Centrum voor Wiskunde en Informatica
Centre for Mathematics and Computer Science

J.A. Bergstra, J.V. Tucker

Algebraic specifications of computable and semicomputable datatypes

Computer Science/Department of Software Technology   Report CS-R8619   May

The Centre for Mathematics and Computer Science is a research institute of the Stichting Mathematisch Centrum, which was founded on February 11, 1946, as a nonprofit institution aiming at the promotion of mathematics, computer science, and their applications. It is sponsored by the Dutch Government through the Netherlands Organization for the Advancement of Pure Research (Z.W.O.).

Algebraic Specifications of

Computable and Semicomputable Datatypes

J.A. Bergstra

Department of Computer Science, University of Amsterdam,
Kruislaan 409, 1098 SJ Amsterdam, The Netherlands.

Department of Philosophy, University of Utrecht,
Heidelberglaan 2, 3584 CS Utrecht, The Netherlands.

J.V. Tucker

Department of Computer Science, University of Leeds,
Leeds LS29JT Britain.

An extensive survey is given of the properties of various specification mechanisms based on initial algebra semantics.

1980 Mathematics Subject Classifications: 68B10 [**Software**]: Analysis of Programs - *Semantics.*
1986 CR Categories: D.2.0 [**Software Engineering**]: Requirements/Specifications - *Languages*; D.2.2 [**Software Engineering**]: Tools and Techniques - *Modules and Interfaces*; D.3.3 [**Programming Languages**]: Languages Constructs - *Modules*; F.3.2 [**Logics and Meanings of Programs**]: Semantics of Programming Languages - *Algebraic Approaches to Semantics.*
Key Words & Phrases: algebraic specification, initial algebra semantics, computable algebra, hidden functions.
Note: This report will be submitted for publication elsewhere. It is a revised and substantially extended version of the report MC/IW115 ([4] in the references).
J.A. Bergstra acknowledges partial support obtained from ESPRIT project no 432 METEOR.

69D22 69D43, 69F32,

Report CS-R8619
Centre for Mathematics and Computer Science
P.O. Box 4079, 1009 AB Amsterdam, The Netherlands

CONTENTS

INTRODUCTION

BACKGROUND Axiomatic methods for data type specification arise from the idea that a data type D in a program language L, or program P, should be formally characterised in L, or P, as a collection Σ of operators which have properties defined by a set E of axioms. The axiomatic specification (Σ,E) is meant to be a contract in which Σ settles the syntactic structure of D and E guarantees a set of features common to all implementations of D. Algebraic specification methods are the simplest of the axiomatic methods in so far as they use the simplest axioms which are algebraic formulae such as equational laws.

Axiomatic data type specifications were first seen in A. van Wijngaarden's study of computer real arithmetic [57], but the full extent of their rôle for general user-defined types emerged later, through the work of C.A.R. Hoare on program specification and correctness [25,26,27] and D. Parnas on modularisation [43,44]. The algebraic specification methods originate in B. Liskov and S. Zilles [34], S. Zilles [59], J.V. Guttag [21] and ADJ [17]. Simple, elegant, and ideally matched to an algebraic view of the semantics of data types, the algebraic specification methods have proved to be a versatile tool for thinking about problems to do with data in the design and implementation of programming languages : see Wulf [58] and the bibliography Kutzler and Lichtenberger [29].

But these investigations, with their diverse programming objectives, have not easily grown into a *theory* of algebraic data type specification. The subject has been made with widely varying standards of conceptual precision and mathematical rigour, and has been troubled by technical problems of an algebraic nature. One thinks of the literature generated by M. Majster's transversable stack [37] which fails to have the much favoured finite equational specification. This important observation signalled a growth to profusion of algebraic specification techniques, many informal and defective, some ad hoc, designed for particular examples.

CLASSIFICATION PROGRAMME The purpose of this paper is to concisely review the mathematical basis of the algebraic approach to data type specification, and organise a proper mathematical analysis and classification of the algebraic specification methods that gives technical insight into the methods, and a theoretical assessment of their scope and limits. We will concentrate on *equational* and *conditional equational specifications, with* *and without hidden operators, using initial algebra semantics,* as developed by the ADJ Group for data types with total operators; see ADJ [17,18,51,52]. However, the tools and techniques can be used to extend the classification programme to include other specification methods.

In Sections 1 and 2 we carefully describe the syntactic and semantic structure of an algebraic specification technique. This leads us to a taxonomy of 27 specification methods : 9 not involving hidden machinery, 9 allowing hidden operators, and 9 allowing hidden types and operators. For these methods we formulate comparison questions of the form : *Given two algebraic specification methods* M *and* M', *is* M *more generally applicable or more powerful than* M', *are they equivalent, or are they disparate in their powers of definition?* In the course of the paper, we completely answer such questions for all the methods not allowing hidden machinery, and we almost complete the classification of the other techniques. The situation is summarised in Figures A and B in Sections 1 and 2.

In making the classification, where possible, we survey relevant information and results existing in the data types literature, and in the mathematical literature, but usually we prove or reprove what we need here. For example, in Section 4, we prove in detail that the simple numerical structure

$$(\{0,1,\ldots\};\ 0,\ x+1,\ x^2)$$

cannot be specified using finitely many equations and initial algebra semantics, unless auxiliary or hidden operators are permitted. Also, in Section 4, we prove in detail that the simple structure

$$(\{0,1,\ldots\},\{true,false\};\ 0,\ x+1,\ p,\ true,\ false),$$

where $p:\{0,1,\ldots\} \to \{true,false\}$ is the characteristic function of the prime numbers, cannot be specified using finitely many conditional equations and initial algebra semantics, but it can be given a specification using finitely many equations and auxiliary functions. These results are related to work on the role of hidden operations by ADJ [52] and Majster [37,38]. Other counter-examples can be found in Section 6.

In Section 3 we carefully define the nature of an effectively calculable data type in terms of *computable* and *semicomputable many-sorted algebras.* This leads to the concepts of *soundness, adequacy* and *completeness* for algebraic specification methods and, in particular, to adequacy and completeness questions of the form : *Can the specification method* M *define all, and only, the data types one wants, at least in principle?*

In Section 5, we prove the following adequacy theorem. *Any computable data type* A *can be algebraically specified by a finite set* E *of equations involving a finite set* Σ *of operators, some external to* A, *using initial algebra semantics for* (Σ,E) (Theorem 5.1). Such specifications can define semicomputable but non-computable types, however. Using the adequacy result, we are able to prove the following completeness theorem (Theorem 5.3).

THEOREM *A data type* A *is semicomputable if, and only if, it can be algebraically specified by a finite set* E *of equations, involving a finite set* Σ *of operators and data domains external to* A, *using initial algebra semantics for* (Σ,E).

The question as to whether or not hidden sorts are necessary for the finite specification of the semicomputable data types if an important open problem (Open Problem 3.15).

The need for a systematic and rigorous survey seems to have been first recognised by S. Kamin whose admirable notes [30] summarised specification techniques, associated with initial algebra semantics, and posed a number of questions about the differences between them. Answers to those questions can be found here along with commentary which settles some other technical matters raised in [30] (hidden function mechanisms; universality).

An objective of this paper is to serve a variety of readers as an essentially self-contained and reliable compendium of theoretical facts about specifications. Part of our material may seem familiar to some readers, but it is a fact that no theorem is given here which has an adequate statement and/or proof elsewhere. For example, our account of the adequacy and completeness theorems and problems, stated above, contradicts a popular and mistaken idea, originating in Guttag [21], that the adequacy of the algebraic specification method is evident from the equational definition of the partial recursive functions.

FURTHER WORK AND PREREQUISITIES This paper is a cornerstone for a series of articles [3-11] which further develops the classification project according to the principles seen here; in particular, it is a second edition of [4]. Among the subjects considered are : implementing equational specifications as rewrite systems and the completeness of the method for computable data types [5]; the size of algebraic specifications and adequacy theorems for computable data types [6,7]; proving specified programs using data type specifications [9]; completeness of methods based on final algebra semantics for the cosemicomputable data types [8,10]; completeness of methods based initial and final algebra semantics for computable data types [11]. See the Concluding Remarks for further comments.

We presume the reader is familiar with the informal issues and basic mathematical ideas about algebraic specifications, for which we follow and recommend ADJ [18,51,52]. The papers Kamin [30] and Majster [37,38] are also useful to have to hand. In addition we use some basic results from recursive function theory for which we recommend Mal'cev [39] or Rogers [47]. The reader will also find the paper Meseguer and Goguen [42] of value in seeing our work in a broad context.

