INFORMATION AND CONTROL **54**, 186–200 (1982)

The Completeness of the Algebraic Specification Methods for Computable Data Types

J. A. BERGSTRA

Department of Computer Science, Mathematical Centre, Kruislaan 413,
1098SJ, Amsterdam, The Netherlands

J. V. TUCKER

Department of Computer Studies, University of Leeds, Leeds LS2 9JT, England

The following fundamental theorem about the adequacy of the algebraic specification methods for data abstractions is proved. Let A be a data type with n subtypes. Then A is computable if, and only if, A possesses an equational specification, involving at most $3(n+1)$ hidden operators and $2(n+1)$ axioms, which defines it under initial and final algebra semantics simultaneously.

INTRODUCTION

Suppose you wish to define a data abstraction as a set of primitive operators Σ whose behaviour satisfies a set of algebraic axioms E. Then *initial* and *final algebra semantics* are two different, though natural, ways of settling on a unique meaning for the specification (Σ, E). As its semantics, they each assign to (Σ, E) a many-sorted algebra, unique up to isomorphism, from the class $\mathrm{ALG}(\Sigma, E)$ of all algebras of signature Σ satisfying the axioms in E. Seen from the syntax of the data type, initial algebra semantics insists that two syntactic operator expressions t, t' over Σ are semantically equivalent if, and only if, $t = t'$ can be *proved* from the axioms E. While final algebra semantics assumes t, t' to be semantically equivalent as long as $t = t'$ does not *contradict* the requirements in E. Here t, t' are called observationally or behaviourly equivalent as far as the axioms of E are concerned; or—as one says in the terminology of logic—$t = t'$ is *consistent* with E.

The two choices have been discussed in the literature on data abstraction with varying degrees of precision and approval. For example, equivalent forms of initial algebra semantics are clearly explained in early articles Zilles (1974, 1975), Liskov and Zilles (1975), and Goguen *et al.* (1975). But Guttag (1975), Guttag and Horning (1978) probably favour final algebra

0019-9958/82 $2.00

semantics: certainly Guttag and Horning (1978) contains a disclaimer about initial semantics and an approximate description of the objectives of the final algebra technique. An early rigorous account of final algebra semantics is Wand (1979) and other exact treatments of this far less well-understood alternative can be seen in Giarratana, Gimona, and Montanari (1976), Hornung and Raulefs (1980), Kamin (1980), Kapur (1980), the Munich Group, Broy *et al.* (1979) and Wirsing and Broy (1980), and our own articles Bergstra and Tucker (1980a, 1983).

Any evaluation of the methods depends on any number of specific questions about data types, of course. And, regrettably, no properly researched comparative study is yet available. The point of this paper is to settle one basic question about the completeness or adequacy of the two specification methods: *Can algebraic specifications under initial and/or final algebra semantics define all the data types one wants, at least in principle*? Recalling that a data type, or data abstraction, is modelled by a many-sorted algebra, finitely generated by elements named in its signature, the following theorem answers that in a fundamental *theoretical* sense one needs, and can rely on, both:

THEOREM. *Let A be an n-sorted algebra finitely generated by elements named in its signature Σ. Then the following are equivalent*:

(1) *A is computable.*

(2) *A possesses an algebraic specification, involving at most* $3(n+1)$ *auxiliary operators and* $2(n+1)$ *equations, which defines A under both its initial and final algebra semantics.*

That (2) implies (1) is a consequence of some straight-forward necessary conditions on the specification methods while the statement that (1) implies (2) is the hard-won answer to our adequacy question.

This paper belongs to a series of articles about the relative power of the various algebraic specification methods for data abstractions (Bergstra and Tucker (1979a, b, 1980a–d, 1983)). In particular, it is a companion to Bergstra and Tucker (1983), where we characterised a cosemicomputable data type A of signature Σ as a structure possessing an algebraic specification (Σ_0, E_0) using final algebra semantics. However, there we required E_0 to contain conditional equations, our bounds on the size of E_0 depended on the number of operators in Σ, and the arguments involved were sufficiently complicated to authorise our working with single-sorted structures only. The corresponding problem about semicomputable data types and initial algebra semantics remains open, but from the proof of the main theorem in Bergstra and Tucker (1983) one could extract a *second* specification of the same size which defines A initially *as long as A is* *computable*. Thus, our new theorem sharpens the corollary in Bergstra and Tucker (1983) in each of the four ways just mentioned and, more importantly, it has its own rather elegant proof which is significantly easier without the overheads of the main theorem in Bergstra and Tucker (1983).We think of our new theorem as a fundamental completeness theorem for the algebraic specification methods.

Readers of this paper are assumed to be well versed in the informal issues and technical foundations of the algebraic specification methods. For this Goguen *et al.* (1978) is essential, and Thatcher *et al.* (1979a, b) is recommended, but knowlege of our previous articles is not, strictly speaking, a prerequisite. A very detailed account of final algebra semantics and of the computability of data abstractions is contained in Bergstra and Tucker (1983) and so in what follows only the proof of our theorem will receive a generous exposition.

1. Data Types and Their Specification

Here we record notation and the technical ideas about data types and their specification which we shall need in proving our theorem. Let us repeat that the reader is supposed to be familiar with the basic principles of the algebraic specification method and to be used to working with the methods in Goguen *et al.* (1978). First we comment on the algebra needed.

