stichting
mathematisch
centrum

MC

AFDELING INFORMATICA
(DEPARTMENT OF COMPUTER SCIENCE)

IW 127/79 DECEMBER

J.A. BERGSTRA & J.V. TUCKER

ON THE ADEQUACY OF FINITE EQUATIONAL METHODS FOR DATA TYPE SPECIFICATION

Preprint

2e boerhaavestraat 49 amsterdam

Printed at the Mathematical Centre, 49, 2e Boerhaavestraat, Amsterdam.

The Mathematical Centre, founded the 11-th of February 1946, is a non-profit institution aiming at the promotion of pure mathematics and its applications. It is sponsored by the Netherlands Government through the Netherlands Organization for the Advancement of Pure Research (Z.W.O).

1980 Mathematics subject classification: 03D45, 03D80, 68B15

ACM-Computing Reviews-category: 4.34

On the adequacy of finite equational methods for data type specification*)

by

J.A. Bergstra**) & J.V. Tucker

ABSTRACT

We report on theoretical investigations into the adequacy of equational techniques for writing data type specification code pioneered in the literature of Programming Methodology.

KEY WORDS & PHRASES: *algebraic data types and data structures, equational specifications with hidden functions and with hidden sorts, computable and semicomputable algebras*

---

*) This report reprints notes which appeared in ACM-SIGPLAN Notices 14 (11) 1979, 13-18. It is not review.

**) Department of Computer Science, University of Leiden, Wassenaarseweg 80, Postbus 9512, 2300 RA LEIDEN, The Netherlands.

INTRODUCTION

The algebraic theory of data type specification, associated with the names J.V. GUTTAG, J.J. HORNING, S. ZILLES, B. LISKOV, M. MAJSTER and the ADJ GROUP, studies algebraic prescriptions (Σ,E) which are intended to explicitly define the operations Σ of a data type τ, by means of algebriac properties E, in any programming system or particular program in which τ might appear. A sound theoretical basis for this enterprise was provided in the initial algebra formalism of the ADJ GROUP [2] although the actual specification of even seemingly innocuous memory structures has proved problematical and occupied many of these pages since, for example, MAJSTER pointed out that a natural kind of traversable stack failed to possess a finite, equational specification [6] (see the most recent contribution of KAPUR [5] and the references there cited; also MAJSTER [7]). One result of these and other complications is a profusion of specification methods some apparently ad hoc, specific to particular examples.

Recently, in *SIGPLAN Notices*, March '79, S. KAMIN introduced a useful classification scheme to organise many of the methods so far applied and summarised what was known about their comparative powers, asking the question: *Given two methods* M,M' *is* M *more generally applicable than* M'? and, implicitly, the question: *Does a given method* M *define all the data types one wants*? Here we report on some investigations, [1], prompted by KAMIN's [4], specifically the theoretical and semantical *problem of adequacy*. This we carefully formulate and settle the fact that everything one could imagine wanting to specify can be done so by finite algebraic specifications. After recalling a little algebra, in section one, we describe our technical approach to the problem of adequacy, in sections two and three, and precisely state the results obtained in section four.

1. BACKGROUND MATERIAL

We address particularly those readers acquainted with the initial algebra methodology of the ADJ GROUP [2], and the useful survey KAMIN [4] though we recall most of what we need of his classification scheme. We consider data structures modelled as heterogeneous algebras in this way: a *data structure* A can be thought to consist of a finite family $A_1,\ldots,A_n$ of *component data domains* together with a finite family of operations of the form

$$\sigma^{\lambda,\mu} = \sigma^{\lambda_1,\ldots,\lambda_k,\mu}: A_{\lambda_1}x\ldots xA_{\lambda_k} \longrightarrow A_\mu$$

for some $k \in \omega$, the *natural numbers* and $\lambda_i,\mu \in \{1,\ldots,n\}$, $1 \leq i \leq k$; we use the terms data structure and algebra interchangably. The *signature* Σ_A of a

data structure A consists of a name, called a *sort*, for each of its domains and a standardised notation for each of its operations which names the sorts on which they are defined.

Data structures are to be specified by means of a signature Σ and a finite set of equations E over Σ where these equations are of two kinds. Let T_Σ be the Σ *term algebra* and $T_\Sigma[X_1,\ldots,X_n]$ the Σ *polynomial algebra* in the variables $X_1,\ldots,X_n$. An identity $t=t'$ is a *simple equation* over Σ if $t,t' \in T_\Sigma$ and is called a (*polynomial*) *equation* over Σ if $t,t' \in T_\Sigma[X_1,\ldots,X_n]$. Given a pair (Σ,E) one proceeds to construct a certain initial algebra $T_{\Sigma,E}$ as a factor algebra of T_Σ by the smallest congruence on T_Σ identifying all terms in T_Σ identified by E.

