Methods and Means of Constructing Plans for Solving Problems in Intelligent Systems on the Example of an Intelligent System on Geometry

Natallia Malinovskaya and Anna Makarenko
Belarusian State University of
Informatics and Radioelectronics
Minsk, Belarus
Email: natasha.malinovskaya.9843@gmail.com, anna.makarenko1517@gmail.com

***Abstract*—In this paper we propose an approach to the development of methods and tools for constructing plans for solving problems in intelligent systems on the example of an intelligent system for geometry. The described approach is aimed at improving the accuracy of answers due to the possibility of decomposition of problems into simpler ones, and also aims to overcome the shortcomings of modern intelligent systems. An intelligent system realizing the proposed approach is described.**

***Keywords*—problem solving, knowledge, knowledge base, intelligent systems.**

I. Introduction

Nowadays, the use of intelligent systems in various fields is becoming relevant. However, the quality of ISs is largely determined by its answers and the ability to solve complex problems, which is why it is necessary that its answers are as accurate and reliable as possible. In order to increase the accuracy of answers it is necessary to be able to decompose problems into simpler ones, in turn, the automation of this process will allow the IS to solve not only simple problems, but also complex, non-trivial ones. The relevance of systems that have the ability not only to perform the above functions, but also have the ability to partially satisfy the human need for quick answers, allowing you to automate some of the routine actions.

Problem solver is one of the key components of ISs, allowing them to solve a wide range of problems. Unlike other modern software systems, the peculiarity of problem solvers in ISs is the necessity to solve problems in conditions when the necessary information for their solution is not explicitly localized in the knowledge base of the IS and must be found in the process of problem solving on the basis of certain criteria.

The composition of the problem solver in each particular system depends on its target, classes of solved problems, subject area and other factors. In general, a problem solver provides the ability to solve problems related both to the core functionality of the system and to ensure its efficient operation and development automation. A problem solver that performs all these functions is called a unified problem solver for a given IS.

Expanding the areas of application of ISs requires their ability to solve complex problems, which involve the joint use of different models of knowledge representation and problem solving models. In addition, solving complex problems involves the use of shared information resources, such as a knowledge base, by different components of the solver that specialize in solving different subproblems. Since a complex problem solver integrates different problem solving models, it is called a hybrid problem solver.

Examples of complex problems are:

- problems related to understanding natural language texts (both printed and handwritten), understanding speech messages and images. In each of these cases it is required to perform syntactic analysis of the processed file or signal, remove insignificant fragments, classify significant fragments, relate them to concepts known to the system, etc.;
- automation of adaptive learning for schoolchildren and students, which implies that the system is capable of autonomously solving various problems from a certain subject area, as well as managing the learning process, generating tasks for students and controlling their independent fulfillment by the student;
- the problems of planning the behavior of intelligent robots, which involve both understanding a variety of external information and making various decisions using both credible methods and methods that rely on probabilistic estimates and plausible assumptions;
- problems related to complex and flexible automation of various enterprises.
- etc.

The use of different problem solving models within an IS implies decomposition of a complex problem into

subproblems that can be solved using one of the known IS problem solving models. Due to the combination of different problem solving models, the set of problems solved by the hybrid solver will be much wider than the combination of sets of problems solved separately by all problem solvers included in its composition [1].

The existing variety of approaches to problem solving in computer systems can be divided into two classes:

- problem solving using stored programs. In this case, it is assumed that the system has a program for solving a problem of a given class in advance, and the solution is reduced to searching for such a program and interpreting it on the given input data. The systems oriented on such approach to problem solving include systems using:
  - programs written in both imperative and declarative programming languages, including logical and functional programming [2];
  - genetic algorithm implementations [3], [4];
  - neural network models of knowledge processing [5], [6], [7].

  It should be noted that even in the case of using a stored program, the solution of the problem is not always trivial, because, first, it is required to find such a stored program on the basis of some specification, and second, to provide its interpretation;
- solving problems in conditions where the solution program is not known.

