intelligent computer systems*, V. Golenkov, Ed. Minsk: Bestprint [Bestprint], 2021.

[11] (2022, September) IMS.ostis Metasystem. [Online]. Available: https://ims.ostis.net

[12] Golenkov V., Korolev V., “Basic transformations of SQL language texts for the implementation of deductive inference mechanisms,” *Minsk: ITC of NAS of Belarus*, 1996.

[13] Averin A.I. and Vagin V.N., “Using parallelism in deductive inference,” *Journal of Computer and Systems Sciences International*, vol. 43, pp. 603–614, 07 2004.

[14] Satya Sundar Sethy. (2021) Mediate inference (syllogism).

[15] Norton John, “A demonstration of the incompleteness of calculi of inductive inference,” *The British Journal for the Philosophy of Science*, vol. 70, pp. 1119–1144, 12 2019.

[16] Yini Zhang and Yilin Wang. (2022) Missing-edge aware knowledge graph inductive inference through dual graph learning and traversing.

[17] Abdul Rahman, Safawi and Ibrahim, Zaharudin and Paiman, Jailani and Bakar, Amzari and Mohd Amin, Zahari, “The decision processes of abductive inference,” *Advanced Science Letters*, vol. 21, pp. 1754–1757, 06 2015.

[18] Gungov, Alexander, “The ampliative leap in diagnostics: The advantages of abductive inference in clinical reasoning,” *History of Medicine*, vol. 5, pp. 233–242, 01 2018.

[19] Geramian A. and Mehregan M.R. and Garousi Mokhtarzadeh and N. and Hemmati M. (2017) Fuzzy inference system application for failure analyzing in automobile industry.

[20] Son L.H. and Van Viet P. and Van Hai P. (2017) Picture inference system: a new fuzzy inference system on picture fuzzy set.

[21] Lupea, Mihaiela, “DARR–a theorem prover for constrained and rational default logics,” vol. 1, 01 2002.

[22] Weydert, Emil, “Defaults, logic and probability – a theoretical perspective,” *KI – Künstliche Intelligenz, v.4/01, 44–49 (2001)*, 11 2022.

[23] Chen, Gang and Wei, Peng and Liu, Mei, “Temporal logic inference for fault detection of switched systems with Gaussian process dynamics,” *IEEE Transactions on Automation Science and Engineering*, vol. PP, pp. 1–16, 05 2021.

[24] Rybakov, V., “Multi-agent temporal nontransitive linear logics and the admissibility problem,” *Algebra and Logic*, vol. 59, 05 2020.

[25] Golenkov V., Gulyakina N., “Grapho-dynamic association models and facilities of parallel information handling in artificial intelligence systems,” *BSUIR Proseedings*, 2003.

[26] Gavrilova T.A., Horoshevski V.F., *Knowledge bases of intelligent systems*, 2000.

[27] Kuznetsov V. E., *Computer representation of informal procedures*, 1989.

Непроцедурные модели решения задач в интеллектуальных компьютерных системах нового поколения

Орлов М.К., Василевская А.П.

В работе рассматривается подход к проектированию решателей задач интеллектуальных систем на основе непроцедурных моделей. Разрабатываемый подход позволяет интегрировать любые модели решения задач, в том числе принципы логического вывода, для решения задач на основе общей формальной модели.

Received 30.10.2022