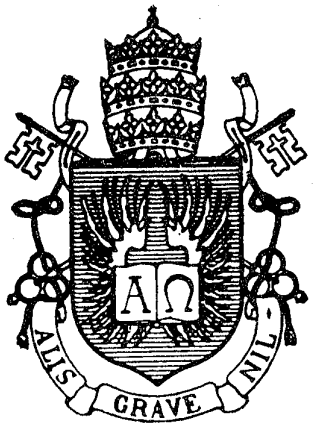

PUC

Series : Monografias em Ciência da Computação

No. 1/85

ON THE CONCEPT OF PROBLEM - SOLVING METHOD

Paulo A. S. Veloso

Departamento de Informática

ONTIFÍCIA UNIVERSIDADE CATÓLICA DO RIO DE JANEIRO

RUA MARQUÊS DE SÃO VICENTE, 225 – CEP-22453

RIO DE JANEIRO – BRASIL

PUC/RJ - DEPARTAMENTO DE INFORMÁTICA

Series : Monografias em Ciência da Computação

No. 1/85 February 1985

Series Editor: Paulo A. S. Veloso

ON THE CONCEPT OF PROBLEM - SOLVING METHOD *

Paulo A. S. Veloso

* Research partly sponsored by FINEP and CNPq .

Abstract

This paper proposes precise formulations, based on abstract data types, for the concepts of problem, problem-solving method, and the application of a problem-solving method to a problem. These formulations are argued to agree with the intuitive understanding of these ideas, thereby formalizing them. This formalization is based on few basic concepts : abstract data type and an extension mechanism (here,a general cluster-like module). Moreover, by embodying ideas related to stepwise refinement they are applicable both to problem solving in general and to the process of program development. Examples are provided to illustrate the main ideas and their application.

Key words : Problem solving,
problem-solving strategy,
problem-solving method,
abstract data type,
program development,
program schema,
knowledge representation.

Sumário

Este trabalho propõe formulações precisas, baseadas em tipos abstratos de dados, para os conceitos de problema, método de resolução de problemas e aplicação de um método de resolução de problemas a um problema. Argumenta-se que estas formulações estão de acordo com o entendimento intuitivo destas idéias, formalizando-as. Esta formalização é baseada em poucos conceitos básicos : tipo abstrato de dados e um mecanismo de extensão (aqui, um módulo que generaliza a idéia de conglomerado de linguagens de programação como CLU, por exemplo).Além disso, por encorporarem idéias relacionadas com refinamentos sucessivos, estas formulações se aplicam tanto a resolução de problemas em geral como ao processo de desenvolvimento de programas.

Palavras chaves : Resolução de problemas,
estratégia para resolução de problemas,
método de resolução de problemas,
tipo abstrato de dados,
desenvolvimento de programas,
esquema de programas,
representação de conhecimento.

Contents

Acknowledgements

Helpful comments by Luiz Carlos Pereira and Sheila R. M. Veloso are gratefully acknowledged.

1. Introduction

The notion of problem-solving strategy or method is fundamental in Computer Science [H+S '78] and in Artificial Intelligence [Ni.'82,So.'84], and some books have been organized around this idea. But it is generally left in somewhat informal terms and is not precisely formulated. This paper proposes a precise formulation for the notion of problem-solving method by means of some concepts related to abstract data types. The basic idea of a problem-solving strategy is argued to be embodied in the concept of a program schema operating on an ADT (short for abstract data type).The program schema itself defines how the solution of the problem is obtained by the method, whereas the ADT specification gives sufficient conditions for the correctness of the solution so obtained. A problem is also defined as an ADT and the idea of applying a problem-solving method to a given problem is argued to amount to implementing the ADT underlying the former on the ADT of the latter.