ACKNOWLEDGEMENTS We are particularly indebited to S. Kamin, J.W. Thatcher, and an anonymous referee, for useful criticism of the first edition of this paper. We thank Ms. Judith Thursby for typing this manuscript.

1. ALGEBRAIC SPECIFICATION METHODS

In this section and the next we shall survey the mathematical foundations of the algebraic specification methods for data types in order to establish notation and terminology, and, in particular, to explain in detail the classification scheme for the methods. A number of subjects require commentaries : axiomatic specifications; algebraic specifications and their algebraic semantics; the use of hidden or auxiliary operators and sorts in specifications; but we begin with a discussion of the concept of an abstract data type.

ABSTRACT DATA TYPES. The term *data type* has many informal usages, and few precise definitions, in the literature about programming languages and methodology. For instance, D. Gries lists seven interpretations in his editorial notes in [20] (pp. 263-268) and all of them can be found supporting rôles to play in the subject of abstract data types and their specification. There is, however, an exact meaning for the term *abstract data type* which is invariably used (often implicitly) in work on algebraic specification methods. The mathematical defintion is essentially due to the ADJ Group and appears in [17] although its essential features are more carefully explained in [18]. We will quickly reconstruct the definition, noting any correspondences between our technical vocabulary and the usages in Gries' list.

First, consider a data type D whose *syntactic* structure is determined by a list of names for different kinds of data (for use in variable declarations) and lists of notations for distinguished data and basic operations (for use in assignments and in tests for control constructs). These items we call *sort, constant* and *operator symbols,* respectively, and the union Σ of the lists we refer to as the *signature* of D. (In Gries' notes, the first interpretation of type is restricted to signatures.)

What makes such a data type D an "abstract" data type is a property of its semantics namely, *the semantics of* D *is defined quite independently of how data and operations are to be represented in implementations.* This criterion is made precise via the semantical concept of a *data structure* which formalises the third, and most popular, informal description in Gries' list (and subsumes the second).

Throughout this paper we assume that every signature permits at least one closed term for each sort.

A *data structure* is a finitary many-sorted algebra which is minimal in a sense to be defined below. Thus, a data structure A consists of a finite family $A_1,\ldots,A_n$ of sets, called *component data domains,* together with a finite list of elements of these sets, and a finite family of (total) functions of the form

$$\sigma_A^{\lambda\,\mu} : A_{\lambda_1} \times \ldots \times A_{\lambda_k} \rightarrow A_\mu$$

where $\lambda=(\lambda_1,\ldots,\lambda_k)$ and $\lambda_1,\ldots,\lambda_k, \mu \in \{1,\ldots,n\}$ and $k \in \omega - \{0\}$. The distinguished elements are called the *initial data* of the structure and the maps are called the *primitive operations* of the structure.

A many-sorted algebra is *minimal,* or *prime,* if it is generated by its distinguished elements; equivalently, if it has no proper subalgebras. This minimality condition in the definition of a data structure ensures that every element of a data structure can be constructed from its initial data by means of its primitive operators.

A data structure A exactly describes how the syntax of a data type D is interpreted in a *concrete implementation* or *particular representation* of the semantics of D. The representation-free picture of the semantics of data type, required in the concept of an *abstract* data type, can be achieved by adopting the following principle:

1.1 ABSTRACTION PRINCIPLE. A property P of a data structure A qualifies as an *abstract semantical property* of the data type D which A represents if, and only if, P is an invariant of algebraic isomorphism i.e. if B is another data structure implementing or representing D, and A and B are isomorphic as algebras, then P is true of B.

For example, *finiteness* is an abstract property and, moreover, any property of a data structure which is *first-order definable* is an abstract property. In a later section we will define the *effective computability* of a data structure in such a way as to make it an abstract property of a data type. With this kind of analysis of the abstract nature of a data type semantics, the ADJ Group gave the following semantical definition of an abstract data type in [18]:

1.2 ABSTRACT DATA TYPE An *abstract data type* is the isomorphism class of a data structure.

For information on the invariance of semantical properties of programs based on abstract data types see Tucker and Zucker [53].

Mathematically, the theory of abstract data types is the theory of finitely generated minimal algebras. We assume the reader is familiar with the basic algebra of congruences, homomorphisms and so on, and can establish, when needed, facts such as:

1.3 LEMMA *Let* A *and* B *be minimal algebras of signature* Σ. *If there are homomorphisms* $\phi : A \to B$ *and* $\psi : B \to A$ *then* $A \cong B$ *via* ϕ *and* ψ.

Nowhere in this paper do we allow partial operations in our types.

AXIOMATIC SPECIFICATIONS A *first-order axiomatic specification* (Σ, T) describes a data type as a signature Σ whose constants and operator symbols satisfy a set T of first-order axioms. In Hoare's seminal paper [25], a specification is a formal documentation for a data type D which guarantees properties of implementations of D for use in proving the correctness of programs using D. The idea of axiomatising implementations is suited to an abstract (read : representation-free) view of data type, but alone, without special algebraic devices, it does not support a method which *uniquely defines* abstract data types. For consider the semantics of a first-order specification (Σ, T).

From the logical point of view, the natural semantics of (Σ, T) is the class $ALG(\Sigma, T)$ of all Σ-structures satisfying the axioms in T. This is because of Gödel's Completeness Theorem:

1.4 COMPLETENESS THEOREM *A first-order statement* p *is provable from* T *if, and only if, it is true in all models of* T; *in the usual notation,*

$$T \vdash p \quad \textit{if, and only if,} \quad T \models p$$

With reference to the Löwenheim-Skolem Theorem, few of the members of $ALG(\Sigma,E)$ can have anything to do with data types. We define, therefore, the class of data structures

$$ALG_m(\Sigma,T) = \{A \in ALG(\Sigma,E) : A \text{ is minimal}\}.$$

The class $ALG_m(\Sigma,T)$ consists of all implementations *consistent* with the conditions in T. As the class is closed under isomorphism, and contains non-isomorphic data structures, $ALG_m(\Sigma,T)$ serves as the semantics of specification (Σ,T) when the latter is thought of as a contract open to interpretation by a number of different abstract data types - an interpretation appropriate to program verification [9,12,13].

However, to be able to define an abstract data type by means of an axiomatic specification (Σ,T) some semantic mechanism M is necessary which chooses, uniquely up to isomorphism, an algebra $M(\Sigma,T) \in ALG_m(\Sigma,T)$ as the meaning of the specification (Σ,T). Given any such mechanism M, we say that an abstract data type D (read : isomophism type of a minimal algebra) is *correctly specified by an axiomatic specification* (Σ,T) *under semantics* M if the algebra $M(\Sigma,T)$ is in D.

This assignment by M cannot be accomplished by logical means for first-order specifications in general; it can be made by algebraic techniques for algebraic specifications.

ALGEBRAIC SPECIFICATIONS According to usage, a first-order specification (Σ,T) is called an *algebraic specification* when the axioms in T "look algebraic". In this paper, we consider specifications made with three simple kinds of algebraic axioms, only : *simple equations*, or *identities; equations;* and *conditional equations*.

Let $T(\Sigma)$ denote the algebra of (closed) terms over Σ and let $T_\Sigma(X_1,\ldots,X_n) = T_\Sigma(X)$ be the algebra of all terms or polynomials over Σ in the indeterminates $X = (X_1,\ldots,X_n)$.

A *simple equation*, also called a *simple identification* is an axiom of the form $t=t'$ where $t,t' \in T(\Sigma)$. An *equation* is an axiom of the form $t(X) = t'(X')$ where $t(X) \in T_\Sigma(X)$ and $t'(X') \in T_\Sigma(X')$. A *conditional equation* is an axiom of the form

$$e_1 \wedge \ldots \wedge e_k \rightarrow e_{k+1}$$

where each e_i, $1 \leq i \leq k$, and e_{k+1} is an equation. In the obvious notations, the sets of such axioms are nested thus:

$$\mathrm{SEQ}(\Sigma) \xrightarrow{i} \mathrm{EQ}(\Sigma) \xrightarrow{i} \mathrm{CEQ}(\Sigma).$$

Here then, an algebraic specification (Σ,E) will be a *simple equational specification* if $E \subset \mathrm{SEQ}(\Sigma)$; an *equational specification* if $E \subset \mathrm{EQ}(\Sigma)$; or a *conditional equation specification* if $E \subset \mathrm{CEQ}(\Sigma)$. In particular, axioms involving negation, explicitly or implicitly, are *not* allowed in specifications: for example, no inequalities $t \neq t'$ or definition-by-cases $t = \underline{\text{if}}\ e\ \underline{\text{then}}\ t_1\ \underline{\text{else}}\ t_2$.

