CONTEMPORARY MATHEMATICS

Methods and Applications of Mathematical Logic

Proceedings of the
VII Latin American Symposium
on Mathematical Logic
held July 29–August 2, 1985

CAN MATHEMATICAL SOCIETY

VOLUME 69

Contemporary Mathematics
Volume 69, 1988

PROBLEM SOLVING BY INTERPRETATION OF THEORIES

Paulo A.S. Veloso

ABSTRACT. Interpretation of theories can be used to solve problems in general. A (concrete) problem, following Polya, consists of two sets, of data and of results, together with a relation between them, the requirement. A solution is a function assigning results to data so as to satisfy the requirement. An abstract problem, e.g. "ordering sequences", is a class of concrete problems specified by axioms. A (problem-solving) method, e.g. "divide-and-conquer", consists of an algorithm together with axioms guaranteeing its correctness. In applying a method to solve a problem one defines the primitive concepts of the former, e.g. "splitting into simpler instances", in terms of those of the latter, so that the axioms of the method so translated can be derived from those of the problem. This amounts to constructing an interpretation of the theory underlying the method into (an extension of) that of the problem. These formulations, which are intended to capture our intuitions, permit a precise investigation of questions related to problems and problem-solving methods. Moreover, they are applicable to the processes of program construction and of problem solving in general.

1. INTRODUCTION

The concept of interpretation of theories has proved to be a powerful tool in mathematical logic, e.g. for establishing relative consistency or undecidability. Here we want to show that it can have even more widespread applications, in solving problems in general. In fact, we shall argue that solving a problem by a method actually *means* constructing an interpretation of theories; or, at least, can be so regarded if properly understood.

The structure of this paper is as follows. In the next section we make precise some ideas of Polya [1957] to define a (concrete) problem as a many-

Partly sponsored by CNPq (grant 40.3012/84) and FINEP.

1980 *Mathematics Subject Classification* (1985 *Revision*). 68T20, 03B70, 68Q65, 68Q60.

sorted structure and an abstract problem as a class of such structures which are the models of a specification. In section 3 the notion of problem-solving strategy is precisely formulated as a method, which is a theory consisting of an algorithm to construct solutions together with axioms guaranteeing the correctness of this procedure. Having formulated both abstract problem and problem-solving method as theories it is quite natural to define the application of the latter to the former in terms of interpretations, which is done in section 4. Then section 5 illustrates these ideas with two examples of methods, namely decomposition and reduction. In section 6 we briefly indicate how the logical concepts underlying these ideas can be made more precise, which leads to the comparison of methods with respect to their power. Finally, section 7 contains some concluding remarks.

2. THE CONCEPT OF PROBLEM

What is a problem? Certainly, everyone can recognize a problem upon seeing one. So, let us start with a simple example. Let us consider the problem of "finding a root of a given polynomial". Is it well formulated? How are we to specify more precisely this problem? Polya [1957] suggests asking three questions in approaching a problem; let us follow his suggestion and ask the following three questions: "What are the data?", "What are the possible results?", "What constitutes a satisfactory solution?".

Polya's first question concerns the data. We must describe the domain of possible data (or problem instances). We want to find roots of polynomials, but of what kind: do they have integral or real coefficients, are they quadratic or can they have arbitrary degree? Clearly, the answers to these questions affect the nature of the problem and its difficulty.

Polya's next question is about the possible results to be expected,whose domain is to be precisely described. What kind of roots do we want: integral, real, or complex? The very solvability of the problem depends strongly on the answer to this question.

Finally, we must have a clear idea of what constitutes a solution, i.e. which results match which problem instances according to the problem requirements. For, a polynomial in general has more than one real root: do we want all of them, any one of them, or the smallest one? This also influences the very nature of the problem and its difficulty.