In this case, it is assumed that the system does not necessarily contain a ready-made solution program for the class of problems to which some formulated problem to be solved belongs. In this connection, it is necessary to apply additional methods of searching for ways to solve the problem, which are not designed for any narrow class of problems (e.g., splitting the problem into subproblems, methods of searching for solutions in depth and width, method of random solution search and trial-and-error method, etc.), as well as various models of logical inference (classical deductive, [8], inductive [9], [10], abductive [8]; models based on fuzzy logics [11], [12], [13], the logic of default [14], temporal logic [15], and many others).

For example, currently one of the most popular approaches to text generation for natural language problems is the use of *large language models* [16], which are models consisting of neural networks with many parameters trained on a large amount of unlabeled text, but these models have a number of disadvantages, partial solution of which can be prevented by integration of such systems with knowledge bases [17], but this approach is not able to solve all the disadvantages of large language models in solving complex problems.

Thus, although there have been significant advances in the development of problem solvers for ISs, there are still outstanding challenges related to provisioning:

- compatibility of different private problem solvers, i.e. the possibility of their coordinated use in solving the same complex problem;
- possibilities to modify the hybrid solver without significant additional costs in the process of operation of the IS. This includes expanding the number of used problem solving models without restrictions on their type. This requirement is due to the fact that when solving a complex problem, it may be unknown what specific problem-solving models and types of knowledge will be needed.

The modern development of *artificial intelligence* is moving towards the creation of intelligent computer systems of a new generation [18]. These systems are capable not only of solving problems from various fields of knowledge, but also of explaining their solutions. However, the disadvantages described above do not allow such systems to be built solely on the basis of existing solutions. Instead, new-generation ISs are built on a unified knowledge base that integrates problems, subject domains, and methods of their solution. Thus, we conclude that the use of modern solutions such as neural networks, models using specialized software interfaces between different system components, large language models can and should become a powerful tool for solving problems of ISs [19], but they cannot completely replace these systems [20].

That is, despite the fact that currently there is a large number of problem-solving models, many of which are implemented and successfully used in practice in various systems, the problem of low consistency of the principles underlying the implementation of such models and the lack of a single unified framework for the implementation and integration of different models remains relevant, which leads to the fact that:

- it is difficult to simultaneously use different models of problem solving within one system when solving the same complex problem;
- it is practically impossible to use technical solutions realized in one system in other systems;
- in fact, there are no complex methods and tools for building problem solvers, which would provide the possibility of designing, realizing and debugging solvers of different kinds.

Therefore, the ability of an IS to independently solve non-trivial problems will expand the possible functionality of the IS, detail and improve the accuracy of its answers.

The purpose of this study is to refine the ontologies of actions and problems, to develop a collective of agents that allows to divide the problem into subproblems, as well as to develop a methodology for the application of the obtained subsystem in specific application systems on the example of an IS for geometry.

II. Proposed approach

In the development of the module of building plans for solving the problems of the IS, it is proposed to use the OSTIS Technology, focused on the development of a class of systems, which are called knowledge-managed computer systems, as well as its basic principles [21], since any *ostis-system* consists of *knowledge base*, *problem solver* and *user interface*, which corresponds to the classical definition of an IS [22].

This technology uses a unified semantic network with a set-theoretic interpretation as a formal basis for knowledge representation. This representation model is called SC-code (Semantic computer code). Elements of such semantic network are called sc-nodes and sc-connectors (sc-arc, sc-edges). Agents in the developed system described by means of SC-code will be called a semantic agent or simply sc-agent.

As it was mentioned earlier, in order to be able to solve some non-trivial problems it is necessary to be able to divide them into trivial ones, i.e. to build a plan for problem solving. When constructing plans for solving the problem of an IDS, it is suggested to divide the main problem into separate subproblems taking into account various separate independent components necessary for solving the main problem.

Let us introduce the concepts *problem*, *subproblem*, *action*, *subaction*.

A. Fragment of the obtained ontology

planning of IDS problems
:= [partitioning a problem into smaller problems - allocation of separate subproblems for solving the main problem]

The concept of **problem** is directly related to the concept of **action**, so let's consider the classification of the concept of **action**.