Semantically, a *data type* or *data abstraction* is identified with (the isomorphism type of) a many-sorted algebra A finitely generated by elements named in its signature Σ. Such structures are called *minimal algebras* because they contain no proper subalgebras. Typically, the many-sorted algebra A consists of a finite family $A_1,\ldots, A_n$ of (*data*) *domains* or (*subtype*) *components* together with a finite collection of distinguished elements, and operators of the form

$$\sigma_A^{\lambda,\mu} = \sigma_A^{(\lambda_1,\ldots,\lambda_k)\mu}: A_{\lambda_1} \times \cdots \times A_{\lambda_k} \to A_\mu,$$

where λ_i, $\mu \in \{1,\ldots, n\}$. The signature Σ of A carries names for its domains, called *sorts*, and notations for the constants and operators; we will use numbers for sorts.

An algebra A is *finite* if each domain A_i is finite; and it is the *unit algebra* if every domain A_i is a singleton. We write the unit algebra as $\mathbb{1}$.

1.1. Lemma. *Let A and B be minimal algebras. Each homomorphism $A \to B$ is an epimorphism and if A and B are homomorphic images of one another, then they are isomorphic.*

If $\phi: A \to B$ is a homomorphism, then the relation $\equiv_\phi$ defined in A by $a \equiv_\phi b$ if, and only if, $\phi(a) = \phi(b)$ in B is a congruence. If ϕ identifies all of A, that is, the relation $\equiv_\phi$ is $A \times A$, then $B \cong \mathbb{1}$.

Next we turn to specifications and their semantics. A *specification* is a pair (Σ, E) composed of a signature Σ and a set of algebraic axioms E. These axioms will always be *equations* over Σ or *conditional equations* over Σ, the latter being formulae of the kind

$$e_1 \wedge \cdots \wedge e_k \to e,$$

where $e_1,..., e_k, e$ are equations over Σ.

If A satisfies the axioms E we call A an *E-algebra* and write $E \models A$. A second set of axioms E' is a *refinement* of E if $A \models E'$ implies $A \models E$; and we write this symbolically as $E' \models E$. If p is a formula provable in the equational calculus or conditional equation calculus from E we write $E \vdash p$.

The starting point for an understanding of initial and final algebra semantics is their description in terms of operator expressions over Σ, stated in the Introduction, rather than their category-theoretic formulations which give the semantics their names. In our proof, we shall use *only* the proof-theoretic characterisation of initial algebra semantics and *only* the category-theoretic definition of final algebra semantics. Since the latter semantics is not well known we will look at it in relation to initial semantics from the category theory point of view.

A specification (Σ, E) for a data type distinguishes the category $\mathrm{ALG}^*(\Sigma, E)$ of all minimal algebras of signature Σ satisfying the axioms E and all morphisms between them. And the semantics of a specification (Σ, E) is designed so as to pick out some algebra from $\mathrm{ALG}^*(\Sigma, E)$ as the *unique meaning* $\mathscr{M}(\Sigma, E)$, where the uniqueness of $\mathscr{M}(\Sigma, E)$ is measured up to algebraic isomorphism. Given a data type semantics (modelled by an algebra) A, a specification (Σ, E) can be said to *correctly define* the data type when $\mathscr{M}(\Sigma, E) \cong A$.

Seen from the category $\mathrm{ALG}^*(\Sigma, E)$, *initial algebra semantics* for algebraic specifications assigns as the meaning of (Σ, E) the initial algebra $I(\Sigma, E)$ in $\mathrm{ALG}^*(\Sigma, E)$; this $I(\Sigma, E)$ always exists and is unique up to isomorphism. On the other hand, *final algebra semantics* would like to pick out the final object from $\mathrm{ALG}^*(\Sigma, E)$ as the meaning of (Σ, E), but clearly this final algebra is in all cases the unit algebra $\mathbb{1} \in \mathrm{ALG}^*(\Sigma, E)$. (Notice $\mathbb{1}$ may not play an initial role in $\mathrm{ALG}^*(\Sigma, E)$ because of the minimality assumption.) Instead, final algebra semantics turns to the category $\mathrm{ALG}_0^*(\Sigma, E)$ which is simply $\mathrm{ALG}^*(\Sigma, E)$ with the unit algebra removed. Unfortunately, $\mathrm{ALG}_0^*(\Sigma, E)$ need not always possess a final object $F(\Sigma, E)$, but when it does this object is unique. Because of this asymmetry, defining and using the final algebra semantics of algebraic specifications can be a rather delicate matter when compared with the initial technique.

The equivalence of the category theory definitions and the logical definitions is represented by this lemma.

1.2. LEMMA. *Let (Σ, E) be a specification, and let t, t' be closed terms over Σ. Then*

(1) *$I(\Sigma, E) \models t = t'$ if, and only if, $E \vdash t = t'$.*

And, assuming $F(\Sigma, E)$ exists,

(2) *$F(\Sigma, E) \models t = t'$ if, and only if, $t = t'$ is consistent with E in the sense that there is some nonunit model $A \in \mathrm{ALG}(\Sigma, E)$, where $A \models t = t'$.*

Let $T(\Sigma)$ be the algebra of all terms over Σ. Let $T_I(\Sigma, E)$ denote the standard syntactic copy of $I(\Sigma, E)$, made by factoring $T(\Sigma)$ be the least E-congruence. The corresponding construction $T_F(\Sigma, E)$ for $F(\Sigma, E)$ can be found in Bergstra and Tucker (1983), but we shall not be needing it. We can now record the definitions governing the ways a specification characterises a data abstraction.