Thus, it would not be reasonable to propose a problem as "finding a root of a polynomial" as it is not clear enough what is meant. We should make clear that what we want is, say, "finding any complex root of a polynomial in one indeterminate with real coefficients and arbitrary degree". Then, we have described the domain of data (or instances) as $D = \mathbb{R}[x]$, that of results as $R = \mathbf{C}$ and the requirement (or condition) for a possible result $c \in R$ to match a data $p[x] \in D$ to be simply that the value of $p[x]$ at c be 0.

With the hindsight provided by this example we can formulate our definition of problem [Veloso and Veloso 1981]. A (*concrete*) *problem* is a many-sorted structure P of the form $\langle D, R, p, \ldots \rangle$, where D and R are nonempty sets (called the domains of data and results, respectively), and p is a binary relation from D to R (called the requirement). A *solution* for P is a function $f : D \longrightarrow R$, assigning to each data $d \in D$ a result $f(d) \in R$ satisfying the requirement in the sense $\langle d, f(d) \rangle \in p$, or in short $f \subseteq p$.

We should remark that the above definition of problem tries to capture the idea of a general problem and not of specific instances: for instance, finding a complex root of $2x^3 + 3x^2 + 5x + 7$ is an instance of the general problem illustrated above of finding a complex root of a polynomial with real coefficients in the indeterminate x. As such, this formulation was influenced by the concept of decision problem from recursive function theory [Rogers 1967]. It is, however, general enough to encompass many kinds of problems, as for instance, Polya's "problems to prove" and "problems to find", as well as "construction problems", puzzles and everyday life problems.

Let us examine another quite simple example of problem, that of "sorting sequences of integers (without repetitions) into increasing order". Here, a possible data is a sequence like $\langle 5, 1, 2, 7 \rangle$, with corresponding result $\langle 1, 2, 5, 7 \rangle$. In *describing* a solution for this problem one sees that the fact that the sequences contain integers does not matter too much. Probably the same basic ideas used in obtaining a solution for it would apply if the sequences contained elements from some other linearly ordered domain E instead. Thus, we might solve a whole class of sorting problems, all at once, namely, the class of all the concrete problems where D and R consist of sequences (without repetitions) of elements from a set E which is linearly ordered and whose requirement p is defined by $\langle d, r \rangle \in p$ iff r is increasing and r has the same elements as d. Thus we are naturally led to consider classes of similar concrete problems sharing a common specification.

An *abstract problem* is a theory A, presented by a set Γ of axioms (called its specification) in a many-sorted language $\mathcal{L}$ including sort symbols D and R and a binary predicate symbol p from D to R. The models of an abstract problem are the concrete problems with this language that satisfy the axioms in the specification. One can solve each one of these concrete problems separately obtaining a solution for it. Usually, each such solution is obtained by describing a function from data to results. What should we regard as a solution for the abstract problem? An apparently natural answer is a common description of the solutions for all of its the concrete problems. Thus we define a *solution* for an abstract problem to be a description f of a function – written in an appropriate formalism, as a programming language or logical definitions – such that, on each concrete problem $P = \langle D, R, p, \ldots \rangle$ satisfying the specification Γ, f defines a function from D to R which is actually a solution for P.

3. THE CONCEPT OF PROBLEM-SOLVING METHOD

As mentioned above, in solving a problem one generally defines a function in a stepwise manner [Nilsson 1971]. There are several strategies for solving problems, one of the most powerful ones being the so-called "divide-and-conquer". The basic idea of this strategy is dividing hard problems into hopefully simpler ones. For instance, the task of proving a theorem may be divided into those of proving some propositions, which in turn may require proving a few lemmas, and so forth; all these proofs, if put together, would provide a proof for the theorem.