1) action classification: Let us consider the specification of the concept of *action* in SCn code.

action
:= [*impact*, in which the *subject'* carries out the *action* purposefully, i.e. according to some *target**]
:= [targeted action performed by one or more actors (cybernetic systems) with the possible application of some tools]
⊂ *impact*
  := [a process in which at least one influencing entity (subject of influence') and at least one entity to be influenced (object of influence') can be clearly distinguished]
  ⊂ *process*
:= [a purposeful ("conscious") process performed (controlled, realized) by a certain subject]
:= [work]
:= [problem solving process]
:= [target-oriented process]
:= [holistic piece of some activity]
:= [A purposeful process controlled by some entity]
:= [the process of performing some action by some subject (executor) on some objects]

target*
:= [target situation*]
⊂ *specification*
:= [description of what is to be obtained (what situation is to be achieved) as a result of performing a given (specified) action*]

Each *action* performed by one or another *subject* is interpreted as a process of solving a certain problem, i.e., a process of achieving a given *target** under given conditions, and, therefore, is performed purposefully. However, an explicit indication of the *action* and its relation to a particular problem may not always be present in memory. Some *problems* may be solved by certain subjects permanently, e.g., optimizing a knowledge base, searching for incorrectness, etc., and for such problems it is not always necessary to explicitly introduce the *structure* that is the formulation of the *problem*.

In its turn, the concept of a *subproblem* or *action* is a separate independent *problem* (action), which is performed within the framework of some other extensive *problem* (action).

B. Architecture of a problem solver for partitioning problems into subproblems

To realize an IS, which includes methods and means of constructing plans for solving problems of the IS, it was necessary to build a decomposition of the problem solver of the system, where the key agent of the IS is an abstract non-atomic sc-agent of constructing a plan for solving problems of the IS, taking into account the above classification.

The following is the decomposition of the IS problem solver in the form of SCn-code:

Problem solver for partitioning a problem into subproblems of an intelligent system
⇒ *decomposition of abstract sc-agent**:
  {• *Abstract sc-agent for classifying a message by topic*
    ⇒ *realization**:
      C++ language
  • *Abstract sc-agent for classifying a message by type*
    ⇒ *realization**:
      C++ language
  • *Abstract sc-agent for generating a problem condition from a template*
    ⇒ *realization**:

C++ language

- *Abstract sc-agent for finding the meaning of a problem*
  ⇒ *realization**:
  C++
- *Abstract sc-agent targeting*
  ⇒ *realization**:
  C++ language
- *Abstract non-atomic sc-agent for constructing a problem-solving plan for an intelligent dialog system*
  ⇒ *realization**:
  C++ language
- *Abstract sc-agent for knowledge base replenishment*
  ⇒ *realization**:
  C++ language
- *Abstract sc-agent for generating a response to a message*
  ⇒ *realization**:
  C++ language

}

In this article we consider the approach to solving complex problems, so we will consider in detail the abstract non-atomic sc-agent of building a plan for solving problems of the IS.

As a part of the abstract non-atomic sc-agent of constructing a plan for solving problems of an intelligent dialog system, the following group of agents that break the problem into separate subproblems was identified:

Abstract non-atomic sc-agent for constructing a plan for solving problems of an intelligent system
⇒ *decomposition of abstract sc-agent**:
{

- *Abstract sc-agent for solving a composite problem*
  ⇒ *realization**:
  C++ language
- *Abstract non-atomic sc-agent for solving a simple problem*
  ⇒ *realization**:
  C++ language
- *Abstract sc-agent for interpreting a non-atomic action*
  ⇒ *realization**:
  C++ language

}

C. Principles of application of the developed solver in application systems

The approach proposes the introduction of separate modules with their ontologies and problem solvers into the developed system, which will increase the range of capabilities of the developed system.

Let us consider *abstract non-atomic sc-agent of constructing a plan for solving problems of an IS* on the example of a system with implemented module of an IS on geometry.