The organization of this paper is as follows. The next section motivates the basic ideas by introducing them informally on a simple example. Then section 3 presents the proposed definitions for the concepts of (abstract) problem and problem-solving method (PSM, for short), as well as that of applying the latter to the former by employing the concepts of ADT and module schema operating on an ADT. Section 4 illustrates how some

usual problem-solving strategies can be precisely formulated with these notions. In section 5 a more elaborate example of application of these ideas is presented. Finally, section 6 concludes with some remarks on the applicability of these ideas in the processes of program development and problem solving, as well as some brief comments on their relationship with knowledge engineering.

2. A preliminary example for motivation

In order to introduce the basic ideas, we start with a very simple example : the problem of sorting a sequence [Kn.'73]. Here the input data is a sequence of elements and the corresponding output data is a sequence consisting of the same elements rearranged into ,say, increasing order. Thus we have two data domains, the input data domain, D, and the result domain, R, both consisting of sequences of elements. In addition, we have the problem requirement, expressed as a relation from D to R, call it q, defined by

q(d,r) <--> ordered(r) & same(d,r)

where the intended meaning of the predicate symbols is what is conveyed by their mnemonical names; namely,

ordered(r) means that r is ordered (in increasing order),

same(d,r) means that d and r have the same elements, in perhaps different orders.

This is the specification of the problem.

In order to solve this problem we can employ the so-called divide-and-conquer strategy [H+S '78] . The basic idea is as follows [Ve.'80] : given an input data d, if it is simple enough (in the sense of satisfying some simplicity criteria expressed by simple(d)) then we know how to sort it directly (by applying the operation direct on d), otherwise we split it into, hopefully less difficult, problem instances (split_1(d), ..., split_n(d)). Each one of these data will in turn be sorted,

giving results r_1,..., r_n ,which will be then recombined (by applying the operation recomb on r_1,..., r_n) to produce a result r for the original data d .

This basic idea can be described by a program schema (PS, for short) as follows

```
sort(d) = if  simple(d) then  direct(d)
              else  recomb(sort(split_1(d)),...,
                                  sort(split_n(d)) )
```

Of course, in the above PS, we encounter symbols, such as simple, direct, etc.,corresponding to calls to procedures,whose bodies have not yet been given. They will be defined in due time and we will see that in so doing we can obtain several of the usual sorting algorithms. However, before performing this refinement step we can ask ourselves under what conditions the above PS will correctly solve the original sorting problem. These conditions should somehow indicate, and constrain, the possible realizations that the so-far undefined symbols may have. As these will be conditions for the correctness of the PS, we will, as usual, divide them into those for partial correctness and those for termination [Ma.'74].

Conditions for partial correctness will be those guaranteeing that , whenever sort(d) does terminate, giving output r, then we have q(d,r). The very text of the PS suggests the following ones

* a direct result satisfies q :

(1) simple(d) ---> q(d,direct(d))

* q is preserved under split-recombinations :

(2) $\neg$ simple(d) ---> [q(split_1(d),r_1)&...&q(split_n(d),r_n) ---> q(d,recomb(r_1,...,r_n))]

In order to ensure termination, we have to guarantee that the recursive calls terminate, that is ; we eventually attain a simple problem instance by means of successive applications of splits. Here the intuitive idea behind the divide-and-conquer strategy suggests that the result of splitting is a problem instance that is in some sense simpler than the original one. In order to capture this idea we consider a binary predicate symbol smllr on the domain D of data, the intuitive intention being that smllr(d,d') means that d is simpler (or easier to solve) than d' . So, for termination we require the following properties

* the results of splitting are smaller than the argument :

(3) $\neg$ simple(d) ---> [smllr(split_1(d),d) &...& smllr(split_n(d),d)]

* splitting cannot go on forever :

(4) smllr is well founded .

It is not difficult to see that these conditions do guarantee the total correctness of the above PS with respect to the input-output behavior expressed by q(d,r) : given any input data d in D, the recursive procedure call sort(d) terminates giving an output r in R such that q(d,r) holds.