Shortly, we will need to discuss computations on syntax in the arguments that follow so we assume that the various sets

$$T(\Sigma)\ ,\quad T_\Sigma(X)\ ,\quad \mathrm{SEQ}(\Sigma)\ ,\quad \mathrm{EQ}(\Sigma)\ ,\quad \mathrm{CEQ}(\Sigma)\ ,\ \text{etc.}$$

have been gödel numbered by means of the set $\omega = \{0,1,\ldots\}$. On being given the gödel number of a term, polynomial, axiom etc. we can primitive recursively calculate gödel numbers for its subterms, and the complexity of its syntax. Furthermore, in saying that $E \subset \mathrm{CEQ}(\Sigma)$ is a recursive or recursively enumerable set (for example) we actually mean that with respect to the gödel numbering $\delta : \omega \to \mathrm{CEQ}(\Sigma)$ the set $\delta^{-1}(E) = \{i : \delta(i) \in E\}$ is recursive or recursively enumerable. And in saying that $E = \{e_i : i \in \omega\}$ is recursively enumerated by $f : \omega \to \mathrm{CEQ}(\Sigma)$, where $f(i) = e_i$, we actually mean that $f : \omega \to \omega$ is a recursive function such that $\delta f : \omega \to E$ is surjective.

<u>SEMANTICS OF ALGEBRAIC SPECIFICATIONS</u>. The choice of an algebra $M(\Sigma,E)$ in $\mathrm{ALG}_m(\Sigma,E)$ as the meaning of the algebraic specification (Σ,E) is most simply made using *initial algebra semantics*.

When E contains conditional equations, the category $\mathrm{ALG}(\Sigma,E)$ of all E-algebras and all homomorphisms between them possesses an initial object $I(\Sigma,E)$, unique up to isomorphism. Furthermore, $I(\Sigma,E) \in \mathrm{ALG}_m(\Sigma,E)$ and we can define $M(\Sigma,E)$ to be $I(\Sigma,E)$: ADJ [17,18].

Let A be a minimal algebra, or data structure, representing the abstract data type D. Then the specification (Σ,E) *correctly defines* the type D *under initial algebra semantics* if

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

The practical effect of this method is to declare two operator terms of $T(\Sigma)$ to be semantically equivalent if, and only if, they can be *proved* equal using the axioms of E and the rules of first-order logic:

1.5 PROVABILITY CRITERION *For any* $t, t' \in T(\Sigma)$

$$E \vdash t = t' \text{ if, and only if, } I(\Sigma,E) \models t = t' \ .$$

Compare this with the Completeness Theorem 1.4.

We must assume that the reader is familiar with the basic algebra and logic involved in constructing and using initial algebra semantics. For example, we will make great use of the construction of $I(\Sigma,E)$ as a factor algebra $T(\Sigma,E)$ of $T(\Sigma)$.

Recall that, for $E \subset CEQ(\Sigma)$, a congruence $\equiv$ on a Σ-algebra A is an *E-congruence* if for each conditional equation e of the form

$$t_1 = t_1' \wedge \ldots \wedge t_k = t_k' \rightarrow t_{k+1} = t_{k+1}'$$

where $t_i, t_i' \in T_\Sigma(X)$, $1 \leq i \leq k+1$, we have that

$$t_1(a) \equiv t_1'(a), \ \ldots, \ t_k(a) \equiv t_k'(a) \text{ implies } t_{k+1}(a) \equiv t_{k+1}'(a)$$

for all $a \in A_{\lambda_1} \times \ldots \times A_{\lambda_n}$. Equivalently, $\equiv$ is an E-congruence if $A/\equiv$ is in $ALG(\Sigma,E)$. The intersection of all E-congruences on A is an E-congruence called the *least E-congruence* on A, and is denoted $\equiv_E$ (when A is understood).

Now consider the least E-congruence on $T(\Sigma)$, and define

$$T(\Sigma,E) = T(\Sigma)/\equiv_E \ .$$

It can be shown that $I(\Sigma,E) \cong T(\Sigma,E)$.

In working with $T(\Sigma,E)$ we will make use of transversals for $\equiv_E$:

Let $\equiv$ be a congruence on A. A *transversal* T for $\equiv$ is a complete family of unique representations of $\equiv$ in the sense that (i) for each $a \in A$ there is $t \in T$ such that $a \equiv t$ and (ii) for each $t,t' \in T$, $t \equiv t'$ implies $t = t'$. (We have here adopted the name *transversal* from the algebra of groups.)

In the case of a transversal T for $\equiv_E$ on $T(\Sigma)$, suppose T satisfies $t_1,\ldots,t_n \in T$ if, and only if, $\sigma(t_1,\ldots,t_n) \in T$. Then T is an Σ-algebra under the application of the operator symbols of Σ to terms in T, that is isomorphic to $T(\Sigma,E)$. This construction has been called a *canonical term algebra* in ADJ[51].

1.6 LEMMA $T(\Sigma,E) = T(\Sigma,\{e \in SEQ : E \vdash e\})$

1.7 LEMMA *If* E *is a set of equations,*

$$T(\Sigma,E) = T(\Sigma,\{e \in SEQ : e \textit{ is a substitution instance of some } e' \in E\}).$$

Initial algebra semantics is the denotational device used to assign a meaning to a specification in the early works of the ADJ Group [17] and

Liskov and Zilles [34,59]. And it is the only semantics considered in this paper. However, initial algebra semantics is *not* the semantics desired by J.V. Guttag [21] (see [22] for an explicit statement to this effect). The semantical status of an algebraic specification is far from clear in Guttag's early work; perhaps his requirements are best met by the *final algebra semantics* of Wand [55] and the Munich Group [14,56]. Final or terminal algebra semantics is a category-theoretic dual to initial algebra semantics in which provability is replaced by logical consistency. We have considered the technique in [8,10,11].

CLASSIFICATION OF SPECIFICATION METHODS

An algebraic specification method is characterised by the nature of its axioms and the nature of its semantical mechanisms. We consider methods based on *initial algebra semantics* and *three* types of axioms. Yet in a specification (Σ,E) the set E of axioms may be a *finite, recursive* or *recursively enumerable* set of simple identities, equations or conditional equations. This amount to 9 possibilities; later we will discuss two further refinements of specification methods, involving auxiliary operations and sorts, that lead to a classification of 27 methods. For the moment, let us make a notation for the 9 : let

FIN, REC and RE

denote finite, recursive and recursively enumerable and let

SEQ, EQ and CEQ

denote simple equations, equations and conditional equations.

Let $\alpha \in \{FIN,REC,RE\}$ and $\beta \in \{SEQ,EQ,CEQ\}$. A specification (Σ,E) is of type (α,β) if E is a set of type α containing axioms of type β.

1.8 DEFINITION An abstract data type represented by many-sorted algebraic structure A has an (α,β) *specification under initial algebra semantics* if there is an (α,β) specification (Σ,E) such that

$$T(\Sigma,E) \cong A.$$

1.9 EXAMPLES Consider the following four important structures on the set $\omega = \{0,1,2,...\}$ of natural numbers:

$$A_1 = (\omega;\ 0,\ x+1)$$
$$A_2 = (\omega;\ 0,\ x+1,\ x+y)$$
$$A_3 = (\omega;\ 0,\ x+1,\ x+y,\ x\cdot y)$$
$$A_4 = (\omega;\ 0,\ x+1,\ x+y,\ x\cdot y,\ x^2)$$

These structures have the following (FIN,EQ) specification:

$$\Sigma_1 = (\mathrm{NAT};\ 0,\ \mathrm{SUCC})$$
$$E_1 = \emptyset$$
$$\Sigma_2 = (\mathrm{NAT};\ 0,\ \mathrm{SUCC},\ \mathrm{ADD})$$
$$E_2 = \{\mathrm{ADD}(X,0) = X,\ \mathrm{ADD}(X,\mathrm{SUCC}(Y)) = \mathrm{SUCC}(\mathrm{ADD}(X,Y))\}$$
$$\Sigma_3 = (\mathrm{NAT};\ 0,\ \mathrm{SUCC},\ \mathrm{ADD},\ \mathrm{MULT})$$
$$E_3 = E_2 \cup \{\mathrm{MULT}(X,0) = 0\ \ \mathrm{MULT}(X,\mathrm{SUCC}(Y)) = \mathrm{ADD}(\mathrm{MULT}(X,Y),X)$$
$$\Sigma_4 = (\mathrm{NAT};\ 0,\ \mathrm{SUCC},\ \mathrm{ADD},\ \mathrm{MULT},\ \mathrm{SQ})$$
$$E_4 = E_3 \cup \{\mathrm{SQ}(X) = \mathrm{MULT}(X,X)\}$$

We leave the task of verifying that for $i=1,\ldots,4$

$$I(\Sigma_i,E_i) \cong T(\Sigma_i,E_i) \cong A_i$$

as an easy, yet essential, exercise.

