The Algorithmic Specification Method of Abstract Data Types: An Overview

by

Jacques Loeckx

A 85/07

May 1985

Universität des Saarlandes
Fachrichtung 10.2 Informatik
D - 6600 Saarbrücken

THE ALGORITHMIC SPECIFICATION METHOD OF ABSTRACT DATA TYPES: AN OVERVIEW

Jacques Loeckx
Fachrichtung 10.2 Informatik
Universität des Saarlandes
D - 6600 Saarbrücken

1. INTRODUCTION

The practical relevance of abstract data types is twofold. First, their use may support the design of modular programs by stepwise refinement. Second, mechanical correctness proofs of these programs may become feasible in practice.

Classically, a data type is considered to be a multisorted algebra. This algebra consists of carrier sets - one for each sort - and of operations on these sets. Essentially, there exist two different specification methods. An *operational* specification [Sh 81, Li 81, NY 83] is embedded in an imperative programming language; the carrier sets are built up with the help of the data structures of this programming language and the operations are defined with the help of program pieces. An *algebraic* specification [GTW 78, GHM 78, BW 82, Re 80, EM 85, etc.] essentially consists of a set of equalities. The algebra thus defined is a special model of this set of equalities, generally the initial model.

A shortcoming of the operational specification method is its lack of abstraction. In fact, an operational specification constitutes an implementation (of a specification) rather than a specification. The main drawback of the algebraic specification method is related to its generality and, in particular, to its non-constructive nature. This leads to the difficult problems of consistency and completeness.

The specification method to be presented here has been called *algorithmic* to stress its constructive nature. In this method each (new) carrier set is defined by a (constructively defined) formal language; each (new) operation is defined by a recursive program. By its constructive nature the algorithmic specification method essentially avoids the problems of the algebraic specification method. On the other hand it differs from the operational specification method by being more "abstract". Being fundamentally different by their very definition the algebraic and algorithmic specification methods are nevertheless related. For instance, in many "simple" cases algorithmic specifications may be transformed into algebraic ones by mere rewriting. Other constructive specification methods may be found in [Ca 80, Kl 84].

In the frame of this paper it is not possible to present a complete description of the algorithmic specification method and of its applications. The interested reader is referred to [Lo 84, Lo 81 a, Lo 81 b, Lo 80, Le 85]. Instead, Section 2 explains the main ideas of the method. Section 3 shortly discusses the logic in which properties of the data types may be expressed and proved. Section 4 provides a glimpse of a specification language based on the algorithmic specification method and of its implementation.

2. THE SPECIFICATION METHOD

2.1 The introduction of new sorts

In order to introduce a new sort it is sufficient to define its carrier set and its operationa.

The carrier set is defined as a term language, i.e. as a particular formal language. In the case of Figure 1 the carrier set is the term language generated by the operation symbols characterized by the word *constructor*. The elements of this carrier set are words such as ε, App(ε,o), App(App(ε,4),2), etc.

The definition of the operations is different for the constructors and the other operation symbols. The interpretation of the constructors is the Herbrand interpretation. For instance, the value of the operation App for the arguments ε and o is the word App(ε,o). The other operation symbolsare defined as recursive programs (in the sense of [Ma 74, LS 84]). For instance, Insert appends an integer to a sequence provided the integer does not yet occur in this sequence; Delete deletes the rightmost occurrence of an integer; Memberof tests whether a (given) integer occurs in a sequence.

2.2 The operations *subset* and *quotient*

The method of Section 2.1 does not allow to specify "elaborate" sorts such as the sort consisting of (finite) sets (of integers). In fact, let us try to modify the specification of Figure 1, being understood that a term such as App(App(ε,o),1) now stands for the set {o,1}. Clearly, a term with "duplicates" such as

App(App(ε,o),o)

```
sort seq
signature
     constructor ε : → seq
     constructor App : seq × int → seq
     Insert : seq × int → seq
     Delete : seq × int → seq
     Memberof : seq × int → bool
     Subset : seq × seq → bool
operations
     Insert(s,i) ⇐ if Memberof(s,i) then s else App(s,i) fi
     Delete(s,i) ⇐ case s of
           ε : s
           App(s',i') : if i=i' then s' else App(Delete(s',i),i') fi
                                                                esac
     Memberof(s,i) ⇐ case s of
           ε : false
           App(s',i') : if i=i' then true else Memberof(s',i) fi
                                                                esac
     Subset(s1,s2) ⇐ case s1 of
           ε : true
           App(s1',i) : if Memberof(s2,i) then Subset(s1',s2)
                                            else false fi esac
```

FIGURE 1: A specification of the sort *seq*, which consists of finite sequences of integers

may not occur in the carrier set. Moreover, terms such as

$$\text{App}(\text{App}(\varepsilon,0),1)$$

and

$$\text{App}(\text{App}(\varepsilon,1),0)$$

have to stand for the same set. The first of these difficulties is solved by introducing an operation which defines a subset of the carrier set; an example of such an operation is the operation Nodup of Figure 2. The second difficulty is solved by introducing an equivalence relation such as Eq of Figure 2; the carrier set then consists of a set of equivalence classes.

It is well-known that such a *subset operation* or *quotient operation* yieldsan algebra only if the *closure condition* and the *congruence condition* respectively are fulfilled (see e.g. [GM 83]). Informally, the closure condition expresses that arguments from the subset lead to a value from the subset; the congruence condition expresses that equi-

```
signature
     Nodup : seq → bool
     Eq : seq × seq → bool
operations
     Nodup(s) ⇐ case s of
         ε : true
         App(s',i) : if Memberof(s',i) then false
                                  else Nodup(s') fi esac
     Eq(s1,s2) ⇐ if Subset(s1,s2) then Subset(s2,s1)
                                   else false fi
```

FIGURE 2: The specification of Figure 1 together with Nodup as subset operation and Eq as quotient operation defines the sort consisting of finite sets of integers. The operation App is <u>not</u> put at the disposal of the user of the specification, as it does not fulfil the closure condition.

valent arguments lead to equivalent values. These conditions have to be fulfilled by all operations put at the disposal of the user of the specification; they have to be proved once and for all by the designer of the specification.