We have been considering the above PS with a parameter n, which indicates the number of subproblem instances into which we are going to split a non-simple data. Let us now fix n to be 2.We can obtain a specific sorting algorithm by supplying realizations for the uninterpreted symbols intervening in the above PS and

axioms. If we do so in such a way that the axioms are satisfied we can be sure of the total correctness of the interpreted program, for it was verified once and for all. Let us now see how some specific sorting algorithms can be thus obtained.

A very familiar sorting algorithm is the so-called "sorting by merging" [Kn.'73]. We can obtain it from the above PS by means of the following definitions, which explain how the previously uninterpreted symbols are supposed to behave. Take

split_1(d) = the first half of d ;

split_2(d) = the second half of d ;

recomb(r_1,r_2) = the merger of r_1 with r_2 ;

direct(d) = d ;

simple(d) <---> d has length at most 1 ;

smllr(d,d') <---> length(d) < length(d') .

With these definitions it is easy to see that the preceding axioms are indeed satisfied , and therefore the above PS is ensured to be totally correct. In addition, this PS with these definitions plugged in is clearly the usual mergesort algorithm.

As another example let us consider the "straight selection sort" [Kn.'73].In this case we take the following definitions

split_1(d) = the sequence consisting solely of the minimum element in d ;

split_2(d) = the sequence obtained from d by the removal of this minimum element ;

recomb(r_1,r_2) = the concatenation r_1 followed by r_2 ;

the other definitions being as before. Under this interpretation the axioms are easily seen to be satisfied and we obtain the

usual straight selection algorithm for sorting.

We mention that we can also obtain other methods of sorting from the above PS, for instance, "quicksort" or "partition-exchange sort" (only we have to take n=3). Still other methods, such as "sorting by insertion" and "tree-sort" , can also be obtained from the above PS coupled with some preparatory steps , which we will see , in sections 4 and 5, to correspond to the general method of reduction of problems.

Let us summarize, for future use, what we have seen in this section. First, we have considered the problem of sorting sequences and showed how this problem can be solved by various particular refinements of the general divide-and-conquer strategy. This strategy was presented in the form of a PS (or abstract program) together with some axioms ensuring its total correctness.Then some well-known sorting methods were shown to be obtained as special interpretations of the general scheme. In order to guarantee the correctness of the PS we wrote some axioms (not all of them necessarily expressible within first-order logic) involving the uninterpreted symbols appearing in the PS as well as some extra symbols, such as smllr.The former, which we shall call "visible", are to be eventually realized by procedures, whereas the latter, which we shall call "hidden",are to be eventually defined as well, but not necessarily by procedures,since they appear only in the axioms, not being invoked by the PS.

3. Problem-solving method and related concepts

Having in mind the ideas informally introduced in the preceding section by means of the illustrative example of sorting we proceed in this section to deal with them in a more general and precise manner.

We shall employ the concept of abstract data type (ADT, for short)[Gu.'77]. For our present purposes, an ADT A is specified by giving :

* a nonempty set S of sorts ;
* a set V of visible predicate and operation symbols ;
* a set H of hidden predicate and operation symbols ;
* a set Ax of axioms involving the preceding symbols .

Thus, an ADT is a presentation of a theory. Its models are the possible realizations of the ADT. So we will be dealing with the so-called loose or incomplete specifications [P+L '79, V+P+M '82] which allow more freedom for the subsequent refinement steps.

Now, let us consider the notion of problem. Polya [Po.'71] suggests that one approaches a problem with the following three questions : what are the data ?, what are the possible results ?, what are the problem conditions ? We can take these questions as a guide in formulating our concept of problem. We can define a concrete problem (CP, for short) [V+V '81] as a two-sorted mathematical structure $CP = \langle D, R, q \rangle$, where

* D is a nonempty set, called the domain of (input) data;
* R is a nonempty set, called the domain of (output) results;
* q is a binary relation from D to R, called the problem requirement.