Let E be a set of equations or conditional equations over the signature Σ and let A be an algebra of signature Σ. The pair (Σ, E) is said to be an *equational* or a *conditional equation specification* of the algebra A with respect to (1) *initial algebra semantics* or (2) *final algebra semantics* if (1) $I(\Sigma, E) \cong A$ or (2) $F(\Sigma, E) \cong A$. When the set of axioms E is finite we speak of *finite specifications* with respect to these semantics.

Finally we must explain how we involve auxiliary or hidden functions in the semantics of specifications. Let A be an algebra of signature Σ_A and let Σ be a signature $\Sigma \subset \Sigma_A$. Then we mean by $A|_\Sigma$ the Σ-algebra whose domain is that of A and whose constants and operators are those of A named in Σ: the Σ-reduct of A; and by $\langle A \rangle_\Sigma$ the Σ-subalgebra of A generated by the constants and operators of A named in Σ viz. the smallest Σ-subalgebra of $A|_\Sigma$.

The following represents the two basic working definitions of specification theory in this paper.

Algebraic Specifications with Hidden Operators

The specification (Σ, E) is said to be a *finite equational* or a *conditional equation hidden enrichment specification* of the algebra A with respect to (1) *initial algebra semantics*, or (2) *final algebra semantics* if $\Sigma_A \subset \Sigma$, and E is a finite set of (conditional) equations over the (finite) signature Σ such that

$$I(\Sigma, E)|_{\Sigma_A} = \langle I(\Sigma, E) \rangle_{\Sigma_A} \cong A \tag{1}$$

or

$$F(\Sigma, E)|_{\Sigma_A} = \langle F(\Sigma, E) \rangle_{\Sigma_A} \cong A. \tag{2}$$

In this paper, all specifications involving hidden operators are made to define data types as described above.

2. Computable Data Types

A many-sorted algebra A is said to be *effectively presented* if corresponding to its component data domains $A_1,..., A_n$ there are mutually recursive sets $\Omega_1,..., \Omega_n$ of natural numbers and surjections $\alpha_i : \Omega_i \to A_i$ $(1 \leqslant i \leqslant n)$ such that for each operation $\sigma_A = \sigma_A^{\lambda,\mu}$ of A there is a recursive *tracking function* $\sigma_\alpha = \sigma_\alpha^{\lambda,\mu}$ which commutes the following diagram

$$\begin{tikzcd}
A_{\lambda_1} \times \cdots \times A_{\lambda_k} \arrow[r, "\sigma_A"] & A_\mu \\
\Omega_{\lambda_1} \times \cdots \times \Omega_{\lambda_k} \arrow[u, "\alpha_{\lambda_1} \times \cdots \times \alpha_{\lambda_k}"] \arrow[r, "\sigma_\alpha"] & \Omega_\mu \arrow[u, "\alpha_\mu"']
\end{tikzcd}$$

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

Now A is *computable* (*semicomputable* or *cosemicomputable*) if, in addition, the relations $\equiv_{\alpha_i}$ defined on Ω_i by

$$x \equiv_{\alpha_i} y \qquad \text{if, and only if,} \quad \alpha_i(x) = \alpha_i(y) \text{ in } A_i$$

are all recursive (r.e. or co-r.e.) for $1 \leqslant i \leqslant n$.

These three notions are the standard formal definitions of constructive algebraic structures and they derive from the work of Rabin (1960) and, in particular, Mal'cev (1961). Their special feature is that they make computability into a *finiteness condition* of algebra: *an isomorphism invariant possessed of all finite structures*. This lemma was proved in Bergstra and Tucker (1979a).

2.1. Representation Lemma. *Every computable many-sorted algebra A is isomorphic to a recursive algebra of numbers Ω each of whose numerical domains Ω_i is the set of natural numbers ω, or the set of the first m natural numbers ω_m, accordingly as the corresponding domain A_i is infinite, or finite of cardinality m.*

The following proposition draws attention to the fundamental difference between initial and final algebra semantics.

2.2. Basic Lemma. *Let (Σ, E) be a specification with E a recursively enumerable set of conditional equations. Then $I(\Sigma, E)$ is semicomputable and $F(\Sigma, E)$ is cosemicomputable, if it exists. In particular, if algebra A possesses* *an r.e. conditional equation hidden enrichment specification with respect to* (1) *initial algebra semantics, or* (2) *final algebra semantics, then* (1) *A is semicomputable, or* (2) *A is cosemicomputable. If A possesses such specifications with respect to both initial and final algebra semantics, then A is computable.*

The proof of Basic Lemma 2.2 is routine once the syntactic algebras $T_I(\Sigma, E)$ and $T_F(\Sigma, E)$ have been constructed. The theorem first appeared in Bergstra and Tucker (1980d) where we used it to find a data type which could not be specified by an r.e. set of algebraic axioms under initial algebra semantics. More examples can be found in Bergstra and Tucker (1983). The next section is given over to proving a strong converse of the last statement of the lemma.

3. Proof of the Theorem

Because of Basic Lemma 2.2, we have only to prove that statement (1) implies statement (2).

