Acta Cybernetica, Tom. 7, Fasc. 4, Szeged, 1986.

Problem solving based on knowledge representation and program synthesis

S. S. LAVROV

Computers cannot solve problems, they are only able to execute programs. A man can solve problems if he has necessary knowledge and experience. In informatics to solve a problem means to find and to describe a sequence of computing operations leading to the intended result. If a problem is solved in that sense then a program capable to get an answer to that single problem is created. Possibly the program can supply solutions of a number of problems differing however only in their input data.

A man can usually do more than this. He knows the field of his activity — an object area as we shall call it. He is able to solve many essentially different problems in this area.

From this standpoint a challenging problem arises — how to transfer human's knowledge and experience to a computer, how to make it capable to solve a large class of problems, not just one, in a specific area. The problem is by no means a new one. It is known a number of ways to solve it.

These traditional ways are: program packages (if one takes an algorithmic approach), data bases (when an informational approach is preferable), expert systems. Every program package or expert system usually has its own built-in control device. In the case of data bases this role is played by a data base management system. Such a system enables us to create different data bases oriented to different object areas. A similar approach is known in connection with expert systems.

There exist also systems based on more abstract form of knowledge representation. Among them the language PROLOG [1] and its implementations and applications should be mentioned first of all.

However in all these cases we have one large program or system which directly uses a computer to solve various problems. The system does not try to generate a program for each specified problem. In other words all these systems are rather interpretive than compilative by their nature.

Only one form of knowledge representation is used in every kind of system. This form is: a computing procedure or a program module in the case of a program package, a table in the case of a data base, a rule of the form "condition→action" in the case of an expert system, a Horn clause in the case of a PROLOG program.

It occurs sometimes that two very similar application systems having almost the same purpose and possibilities are classified differently by their authors, e.g. as an expert system and a data base, depending on the authors' tastes and points of view.

A very interesting problem solving system called PRIZ was developed in Tallinn by E. H. Tyugu and his colleages as early as the first half of 60-ies [2—4]. The system has very much progressed since then of cause. The work which will be reported here was inspired in many aspects by this system.

The approach adopted in the informatics division of our institute is intended to overcome the drawbacks and the restrictions mentioned above. In our system called SPORA ("СПОРА") we wanted to develop a unified approach combining the principles accepted by the designers of many application systems. On the other hand we tried not to mix up different concepts. Moreover our intention was to find the most appropriate place in the system for every independent concept known. We wanted to use any such concept with maximal effect.

Similarly we tried to take the greatest possible advantage of different experience and tastes of different people working in any chosen area. There are always people with strong mathematical attitude and a good mathematical education. There are people also who like computer programming and are eager to contact, to cooperate with a computer. Surely most people are using computers only by necessity because without them they could not reach the desirable result.

Starting from all these considerations we have built our system in the following way.

The main part of the system is a knowledge base. We distinguish at least three kinds of knowledge: conceptual, algorithmic and factual ones. Conceptual knowledge is a set of terms (words) naming the basic concepts or notions of a given object area together with their properties and relationships. All this is expressed on the most abstract level. A description of an object area on this level is called conceptual model of the area. Abstraction is made from physical representation of entities (i.e. from their measurement units), from their programming representation (possible data types in some algorithmic language) and even from their mathematical representation. E.g. we prefer to write the Ohm law in the form

Ohm (voltage, current, resistance)

or

voltage=times (current, resistance)

where the word "times" denotes a map with no predefined mathematical properties instead of the usual $u=i*r$.

A conceptual model contains some entity types and functional dependencies (maps) called primary. They are just names and on the abstract level do not posses any directly stated properties. However we consider the possibility to add a sort of axiomatics describing such properties to a model.

The model also contains secondary types and maps which are described in a relational manner. The components of an object, i.e. the attributes of a type or the arguments and the result of a map should be explicitly listed. In addition to this the dependencies between the components should be described.

There are tree kinds of dependencies. A functional dependency has the form $v:=t$ where v is a component of an object, t is a functional term constructed from such components. Such a dependency prescribes the value of v to be computed as a result of the term t evaluation.

An equational dependency (or simply an equation) having the form $t_1 = t_2$ prescribes values of functional terms t_1 and t_2 to be equal.

A relational dependency looks like this:

$$\text{is } R_1(v_{i_1} = t_1, \ldots, v_{i_k} = t_k)$$

where R_1 is another type described elsewhere, $v_{i_1}, \ldots, v_{i_k}$ are some of the attributes of R_1 and $t_1, \ldots, t_k$ are functional terms constructed from the components of the currently defined object. Such a dependency allows one to use the dependencies associated with the type R_1 in the actual definition. It means that an object with the components $v_{i_1}, \ldots, v_{i_k}$ having values supplied by the terms $t_1, \ldots, t_k$ must belong to the type R_1.

There is a possibility for an object to have optional components and conditional dependencies between them. The definition of either a type or a map may be recursive.

This approach has his pros and cons. The main gain is that abstraction from many details usually opens the shortest and the most natural way to a solution of the given problem. The main drawback is that a solution (if one is found) is an abstract one and cannot be directly used for computation. Another difficulty arises from the fact that equational dependencies cannot be resolved on the abstract level. This is so because on that level we abstract from the mathematical representation of primary maps and cannot use their mathematical properties.