Let us illustrate how the divide-and-conquer strategy works on our example of sorting. In order to sort a sequence as $\langle 5, 1, 2, 7 \rangle$, which is not simple enough, we split it into two halves $\langle 5, 1 \rangle$ and $\langle 2, 7 \rangle$. This process of splitting is repeated until we have only simple data, in this case the 4 unit-length sequences $\langle 5 \rangle$, $\langle 1 \rangle$, $\langle 2 \rangle$, and $\langle 7 \rangle$, which are simple enough to be sorted immediately, actually resulting in themselves. Now we start merging these results: $\langle 5 \rangle$ and $\langle 1 \rangle$ into $\langle 1, 5 \rangle$, and $\langle 2 \rangle$ and $\langle 7 \rangle$ into $\langle 2, 7 \rangle$. This process of merging continues until we recombine the results for all the data that were split, obtaining in the end a result for the original problem instance. The final step, in this case, consists of merging $\langle 1, 5 \rangle$ and $\langle 2, 7 \rangle$ into $\langle 1, 2, 5, 7 \rangle$, which is a result for the original data. This algorithm for sorting is as follows: given a sequence, if it is not simple then split it into two halves and recursively sort these halves and merge the sorted results, otherwise the result is the input sequence. This is the well-known "mergesort" algorithm for sorting [Horowitz and Sahni 1978].

The above method is naturally formulated as a (recursive) algorithm for sorting sequences. It does embody the basic ideas of the divide-and-conquer strategy. However, it is not the strategy itself, being rather the application of the strategy to the problem of sorting. We would like to consider the strategy per se in order to study it independently of its many possible applications.

The basic idea of divide-and-conquer can be expressed by the following (recursive) definition of a function (symbol)

$$f = (\lambda d : D)\ [\ b(d) \Rightarrow k(d) \\ \neg b(d) \Rightarrow c(f(h_1(d), f(h_2(d)))]$$

The intended meaning of the predicate symbol b on sort D is that of "being a simple data" whereas those of the function symbols h_1 and c are, respectively, "the first split of a data" and "the recombination of the results". These intended meanings can be expressed by axioms including e.g. the following two

$$(\forall d : D)[b(d) \longrightarrow p(d,k(d))]$$
$$(\forall d : D)(\forall r_1, r_2 : R)\{\neg b(d) \longrightarrow [p(h_1(d),r_1) \,\&\, p(h_2(d),r_2) \longrightarrow p(d,c(r_1,r_2))]\}$$

The idea is that on any concrete problem which satisfies the axioms the algorithm defines a function which is actually a solution for the concrete problem.

We define a *(problem-solving) method* M to consist of an algorithm G defining a function symbol f (perhaps together with and in terms of some auxiliary ones) together with a specification Λ consisting of axioms guaranteeing its correctness in the sense that on any concrete problem $P = \langle D, R, p, \dots \rangle$ which is a model of Λ the algorithm G will define a total function from D to R which is a solution for P. The language of the method M consists of the (nonlogical) symbols of the algorithm together with those of the specification.

4. THE CONCEPT OF APPLICATION OF A METHOD TO A PROBLEM

The basic idea of the "divide-and-conquer" strategy was captured in a method as an algorithm describing how the solution is obtained (in terms of some auxiliary function and predicate symbols) and axioms (involving these auxiliary function and predicate symbols) which guarantee the correctness of this algorithm. In applying this idea to the problem of sorting we defined these auxiliary functions and predicates in terms of those of the problem, namely sequences, thereby obtaining the mergesort algorithm. For instance, we defined the predicate symbol b to mean that the sequence has length (at most) one and the function symbol c to mean merging of sequences. In order to ensure that this algorithm does sort correctly it suffices to show that the specification of the method so translated does hold, i.e. it can be derived from the specification of the problem. This is the idea behind interpreting a theory into another [Enderton 1972, Shoenfield 1967].

A method M consists of a syntactic part (its language) together with an algorithm G and a specification Λ, the latter guaranteeing correctness of the former. Similarly, an abstract problem A consists of a syntactic part (its language $\mathcal{L}$) together with a specification Γ. Now, the application of a method to a problem will consist of a syntactic part enabling the translation from the language of M into $\mathcal{L}$ together with the condition that this translation preserves specifications.