Let A be a computable many-sorted algebra finitely generated by elements named in its signature Σ. By the Representation Lemma 2.1, A can be identified with a recursive number algebra R each of whose domains is either ω or some finite initial segment ω_m of ω. It is sufficient to build an appropriate specification for R and this task we organise into some semantical constructions followed by some syntactical constructions.

First, we add *enumeration operators* to R to make a new algebra R_{e} with the special property that *any specification which defines R_{e} (and hence R) under initial algebra semantics will also define R_{e} (and hence R) under final algebra semantics*. Next, R_{e} is augmented with *arithmetical* and *conditional operators* to make a second algebra R_0. To complete the proof of the theorem it will be sufficient to provide a concise equational specification (Σ_0, E_0) which defines R_0 under initial algebra semantics: this is the objective of the syntactical constructions.

Semantical Constructions

Let D and $D_1, \ldots, D_{n-1}$ denote the n domains of R with $\mathrm{card}(D) \geqslant \mathrm{card}(D_\lambda)$ for $1 \leqslant \lambda \leqslant n-1$; call D the *principal domain* of R and notice that R is finite if, and only if, D is finite. To R we add the following constant and operators to form a new algebra R_{e} of signature Σ_{e} in which all domains can be accessed and enumerated from D.

Principal Enumeration Operators. For the principal domain D, add to R the element $0 \in D$ as a constant together with the map *succ*: $D \rightarrow D$ defined by $\mathrm{succ}(x) = x + 1$ if $D = \omega$ or by $\mathrm{succ}(x) = \min(x + 1, m)$ if $D = \omega_m$; and the map *pred*: $D \rightarrow D$ defined by $\mathrm{pred}(x) = x \dot{-} 1$.

Access Operators. For each nonprincipal domain D_λ $(1 \leqslant \lambda \leqslant n-1)$, add to R the map *fold*$_\lambda : D_\lambda \to D$ defined by $\text{fold}_\lambda(x) = x$; and the map *unfold*$_\lambda$: $D \to D_\lambda$ defined by $\text{unfold}_\lambda(x) = x$ if $D_\lambda = \omega$ or by $\text{unfold}_\lambda(x) = \min(x, m(\lambda))$ if $D = \omega_{m(\lambda)}$.

Clearly, R_e possesses 1 constant and $2 + 2(n-1) = 2n$ operators more than R, and $R_e|_\Sigma = \langle R_e \rangle_\Sigma = R$.

3.1. Lemma. *If B is a homomorphic image of R_e then either $B \cong R_e$ or $B \cong \mathbb{1}$.*

Proof. Let $\phi: R_e \to B$ be an epimorphism and suppose it is not injective; we show ϕ is trivial. There are two cases depending upon whether ϕ identifies distinct points in the principal domain or in some nonprincipal domain.

Case 1. Suppose $i, j \in D$ and $i \neq j$ but $\phi(i) = \phi(j)$. Let $i > j$ and write $i = \text{succ}^i(0)$ and $j = \text{succ}^j(0)$. Then $\text{succ}^i(0) \equiv_\phi \text{succ}^j(0)$ implies $\text{pred}^{i-1}(\text{succ}^i(0)) \equiv_\phi \text{pred}^{i-1}(\text{succ}^j(0))$ because $\equiv_\phi$ is a congruence. Thus, $\text{succ}(0) \equiv_\phi 0$ and, in fact,

$$0 \equiv_\phi \text{succ}(0) \equiv_\phi \text{succ}^2(0) \equiv_\phi \cdots$$

so all of D is identified in B under ϕ. Now, for any $x, y \in D_\lambda$ $(1 \leqslant \lambda \leqslant n-1)$ we can write $x = \text{unfold}_\lambda(x)$ and $y = \text{unfold}_\lambda(y)$. Since $x \equiv_\phi y$ in D we know that $\text{unfold}_\lambda(x) \equiv_\phi \text{unfold}_\lambda(y)$ in D_λ : that is, $x \equiv_\phi y$ in D_λ. Thus, all of D_λ is identified in B under ϕ and B is the unit algebra.

Case 2. Suppose $i, j \in D_\lambda$ and $i \neq j$ but $\phi(i) = \phi(j)$ for some $1 \leqslant \lambda \leqslant n-1$. Since $i \equiv_\phi j$ in D_λ we know that $\text{fold}_\lambda(i) \equiv_\phi \text{fold}_\lambda(j)$ in D because $\equiv_\phi$ is a congruence. Thus, two distinct elements of D are identified and we are in Case 1 again. ▮

3.2. Corollary. *If R_e is the initial object of some* $\text{ALG}(\Sigma_e, E_e)$, *then R_e is the final object of* $\text{ALG}_0^*(\Sigma_e, E_e)$, *too*; *in fact,* $\text{ALG}_0^*(\Sigma_e, E_e)$ *is merely the isomorphism type of R_e.*

The corollary is immediately deducible from Lemma 3.1. And it follows that if R_0 is an algebra of signature Σ_0 such that $\Sigma_e \subset \Sigma_0$ and

$$R_0|_{\Sigma_e} = \langle R_0 \rangle_{\Sigma_e} = R_e,$$

then *if R_0 is the initial object of some* $\text{ALG}(\Sigma_0, E_0)$, *then R_0 is the final object of* $\text{ALG}_0^*(\Sigma_0, E_0)$ *too*; *and again* $\text{ALG}_0^*(\Sigma_0, E_0)$ *contains only R_0 up to isomorphism.* This is simply because each Σ_0-homomorphism is necessarily a Σ_e-homomorphism.