These 9 types of specification are completely classified in Figure A, wherein a single arrow $(\alpha,\beta) \to (\alpha',\beta')$ indicates that any data type that can be given an (α,β) specification can be given an (α',β') specification, but not conversely; and a double arrow $(\alpha,\beta) \rightleftarrows (\alpha',\beta')$ indicates that the two kinds of specification define the same data types. The figure also registers the adequacy of the methods with respect to the finite, computable and semicomputable data types; this is taken up later in Section 3. Figure A conveniently records many theorems, of varying difficulty, distributed throughout the paper. Some results are easy and can be proved here, for illustration. However most results are best established with the semantical concepts of computable and semicomputable data type at hand; we will comment on Figure A then.

<u>1.10 THEOREM</u> *Let* A *be a many-sorted algebra of signature* Σ. *Then the following are equivalent :*

(i) A *has an* (RE,SEQ) *specification;*

(ii) A *has an* (RE,EQ) *specification;*

(iii) A *has an* (RE,CEQ) *specification.*

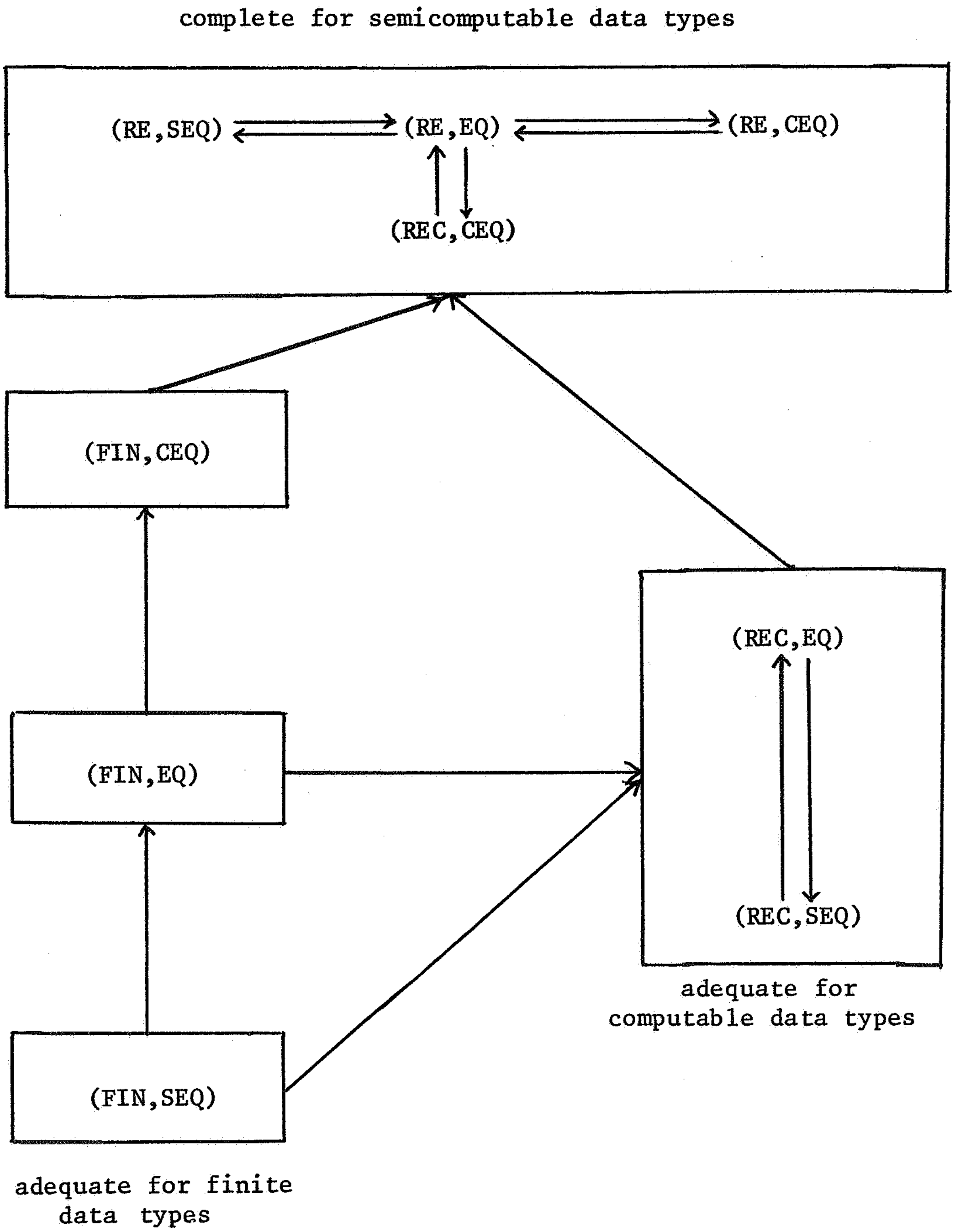


Figure A : Classification of the 9 algebraic methods (without hidden machinery)

PROOF By virtue of the definitions, it is sufficient to prove statement (iii) implies statement (i). Suppose $A \cong T(\Sigma,E)$ where E is an r.e. set of conditional equations. Then define

$$E' = \{t = t' : t, t' \in T(\Sigma) \text{ and } E \vdash t = t'\}$$

By Lemma 1.6, $T(\Sigma,E) \cong T(\Sigma,E')$. Clearly, E' is r.e. and hence A has an (RE,SEQ) specification.

□

1.11 THEOREM *Let A be a many-sorted algebra of signature Σ. Then A has a (REC,EQ) specification if, and only if, A has a (REC,SEQ) specification.*

PROOF Clearly every (REC,SEQ) specification is a (REC,EQ) specification. To prove the converse, suppose that A has a (REC,EQ) specification (Σ,E) i.e. E is a recursive set of equations over Σ and $A \cong T(\Sigma,E)$.

Define E' to be the set of all simple equations obtained by substituting all closed terms over Σ into the equations of E. Thus

$$E' = \{t=t' : \text{for some } e_1 = e_2 \in E \text{ and } t_1,\ldots,t_n \in T(\Sigma)$$
$$t = e_1(t_1,\ldots,t_n) \text{ and } t' = e_2(t_1,\ldots,t_n)\}$$

By Lemma 1.7, we have $T(\Sigma,E) \cong T(\Sigma,E')$. We claim E' is recursive.

Now given $t=t'$ with $t,t' \in T(\Sigma)$ there are finitely many equations $e_1=e_2 \in EQ(\Sigma)$, the set of all equations over Σ, such that there could exist $t_1,\ldots,t_n \in T(\Sigma)$ with

$$t = e_1(t_1,\ldots,t_n) \text{ and } t' = e_2(t_1,\ldots,t_n)$$

The length of the equations are constrained by the length of the terms t,t'; in fact :

$$|e_1| + |e_2| + 2(|t_1| + \ldots + |t_n|) \leq |t| + |t'|$$

Thus we can search through all equations and find those $e_1=e_2$ for which $t=t'$ is a substitution instance, and decide whether or not $e_1=e_2 \in E$ since E is recursive. Thus, E' is recursive. □

1.12 PROPOSITION *Let A be a finite many-sorted minimal structure. Then A has a (FIN,SEQ) specification.*

PROOF For each element b of A, let $t_b \in T(\Sigma)$ be a term that evaluates to b in A (using minimality). Let us axiomatise each operation σ_A of A

$$E_\sigma = \{\sigma(t_{b_1},\ldots,t_{b_n}) = t_b : \sigma_A(b_1,\ldots,b_n) = b\}$$

and set

$$E = \bigcup_{\sigma\in\Sigma} E_\sigma$$

Then it may be proved that $T(\Sigma,E) \cong A$. □

<u>CONSTRUCTING SPECIFICATIONS</u> We will state a series of theorems about constructing algebras and their specifications which are of general interest and which will be employed in many of the proofs of this paper.