A has *finite, simple equational* specification (Σ,E) if E is a finite set of simple equations over Σ and $T_{\Sigma,E} \cong A$. Similarly, A has *finite, equational* spcification (Σ,E) if E is a finite set of polynomial equations over Σ and $T_{\Sigma,E} \cong A$. KAMIN abbreviates these (F,S,N) and (F,V,N) specifications respectively, where N stands for *no* hidden operations or hidden sorts, that is, $\Sigma = \Sigma_A$.

The (F,S,N) specifications are rather weak (see 4.1) while the (F,V,N) specifications are very powerful, admitting all kinds of complicated, non constructive structures, still they, nor the (F,C,N) specifications of ADJ[10], are adequate, so to consider hidden operators and structures is quite natural and, in any case, cannot be avoided; we leave these definitions until section four.

2. DATA TYPES AND DATA STRUCTURES

> *My own pet notion is that in the world of human thought generally, and in physical science particularly, the most important and most fruitful concepts are those to which it is impossible to attach a well-defined meaning.*
>
> H.A. KRAMERS (1947)

Whatever a data type τ is, we shall assume it is given a syntactical definition in any programming language or system, or particular program in which it appears. Moreover we assume it is structured in that it possesses various operations Σ_τ explicitly defined by a piece of program E_τ with the result that if a data type τ appears in a program P then the computations of P on various inputs will generate a class $K_\tau(P)$ of (in general, infinite) *data structures* A of type τ which completely determines the semantics of the type τ as it is defined in P by (Σ_τ,E_τ)*. The problem of algebraic specification is to design pairs (Σ,E) which put (Σ_τ,E_τ) into an algebraic normal form so that $K_{\Sigma,E}(P)$ represents sufficiently well the desired semantics $K_\tau(P)$. Obviously, the requirement that E be finite is highly desirable**.

A discussion of the adequacy of any method of data type specification rests inevitably on a sound delimitation of the semantical limitations of τ, and to present a convining theoretical statement on the adequacy of the

---

* We use data structure in the third sense of data type appearing in the list of D. GRIES [3,p.267].

** If E is not finite then it will contain a piece of program to generate sufficiently much of E as required.

finite, equational methods we shall choose an extremely general measure (from the practical point of view) of the acceptability of a data structure A, and address the hardest question of specifying its type when $K_\tau = \{A\}$: the semantics of τ is arbitrary but must be uniquely characterised. First we must remark on a basic algebraic property of data structures which may be a little confused in some of the literature.

Let A be a data structure. It is reasonable to suppose it is finitely generated in any computation in which it appears by initial values $a_1,\ldots,a_n$ which are either fixed by the type or are presented to the (specification of the) type as input. In either case the signature of any specification (Σ,E) for A, via its type, should carry names $x_1,\ldots,x_n$ for otherwise programming with the specification would not allow access to all of A. (Note that running a program scheme over an algebra A computes solely within the subalgebra of A finitely generated by its input from A.) This means that if a structure A finitely generated by $a_1,\ldots,a_n$ is considered acceptable as a data structure then one should ask for a specification for it in the form $(A,a_1,\ldots,a_n)$: specifying an algebra *prime* in the sense that it has no proper subalgebras or, equivalently, that it is generated by the constants named in its signature. This is implicit in the direct specifications of section one but unclear in the case of hidden function and hidden sort specifications, see section four.

3. COMPUTABLE AND SEMICOMPUTABLE DATA STRUCTURES

Our semantic measures of adequacy are invested in the concepts of *finite, computable* and *semicomputable algebras*. We define these latter two categories in a moment. Their formulations are based upon work of M.O. RABIN [9] and A.I. MAL'CEV [8] devoted to inventing a theory of computable algebraic systems and they represent a distinct improvement on other definitions, such as MAJSTER's definition of a computable data type in [7], because they are completely formal and give concepts which are isomorphism invariant: the hall-mark of genuine algebraic properties and indispensible in our treatment of the adequacy problem.