Our aim is to create such an enrichment R_0 of R_e and give it a concise algebraic specification (Σ_0, E_0) without hidden functions. Clearly, we need only bother about initial algebra semantics in such circumstances.

We complete the semantical foundations of the proof by adding arithmetic to the principal domain in R_e, and a selection of conditional operators to both principal and nonprincipal domains in R_e.

Arithmetic Operators. For the principal domain D, add to R_e the map *add*: $D \times D \rightarrow D$ defined by $\text{add}(x,y) = x+y$ if $D = \omega$ or by $\text{add}(x,y) = \min(x+y, m)$ *if* $D = \omega_m$; and the map *mult*: $D \times D \rightarrow D$ defined by $\text{mult}(x,y) = x \cdot y$ if $D = \omega$ or by $\text{mult}(x,y) = \min(x \cdot y, m)$ if $D = \omega_m$.

Conditional Operators. For the principal domain D, add to R_e the maps $c: D \times D \times D \rightarrow D$ and $h: D \times D \times D \rightarrow D$ defined by

$$\begin{aligned} c(x,y,z) &= 0 && \text{if} \quad x = y \quad \text{and} \quad z = 0, \\ &= 1 && \text{otherwise}, \\ h(x,y,z) &= z && \text{if} \quad x = y, \\ &= 0 && \text{otherwise}. \end{aligned}$$

And for each nonprincipal domain D_λ $(1 \leqslant \lambda \leqslant n-1)$ add to R_e the map $h_\lambda: D \times D \times D_\lambda \rightarrow D$ defined by

$$\begin{aligned} h_\lambda(x,y,z) &= z && \text{if} \quad x = y, \\ &= 0 && \text{otherwise}. \end{aligned}$$

(Beware of the change of sort when dealing with h_λ!)

R_e augmented by these $4 + (n-1)$ operators results in the algebra R_0 of signature Σ_0. Clearly, R_0 posseses 1 constant and $3(n+1)$ operators more than R, and $R_0|_\Sigma = \langle R_0 \rangle_E = R$.

It now remains for us to build an algebraic specification (Σ_0, E_0) involving $2(n+1)$ equations and *no* hidden functions, which defines R_0 under initial algebra semantics. This task is divided into two stages: we begin by finding an algebraic specification (Σ_0, E_1) for R_0 which uses *conditional* equations of a special kind. The role of this (Σ_0, E_1) is to act as a template for a sequence of transformations which will compress E_1 into the required E_0.

Syntactical Constructions: The Template

Remember that R_0 is R augmented by the constant and operators

$$0, \quad \text{succ}, \quad \text{pred}, \quad \text{add}, \quad \text{mult}, \quad c, \quad h$$

on the principal domain D; and

$$\text{fold}_\lambda, \quad \text{unfold}_\lambda, \quad h_\lambda$$

for each other domain D_λ $(1 \leqslant \lambda \leqslant n-1)$. Let the signature Σ_0 of R_0 contain the following notations for the extra operators:

$$O,\ \mathrm{SUCC},\ \mathrm{PRED},\ \mathrm{ADD},\ \mathrm{MULT},\ D,\ H,\ \mathrm{FOLD}_\lambda,\ \mathrm{UNFOLD}_\lambda,\ H_\lambda.$$

3.3. LEMMA. *There is a finite algebraic specification* (Σ_0, E_1) *involving equations, and conditional equations of the form*

$$t = t' \to r = s,$$

where the premiss $t = t'$ *is an equation over the principal sort in* Σ_0, *which defines* R_0 *under initial algebra semantics.*

Proof. If R_0 is a finite algebra, then it is straightforward to make a specification by enumerating the graphs of the operations of R_0 and translating these relations into formal syntactical identities. Such a specification will satisfy the requirements of the lemma. (We had occasion to write out this observation in our study (Bergstra and Tucker (1980c).)

Assume R_0 is an infinite algebra so that, in particular, D is infinite. Here are the equations making up E_1. For enumerations and arithmetic on D we take

$$\begin{aligned} &\mathrm{PRED}(O) = O, && \mathrm{PRED}(\mathrm{SUCC}(X)) = X,\\ &\mathrm{ADD}(X, O) = X, && \mathrm{ADD}(X, \mathrm{SUCC}(Y)) = \mathrm{SUCC}(\mathrm{ADD}(X, Y)),\\ &\mathrm{MULT}(X, O) = O, && \mathrm{MULT}(X, \mathrm{SUCC}(Y)) = \mathrm{ADD}(X, \mathrm{MULT}(X, Y)). \end{aligned}$$

For the access operators we take

$$\mathrm{UNFOLD}_\lambda(\mathrm{FOLD}_\lambda(X^\lambda)) = X^\lambda$$

for each λ $(1 \leqslant \lambda \leqslant n-1)$; and for each unfolding of D into a *finite domain* $D_\lambda = \omega_{m(\lambda)}$ we use these special equations

$$\mathrm{UNFOLD}_\lambda(\mathrm{SUCC}^{m(\lambda)}(O)) = \mathrm{UNFOLD}_\lambda(\mathrm{SUCC}^{m(\lambda)+1}(O)).$$

The various conditional operators c, h, and h_λ and the original operators of R can all be treated in the same way.

