On the Problem of Logical Recognition in the Dynamic Internet Environment[1]

A. A. Morozov and Yu. V. Obukhov

Institute of Radio Engineering and Electronics, Russian Academy of Sciences,
ul. Mokhovaya 11, GSP-3, Moscow, 103907 Russia
e-mail: {morozov, obukhov}@mail.cplire.ru

Abstract—The problem of formalized logical recognition in conditions of permanent modification of information content on the Internet is considered. An approach based on the idea of modifiable reasoning is proposed. As a mathematical technique for this approach, we propose to use the method of logical actors. This method is compared with the nonmonotonic reasoning.

INTRODUCTION

The complexity of the retrieval and recognition of information entities on the Internet is caused by the following reasons.

1. The information on the Internet has a complex structure permanently being renovated.

2. The space of informational search is extremely large.

Since the features of sought for entities have a complex structure, it is expedient to use logical models and languages when solving the problems of information retrieval and recognition in the Internet environment. However, the use of logical reasoning is substantially complicated by the necessity of ensuring the soundness and completeness of logical inferences under conditions of permanently updating information on the Internet. The data may undergo changes in the course of the execution of a logic program, which may result in recognition errors.

In this paper, we discuss the problems of developing a mathematical tool for the modifiable reasoning that would ensure the validity of logic programs (intelligent agents) that function in the dynamic Internet environment, where the informational content is permanently changing and augmented. We compare the nonmonotonic reasoning and the method of logical actors as possible bases for the tool mentioned above.

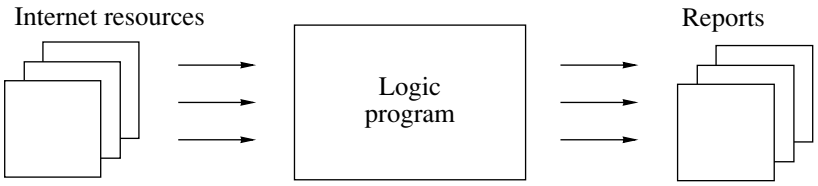
1. THE MODEL OF AN INTELLIGENT AGENT

Consider a simplified model of an intelligent agent that performs the information retrieval and recognition on the Internet (see figure).

Let us assume that our agent is a certain logic program that processes data extracted from the Internet and outputs reports on the gathered data. In the context of such a model, the following questions are of vital importance:

1. Does the intelligent agent work with certain Internet resources specified in advance or do the resources depend on the incoming data and are they selected in the course of program execution?
2. How long should this agent operate? Certain agents are used only occasionally to carry out one-time search operations on the Web (by the way, this task is quite adequately executed by conventional search engines like Rambler, Alta Vista, etc.), while others continually operate for months or even years.

Obviously, when an agent works with certain resources specified in advance or when it is used for one-time search operations, the new report based on the



A simplified model of the Internet intelligent agent.

[1] The work was supported by the Russian Foundation for Basic Research, project no. 00-01-00560a.

Received October 25, 2000

Pattern Recognition and Image Analysis, Vol. 11, No. 2, 2001, pp. 454–457.

modified data from the Web can be obtained simply by running the logic program all over again. However, if the necessary resources are selected by the agent (i.e., if the *active* information retrieval is performed), a simple repeated run of a logic program may fail because part of the necessary resources may be inaccessible at that very moment, as often happens in the Web. If, in addition, the agent is designed for long-term data collection, the logic program restart will lead to the wasting of computational resources and to the loss of collected data.

Thus, the logic programming of intelligent agents that perform the active information retrieval and operate over relatively long periods of time must be based on a certain inference mechanism allowing one to update original data and to modify the statements that were inferred on the basis of these data with all other results being preserved. In other words, what we need is a certain mechanism of modifiable reasoning.

2. FORMALIZATION OF MODIFIABLE REASONING ON THE INTERNET

Regarding modifiable reasonings, the first thing that comes to mind is nonmonotonic reasoning. The idea of *nonmonotonic logical systems* [1], by and large, consists in formalizing the reasonings that allow for the construction of satisfiable but not valid statements. Such logical schemes allow one to formalize the rejection of certain statements that were inferred earlier whenever new data that contradict these statements become available. That is why nonmonotonic reasoning is seen as the best (and sometimes, the only possible) tool for the formalization of modifiable reasonings. Nevertheless, starting from our experience of work with inference systems [3, 5, 6], we can assert that this idea is wrong or, at least, does not reflect the entire range of methods that can be used to formalize modifiable reasoning.