Examples will be considered on the basis of dialog-based ISs.

The abstract sc-agent for solving a composite problem takes as input one parameter, which specifies the condition of the composite problem written in the form of sc-code and a node denoting the target of solving the problem. The task of this agent is to divide the original problem into simple subproblems, and also to call for each simple subproblem one of the agents for solving simple problems in a certain sequence, which is set by the abstract sc-agent of the interpretation of non-atomic action, in some cases there is a need to execute the agent of the application of the logical formula [23]. The result of solving the subproblem is recorded in the decision tree, which is ultimately the solution of the user's problem.

That is, the abstract non-atomic sc-agent of constructing a plan for solving the problems of the IS involves breaking the main problem-request into separate subproblems to form a better response to the user's message, i.e., solving smaller problems allows increasing the quality of knowledge immersed in the database. As an input construct, we obtain a situation that is formed due to an abstract sc-agent of target-setting developed within the framework of the IS under development, the input of which is a user's user's message. As an example, let us take the user's request "What is the area of the figure?".

Depending on the knowledge in the KB, the formed target may not be deployed at all, or it may be more deployed, which allows to break the problem into subproblems more qualitatively, i.e., to solve the problem in the most accurate way in the future.

The figure 1 shows an example of a more extended generated situation, namely the target of "find out the area of a figure".

When receiving a question from a user about the area of a figure without additional instructions, it is impossible to answer it directly. As a result of the abstract target-setting agent's work in the IS, a situation arises that requires solving a non-trivial problem. To break this problem into simpler subproblems, an abstract non-atomic sc-agent of constructing a problem-solving plan is used.

We apply a depth-first search method over a graph in which we define the corresponding subproblems that are trivial. Unknown values for these subproblems can be discovered by searching the KB or by using an agent included in the IS module on geometry. If it is still not possible to solve the problem, we ask for clarifications from the user to obtain additional information that will contribute to expanding the description about the situation.

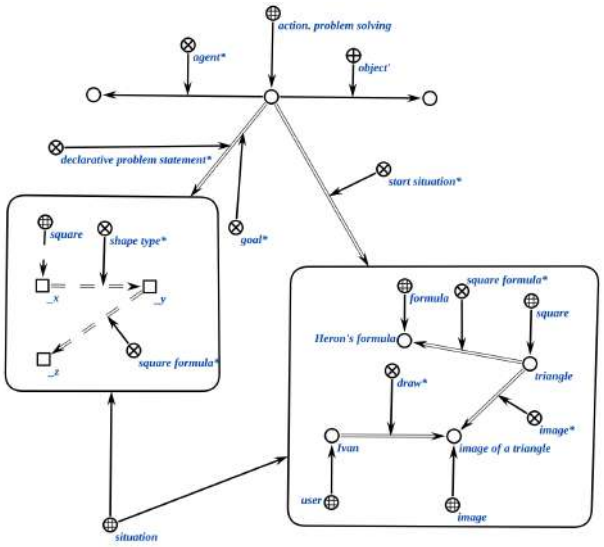

Figure 1. Example of a detailed situation generated as a result of a dialog with a user

Therefore, the agent of building a plan for solving problems of the IS in turn consists of an agent for solving a composite problem and an agent for solving a simple problem. That is, in the context of the developed dialog system, a simple problem will be an indivisible problem that does not require additional information, such a problem, the result of which can be solved as a result of a search in the KB, or as a result of the work of an abstract sc-agent from additional implemented modules.

In general, a *problem solving plan* consists of a sequence of simple and compound problems. The composite problems are solved by *Abstract sc-agent for interpreting non-atomic action*. It receives 1) a processing program in the form of a template and 2) sets of arguments that it retrieves from the semantic neighborhood of the composite problem. The template data is matched with the arguments and the corresponding agents are invoked, to solve the atomic actions belonging to the composite problem. The invoked agents perform the actions of the composite problem.

In essence, this agent is an interpreter that generates a processing program, i.e., a program that includes a sequence of agent executions. A processing program — is a description of a non-atomic action to be performed to solve a composite problem. Sequential execution of agents is produced by sequential processing of transitions between agents.