Let $F \in \Sigma \cup \{C, H, H_\lambda\}$ name function $f\colon D_{\alpha(1)} \times \cdots \times D_{\alpha(k)} \to D_\beta$, where $\alpha(1),\ldots, \alpha(k)$, $\beta \in \{0, 1,\ldots, n-1\}$ and $D_0 = D$. For convenience in notations, let us introduce $\mathrm{unfold}_0\colon D \to D$, defined by $\mathrm{unfold}_0(x) = x$, and give it the syntactic name UNFOLD_0; now we can write

$$\begin{aligned} \mathrm{graph}(f) = \{(x_1,\ldots, x_k, y) \in D^{k+1}\colon & f(\mathrm{unfold}_{\alpha(1)}(x_1),\ldots, \mathrm{unfold}_{\alpha(k)}(x_k))\\ &= \mathrm{unfold}_\beta(y)\}. \end{aligned}$$

Remember that $D=\omega$ and notice that graph(f) is a recursively enumerable set.

Using Matijacevič's Diophantine theorem (see Manin, 1977) one can find polynomials p_f and q_f in variables $X=(X_1,\dots,X_k)$, Y and $Z=(Z_1,\dots,Z_l)$ such that

$$\text{graph }(f)=\{(x,y)\in\omega^k\times\omega\colon \exists z\in\omega^2\cdot[p_f(x,y,z)=q_f(x,y,z)]\}.$$

Let P_f and Q_f be formal translations of p_f and q_f to polynomials over the enumeration and arithmetic operator names $\{O$, SUCC, PRED, ADD, MULT$\}$. Now we take the following conditional equations to govern F:

$$\begin{aligned}P_f(X,Y,Z)=Q_f(X,Y,Z)\to F(\mathrm{UNFOLD}_{\alpha(1)}(X_1),\dots,\mathrm{UNFOLD}_{\alpha(k)}(X_k))\\ =\mathrm{UNFOLD}_\beta(Y).\end{aligned}$$

To complete the construction of E_1 it remains to consider the constants of Σ. If $\underline{c}\in\Sigma$ is a constant of the principal sort naming element $c\in D$, then take

$$\underline{c}=\mathrm{SUCC}^c(O).$$

If $\underline{c}\in\Sigma$ is a constant of a nonprincipal sort naming $c\in D_\lambda$, then take

$$\underline{c}=\mathrm{UNFOLD}_\lambda(\mathrm{SUCC}^c(O)).$$

Clearly $R_0\models E_1$ and by initiality there is an epimorphism $T_I(\Sigma_0,E_1)\to R_0$, but one needs to give the reverse map $R_0\to T_I(\Sigma_0,E_1)$ in order to prove $T_I(\Sigma_0,E_1)\cong R_0$ (Lemma 1.1). The inverse $\Phi\colon R_0\to T_I(\Sigma_0,E_1)$ is the family of maps $(\phi,\phi_1,\dots,\phi_{n-1})$ defined by

$$\begin{aligned}\phi(x)&=[\mathrm{SUCC}^x(O)] &&\text{for}\quad x\in D,\\ \phi_\lambda(x)&=[\mathrm{UNFOLD}_\lambda(\mathrm{SUCC}^x(O))] &&\text{for}\quad x\in D_\lambda,\end{aligned}$$

where $1\leqslant\lambda\leqslant n-1$ and $[t]$ denotes the equivalence class of terms determined by $t\in T(\Sigma_0)$ under the congruence $\equiv_{E_1}$. The proof that this Φ is a homomorphism is a lengthy exercise which is entirely routine for any reader with some experience in many-sorted algebra: we take the liberty of omitting it, leaving the reader to consult some of our earlier articles such as Bergstra and Tucker (1979) and (1980a, b) if necessary.

Syntactical Constructions: Compression

The specification (Σ_0,E_1) is not particularly concise: if R_0 is finite, then the number $|E_1|$ of algebraic axioms in E_1 is comparable with the cardinality $|R_0|$ of R_0; and if R_0 is infinite, then $|E_1|$ is a function of $|\Sigma_0|$ and, hence, of $|\Sigma|$. The compression of E_1 is based upon this simple, but important, tool:

3.4. REFINEMENT LEMMA. *Let (Σ, E) be an algebraic specification for some data type A and assume $I(\Sigma, E) \cong A$. Suppose (Σ, E') is another algebraic specification such that*

(i) $E' \models E$

and

(ii) $A \models E'$.

Then $I(\Sigma, E') \cong A$.

Proof. By hypothesis (ii), A is an E'-algebra and so there is an epimorphism $I(\Sigma, E') \to A$. On the other hand, hypothesis (i) implies $I(\Sigma, E')$ is an E-algebra and so initiality again implies there is an epimorphism $A \cong I(\Sigma, E) \to I(\Sigma, E')$. By Lemma 1.1, $I(\Sigma, E') \cong A$. ∎

Starting with E_1, we shall generate a sequence of refined specifications for R_0,

$$E_5 \models E_4 \models E_3 \models E_2 \models E_1$$

by replacing one axiomatisation by another and checking conditions (i) and (ii) of the Refinement Lemma 3.4.