As an alternative to the nonmonotonic logical schemes, we have elaborated the technique of *logical actors* [2, 3, 6, 7]. This technique is free from the drawbacks of nonmonotonic reasoning such as

(1) the lack of validity of derived formulas;

(2) the possibility of obtaining mutually exclusive results when the inference rules are applied in a different order;

(3) the danger of looping when determining and interpreting the deducibility relation.

Moreover, our method has certain advantages, which are illustrated by the following example.

We write the same canonical example (about the ostrich Titi [1]) in two different ways. First, we use the logic program with the operator *not*, which is a certain approximation for nonmonotonicity relation in the Prolog.

```
?-can_fly (“Titi,” Answer).          (Goal statement)
can_fly (Name, “yes”):–
      bird (Name),
      not ostrich (Name).            (Negation through failure)
can_fly(Name, “not”):–
      bird (Name),
      ostrich (Name).
bird (“Titi”).
ostrich (“Titi”).                    (Appended statement)
```

If the database does not contain the fact formulated as *ostrich (“Titi”)*, then the proof of the statement **not** *ostrich (“Titi”)* will succeed and the Prolog will read out *Answer* = “*yes*,” meaning that Titi can fly. However, when the fact *ostrich (“Titi”)* is appended, this negation becomes inderivable, and the Prolog will answer that Titi cannot fly. Notice that the new information is appended as a fact of the database; that is, the *program text* is modified. The *query text* remains unchanged under this procedure.

Now, let us write the same example in terms of logical actors. *Logical actors* [2] are the logic program’s subgoals that can be solved *repeatedly* without logic program *backtracking*.

```
?-can_fly (“Titi,” Order, Answer).          (Goal statement)
can_fly (Name, Order, Answer):–
      bird (Name),
      @verify_order (Order, Answer).        (Logical actor)
verify_order (“conventional,” “yes”).
verify_order (“ostriches,” “not”).
bird (“Titi”).
```

This program does the same, but the principle of its operation is different. The main distinction is that not the *program text* but the *query text* is modified (the text of the goal statement). The new information comes not in the form of logical statements but in the form of terms, i.e., in the form of new values of variables in the goal statement.

This is done as follows. In the course of executing the logic program, the actor *verify_order* will be solved as a usual subgoal; the Prolog will output the following result: *Answer* = "*yes*," and, moreover, it will assign the value to the variable *Orde*r, i.e., *Order* = "*conventional*." Subsequently, if it turns out that Titi is an ostrich, this information must be communicated to the program via the following *destructive assignment* operation:

Order: = "ostriches."

The correctness of logical inference is certainly violated by this destructive assignment. However, once the value of the variable *Order* has been changed, the logical actor mechanism will restore the correctness of the inference by the repeated proof of a separate subgoal, namely, the logical actor *verify_order.* This time, the fact *verify_order ("ostriches," "not")* will be selected in the database and the program will output the new result: *Answer* = "*no*."

The second formalization method has the following merits:

1. The new data are arriving in the form of terms (data items). This is more convenient from the standpoint of real-life program designing.

2. If the source data have changed, it is necessary to resolve only some separate program subgoals; moreover, these subgoals can easily be found by analyzing which predicates depend on which variables.

3. Finally, the most important advantage is that the reasoning is carried out in the framework of classical monotonic logic, and the modifiable reasoning implementation becomes, in fact, purely an engineering problem.

To implement the mechanism of logical actors, it is necessary to work out an appropriate inference strategy that supports the repeated solving of program subgoals. In [6], it is proven that such strategies do exist; they have the *completeness* (provided there is no infinite looping) and *soundness* properties.

We implemented one of the possible strategies in the Actor Prolog language [2, 4, 7]; this strategy was tested for the problem of logical analysis of functional (SADT) diagrams of information systems [5, 6].

3. SCHEMES FOR USING THE LOGICAL ACTORS

The simplest scheme of using the logical actor technique on the Internet is the application of the Actor Prolog language for programming intelligent agents [8]. In this process, the logic program creates a set of logical actors that are related to each other and to external information resources by common variables. If some of the Internet resources being used undergo changes, the repeated solving of the entire logic program is not necessary because only several actors are affected.