The transition to the next action depends on the result of the previous one. After the action is completed, i.e. after it is added to the problem_executed class, its success is checked to determine the required transition. There are 3 variants of transition:

1 Transition on successful completion of the action. It is defined by the then* relation. Transition by this relation is performed at successful completion of the action from which the transition is performed (its addition to the class of successfully completed action).

2 Transition at unsuccessful completion of the action. It is set by the then* relation. The transition by this relation is performed at the successful completion of the action from which the transition is performed (its addition to the class of unsuccessfully completed action).

3 Unconditional transition.

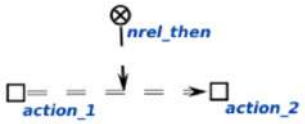

Figure 2. An example of recording a transition between two actions, depending on the result of the previous action

The figure 2 shows an example of recording a transition between two actions, depending on the result of the previous action.

In addition to transitions depending on the result of the previous action, conditional transitions are introduced by means of the state* relation. The first element of pairs of this relation are pairs (arcs) of transitions according to the success of action completion, the second element is a logical formula. In this case, an additional condition is imposed on the transition pair (besides the success / failure of action completion) - the truth of the logical formula. In this case, the truth of the formula is calculated for the same substitutions that were used to generate the process by the program.

The figure 3 shows an example of writing a conditional transition between two actions.

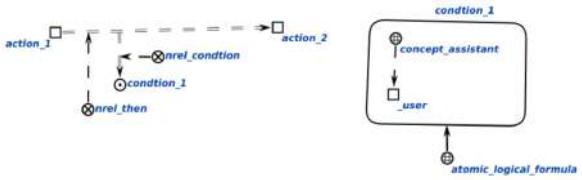

Figure 3. Example of writing a conditional transition between two actions

However, in order to constantly replenish the *knowledge base* during the dialog process, it is necessary to extract *knowledge* from the user's messages and immerse them into the *knowledge base*. For this purpose, the problem solver of the described system implements an action whose problem is to transform the natural-language text of messages into *knowledge base* constructs.

Such a solution allows the system to better utilize knowledge for a more accurate and coherent answer,

taking into account the previously mentioned entities and understanding how they relate to the current topic of conversation. This makes the dialog with the system more productive and natural, similar to talking to a person who remembers all the details to solve a problem.

III. Example of application of the results obtained

An example of realization of the proposed approach is an intelligent geometry learning system consisting of three components: a KB, a problem solver and a UI.

Below is a decomposition of the *Geometry Intelligent System Problem Solver*. It consists of a search module — agents that search for constructs in the KB and a computational problem solver, i.e., agents that implement algorithms for solving geometry problems.

Solver of intelligent system problems in geometry
⇒ *decomposition of an abstract sc-agent**:
{• *Abstract non-atomic sc-agent search agent*
• *Computational Problem Solver*
⇒ *decomposition of an abstract sc-agent**:
{• *Abstract sc-agent for interpreting arithmetic expressions*
• *Abstract non-atomic sc-agent for interpreting logical rules*
• *Abstract sc-agent of constructing a strategy for finding a solution to a problem in width*
}
}

Next, a decomposition of *Abstract non-atomic sc-agent search* is presented. It consists of a necessary set of sc-agents that can be used in solving specific problems. For example, if a theorem proving problem needs to be solved, it is appropriate to use *Abstract sc-agent for searching axioms of a given ontology* or *Abstract sc-agent for searching theorems of a given ontology*.