Let A and B be algebras with signatures Σ and Σ' respectively; and suppose that $\Sigma \cap \Sigma' = \emptyset$. The *join* $[A,B]$ of A and B is the algebra of signature $\Sigma \cup \Sigma'$ obtained by taking all the domains, constants, and operations of A and B together to form one algebra. The effect of this operation on algebraic specifications is this:

<u>1.13 JOIN LEMMA</u> *Suppose* $A \cong T(\Sigma,E)$ *and* $B \cong T(\Sigma',E')$. *Then* $[A,B] \cong T(\Sigma \cup \Sigma', E \cup E')$.

Let A be an algebra with signature Σ. Let E' be a set of conditional equations over Σ, and let $\equiv_{E'}$ denote the least **E'-congruence** on A.

<u>1.14 FACTOR LEMMA</u> *Suppose* $A \cong T(\Sigma,E)$. *Then*

$$A/\equiv_{E'} \cong T(\Sigma,E \cup E')$$

<u>1.15 REFINEMENT LEMMA</u> *Suppose* $A \cong T(\Sigma,E)$. *If* $E' \vdash E$ *and* $A \models E'$ *then* $A \cong T(\Sigma,E')$.

Let A be an algebra with signature Σ and let $f : A_{\lambda_1} \times \ldots \times A_{\lambda_n} \to A_\mu$ be a function on A. On adding f as an operation we obtain an algebra A_f with signature $\Sigma_f = \Sigma \cup \{F\}$.

Suppose $A \cong T(\Sigma,E)$. We can algebraically specify A_f by a straight-forward representation of the graph of f. Let T be a set of canonical term representatives, or a transveral, of $\equiv_E$. The map f on A uniquely induces maps $\hat{f}$ and $\hat{f}_T$ on $T(\Sigma,E)$ and T in the obvious way

$$\begin{tikzcd}
A_{\lambda_1} \times \ldots \times A_{\lambda_n} \arrow[r, "f"] & A_\mu \\
T(\Sigma,E)_{\lambda_1} \times \ldots \times T(\Sigma,E)_{\lambda_n} \arrow[u, "\phi"] \arrow[r, "\hat{f}"] & T(\Sigma,E)_\mu \arrow[u, "\phi"'] \\
T_{\lambda_1} \times \ldots \times T_{\lambda_n} \arrow[u, "\nu"] \arrow[r, "\hat{f}_T"] & T_\mu \arrow[u, "\nu"']
\end{tikzcd}$$

where $\nu : T(\Sigma) \to T(\Sigma)/{\equiv_E}$ is the canonical factor map $\nu(t) = [t]$ which is a bijection on T. We define

$$E_f(T) = \{F(t_1,\ldots,t_n) = t : \hat{f}_T(t_1,\ldots,t_n) = t\}$$

which represents the graph of f on T.

1.16 FUNCTION LEMMA *Suppose* $A \cong T(\Sigma,E)$ *and* f *is a map on* A. *For any transversal* T,

$$A_f \cong T(\Sigma_f, E \cup E_f(T))$$

2. SPECIFICATIONS WITH HIDDEN FUNCTIONS AND HIDDEN SORTS

The specification methods classified by Definition 1.8 have the property that only the sorts and operations of the data type signature are allowed in specifications of the data type : if A is of signature Σ then A must be axiomatised by a set E using the operations in Σ only. These methods can be augmented usefully by allowing extra, auxiliary sorts and functions in specifications (Σ',E') that are not required in A, so $\Sigma \subset \Sigma'$.

2.1 EXAMPLE Consider the algebra

$$A_5 = (\omega;\ 0,\ x+1,\ x^2)\ .$$

The natural way to specify A_5 is to specify the algebra

$$A_4 = (\omega;\ 0,\ x+1,\ x+y,\ x\cdot y,\ x^2)$$

by means of the (FIN,EQ) specification (Σ_4,E_4) in Examples 1.9 and then to *forget* or to *hide* the operations of addition and multiplication. Later we will prove that it is not possible to specify A_5 without recourse to hidden operations.

To put such techniques on a proper foundation we must define the mechanisms of hiding the auxiliary operations.

Let B be an algebra of signature Σ and let $\Sigma_0 \subset \Sigma$. We define two algebras:

$B|_{\Sigma_0}$ is the algebra consisting of the domains, constants and operations of B named in Σ_0; and

$\langle B\rangle_{\Sigma_0}$ is the subalgebra of $B|_{\Sigma_0}$ generated by elements named in Σ_0.

2.2 LEMMA *The following are equivalent :*

(i) $B|_{\Sigma_0}$ *is minimal;*

(ii) $B|_{\Sigma_0} = \langle B\rangle_{\Sigma_0}$;

(iii) $B|_{\Sigma_0} \cong \langle B\rangle_{\Sigma_0}$.

2.3 LEMMA *Let* B *be of signature* Σ *and let* $\Sigma_0 \subset \Sigma_1 \subset \Sigma$. *Then*

(i) $(B|_{\Sigma_1})|_{\Sigma_0} = B|_{\Sigma_0}$;

(ii) $\langle\langle B\rangle_{\Sigma_1}\rangle_{\Sigma_0} = \langle B\rangle_{\Sigma_0}$.

2.4 LEMMA *Let* B *and* B' *be* Σ*-algebras. Let* $\phi : B \to B'$ *be a* Σ*-homomorphism. Then the restrictions*

$$\phi : B|_{\Sigma_0} \to B'|_{\Sigma_0} \quad \textit{and} \quad \phi : \langle B\rangle_{\Sigma_0} \to \langle B'\rangle_{\Sigma_0}$$

are Σ_0*-homomorphisms.*

These two contraction methods lead to *three* kinds of specifications allowing hidden functions and *three* kinds of specifications allowing hidden sorts and functions. For in either case, in the specification of A of signature Σ_A, an algebra B of signature Σ_B, with $\Sigma_A \subset \Sigma_B$, is constructed and specified, and we may choose one of the following :

(i) $B|_{\Sigma} \cong A$

(ii) $\langle B\rangle_{\Sigma} \cong A$

(iii) $B|_{\Sigma} = \langle B\rangle_{\Sigma} \cong A$.

Let $\alpha \in \{FIN, REC, RE\}$ and $\beta \in \{SEQ, EQ, CEQ\}$. Let A be a many-sorted algebra of signature Σ representing an abstract data type.

2.5 DEFINITIONS An (α,β) *hidden function specification of type* I, II *or* III for A consists of an algebraic specification (Σ_0, E_0) of type (α,β) such that $\Sigma \subset \Sigma_0$, and Σ_0 contains exactly the sorts of Σ, and which defines A by means of initial algebra semantics in one of the following three ways, respectively:

Type I $\quad T(\Sigma_0,E_0)|_{\Sigma} \cong A$

Type II $\quad \langle T(\Sigma_0,E_0)\rangle_{\Sigma} \cong A$

Type III $\quad T(\Sigma_0,E_0)|_{\Sigma} = \langle T(\Sigma_0,E_0)\rangle_{\Sigma} \cong A$

2.6 DEFINITIONS An (α,β) *hidden sorts specification of type* I, II *or* III for A consists of a specification (Σ_0,E_0) of type (α,β) such that $\Sigma \subset \Sigma_0$ and which defines A by means of initial algebra semantics in one of the following three ways, respectively:

Type I $\quad T(\Sigma_0,E_0)|_\Sigma \cong A$

Type II $\quad <T(\Sigma_0,E_0)>_\Sigma \cong A$

Type III $\quad T(\Sigma_0,E_0)|_\Sigma = <T(\Sigma_0,E_0)>_\Sigma \cong A.$

In Kamin [30], type I specifications are said to be the *usual interpretation* of hidden functions and sorts; and type II specifications are said to be the *subalgebra interpretation*. In the standard case that A is a Σ-minimal algebra, a type I specification is also a type II specification and a type III specification (Lemma 2.2).

In this paper we will consider only specifications of type III, and introduce the following terminology ([4]).

2.7 DEFINITIONS An (α,β) *hidden enrichment specification* for A is an (α,β) hidden functions specification of type III for A.

An (α,β) *hidden enrichment with sorts specification* for A is an (α,β) hidden sorts specification of type III for A.

For example, (Σ_4,E_4) is a (FIN,EQ) hidden enrichment specification of the algebra A_5 of Example 2.1.