Consider a (problem-solving) method M consisting of an algorithm G and specification Λ and an abstract problem A with language $\mathcal{L}$ and specification Γ. An *application* of the method to the abstract problem is an algorithmic interpretation of the theory Λ into Γ. As such, it consists of an mapping I assigning to each symbol of the language of M a definition for it in terms of the symbols of $\mathcal{L}$ and satisfying a correctness criterion. This mapping I induces a translation of expressions of the language of A into corresponding ones of $\mathcal{L}$. The correctness criterion is that each axiom of Λ so translated is a consequence of the specification Γ, in short $\Gamma \models I(\Lambda)$.

As a net result we have the following consequence of an analog of the interpretation theorem:

"If I is an application of the method $M = \langle G, \Lambda\rangle$ to the abstract problem $A = \langle \mathcal{L}, \Gamma\rangle$ then on any model P of Γ, I(f) defines a function which is a solution for P, where f is the function (symbol) defined by G."

5. EXAMPLES OF PROBLEM-SOLVING METHODS

We have sketched the formulation of the divide-and-conquer strategy as a problem-solving method, where each data is split into two. In general, we may want to split each problem instance into n instances. The corresponding method is called *n-ary decomposition* and is denoted by $\mathcal{D}c_n$ [Veloso 1984]. Its language consists of 2 sort symbols D and R; a unary predicate symbol b (on D) and a binary predicate symbol m (from D to D); a unary function symbol k (from D to R), an n-ary function symbol c (from R^n to R), and n unary function symbols $h_1, \dots, h_n$ (from D to D). Its algorithm is

$$\begin{aligned} f = (\lambda d{:}D)\ [\, & b(d) \Rightarrow k(d) \\ & \neg b(d) \Rightarrow c(f(h_1(d)), \dots, f(h_n(d)))] \end{aligned}$$

Its specification consists of the following 4 axioms

(1) $(\forall d : D)\ [b(d) \longrightarrow p(d, k(d))]$

(2) $(\forall d : D)(\forall r_1, \dots, r_n{:}R)\{\neg b(d) \longrightarrow [p(h_1(d), r_1) \,\&\, \dots \,\&\, p(h_n(d), r_n) \longrightarrow$
$\longrightarrow p(d, c(r_1, \dots, r_n))]\}$

(3) $(\forall d{:}D)\ [\neg b(d) \longrightarrow m(h_1(d),d)\ \&\ \ldots\ \&\ m(h_n(d),d)]$

(4) m is well founded.

It is easy to see that on any model $P = \langle D, R, p, \ldots \rangle$ of these axioms the algorithm for f actually defines a total function $f : D \longrightarrow R$ [with domain D, because of (3) and (4)] which is a solution for P [because of (1) and (2)].

A very useful strategy in approaching a problem is trying to look for a similar one. In the words of Polya "Here is a problem related to yours and solved before". Let us formulate it as a method, *reduction*, in our sense [Veloso 1984]. Its language consists of 4 sort symbols D, R, E, and S; 2 binary predicate symbols p (from D to R) and q (from E to S) and a unary function symbol g (from E to S) and its algorithm defines f as the composite $v \cdot g \cdot t$

$$f = (\lambda\, d : D)\ [v(g(t(d)))]$$

Its specification consists of the following two axioms

$(\forall d{:}D)(\forall s{:}S)\ [q(t(d),s) \longrightarrow p(d,v(s))]$
{"... a problem related to yours ..."}

$(\forall e{:}E)\ q(e,g(e))$ {"... solved before ..."}

A model for this specification can be regarded as consisting of 2 concrete problems $\langle D, R, p, \ldots \rangle$ and $\langle E, S, q, \ldots \rangle$, the latter being solved by g (in view of the second axiom). The first axiom guarantees that the composition effects a correct transfer of solutions.

6. SOME LOGICAL ASPECTS

We have been deliberately vague about some details concerning formalisms and notations. In this section we shall be a little more precise about these matters in order to provide a framework where the concepts of the preceding sections can be formulated with more rigor.