First Step

For purely technical reasons, the first refinement of E_1 leads to a set of equations E_2. If R_0 is finite then set $E_2 = E_1$. If R_0 is infinite, then let E_2 contain all the *equations* in E_1 together with the n new equations

$$H(X, X, Z) = Z, \qquad H_\lambda(X, X, Z^\lambda) = Z^\lambda,$$

where Z^λ is a variable of sort λ and $1 \leqslant \lambda \leqslant n-1$. And now replace each conditional equation of the form

$$t = t' \to r = s \qquad \text{or} \qquad t = t' \to r^\lambda = s^\lambda$$

in E_1 by the equation

$$H(t, t', r) = H(t, t', s) \qquad \text{or} \qquad H_\lambda(t, t', r^\lambda) = H_\lambda(t, t', s^\lambda),$$

respectively. This is all of E_2, and clearly $E_2 \models E_1$ and $R_0 \models E_2$.

Second Step

From E_2 we make a new axiomatisation E_3 with the special feature that most formulae are equations which govern the behavior of the principal domain and those formulae which remain are the simple conditional equations

$$\mathrm{FOLD}_\lambda(X^\lambda) = \mathrm{FOLD}_\lambda(Y^\lambda) \to X^\lambda = Y^\lambda.$$

The set E_3 contains all those equations in E_2 *over the pincipal sort*; and each equation $r^\lambda = s^\lambda$ in E_2 over sort λ $(1 \leqslant \lambda \leqslant n-1)$ is replaced by the equation

$$\mathrm{FOLD}_\lambda(r^\lambda) = \mathrm{FOLD}_\lambda(s^\lambda).$$

Adding the $n-1$ simple conditional equations completes E_3 and it is clear that $E_3 \vDash E_2$ and $R_0 \vDash E_3$.

Third Step

From E_3 we make a concise axiomatisation E_4 which involves 1 equation and $n+1$ conditional equations. The set E_4 contains the $n-1$ simple conditional equations of E_3 and, in addition, these two new conditionals

$$C(X, Y, Z) = O \to X = Y, \qquad (\mathrm{ce}_1)$$

$$C(X, Y, Z) = O \to Z = O. \qquad (\mathrm{ce}_2)$$

Thus to complete E_4 it remains for us to construct one *master equation.*

Let $\{t_i = t'_i \colon 1 \leqslant i \leqslant l\}$ be an enumeration of all the equations in E_3; as we know, these are equations over the principal sort. Inductively define a master polynomial M by

$$M_0 = O, \qquad M_{i+1} = C(t_{i+1}, t'_{i+1}, M_i),$$

for $0 \leqslant i \leqslant l-1$ and set $M = M_l$. The master equation is simply

$$M = O. \qquad (\mathrm{me})$$

Now to verify that $E_4 \vDash E_3$ and $R_0 \vDash E_4$ one checks with induction that for each i

$$\{\mathrm{ce}_1, \mathrm{ce}_2, M_{i+1} = 0\} \vDash M_i = O \qquad \text{and} \qquad \{\mathrm{ce}_1, \mathrm{ce}_2, M_{i+1} = O\} \vDash t_i = t'_i$$

and that $\quad R_0 \vDash M_i = O.$

Last Step

The last refinement step turns E_4 into a set of $2(n+1)$ equations and this set E_5 is the axiomatisation E_0 required in the theorem. The set E_5 contains the master equation (me) of E_4, but the pair of conditional equations

$$C(X, Y, Z) = O \rightarrow X = Y,$$

$$C(X, Y, Z) = O \rightarrow Z = O,$$

is replaced by the triple of equations

$$H(X, X, Z) = Z,$$

$$H(C(X, Y, Z), O, X) = H(C(X, Y, Z), O, Y),$$

$$H(C(X, Y, Z), O, Z) = H(C(X, Y, Z), O, O).$$

And, instead of the $n-1$ conditional equations,

$$\mathrm{FOLD}_\lambda(X^\lambda) = \mathrm{FOLD}_\lambda(Y^\lambda) \rightarrow X^\lambda = Y^\lambda$$

in E_4, the set E_5 contains the $2(n-1)$ equations

$$H_\lambda(X, X, Z^\lambda) = Z^\lambda,$$

$$H_\lambda(\mathrm{FOLD}_\lambda(X^\lambda), \mathrm{FOLD}_\lambda(Y^\lambda), X^\lambda) = H_\lambda(\mathrm{FOLD}_\lambda(X^\lambda), \mathrm{FOLD}_\lambda(Y^\lambda), Y^\lambda).$$

Clearly, $|E_5| = 4 + 2(n-1) = 2(n+1)$ and it is straightforward to check that $E_5 \models E_4$ and $R_0 \models E_5$. Thus, taking $E_0 = E_5$ we have the concise initial and final semantics specification (Σ_0, E_0) of R_0 which is a hidden function specification of R under both initial and final algebra semantics. ∎