In addition to the 9 types of specification methods, defined in the last section, we can consider a further 9 types of specification method that allow hidden functions and 9 types of specification method that allow hidden sorts and functions. This makes 27 methods in total, all based on initial algebra semantics.

Thus, let $\alpha \in \{FIN,REC,RE\}$ and $\beta \in \{SEQ,EQ,CEQ\}$ and let $\gamma \in \{HE,HES\}$, where HE and HES stand for *hidden enrichment* and *hidden enrichment with sorts*, respectively. The first 9 types of specification are abbreviated (α,β) as before; the 18 new types of specification are abbreviated (α,β,γ).

Once again we will summarise what is known of the classification in Figure B. The majority of equivalences will follow easily from our discussion of computability : see the next section.

We conclude with two lemmas.

2.8 LEMMA *Let* (Σ,E) *and* (Σ',E') *be specifications with* $\Sigma \subset \Sigma'$ *and* $E \subset E'$. *Suppose that* Σ *and* Σ' *contain the same sorts. Suppose there exists a transversal* $T \subset T(\Sigma)$ *for* $\equiv_E$ *such that*

(i) *for distinct* $t_1,t_2 \in T$, $t_1 \not\equiv_{E'} t_2$

(ii) *for each constant* $c \in \Sigma' - \Sigma$, *there is* $t \in T$ *such that* $c \equiv_{E'} t$;

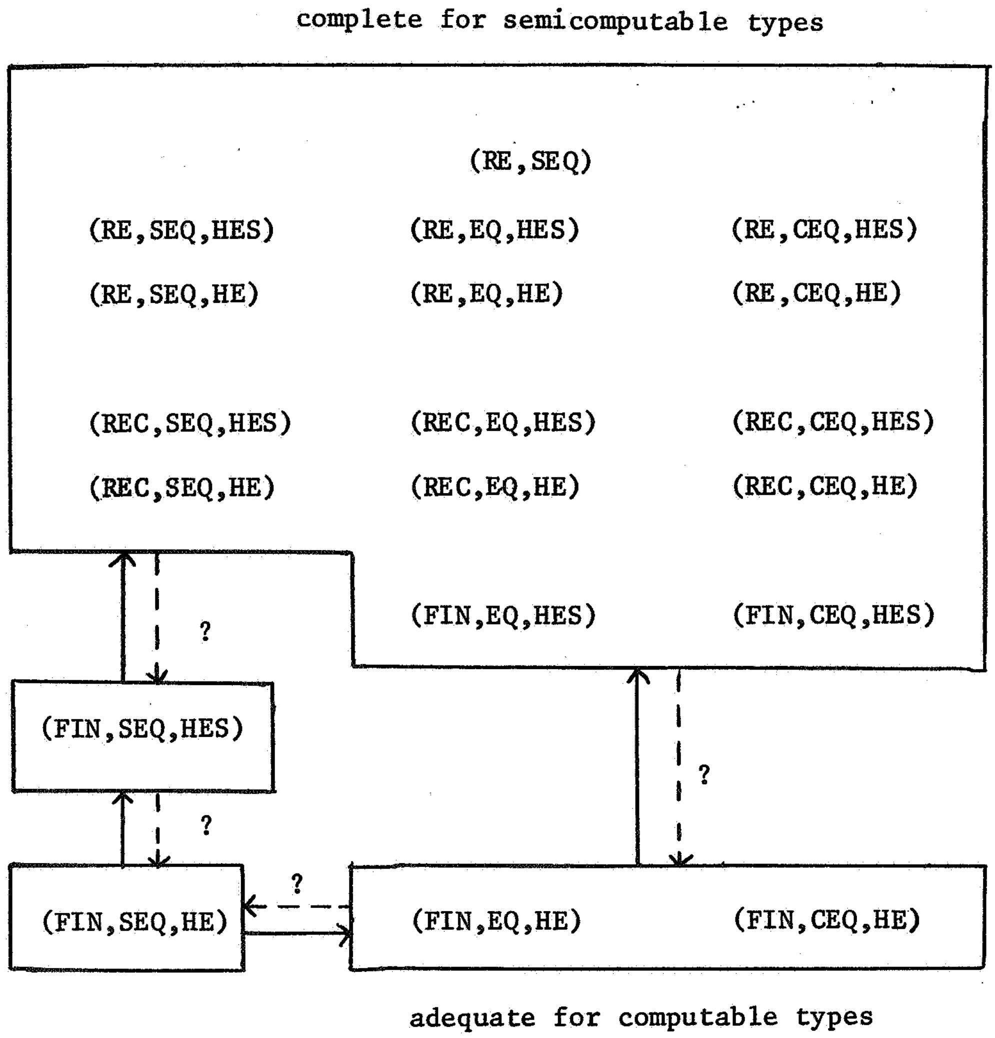


Figure B : Classification of the 18 algebraic methods (using hidden machinery) *(iii)* *for each* k*-ary operation* $\sigma \in \Sigma' - \Sigma$ *and any* $t_1,\ldots,t_k \in T$ *there is* $t \in T$ *such that* $\sigma(t_1,\ldots,t_k) \equiv_{E'} t$.

Then $T(\Sigma',E')|_\Sigma \cong T(\Sigma,E)$.

PROOF Since $E \subset E'$ and T is a transversal, it is easy to see that $\phi([t]_E) = [t]_{E'}$, for $t \in T$, well defines a Σ-homomorphism

$$\phi : T(\Sigma,E) \to T(\Sigma',E')|_\Sigma \quad .$$

Condition (i) implies ϕ is injective, because if $t_1,t_2 \in T$ and $[t_1]_E \neq [t_2]_E$ then $[t_1]_{E'} \neq [t_2]_{E'}$. Conditions (ii) and (iii) imply that ϕ is surjective as follows: we show that for each $t' \in T(\Sigma')$ there is $t \in T$ such that $t \equiv_{E'} t'$.

Now if $t' \in T(\Sigma)$ then $t \equiv_E t'$ for some $t \in T$ and so $t \equiv_{E'} t'$ as $\equiv_E$ is contained in $\equiv_{E'}$. Assume $t' \in T(\Sigma') - T(\Sigma)$. We argue by induction on the complexity of t'.

The basis case has t' a constant in $\Sigma'-\Sigma$ and is immediate from the condition (ii).

Let $t' = \sigma(t_1',\ldots,t_k')$ for some $\sigma \in \Sigma'$ and assume there exist $t_1,\ldots,t_k \in T$ such that $t_i \equiv_{E'} t_i'$ for $1 \le i \le k$. Then $t' \equiv_{E'} \sigma(t_1,\ldots,t_k)$. If $\sigma \in \Sigma$ then $\sigma(t_1,\ldots,t_k) \in T(\Sigma_0)$ and obviously $t \equiv_{E'} t'$ for some $t \in T$. If $\sigma \in \Sigma' - \Sigma$ then $\sigma(t_1,\ldots,t_k) \equiv_{E'} t$ for some $t \in T$ by condition (iii). Thus, $t \equiv_{E'} t'$. □

2.9 LEMMA *Let* (Σ,E) *and* (Σ',E') *be specifications with* $\Sigma \subset \Sigma'$, $E \subset E'$. *Suppose that* Σ *and* Σ' *contain the same sorts and that*

$$T(\Sigma',E')|_\Sigma \cong T(\Sigma,E).$$

Let A *and* A' *be* Σ *and* Σ'*-algebras such that*

$$A'|_\Sigma \cong A \quad .$$

If $A \cong T(\Sigma,E)$ *and* A' *is an* E'*-algebra then*

$$B \cong T(\Sigma',E') \quad .$$

PROOF The hypotheses imply there is a Σ-isomorphism

$$\phi : T(\Sigma',E')|_\Sigma \to A'|_\Sigma \quad .$$

By the initiality of $T(\Sigma',E')|_\Sigma$ for E-algebras - inherited from $T(\Sigma,E)$ - the map ϕ is unique as a homomorphism. Since A' is an E'-algebra there exists a Σ'-homomorphism $\psi : T(\Sigma',E') \to A'$ which restricts to a Σ-homomorphism

$\psi : T(\Sigma',E')|_{\Sigma} \to A|_{\Sigma}$. Thus $\psi = \phi$ and ψ must be bijective and hence a Σ'-isomorphism.

□

3. COMPUTABLE AND SEMICOMPUTABLE ALGEBRAS

In this section we define the computable and semicomputable data types and explain their role in the theory. A number of basic properties of these notions are described and we are then able to review the Figures A and B of Sections 1 and 2 in order to set the scene for the rest of the paper.