Abstract non-atomic sc-agent search agent
⇒ *decomposition of an abstract sc-agent**:
{• *Abstract sc-agent for finding an annotation for a given section*
• *Abstract sc-agent for searching axioms of a given ontology*
• *Abstract sc-agent for searching theorems of a given ontology*
• *Abstract sc-agent for finding direct links between two objects*
• *Abstract sc-agent for searching concepts through which a given concept is defined*
• *Abstract sc-agent for searching the scope of a relation definition*
• *Abstract sc-agent to find a definition or explanation for a given object*
• *Abstract sc-agent for finding examples for a given concept*
• *Abstract sc-agent for finding a formal statement record for a given statement sign*
• *Abstract sc-agent for finding illustrations for a given object*
• *Abstract sc-agent of finding key sc-elements for a given subject area*
• *An abstract sc-agent searches for concepts that are defined on the basis of a given*
• *Abstract sc-agent search for all constructs isomorphic to a given pattern*
• *Abstract sc-agent for finding the sc-text of a proof for a given assertion*
• *Abstract sc-agent for searching relations defined on a concept*
• *Abstract sc-agent for searching sc-text of condition and problem solution*
• *Abstract sc-agent for searching statements about an object*
}

The following is a decomposition of the *Abstract non-atomic sc-agent problem solver*.

Abstract non-atomic sc-agent problem solving agent
⇒ *decomposition of an abstract sc-agent**:
{• *Abstract sc-agent for searching the value of an unknown quantity*
• *Abstract sc-agent for verifying the truth of an assertion*
• *Abstract sc-agent application of problem-solving strategies*
• *Abstract sc-agent of performing logical inference*
• *Abstract non-atomic sc-agent for calculating mathematical expressions*
⇒ *decomposition of an abstract sc-agent**:
{• *Abstract sc-agent for coordinating the calculation of mathematical expressions*
• *Abstract sc-agent for degree expansion, root extraction and finding the natural logarithm*
• *Abstract sc-agent for addition and subtraction of quantities and numbers*
• *Abstract sc-agent of product and division of quantities and numbers*
• *Abstract sc-agent for comparing quantities and numbers*
• *Abstract sc-agent for calculating trigonometric expressions*
}
}

Abstract sc-agent for coordinating the calculation of mathematical expressions takes a formula as a parameter. An example is shown below in Fig. 4.

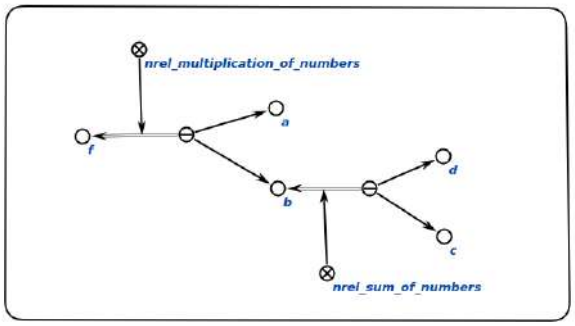

Figure 4. Example formula as input parameter of sc-agent

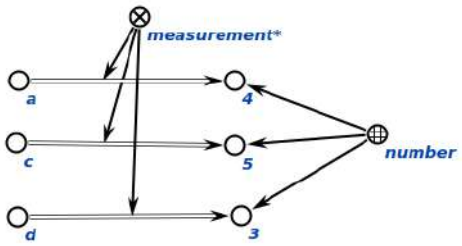

Figure 5. Example formula as input parameter for sc-agent

A formula is represented as an sc-structure, which contains sc-bindings of mathematical operation relations and sc-nodes, which are signs of numbers or variables whose value is known or to be calculated. In this example, the formula consists of:

- variables:
  - a,
  - b,
  - c,
  - d,
  - f,
- relations:
  - nrel_sum_of_numbers,
  - nrel_multiplication_of_numbers,
- arcs and edges.

The abstract sc-agent of coordination of calculation of mathematical expressions of formula processing after initiation searches for all sc-edges of relations of arithmetic operations and forms a structure for calling the operation calculation agent with the parameter of the sc-edges connecting the node of the relation and the binary arc of the basic kind. In turn, each of the sc-agents for operation computation checks whether it can compute an operation of the given type. If it can, it computes the operation and creates an sc-node with the answer, otherwise it does not continue its work.

Thus, the abstract sc-agent coordinating the computation of mathematical expressions processing formula does not know in advance which agent to call specifically. All agents react to the initiated action by checking the input parameters as the initial condition of the problem.