Our theories were seen to involve two kinds of definitions, namely, algorithm-like constructions and formula-like axioms. Accordingly, we consider the symbols of a (many-sorted) *language* $\mathcal{L}$ to be categorized into two sets: a set $\mathcal{L}'$ of *constructive symbols* and a set $\mathcal{L}''$ of *declarative symbols*. The idea is that constructive symbols should be given algorithmic definitions, say by means of procedures, whereas declarative symbols can be given classical definitions by means of formulas.

A *theory* W in a language $\mathcal{L} = \langle \mathcal{L}', \mathcal{L}'' \rangle$ consists of a set F of explicit algorithmic definitions for the symbols of $\mathcal{L}'$, and a set of Γ of axioms stating properties of the symbols of $\mathcal{L}'$ and $\mathcal{L}''$. A *model* for such a theory W is a structure P for the declarative (sub)language $\mathcal{L}''$ – thus giving realizations for the declarative symbols – such that all the algorithms in F actually define total functions and predicates on P, thus expanding it to a structure $P \uparrow F$ for the language $\mathcal{L}$, which satisfies all the axioms in Γ. Now, given two theories W and V in a language $\mathcal{L}$ we say that V is a *consequence* of W and write $W \models V$ iff whenever P is a model of W then it is a model of V as well.

When $N' \subset \mathcal{L}'$ and $N'' \subset \mathcal{L}''$ a theory W in the language $N = \langle N', N'' \rangle$ can be *extended by definitions* to a theory in the language $\mathcal{L} = \langle \mathcal{L}', \mathcal{L}'' \rangle$ by adding a theory V consisting of, for each symbol $y' \in (\mathcal{L}' - N')$, an algorithm in N' defining it in terms of the symbols already in N' and, for each symbol $y'' \in (\mathcal{L}'' - N'')$, a formula in N'' defining it in terms of those already in N''. The resulting theory is called the *extension of* W *by the definitions* V and is denoted by W[V]. In this case every model P for W expands uniquely to a model, denoted $P \uparrow V$, for the extended theory.

An *interpretation* I of a language $N = \langle N', N'' \rangle$ into a theory $A = \langle G, \Gamma \rangle$ in a language $\mathcal{L} = \langle \mathcal{L}', \mathcal{L}'' \rangle$ consists of two mappings: J assigning to each symbol $y' \in N'$ a symbol $J(y')$ in $\mathcal{L}'$ and K assigning to each symbol $y'' \in N''$ a symbol $K(y'')$ in $\mathcal{L}''$, satisfying the usual closure conditions. These mappings permit syntactical translations of algorithms in N' to algorithms in $\mathcal{L}'$ and of formulas in N'' to formulas in $\mathcal{L}''$; thus any theory M in N is translated into a theory $I(M)$ in $\mathcal{L}$. Also, each model P of A is a structure for $\mathcal{L}$ which induces a structure $I^{-1}(P)$ for N. Now, an *interpretation* I of a theory $M = \langle F, \Lambda \rangle$ in a language $N = \langle N', N'' \rangle$ into a theory $A = \langle G, \Gamma \rangle$ in a language $\mathcal{L} = \langle \mathcal{L}', \mathcal{L}'' \rangle$ is an interpretation of the language N into the theory A such that $A \models I(M)$. Finally, an *algorithmic interpretation* $\langle I, V \rangle$ of a theory M in the language N into a theory A in the language $\mathcal{L}$ consists of an interpretation I of the theory M into the extension $A[V]$ by definitions of A.

We have an analog of the interpretation theorem, namely:

"If $\langle I, V \rangle$ is an algorithmic interpretation of a theory M in the language N into a theory A in the language $\mathcal{L}$ then every model P for A induces a model $I^{-1}(P \uparrow V)$ for M such that for every classical sentence σ of the declarative (sub)language of $\mathcal{L}$ $I^{-1}(P \uparrow V) \models \sigma$ iff $P \models I(\sigma)$."

We have defined both abstract problem and (problem-solving) method in terms of theories and the application of a method to an abstract problem as an algorithmic interpretation. Thus, we have the following consequence:

"If M is a method with function symbol f defined by its algorithm and (I, V) is an application of M to an abstract problem A then $I(f)$ is a solution for A."