Accordingly,an abstract problem (AP, for short) is an abstract data type AP , whose language has the following symbols
* D for the sort of input data;
* R for the sort of output results;
* g for a (hidden) binary predicate symbol from sort D to sort R.

The realizations of an AP are the corresponding CP's.

The examples of the preceding section suggest that applying a problem-solving method to a problem amounts to interpreting the undefined symbols of the PSM in terms of the problem. We shall now make this suggestion more precise. The basic idea consists of extending the ADT of the problem so that it can correctly support the PSM.There are three natural ways to extend an ADT : by sort definitions, by procedural definitions (say, by programs) ,and by non-procedural definitions (say, by logical formulas), which match naturally with the sorts,visible and hidden symbols.

A module schema (MS, for short) is a generelized cluster-like [L+Z '74] program text consisting of
* sort definitions by means of type declarations (a la Pascal)
* visible symbol definitions by means of procedure declarations;
* hidden symbol definitions by means of formulas.

This text may also involve some symbols not explicitly defined therein. It is thus a mechanism for defining new symbols in terms

of some others, supposedly defined elsewhere. We say that an MS M operates on (or manipulates) an ADT A iff

* every sort (respectively visible, hidden symbol) occurring in M but not explicitly defined in it is a sort (respectively visible, hidden symbol) of A;
* the axioms of A guarantee the strong termination of the procedures of M, in the sense that on any realization of A the interpreted procedures terminate for all inputs.

The language of such an MS operating on an ADT is the union of the languages of M and A . In view of the above conditions, every realization of the ADT A has a unique expansion to a realization of this extended language. This gives rise to a new ADT, called the extension of A by M.

By a program schema (PS, for short) we mean an MS with a designated main procedure f. Finally, a problem-solving method(PSM, for short) is a PS operating on an ADT, called its underlying ADT.

In order to clarify what we mean by applying a PSM to a problem we need the concept of implementation of ADT's.

An implementation of an ADT A on another one, C, is an MS operating on C which defines all the sorts and symbols of A in terms of those of C, so that with these definitions the axioms of A are derivable from the axioms of C.(The similarity with the logical concept of interpretation of theories [En.'72] is clear. We just mention that another, more flexible,notion of implementation, based on the logical concept of conservative

extension is also useful in this context [V+P+M '82 , M+S+V '84, M+V '85] .)

Now it is clear what we mean by applying a PSM P to an AP : it consists of implementing the ADT underlying the PSM on the ADT of the AP [Ve.'80]. Then we can derive

$(\forall d{:}\underline{D})\, g(d, \underline{f}(d))$

In this case, this PSM will give a correct solution for every concrete problem realizing this AP .

4. Examples of some problem-solving methods

In this section we briefly indicate how the general concept of PSM can be used to formulate some usual problem-solving strategies.

Decomposition

We have already taken a look at the problem-solving strategy of divide-and-conquer in section 2 as an aid in introducing the basic ideas. The problem-solving method embodying this strategy will be called decomposition. Given a natural number n, called the index, by an *n-ary decomposition* [Ve.'80] we mean the PSM consisting of the (recursive) program schema

```
f(d) = if simple(d) then direct(d)
                    else recomb(f(split_1(d)),...,
                                f(split_n(d)))
```

operating on the ADT with sorts D and R, having as visible symbols those appearing in the above PS, and as hidden symbols smllr and g, and as axioms those in section 2, namely

```
simple(d) ---> g(d,direct(d))

¬ simple(d) ---> [ g(split_1(d),r_1)&...&g(split_n(d),r_n)
                        --->g(d,recomb(r_1,...,r_n)) ]

¬ simple(d) ---> [ smllr(split_1(d),d)&...&
                        smllr(split_n(d),d) ]
```

smllr is well founded .

These axioms guarantee the correctness of the PSM, as outlined in section 2 .