ADEQUACY AND COMPLETENESS

Our semantic measures of adequacy for the specification methods are the classes of *computable* and *semicomputable data types*.

3.1 DEFINITIONS A many-sorted algebra A is said to be *effectively presented* when it is given an *effective coordinatization* (α, Ω) consisting of recursive sets $\Omega_1, \ldots, \Omega_n$, $\Omega_i \subset \omega$ for $1 \le i \le n$, corresponding with the domains $A_1, \ldots, A_n$ of A; surjections $\alpha_1, \ldots, \alpha_n$, $\alpha_i : \Omega_i \to A_i$ for $1 \le i \le n$; and, for each operation σ,

$$\sigma : A_{\lambda_1} \times \ldots \times A_{\lambda_k} \to A_{\mu}$$

a recursive function $\bar{\sigma}$

$$\bar{\sigma} : \Omega_{\lambda_1} \times \ldots \times \Omega_{\lambda_k} \to A_{\mu}$$

that *tracks* σ in the sense that the following diagram commutes:

$$\begin{tikzcd}
A_{\lambda_1} \times \ldots \times A_{\lambda_k} \arrow[r, "\sigma"] & A_{\mu} \\
\Omega_{\lambda_1} \times \ldots \times \Omega_{\lambda_k} \arrow[u, "\alpha_{\lambda_1} \times \ldots \times \alpha_{\lambda_k}"] \arrow[r, "\bar{\sigma}"] & \Omega_{\mu} \arrow[u, "\alpha_{\mu}"']
\end{tikzcd}$$

wherein $(\alpha_{\lambda_1} \times \ldots \times \alpha_{\lambda_k})(x_1, \ldots, x_k) = (\alpha_{\lambda_1}(x_1), \ldots, \alpha_{\lambda_k}(x_k))$. We sometimes write $\alpha : \Omega \to A$ or, simply, α for an effective coordinatization (α, Ω).

The algebra A is said to be *computable, semicomputable* or *cosemicomputable*, if there *exists* an effective presentation $\alpha : \Omega \to A$ for which the relations $\equiv_i$ defined on Ω_i by

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

for $1 \le i \le n$, are recursive, r.e., or co-r.e., respectively.

These three notions are the standard formal definitions of constructive algebraic structures currently in use in mathematical logic and they derive from the work of M.O. Rabin [46] and, in particular, A.I. Mal'cev [39]; they possess the following *essential* property:

3.2 LEMMA *Computability, semicomputability and cosemicomputability are isomorphism invariants.*

Thus, the three notions qualify as abstract semantical properties for data types, according to the Abstraction Principle 1.1.

3.3 DEFINITION A data type D is *computable, semicomputable,* or *cosemicomputable,* if there exists an algebra A representing D that is computable, semicomputable or cosemicomputable.

By Lemma 3.2, if one algebra represents D and is effectively computable then all representing algebras of D are effectively computable.

Shortly, in Lemma 3.12 and 3.14, we will see that, *under initial algebra semantics, all the algebraic specification methods define semicomputable data types.* Thus, given the independent interest of the notion, it is natural to seek to determine which specification methods are capable of defining all semicomputable data types.

More generally, let M be a data type specification method and let K be a class of data types.

3.4 DEFINITIONS The method M is *sound* for K if each data type D defined by M is in K.

The method M is *adequate* for K if each data type D in K can be defined by M.

The method M is *complete* for K if M is sound and adequate for K.

Notice that two methods that are complete for the same class are equivalent.

In this paper we are often concerned with methods that are complete for the semicomputable data types and adequate (but not sound) for the computable data types. For information on methods that are complete for computable and cosemicomputable data types, see the Concluding Remarks.

BASIC TECHNICAL IDEAS

Combining the components of an effective coordinatization (α,Ω) of A we can make a recursive algebra Ω of numbers from $\Omega_1,\ldots,\Omega_n$ and the recursive tracking operations, of signature Σ. With respect to this algebra, the maps $\alpha_1,\ldots,\alpha_n$ constitute a Σ-epimorphism $\alpha : \Omega \to A$. Thus, A is the homomorphic image of a recursive algebra Ω of numbers and $A \cong \Omega/\equiv_\alpha$.

3.5 REPRESENTATION LEMMA *A computable algebra* A *is isomorphic to a recursive algebra* R *of numbers each of whose domains* R_i *is the set* ω *of natural numbers, or the set* ω_m *of the first* m *natural numbers, accordingly as the corresponding domain* A_i *of* A *is infinite, or finite of cardinality* m.

PROOF Let A be computable under $\alpha : \Omega \to A$. For each $1 \le i \le n$, define the recursive set $\Gamma_i \subset \Omega_i$ by

$$x \in \Gamma_i \Leftrightarrow x \in \Omega_i \ \& \ (\forall y{<}x)[y \in \Omega_i \to y \not\equiv_{\alpha_i} x]$$

so that $\alpha_i : \Gamma_i \to A_i$ is bijective. Let $f_i : \omega \to \Gamma_i$ be a recursive bijection if Γ_i is infinite; and let $f_i : \omega_m \to \Gamma_i$ be a bijection if Γ_i is finite. Define $R_i = \mathrm{dom}(f_i)$ and $\beta_i : R_i \to A_i$ by

$$\beta_i = \alpha_i f_i : R_i \to \Gamma_i \to A_i .$$

Now for each recursive tracking function

$$\bar{\sigma} : \Omega_{\lambda_1} \times \ldots \times \Omega_{\lambda_k} \to \Omega_\mu$$

of operation σ of A we define a recursive function

$$\sigma_R : R_{\lambda_1} \times \ldots \times R_{\lambda_k} \to R_\mu$$

by

$$\sigma_R(x_1,\ldots,x_k) = f_\mu^{-1}\bar{\sigma}(f_{\lambda_1}(x_1),\ldots,f_{\lambda_k}(x_k)).$$

It is easy to see that σ_k tracks σ with respect to β. It follows that combining the domains R_i and operations σ_R forms the required algebra R isomorphic to A under β. □

Obviously, such an isomorphic algebra of numbers can be provided for a semicomputable or cosemicomputable algebra A if, and only if, A is computable.

We will now discuss the invariance of computability in terms of the uniqueness of the coordinatizations.

<u>3.6 DEFINITIONS</u> Let α and β be effective presentations of an algebra A. Then α *recursively reduces* to β (in symbols : $\alpha \le \beta$) if there exist recursive functions $f_1,\ldots,f_n$ where

$$f_i : \Omega_i^\alpha \to \Omega_i^\beta$$

that commute the following diagrams

$$\begin{tikzcd} & A_i \\ \Omega_i^\alpha \arrow[ur, "\alpha_i"] \arrow[r, "f_i"'] & \Omega_i^\beta \arrow[u, "\beta_i"'] \end{tikzcd}$$

for $1 \le i \le n$.

And α is *recursively equivalent* to β if both $\alpha \le \beta$ and $\beta \le \alpha$.

Now recursive equivalence is the basic identity relation between coordinatizations and establishes the uniqueness of computability concepts in the algebraic setting.

Let $R \subset A_{\lambda_1} \times \ldots \times A_{\lambda_k}$ be a relation on A and let A be effectively presented by α. Then R is said to be *α-computable* if its preimage

$$\alpha^{-1}(R) = \{(x_1,\ldots,x_k) : (\alpha_{\lambda_1}(x_1),\ldots,\alpha_{\lambda_k}(x_k)) \in R\}$$

is recursive. The definitions of *α-semicomputable* and *α-cosemicomputable* relations follow mutato nomine. The following fact is easy to check.

<u>3.7 LEMMA</u> *Let* R *be an α-computable (α-semicomputable or α-cosemicomputable) relation on* A. *If β is another effective presentation for* A *and β recursively reduces to α then* R *is β-computable (β-semicomputable or β-cosemicomputable).*

To what extent is the computability of an algebra, and its various relations, dependent upon the choice of a coordinatization? We will show that as far the theory of data types is concerned, the computability theory is independent of coordinatizations.

Henceforth, we consider minimal algebras only : let A be a minimal algebra of signature Σ.

Clearly, the term algebra $T(\Sigma)$ is computable under any natural gödel numbering of terms. By Lemma 3.5 , we can choose a computable coordinatization

$\gamma_* : R \to T(\Sigma)$, with the domains of R each being ω. Let $v : T(\Sigma) \to A$ be the unique term evaluation homomorphism. We define the *standard effective presentation* of A, derived from γ_*, to be the composition

$$\gamma_A = v\gamma_* : R \to T(\Sigma) \to A .$$

3.8 REDUCTION LEMMA *The standard effective presentation* γ_A *of minimal algebra* A *recursively reduces to every effective presentation* α *of* A.