The values of the variables in the formula can be specified in advance, or they can be found in the course of solving the problem. Below are the steps of calculating the values of the variables in the formula, if the values of a, d, c are known in advance (otherwise the formula would not have a specific value).

Since the values of a and b must be known to compute f, the sc-agent checks if their values are known. Since the value is known only for a, the agent will generate a

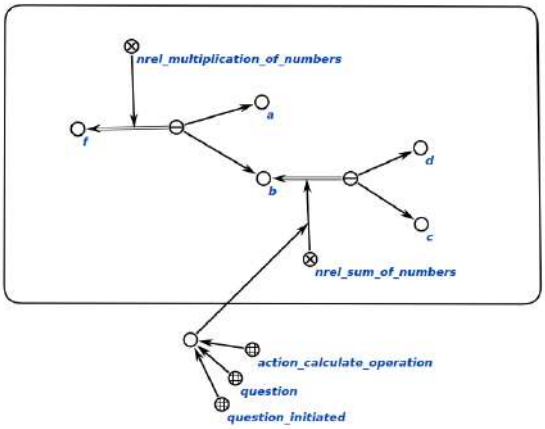

Figure 6. Example of operation calculation agent initiation

structure to initiate the agent to compute an arithmetic operation, after which the value of b will be known.

An example of such a structure for initiating the sc-agent is given below in Fig.6.

Once completed, the agent will create the following construct in the KB. 7

Thus, if all initiated agents are successfully executed, the value of the value of f.

This agent can be used to calculate the values of area, perimeter, etc. using predetermined formulas.

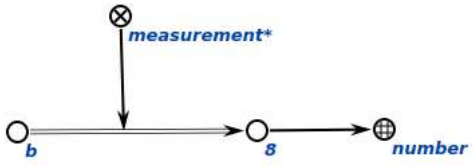

Figure 7. Example of the result of the operation calculation agent execution

IV. Conclusion

The paper proposes an approach to the development of methods and means of constructing plans for problem solving in ISs, which allows us to improve the accuracy of answers, as well as to overcome the shortcomings of modern ISs.

The proposed model allows us to consider the developed problem solver at different levels of detail, which provides the possibility of step-by-step design of solvers, as well as their modifiability.

Classification and specification of actions, problems are specified.

The architecture is considered and the IS itself, realizing the proposed approach, is described.

The obtained results will allow to increase the efficiency of designing ISs and means of automating the development of such systems, as well as to provide an opportunity not only for the developer, but also for the IS to automatically supplement the system with new knowledge and skills.

Acknowledgment

The authors would like to thnk the research groups of the Department of Intelligent Information Technologies of the Belarusian State University of Informatics and Radioelectronics.

References

[1] I. Z. Batyrshin, *Fuzzy hybrid systems. Theory and practice*, H. G. Y. M., Ed. Physmatlit, 2007.

[2] T. Pratt, *Programming languages: development and realization : transl. from English / T. Pratt, M. Zelkovits*, A. Matrosov, Ed. SPb. Peter Print, 2002.

[3] L. A. Gladkov, *Genetic algorithms : textbook*, V. M. K. n. r. L. A. Gladkov, V. V. Kureichik and supplementary ed. M., Eds. Physmatlit, 2006.

[4] V. V. Emelyanov, *Theory and practice of evolutionary modeling*, V. M. K. V. V. Emelyanov, V. V. Kureichik, Ed. Physmatlit, 2003.

[5] M. B. Berkinblit, *Neural networks : an experimental textbook*, M. B. Berkinblit, Ed. MIROS : All-Russian extramural multidisciplinary school of the Russian Academy of Sciences., 1993.

[6] V. A. Golovko, *Neural networks: training, organization, and applications : a textbook*, V. A. Golovko, Ed. Journal. "Radiotekhnika", 2001.

[7] A. N. Gorban, *Neural networks on a personal computer*, D. A. R. A. N. Gorban, Ed. Novosibirsk : Nauka, 1996.