The following more complex schemes of using logical actors on the Internet are believed to be of practical interest. The Internet agents are conveniently constructed of individual blocks, i.e., smaller agents (concurrent processes) that transfer data flows to one another. If each elemental agent of such a network is represented by a program written in the Actor Prolog language, then the declarative semantics of the agent as a whole can be characterized via the AND operation on Horn clauses corresponding to individual blocks. In particular, this scheme fits together with the ideas of visual programming via *data-flow diagrams*.

Data-flow diagrams can be used not only as a visual programming language, but simply as a *graphic interface for the interaction with a user.* In so doing, the states of individual blocks (validated/not validated) can be marked by different colors. The validation of separate blocks can be performed manually, for instance, clicking the mouse on corresponding boxes of the screen. Note that such a scheme presumes a clear-cut division of functions between a human and a computer; that is, the human is responsible for solving the *completeness* and *termination* manually handling the order of the block validation, while the computer ensures the *correctness* of the inference implementing sound inference control strategy.

One more scheme of using logical actors that deserves an independent examination is the logic programs that *dynamically* generate new agents (Concurrent Processes) in the course of their execution. The construction of new agents in the course of logical program execution can be considered as the transformation of its declarative semantics.

CONCLUSION

In this paper, we have considered the problem of developing a modifiable reasoning technique that ensures the correctness of logic programs (intelligent agents) functioning in the Internet environment under conditions of permanent changes in the information content. We have proposed a new version of modifiable reasoning, namely, the *logical inference with the modification of the goal statement*. Certain schemes of using the mechanism of logical actor to program Internet agents are described.

REFERENCES

1. Thayse, A., Gribomont, P., Hulin, G, *et. al.*, *Approche Logique de l'Intelligence Artificielle*, vol. 2: *De la Logique Modale à la Logique des Bases de Données*, Paris: Dunod, 1989.

2. Morozov, A. A., Actor Prolog, *Programmirovanie*, 1994, no. 5, pp. 66–78.
3. Morozov, A. A., Obukhov, Yu. V., and Oleinikov, A. Ya., Logic Programming of Open Systems, in *Logika, metodologiya, filosofiya nauki: tez. dokl. XI mezhd. konf.*, (Proc. 9th Int. Conf. on Logics, Methodology, and Philosophy of Science), Obninsk, 1995, pp. 153–156. (http://www.cplire.ru/Lab144/obninsk.html).
4. Morozov, A. A. and Obukhov, Yu. V., *Actor Prolog. Opredelenie yazyka programmirovaniya* (Actor Prolog. Programming Language Definition), Moscow, 1996, Preprint 2(613) of Institute of Radio Engineering and Electronics, Russian Academy of Sciences. (http://www.cplire.ru/Lab144/index.html).
5. Morozov, A. A. and Obukhov, Yu. V., Semantic Analysis of Functional Diagrams of Information Systems with the Means of Object-Oriented Logic Programming, in *Razvitie i primenenie otkrytykh sistem: tez. dokl. IV mezhd. konf.* (Proc. 4th Int. Conf. on Open System Development and Application), Nizhni Novgorod, 1997, pp. 61–64. (http://www.rapros97.nnov.ru/reports/9.html).
6. Morozov, A. A., Logical Analysis of Functional Diagrams in the Interactive Design of Information Systems, *Cand. Sc. (Phys.Math.) Dissertation*, Moscow, 1998. (http://www.cplire.ru/Lab144/auto.html).
7. Morozov, A.A., Actor Prolog: an Object-Oriented Language with the Classical Declarative Semantics, *Proc. of IDL'99 workshop*, Paris, 1999. (http://www.cplire.ru/Lab144/paris.pdf).
8. Gulyaev, Yu.V., Morozov, A.A., and Obukhov, Yu.V., On the Problem of Using Logic Object-Oriented Programming in the World Wide Web, *Proc. of the Special Russian Session "The Internet Developments in Russia." First IEEE/Popov Workshop on Internet Technologies and Services*, Moscow, 1999, pp. 54–59. (http://www.cplire.ru/Lab144/internet.html).