3. THE LOGIC OF STRICT ALGEBRAS

The *logic of strict algebras* [Lo 84] allows to express and prove properties of the data types introduced by algorithmic specifications. The logic is similar to LCF (see [Mi 72 , LS 84]) and takes account of the fact that the operations constitute partial computable functions. An example of a formula referring to Figure 1 is:

$$\forall s \in seq,\ i \in int.\ \mathrm{Memberof}(s,i) = \underline{\mathrm{false}} \supset \mathrm{Delete}(s,i) = s$$

An important proof rule is structural induction on the carrier set [Lo 80, Lo 84]. For "non-trivial" properties it may be necessary to use fixpoint induction [Lo 81 b].

4. THE SPECIFICATION LANGUAGE OBSCURE

4.1 The language

The specification language OBSCURE [Le 85] is similar to CLEAR [Sa 84] but is based on the algorithmic specification method instead of the algebraic one. Essentially OBSCURE allows to construct an algebra from a given one.

OBSCURE contains among others three constructs which correspond to the extension described in Section 2.1, to the subset operation and to the quotient operation respectively. Moreover OBSCURE allows the specification and use of parameterized data types. Finally, it provides the possibility to define implementations of abstract data types.

4.2 The implementation

An implementation of OBSCURE is under construction at the university of Saarbrücken. It consists of a programming and a verification part.

The programming part supports the top-down development of programs by stepwise refinement. In particular, it generates the closure and congruence conditions which are to be proved.

The verification part allows interactive correctness proofs and bears similarities with the AFFIRM system [Mu 80, Th 79, Lo 80]. Essentially, the computer performs the formula manipulation and takes care of the administration; the user is responsible for the choice of the proof strategy. The readability of the intermediate results produced by the computer is given particular attention to.

REFERENCES

[BW 82] Broy, M., Wirsing, M., Partial abstract types, *Acta Inform.* 18 (1982), 47 - 64

[Ca 80] Cartwright, R., A constructive alternative to abstract data type definitions, Proc. 1980 LISP Conf., Stanford Univ. (1980), 46 - 55

[EM 85] Ehrig, H., Mahr, B., Fundamentals of Algebraic Specification, Springer-Verlag, 1985

[GHM 78] Guttag, J.V., Horowitz, E., Musser, D.R., Abstract data types and software validation. *Comm. ACM* 21 (Dec. 1978), 1048 - 1069

[GM 83] Goguen, J.A., Meseguer, J., *Initiality, Induction and Computability*, SRI-CSL Techn. Rep. 140, Stanford Research Institute, December 1983

[GTW 78] Goguen, J.A., Thatcher, J.W., Wagner, E.G., An initial algebra approach to the specification, correctness and implementation of abstract data types. *Current Trends in Programming Methodology IV* (Yeh, R., Ed.), Prentice-Hall, 1978, 80 - 149

[Kl 84] Klaeren, H.A., A constructive method for abstract algebraic software specification, *Theor. Comp. Sc.* 30, 139 - 204 (1984)

[Le 85] Lermen, C.W., OBSCURE, a specification language for algorithmic specifications, Intern. Rep., FR 10.2, Univ. Saarbrücken (in preparation)

[Li 81] Liskov, B., et al, *CLU Reference Manual*, LNCS 114, 1981

[Lo 80] Loeckx, J., Proving properties of algorithmic specifications of abstract data types in AFFIRM. AFFIRM-Memo-29-JL, USC-ISI, Marina del Rey, 1980

[Lo 81 a] Loeckx, J., Algorithmic specifications of abstract data types, Proc. ICALP 81, LNCS 115 (1981), 129 - 147

[Lo 81 b] Loeckx, J., Using abstract data types for the definition of programming languages and the verification of their compilers, Int. Rep. A 81/13, FB 10, Univ. Saarbrücken (1981)

[Lo 84] Loeckx, J., Algorithmic specifications: A constructive specification method for abstract data types, to appear in *TOPLAS*

[LS 84] Loeckx, J., Sieber, K., *The Foundations of Program Verification*, J. Wiley/Teubner-Verlag, New York/Stuttgart (1984)

[Ma 74] Manna, Z., *Mathematical Theory of Computation*, McGraw-Hill (1974)

[Mi 72] Milner, R., Implementation and application of Scott's logic for computable functions. Proc. ACM Conf. on Proving Assertions about Programs, SIGPLAN Notices 7 (Jan. 1972), 1 - 6

[Mu 80] Musser, D.R., Abstract data type specification in the AFFIRM System. *IEEE Trans. on Softw. Eng.* SE-6 (1980), 24 - 32

[NY 83] Nakajima, R., Yuasa, T., *The IOTA Programming System*, LNCS 160, 1983

[Re 80] Reichel, H., Initially-restricting algebraic theories, Proc. 9th FCS, LNCS 88, 460 - 473 (1980)

[Sa 84] Sannella, D., A set-theoretic semantics for CLEAR, *Acta Inform.* 21, 5, 443 - 472 (1984)

[Sh 81] Shaw, M. (ed.), *ALPHARD, Form and Content*, Springer-Verlag, 1981

[Th 79] Thompson, D.H. (Ed.), AFFIRM Reference Manual. Internal Report, USC-ISI, Marina del Rey (1979)