Reduction

Another important example of PSM is reduction [V+V '81], consisting of the PS

$f(d) = retr(f'(ins(d)))$

operating on the ADT with 4 sorts D, R, D', R', having as symbols the following ones

hidden predicate symbols $g : D \times R$ and $g' : D' \times R'$;

visible operation symbols $ins : D \longrightarrow D'$,

$retr : R' \longrightarrow R$,

$f' : D' \longrightarrow R'$;

and the following two axioms

$g'(ins(d), r') \longrightarrow g(d, retr(r'))$

$g'(d', f'(d'))$

This PSM is called reduction because it embodies the strategy of reducing a problem (the AP $\langle D, R, g \rangle$) to another one (the AP $\langle D', R', g' \rangle$). As such it might be called a problem-transforming method, rather than a problem-solving method. We call it a PSM because we formulate it as already having a solution f' for the second problem. In the words of Polya [Po.'71] : "Here is a problem related to yours and solved before".

Other problem-solving methods

Many of the usual problem-solving strategies can be formulated as a PS operating on an ADT. For instance, the greedy method, dynamic programming, backtracking, etc.,[H+S '78] have been formulated in this manner. We omit their detailed formulation,which is not relevant for our present purposes. We just mention that these formulations capture the intuitions behind the corresponding strategies. Some of these methods formulated along these lines, together with illustrative examples, can be found in [Wa.'84].

5. An illustrative example of application

In this section we illustrate how the preceding ideas can be used by applying them to a simple but interesting example. The example we chose is, once again, the problem of sorting, but now we are going to solve by a more sophisticate method.

We have an AP **SORT** = <Seq[El],Seq[El],is-sort>,where the axioms Ax specify that the two sorts are to be realized as domains consisting of sequences of elements from some linearly ordered domain, El, and define the predicate symbol is-sort as before.

In order to illustrate more clearly the application of the preceding ideas and concepts we shall indicate the solution of this problem by means of the tree-sort method. We recall that this method solves the problem of sorting a sequence by storing it into an auxiliary structure, an ordered tree, which is then traversed to produce the sorted result.In other words, the problem of sorting sequences is reduced to that of merging sequences into binary search trees.

In order to perform this reduction we implement the ADT

underlying the PSM reduction on the ADT SORT. For this purpose, we extend SORT by an MS containing the definitions of two new sorts, Tree[El]×Seq[El] and Tree[El], and a hidden predicate symbol is-merger from the former to the latter, defined by

is-merger((t,s),t') iff t' is a merger of the tree t with the sequence s.

In addition, this MS is to include definitions for the remaining symbols of reduction, so that the axioms of reduction hold.

It is quite simple to write a procedure for ins, say, ins(s) = (null_tree ,s) . Those for retr and f' generate two new problems, respectively, traversing a tree in inorder and merging a sequence into a tree. A natural way to solve them is by the divide-and-conquer strategy. We shall illustrate this with the latter.

In order to solve the AP < Tree[El]×Seq[El], Tree[El], is-merger > by divide-and-conquer we have to implement on the ADT of this AP the ADT underlying unary decomposition.

For this implementation we write an MS consisting of

simple(t,s) <---> length(s) = 0 ;

direct(t,s) = t ;

split_1(t,s) = (put(hd(s),t),tl(s));

recomb(t) = t ;

smllr((t,s),(t',s')) <---> length(s) < length(s') .

Here the operation put(e,t) is intended to insert the element e into the tree t so as to produce a new ordered tree. The definition of this operation generates a new problem, which can

also be solved by decomposition, as is well known.

This implementation thus solves the problem of defining the operation symbol f'. Similarly, we can apply decomposition to solve the problem of defining retr, obtaining the usual recursive definition of tree traversal in inorder.We have thus completed the definition of an MS which applies the PSM.reduction to the AP **SORT.** In doing so we resorted several times to the PSM (unary) decomposition, in order to solve the auxiliary problems posed by the definitions of the symbols intervening in this implementation.

6. Concluding remarks

The intuitive notions of problem, problem-solving stategy and application of a problem-solving strategy to a problem have been examined and precisely formulated in terms of abstract data types.