Given some problem-solving methods one is led to ask about their relative power. When should say that a method is more powerful than another one? A natural answer is provided in the following definition. A method M is *at least as powerful as* a method M' (denoted $M \geqslant M'$) iff M applies to solve any abstract problem that M' does. Naturally enough, we say that M is *as powerful as* M' (denoted $M \equiv M'$) iff $M \leqslant M'$ and $M' \leqslant M$.

The above concepts of method as theory and application as algorithmic interpretation provide a sufficient condition for a method to be at least as powerful as another one, in fact in a strong sense in that the comparison is "uniform":

"If there exists an algorithmic interpretation of a method M into a method M' then $M \geqslant M'$."

For, whenever M' is applied, via an algorithmic interpretation, to solve an abstract problem A, then we can construct a composite algorithmic interpretation to apply M directly to A.

Let us examine an application of these ideas. In section 5 we have presented the method of n-ary decomposition as a theory $\mathcal{D}c_n$. It is clear that unary decomposition $\mathcal{D}c_1$ is a special case, thus $\mathcal{D}c_1 \leqslant \mathcal{D}c_n$. On the other hand, we can define an algorithmic interpretation of $\mathcal{D}c_n$ into $\mathcal{D}c_1$ (based on the idea of replacing the problem domains by domains consisting of finite sequences of these elements) [Veloso and Veloso 1981], thus $\mathcal{D}c_n \leqslant \mathcal{D}c_1$. Therefore, unary decomposition is as powerful as n-ary decomposition.

7. CONCLUSION

Based upon some ideas of Polya [1957] we have presented some precise definitions for the concepts of problem and problem-solving method in order to show that applying a method to a problem amounts to interpreting theories. Both abstract problems and methods were defined as theories and the application of a method to an abstract problem as an appropriate interpretation of their theories.

At first it seems that we followed a path composed of many degrees of abstraction: first a concrete problems as a many-sorted structure, then an abstract problem as a class of concrete problems satisfying a specification, followed by a problem-solving method which is supposed to solve many diverse abstract problems; and we also wanted to compare methods. However, our formulation flattened these various levels to just two: structures (concrete problems) and theories (abstract problems and methods). Thus, it was quite natural to employ the concept of interpretation both to define application of a method to an abstract problem and to compare methods with respect to their relative power.

The formulations presented here are intended as "rational reconstructions" of our intuitions about these ideas permitting precise investigation of questions related to them. They appear to be applicable to program construction and to problem solving in general.

REFERENCES

H.B. ENDERTON — A *Mathematical Introduction to Logic*; Academic Press, New York, 1972.

E. HOROWITZ and S. SAHNI — *Fundamentals of Computer Algorithms*; Computer Science Press, Potomac, 1978.

N.J. NILSSON — *Problem Solving Methods in Artificial Intelligence*; McGraw-Hill, New York, 1971.

G. POLYA — *How to Solve it: a new aspect of the mathematical method*; Princeton Univ. Press, Princeton, 1957.

H. ROGERS — *Theory of Recursive Functions and Effective Computability*; McGraw-Hill, New York, 1967.

J.R. SHOENFIELD — *Mathematical Logic*; Addison-Wesley, Reading, 1967.

P.A.S. VELOSO and S.R.M. VELOSO — "Problem decomposition and reduction: applicability, soundness, completeness";;R. Trappl, J. Klir, F.Pichler (eds.) *Progress in Cybernetics and Systems Research*, vol. VIII,Hemisphere, 1981.

P.A.S. VELOSO — "Outlines of a Mathematical Theory of General Problems"; *Philosophia Naturalis*, vol. 21 (nº 2-4), 1984, p. 354-367.

DEPARTAMENTO DE INFORMÁTICA
PONTIFÍCIA UNIVERSIDADE CATÓLICA
22453 - RIO DE JANEIRO - RJ
BRASIL