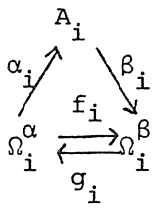
A data structure A is said to be *effectively presented* when corresponding to its family of component data domains $A_1,\ldots,A_n$ there are mutually disjoint recursive sets $\Omega_1,\ldots,\Omega_n$, $\Omega_i \subset \omega$ for $1 \le i \le n$, and surjections $\alpha_i: \Omega_i \to A_i$, $1 \le i \le n$, such that for each operation $\sigma^{\lambda,\mu}$ of A there is a recursive tracking function $\sigma_\alpha^{\lambda,\mu}$ which commutes the following diagram:

$$\begin{array}{ccc} A_{\lambda_1}\times\ldots\times A_{\lambda_k} & \longrightarrow & A_\mu \\ \uparrow{\scriptstyle \alpha_{\lambda_1}\times\ldots\times\alpha_{\lambda_k}} & & \uparrow{\scriptstyle \alpha_\mu} \\ \Omega_{\lambda_1}\times\ldots\times\Omega_{\lambda_k} & \longrightarrow & \Omega_\mu \end{array}$$

wherein $\alpha_{\lambda_1}\times\ldots\times\alpha_{\lambda_k}(x_{\lambda_1},\ldots,x_{\lambda_k}) = (\alpha_{\lambda_1}(x_{\lambda_1}),\ldots,\alpha_{\lambda_k}(x_{\lambda_k}))$.

Now, A is a *computable data structure* if for each $1 \le i \le n$ the relation $\equiv_{\alpha_i}$, defined on Ω_i by $x \equiv_{\alpha_i} y$ iff $\alpha_i(y)$ in A_i, is recursive. And A is a *semi computable data structure* if each of these $\equiv_{\alpha_i}$ is recursively enumerable.

Three facts are worthy of mention here: (i) every countable data structure possesses an effective coordinatisation, (ii) if A is computable, or semicomputable, and B is isomorphic to A, then B is computable, or semicomputable, and (iii) if A is a finitely generated data structure, computable or semicomputable, under *coordinatisations* $\alpha = (\Omega_i^{\alpha},\alpha_i)_{1\leq i\leq n}$ and $\beta = (\Omega_i^{\beta},\beta_i)_{1\leq i\leq n}$ then α and β are recursively equivalent in the sense that for each $1 \leq i \leq n$ there exist recursive functions f_i, g_i which commute the diagram



See MAL'CEV [8].

4. ADEQUACY

4.1. *Finite data structures*

It is not difficult to show that *if* A *is a finite algebra, finitely generated by* $a_1,\ldots,a_n$, *then* $(A,a_1,\ldots,a_n)$ *has a* (F,S,N) *specification.* And obviously some infinite structures have too: for example, $\underline{\omega} = (\omega;0,+1)$ where $+1(n) = n+1$ is specified by $(\{\underline{0},\underline{S}\},\emptyset)$ where $\underline{0}$ is a constant and $\underline{S}$ a unary function symbol. We offer this characterisation of those algebras possessing (F,S,N) specifications which, although slightly clumsy, does make the point that if a data structure has a (F,S,N) specification then it is necessarily computable and is almost isomorphic to a term algebra and so is essentially a tree structure.

Let Σ be a signature containing n sorts and let $c_1^i,\ldots,c_{m_i}^i$ be the m_i constants of sort i in Σ, $1 \leq i \leq n$. Let A be any algebra with signature Σ. The *signature diagram* of A is the set D(A) of all simple equations and inequalities of the following forms which are valid in A:

$$\sigma^{\lambda,\mu}(c_{i_1}^{\lambda_1},\ldots,c_{i_k}^{\lambda_k}) = c_i^{\mu}$$

$$\sigma^{\lambda,\mu}(c_{i_1}^{\lambda_1},\ldots,c_{i_k}^{\lambda_k}) \neq c_i^{\mu}$$

for $1 \leq \lambda_j,\mu \leq n$, $1 \leq i_j \leq m_{\lambda_j}$, $1 \leq j \leq k$ and each k-ary operation of Σ.

Let K(D(A)) be the class of all Σ algebras satisfying all the formulae in D(A).

THEOREM. *Let* A *be an algebra finitely generated by* $a_1,\ldots,a_n$. *Then the following are equivalent:*

(i) $(A,a_1,\ldots,a_n)$ *has a* (F,S,N) *specification;*

(ii) *there exist* $b_1,\ldots,b_m \in A$ *such that* $B = (A,a_1,\ldots,a_n,b_1,\ldots,b_m)$ *is initial in* K(D(B)).