The definition of problem [V+V '81], consisting of data, results and a relation between them embodying the conditions of the problem, is based on ideas of Polya [Po.'71] about Heuristics. A problem-solving method is defined as a program schema operating on an abstract data type. The former computes the solution by the method, whereas the specification of the latter guarantees the total correctness of the program schema. Applying a problem-solving method to a problem consists of implementing the abstract data type underlying the former on the abstract data type of the latter.

These formulations were argued to capture the basic intuitions behind these vague, but fruitful, notions. A problem-solving strategy embodies some general knowledge about how to solve some classes of problems. This knowledge is represented in the corresponding problem-solving method in two parts : the

procedural aspects in the program schema and the declarative aspects in the specification of the underlying abstract data type.

The examples presented, though simple, illustrate the usefulness of these ideas. Regarding the application of a problem-solving method as an implementation of abstract data types suggests a methodology for problem solving, especially in program development. The example in section 5 also illustrates a key feature of these ideas, namely, they can be applied repeatedly in a stepwise manner, corresponding to successive refinements. This appears to be a crucial aspect both of problem solving in general and of the process of program development.

References

[En.'72] H. B. Enderton - A Mathematical Introduction to Logic. Academic Press, New York,1972.

[Gu.'77] J. V. Guttag - Abstract Data Types and the Development Data Structures,Comm. ACM 20(6),1977.

[H+S '78] E. Horowitz, S. Sahni - Fundamentals of Computer Algorithms. Computer Science Press, Potomac, 1978.

[Kn.'73] D. E. Knuth - The Art of Computer Programming, vol. 3 : Sorting and Searching. Addison-Wesley, Reading, 1973.

[L+Z '74] B. Liskov, S. Zilles - Programming with Abstract Data Types,SIGPLAN Notices 4(4), 1974 .

[Ma.'74] Z. Manna - The Mathematical Theory of Computation. McGraw-Hill, New York, 1974 .

[M+S+V '84] T. S. E. Maibaum, .M. R. Sadler, P. A. S. Veloso - Logical Specification and Implementation. 4th. Conf. on Foundations of Software Technology and Theoretical Computer Science, Bangalore,1984.

[M+V '85] T. S. E. Maibaum, P. A. S. Veloso - A Theory of Abstract Data Types for Program Development : bridging the gap ? Joint Conf. on Theory and Practice of Software Development (TAPSOFT) , Berlin, 1985.

[Ni.'82] N. J. Nilsson - Principles of Artificial Intelligence. Springer-Verlag, Berlin, 1982 .

[P+L '79] T. H. Pequeno, C. J. Lucena - An Approach for Data Type Specification and its Use in Program Verification, Inform. Proc. Letters 8 (2), 1979 .

[Po.'71] G. Polya - How to Solve it : a new aspect of the mathematical method. Princeton Univ. Press,Princeton, 1971 .

[So.'84] J. F. Sowa - Conceptual Structures : information processing in mind and machine. Addison-Wesley, Reading, 1984 .

[Ve.'80] P. A. S. Veloso - Divide-and-conquer via Data Types. Proc. VII Conf. Latinoamericana de Informática, Caracas, 1980.

[Ve.'84] P. A. S. Veloso - Outlines of a Mathematical Theory of General Problems.Philosophia Naturalis 23 (2/3), 1984 .

[V+P+M] P. A. S. Veloso, F. E. P. Pessoa, T. S. E. Maibaum - Teoria de Tipos Abstratos de Dados para Programação : um enfoque lógico. Proc. IX Conf. Latinoamericana de Informática, Lima, 1982 .

[V+V '81] P. A. S. Veloso, 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 Publ. Co., Washington, DC, 1981 .

[Wa.'84] C. F. E. M. Waga - Métodos para Resolução de Problemas. Dept. Informática, PUC-RJ, M.Sc. diss., 1984 .