[8] V. N. Vagin, *Reliable and plausible inference in intelligent systems*, 2nd ed., D. A. P. B. N. Vagina, Ed. Fizmatlit, 2008.

[9] B. A. Kulik, *The logic of natural reasoning*, B. A. Kulik, Ed. St. Petersburg. : Nev. dialect, 2001.

[10] D. Poya, *Programming languages: design and implementation : transl. from English*, M. Z. e. b. A. M. T. Pratt, Ed. Peter Print, 2002.

[11] I. Z. Batyrshin, *Basic operations of fuzzy logic and their generalizations*, I. Z. Batyrshin., Ed. Kazan : Fatherland, 2001.

[12] N. P. Demenkov, *Fuzzy control in technical systems : textbook*, N. P. Demenkov, Ed. Moscow State Technical University Publishing House, 2005.

[13] D. A. Pospelov, *Modeling reasoning: experience in analyzing thought acts*. Radio and communications, 1989.

[14] R. A. Reiter, *A logic for default reasoning*, R. Reiter, Ed. Artificial Intelligence., 1980, vol. 13, no. 13.

[15] A. P. Eremeev, *Construction of ternary logic-based decision functions in decision-making systems under uncertainty*, A. P. Eremeev, Ed. Famous Russian Academy of Sciences. Theory and systems of management., 1997.

[16] A. Tamkin, M. Brundage, J. Clark, and D. Ganguli, "Understanding the capabilities, limitations, and societal impact of large language models," *CoRR*, vol. abs/2102.02503, 2021. [Online]. Available: https://arxiv.org/abs/2102.02503

[17] K. Bantsevich, M. Kovalev, and N. Malinovskaya, "Integration of large language models with knowledge bases of intelligent systems," *Otkrytye semanticheskie tekhnologii proektirovaniya intellektual'nykh system [Open semantic technologies for intelligent systems]*, pp. pp 213–219, 2023.

[18] V. V. Golenkov and N. A. Gulyakina, "Next-generation intelligent computer systems and technology of complex support of their life cycle," *Otkrytye semanticheskie tekhnologii proektirovaniya intellektual'nykh system [Open semantic technologies for intelligent systems]*, pp. 27–40, 2022.

[19] M. V. Kovalev, "Convergence and integration of artificial neural networks with knowledge bases in next-generation intelligent computer systems," *Otkrytye semanticheskie tekhnologii proektirovaniya intellektual'nykh system [Open semantic technologies for intelligent systems]*, pp. 173–186, 2022.

[20] (2023, March) Opinion | Noam Chomsky: The False Promise of ChatGPT. [Online]. Available: https://www.nytimes.com/2023/03/08/opinion/noam-chomsky-chatgpt-ai.html

[21] V. Golenkov, N. Gulyakina, and D. Shunkevich, *Open technology for ontological design, production and operation of semantically compatible hybrid intelligent computer systems*, G. V.V., Ed. Minsk: Bestprint, 2023.

[22] A. N. Averkin, *A Comprehensive Dictionary of Artificial Intelligence*, H. G. Yarushina, Ed. Radio and communications, 1992.

[23] "Github:scl-machine [electronic resource]." [Online]. Available: https://github.com/ostis-ai/scl-machine

МЕТОДЫ И СРЕДСТВА ПОСТРОЕНИЯ ПЛАНОВ РЕШЕНИЯ ЗАДАЧ В ИНТЕЛЛЕКТУАЛЬНЫХ СИСТЕМАХ НА ПРИМЕРЕ ИНТЕЛЛЕКТУАЛЬНОЙ СИСТЕМЫ ПО ГЕОМЕТРИИ

Малиновская Н. В., Макаренко А. И.

В данной работе предлагается подход к разработке методов и средств построения планов решения задач в интеллектуальных системах на примере интеллектуальной системы по геометрии. Описанный подход направлен на повышение точности ответов засчет возможности декомпозиции задач на более простые, а также направлен на преодоление недостатков современных интеллектуальных систем. Описана интеллектуальная система, реализующая предлагаемый подход.

Received 01.04.2024