References

1. Bergstra, J. A. and Tucker, J. V. (1979a), "Algebraic Specifications of Computable and Semicomputable Data Structures," Mathematical Centre, Department of Computer Science Research Report IW 115, Amsterdam.
2. Bergstra, J. A. and Tucker, J. V. (1979b), On the adequacy of finite equational methods for data type specification, ACM-SIGPLAN *Notices* **14** (11), 13–18.
3. Bergstra, J. A. and Tucker, J. V. (1980a), A characterisation of computable data types by means of a finite, equational specification method, *in* "Automata, Languages, and Programming, Seventh Colloquium, Noordwijkerhout" (J. W. de Bakker and J. van Leeuwen, Eds.), pp. 76–90, Springer-Verlag, Berlin.
4. Bergstra, J. A. and Tucker, J. V. (1980b), "Equational Specifications for Computable Data Types: Six Hidden Functions Suffice and Other Sufficiency Bounds," Mathematical Centre, Department of Computer Science Research Report IW 128, Amsterdam.
5. Bergstra, J. A. and Tucker, J. V. (1980c), "On bounds for the Specification of Finite Data Types by Means of Equations and Conditional Equations," Mathematical Centre, Department of Computer Science Research Report IW 131, Amsterdam.
6. Bergstra, J. A. and Tucker, J. V. (1980d), A natural data type with a finite equational final semantics specification but no effective equational initial semantics specification, *Bull. European Assoc. Theoret. Comput. Sci.* **11**, 23–33.
7. Bergstra, J. A. and Tucker, J. V. (1983), Initial and final algebra semantics for data type specifications: Two characterisation theorems, *SIAM J. Comput.* **12**, 366–387.
8. Broy, M., Dosch, W., Partsch, H., Pepper, P., and Wirsing, M. (1979), Existential

quantifiers in abstract data types, *in* "Automata, Languages, and Programming, Sixth Colloquium, Graz" (H. Maurer, Ed.), pp. 72–87, Springer-Verlag, Berlin.

9. Goguen, J. A., Thatcher, J. W., Wagner, E. G., and Wright, J. B. (1975), Abstract data types as initial algebras and correctness of data representations, *in* "Proceedings ACM Conference on Computer Graphics, Pattern Recognition, and Data Structure," pp. 89–93, Assoc. Comput. Mach., New York.
10. Goguen, J. A., Thatcher, J. W., and Wagner, E. G. (1978), An initial algebra approach to the specification, correctness and implementation of abstract data types, *in* "Current Trends in Programming Methodology IV, Data Structuring" (R.T. Yeh, Ed.) pp. 80–149, Prentice-Hall, Englewood Cliffs, N.J.
11. Guttag, J. V. (1975), "The Specification and Application to Programming of Abstract Data Types," Ph.D thesis, Univ. of Toronto.
12. Guttag, J. V. and Horning, J. J. (1978) The algebraic specification of abstract data types, *Acta Inform.* **10**, 27–52.
13. Hoare, C. A. R. (1969), An axiomatic basis for computer programming, *Comm. ACM* **12**, 576–580.
14. Hornung, G. and Raulefs, P. (1980), Terminal algebra semantics and retractions for abstract data types, *in* "Automata, Languages, and Programming, Seventh Colloquium, Noordwijkerhout" (J. W. de Bakker and J. van Leeuwen, Eds.), pp. 310–325, Springer-Verlag, Berlin.
15. Kamin, S (1980) Final data type specifications: A new data-type specification method, *in* "Seventh ACM Principles of Programming Languages Conference, Las Vegas," pp. 131–138, Assoc. Comput. Mach., New York.
16. Kapur, D. (1980), "Towards a Theory for Abstract Data Types," MIT/LCS/TR-237, Cambridge, Mass.
17. Liskov, B. and Zilles, S. (1975), Specification techniques for data abstractions, *IEEE Trans. Software Engrg.* **1**, 7-19.
18. Mal'cev, A. I. (1961), Constructive algebras. I. *Russian Math. Surveys*, **16**, 77–129.
19. Manin, Y. (1977), "A Course in Mathematical Logic," Springer-Verlag, New York.
20. Rabin, M. O. (1960), Computable algebra, general theory and the theory of computable fields, *Trans. Amer. Math. Soc.* **95**, 341–360.
21. Thatcher, J. W., Wagner, E.G., and Wright, J. B. (1979a), "Specification of Abstract Data Types Using Conditional Axioms," IBM Research Report RC 6214, Yorktown Heights.
22. Thatcher, J. W., Wagner, E. G., and Wright, J. B. (1979b), "Data Type Specification: Parametrization and the Power of Specification Techniques," IBM Research Report RC 7757, Yorktown Heights.
23. Wand, M. (1979), Final algebra semantics and data type extensions, *J. Comput. Systems Sci.* **19**, 27–44.
24. van Wijngaarden, A. (1966), Numerical analysis as an independent science, *BIT* **6**, 66–81.
25. Wirsing, M. and Broy, M. (1980), Abstract data types as lattices of fintely generated models, *in* "Mathematical Foundations of Computer Science, Eight Symposium Rydzyna," Springer-Verlag, Berlin.
26. Zilles, S. (1974), "Algebraic Specification of Data Types," Project MAC Progress Report 11, M.I.T., Cambridge, Mass.
27. Zilles, S. (1975), "An Introduction to Data Algebras," working paper, IBM Research Laboratory, San José, Calif.
28. Giarratana, V., Gimona, F., and Montanari, U. (1976), Observability concepts in abstract data type specification, *in* "Mathematical Foundations of Computer Science" (A. Mazurkievicz, Ed.), Lecture Notes in Computer Science, No. 45, Springer-Verlag, Berlin.