4.2. *Computable data structures*

Since (F,V,N), and (F,C,N), specifications are inadequate for the definition of all computable data structures, in order to preserve the finiteness of specifications one is lead to consider specifications involving

hidden operators and even hidden sorts. The idea here is that the construction of the data structure A involves the construction of a data structure B having a (F,V,N) specification and which "contains" the data structure A and from which A is easily "removed". KAMIN identifies two distinct interpretations of containment/removal, to explain them we introduce these constructions: let A be an algebra and $\Sigma_A \supset \Sigma$, then $A|_\Sigma$ denotes the algebra with domains those of A named by the sorts of Σ and operations only those of A named in Σ;

$\langle A \rangle_\Sigma$ denotes the prime subalgebra of $A|_\Sigma$.

KAMIN [4,p.37] defines a data structure A to have a *finite, equational hidden function specification* (i) under the *usual interpretation* or (ii) under the *subalgebra interpretation* if there is a $\Sigma \supset \Sigma_A$, containing exactly the sorts of Σ_A, and a finite set of equations E over Σ such that (i) $T_{\Sigma,E}|_{\Sigma_A} \cong A$ or (ii) $\langle T_{\Sigma,E} \rangle_{\Sigma_A} \cong A$ respectively. His notation (F,V,HF) refers to hidden function specifications in the first sense.

In view of our discussion concerning primeness in section two, we might be expected to rely on the subalgebra interpretation, observe, however, that if A is prime then a specification of the usual kind is at the same time one of the subalgebra kind hence we design this concept:

A data structure A has a *finite, equational hidden enrichment specification* if there is a $\Sigma \supset \Sigma_A$, containing exactly the sorts of Σ_A, and a finite set of equations E over Σ such that $T_{\Sigma,E}|_{\Sigma_A} = \langle T_{\Sigma,E} \rangle_{\Sigma_A} \cong A$; this we abbreviate a (F,V,HE) specification.

This theorem shows that (F,V,HE) specifications are adequate for all data structures arising in Computer Science.

THEOREM. *Let* A *be a computable algebra finitely generated by* $a_1,\ldots,a_n$. *Then* $(A,a_1,\ldots,a_n)$ *has a* (F,V,HE) *specification.*

4.3. *Semicomputable data structures*

Let A be a finitely generated algebra. Then A is said to possess a *finite, equational hidden enrichment by sorts specification* if there is a $\Sigma \supset \Sigma_A$ and a finite set of equations E over T_Σ such that

$$T_{\Sigma,E}|_{\Sigma_A} = \langle T_{\Sigma,E} \rangle_{\Sigma_A} \cong A;$$

this we abbreviate by saying A has a (F,V,HES) specification.

THEOREM. *Let* A *be a semicomputable algebra finitely generated by* $a_1,\ldots,a_n$. *Then* $(A,a_1,\ldots,a_n)$ *has a* (F,V,HES) *specification.*

An open question is: *Does every finitely generated semicomputable algebra have a* (F,V,HE) *specification?*

REFERENCES

[1] J.A. BERGSTRA & J.V. TUCKER, Algebraic specifications of computable and semicomputable data structures, *Mathematical Centre, Computer Science Report*, Amsterdam, July 1979. (IW115)

[2] J.A. GOGUEN, J.W. THATCHER, E.G. WAGNER, An initial algebra approach to the specification, correctness and implementation of abstract data types, in R.T. Yeh (ed.) *Current trends in programming methodology. IV, Data structuring*, Prentice Hall, Englewood Cliffs, New Jersey, 1978.

[3] D. GRIES (ed.) *Programming methodology*, Springer-Verlag, New York, 1978.

[4] S. KAMIN, Some definitions for algebraic data type specifications, *SIGPLAN Notices* 14(3) (1979) 28-37.

[5] D. KAPUR, Specifications of Majster's Traversable Stack and Veloso's Traversable Stack, *SIGPLAN Notices* 14(5) (1979) 46-53.

[6] M.E. MAJSTER, Limits of the "algebraic" specification of abstract data types, *SIGPLAN Notices* 12(10) (1977) 37-42.

[7] _____ , Data types, abstract data types and their specification problem, *Theoretical Computer Science* 8 (1979) 89-127.

[8] A.I. MAL'CEV, Constructive algebras, I., *Russian Mathematical Surveys* 16 (1961) 77-129.

[9] M.O. RABIN, Computable algebra, general theory and the theory of computable fields, *Transactions American Mathematical Society* 95 (1960) 341-360.

[10] J.W. THATCHER, E.G. WAGNER, J.B. WRIGHT, Specification of abstract data types using conditional axioms, *IBM Research Report*, RC 6214, Yorktown Heights, 1976.