Therefore an abstract conceptual model of an object area needs an algorithmic and informational support. At this point two other kinds of knowledge mentioned above step on the scene.

Algorithmic knowledge is a collection of ways to represent each entity as an object of some algorithmic language and each map as a procedure written in such a language. Such a representation is needed only for the entities of primary types and for primary maps. The secondary types and maps have a standard representation based on their desription sketched above. The algorithmic languages used for the representation of algorithmic knowledge are called base languages of the system. Currently base languages are Pascal, FORTRAN and ALGOL 60. The system itself is written mainly in Pascal and partially in the assembly language of the BESM 6 computer.

Factual knowledge is a set of values and qualitative characteristics of objects under consideration. The most natural way to represent the factual knowledge is to put it in a data base. The data manipulation language of the data base is also considered as a base language of the system.

Thus the whole knowledge base consists of a conceptual model, a program modules package forming the algorithmic support of the model and a data base forming its informational support. Certainly an interface between these three parts of the knowledge base should be described. We consider both the program package and the data base equally important parts of the knowledge base having equal rights and status.

Our problem solving system has a number of input languages oriented to different needs and to different classes of users.

The most complicate and still rather simple language is the language for object area description. People using this language to construct conceptual models should be experts in the object area. They must be mathematically educated as well. We call them model designers. They are working in close contact with (if not being the

same) people creating algorithmic and informational support of the model, i.e., program packages and data bases.

To describe the interface between the model and its support one uses another input language called representation language or more exactly — primary types and maps representation language. This language is essentially a kind of universal macro-language. For each primary data type one has to describe a way to translate a name of an object having this type into a base language construction. For each primary map a way to call corresponding procedure should be described.

Users of the representation language should be good programmers first of all. Their main task is to describe the interface between the conceptual model and its algorithmic and informational support. Both program modules and contents of a data base included in the support may be written by other people. The use of a macro-language allows one to include arbitrary modules or data collections in the conceptual model support.

A knowledge base of an object area being constructed, it is a rather simple task to specify a problem from the area. Essentially one has to list all the input quantities and the desirable result, in other words, to point out the places which these data occupy in the model. Additionally if the data base part does not contain the numerical values of some input quantities one has to supply the system with these values.

All this information may be expressed in a rather simple language called request language and oriented to the most numerous category of users, viz. terminal ones.

A closer consideration reveals however a gap between the conceptual model and a problem specification existing in that scheme of problem solving. In fact there may be no places for the input data and the result of a problem in the model. Let us take geometry as an example of an object area. Its model contains such notions as a point, a straight line, a triangle, a circle, a distance, an angle and so on, such relations as the incidence of a point and a line, the tangency of a line to a circle etc. To state a problem however one needs a collection of objects having various relationships between themselves to be described or drafted. The conceptual model of geometry cannot contain all such intented collections.

Therefore before stating a problem one has to describe a more or less concrete object on which the problem will be stated. We meet here another kind of knowledge which may be called a constructive one. Constructive knowledge is a set of rules and methods allowing one to describe an object under investigation on the basis of a conceptual model. A number of problems may be posed with respect to the object.

In our system we have a device useful by itself which can serve this purpose. The device is that of submodels related to a given model. Indeed the means to describe an investigated object are essentially the same as those used in the description of any secondary object.

To conclude a couple of words about the system functioning should be said.

A problem specification together with the model of the corresponding object area (or part of the model) is translated into a logical language. The result of the translation is an existence theorem for a solution of the given problem. Then the system tries to prove the theorem. From the proof if one is found the system extracts an algorithm leading to a solution of the problem. We call this algorithm an abstract program because it is expressed in terms of the conceptual model.

Next the abstract program is translated into one of the base languages of the system. The interface between the model and its support is also used on this stage.

The resulting base language program can be compiled into machine code and then be executed by computer. The input data for the computation can be either taken from the data base or given by the author of the problem specification.

Variants of the scheme just described are possible. The abstract program can be printed for examination by the user instead of being processed further. If the problem stated is of a rather general nature then the abstract program can be added to the conceptual model while its translation into a base language is added to the algorithmic support of the model. The compiled program can be included into a library. Thus not only the final result of computation but all the intermediate ones starting from the abstract program can be considered as a solution of the problem.

When one of the steps described above fails the user gets a diagnostic message.

The first version of the system SPORA was written in 1977—82. Recently the second version was developed and tested.

References

[1] Van Emden, M. E., Kowalsky R. A., The semantics of predicate logic as a programming language. — J. of the ACM, 1976, v. 23, N 4, p.p. 23—32.

[2] Tyugu, E. H., A data base and problem solver for computer-aided design. — *In:* Information Processing 71. — North-Holland Publ. Co., 1972, p.p. 1046-49.

[3] Tyugu, E., Towards practical synthesis of programs. — *In:* Information Processing 80. — North-Holland Publ. Co., 1980, p.p. 207—19.

[4] *Тыугу Э. Х.* Концептуальное программирование. — Москва, изд—во „Наука", 1984.

(Received Sept. 26, 1985.)

ИНСТИТУТ ТЕОРЕТИЧЕСКИЙ АСТРОНОМИИ
АКАДЕМИИ НАУК СССР
191187 ЛЕНИНГРАД
НАБЕРЕЖНАЯ КУТУЗОВА, 10
СССР