A proof of this fact can be found in Mal'cev [39]; coupled with Lemma 3.6, it leads to several important results:

3.9 INVARIANCE THEOREM *The minimal algebra* A *is computable, semicomputable or cosemicomputable if, and only if, it is so under the standard effective presentation* γ_A.

3.10 UNIQUENESS THEOREM *Any two semicomputable coordinatizations of the minimal algebra* A *are recursively equivalent.*

3.11 REPRESENTATION LEMMA *Let* A *be a minimal algebra. If* A *is semicomputable, or cosemicomputable, then it can be represented as the image of a recursive algebra* R *of numbers, all of whose domains are* ω, *and such that epimorphism* $\alpha : R \to A$ *has congruence* $\equiv_\alpha$, *defined by*

$$x \equiv_\alpha y \textit{ if, and only if, } \alpha(x) = \alpha(y) \textit{ in } A$$

is r.e., or co-r.e., respectively.

CLASSIFICATION OF METHODS : NO HIDDEN MACHINERY Let us begin to apply these concepts in the classification of specification methods, and comment on Figure A.

Let A be minimal and define

$$\begin{aligned} S_A &= \{t = t' \in SEQ : A \models t = t'\} \\ &= \{t = t' \in SEQ : v(t) = v(t') \text{ in } A\} \\ &= \{\gamma_*(x) = \gamma_*(y) : (x,y) \in \equiv_{\gamma_A}\} \end{aligned}$$

where $\gamma_A = v\gamma_* : R \to A$ is the standard effective presentation for A constructed above. Clearly, by the definitions,

$$T(\Sigma, S_A) \cong A$$

Suppose that A has an (RE,CEQ) specification (Σ,E) so that $A \cong T(\Sigma,E)$. By the Provability Criterion 1.5,

$$E \vdash t = t' \Leftrightarrow A \models t = t' \Leftrightarrow t = t' \in S_A$$

Since E is an r.e. set of axioms, S_A is r.e. Thus we may deduce that A has an (RE,SEQ) and hence an (RE,EQ) specification. Furthermore, since S_A is r.e. we know that $\equiv_{\gamma_A}$ is r.e. and, in particular, that A is semicomputable.

Conversely, suppose A is semicomputable. Then, by the Invariance Theorem 3.9, A is semicomputable under γ_A and S_A is r.e. Thus, we know A has an (RE,SEQ) specification and so an (RE,EQ) and an (RE,CEQ) specification.

3.12 COMPLETENESS LEMMA *Let* A *be a minimal many-sorted algebra. Then the following are equivalent :*

(i) A *is semicomputable;*

(ii) S_A *is r.e.;*

(iii) A *has an* (RE,SEQ) *specification;*

(iv) A *has an* (RE,EQ) *specification;*

(v) A *has an* (RE,CEQ) *specification;*

Suppose that A is computable then, by a similar argument, we can conclude that S_A is recursive. Conversely, if S_A is recursive then A is computable under γ_A.

3.13 LEMMA *Let* A *be a minimal many-sorted algebra. Then the following are equivalent :*

(i) A *is computable;*

(ii) S_A *is recursive.*

If A *is computable then* A *has a* (REC,SEQ) *specification.*

Later we will show that the converse of this adequacy fact is false (Corollary 6.3) ie. that (REC,SEQ) specifications are *not* sound for computable types. In addition, we will show that the (REC,SEQ) and (REC,EQ) specifications, shown to be equivalent in Theorem 1.11, are *not* adequate for the semicomputable types (Theorem 6.5). However, we will show that the (REC,CEQ) specifications are complete for the semicomputable types (Theorem 6.1). This concludes the case of infinite specifications (without hidden operators or sorts). To complete our commentary on Figure A, we note that the (FIN,SEQ), (FIN,EQ) and

(FIN,CEQ) specifications are rather weak and are not adequate even for the computable data types; a full discussion of these finite methods can be found in the next section.

CLASSIFICATION OF METHODS : HIDDEN MACHINERY Given the Completeness Lemma 3.12, it is a routine matter to extend it to the following:

3.14 COMPLETENESS LEMMA *Let* A *be a minimal many-sorted algebra. Then the following are equivalent*

(i) A *is semicomputable;*

(ii) A *has an* (RE,β,γ) *specification for any* $\beta \in$ {SEQ,EQ,CEQ} *and* $\gamma \in$ {HE,HES}.

Later, we will show that the (REC,S,HE) specifications and hence all the (REC,α,β) specifications are complete for the semicomputable types (Theorem 6.2). Thus, on allowing hidden sorts or operators and infinitely many axioms, completeness for a method duly follows, and all methods are equivalent.

We are left with two basic questions:

Are any finite algebraic specifications either complete for the semicomputable data types, or adequate for the computable data types?

In Theorem 5.3, we will prove (FIN,EQ,HES), and hence (FIN,CEQ,HES), specifications complete for the semicomputable data types. In Theorem 5.1 we will prove (FIN,EQ,HE), and hence (FIN,CEQ,HE), specifications are adequate (but not sound) for the computable data types. As illustrated in Figure B we have no information on the (FIN,SEQ,HE) and (FIN,SEQ,HES) specifications and, more importantly, must record that the following problem from [4] is still open:

3.15 OPEN PROBLEM *Are the* (FIN,EQ,HE) *specifications complete for the semicomputable algebras?*

Some work on this problem can be found in [10].

WORD PROBLEMS The mathematical tools of computability we employ are used in studying algorithmic questions in algebra, mainly in combinatorial aspects of group theory (Lyndon and Schupp [36]) and universal algebra (Mal'cev [40]); and in ring and field theory (Rabin [46], Stoltenberg-Hansen and Tucker [50]). The equivalence of (i) and (ii) in Lemmas 3.12 and

class \ method	(FIN,EQ)	(FIN,CEQ)	(FIN,EQ,HE)	(FIN,CEQ,HE)	(FIN,EQ,HES)
finite data types	✓	✓	✓	✓	✓
computable data types	X	X	✓	✓	✓
semicomputable data types	X	X	?	?	✓

Figure C : The adequacy of finite algebraic axiomatisations

3.14 establish their connection with the decidability of word problems.

With reference to the literature (Gratzer [19], Cohn [16] or Mal'cev [40]) we note the following :

3.16 LEMMA *Let* V *be a variety of algebraic structures of signature* Σ *defined by a finite set* L *of laws. Then* $A \in V$ *is finitely presented with respect to* V *by* (X,R) *if, and only if, the pair* $(\Sigma \cup X,\ L \cup R)$ *is a* (FIN,EQ) *specification for* A *on adjoining the generators of* A *as constants.*

Thus, the existence of finitely presented semigroups and groups with unsolvable word problems (Lallement [33], Rotman [48]) implies that (FIN,EQ) specifications define non-computable algebras and are not sound for computable semigroups and groups. Not all finitely generated semigroups and groups are finitely presented, and indeed there exist semicomputable semigroups and groups that are not finitely presented : thus (FIN,EQ) specifications are not complete.

On the other hand every finitely generated abelian group is finitely presented with respect to the variety of abelian groups and indeed is computable. By the Hilbert Basis Theorem, the same is true of the finitely generated commutative rings.

4. LIMITATIONS OF SPECIFICATIONS WITHOUT HIDDEN MECHANISMS

The main tasks of this section are to construct two simple algebras and prove in detail that they fail to possess (FIN,EQ) and (FIN,CEQ) specifications, respectively. Both algebras are computable so from these theorems we can deduce a number of non-equivalence results with methods that are adequate for computable data types.

Hidden operations can be used to give simple specifications, as we saw in Example 2.1. In Majster [37], there appeared the first example of a type which cannot be specified by a (FIN,EQ) specification. The type is an interesting stack, but its complexity precluded a full proof of its non-definability. Attempts and suggestions aimed at giving a specification of Majster's stack, using extra machinery, are found in Kapur [31], Jones [28], Hilfinger [24] and Subrahmanyam [49], Veloso [54]; see, too, Majster [38] which includes another example we will take up shortly.

In ADJ [52], there is a critique of this situation. In particular, a simpler toy-stack, based on Majster's stack, is constructed and carefully proved not to have a (FIN,EQ) specification and yet to have a (FIN,EQ,HE) specification. Thus, it is known that there are data types that one desires to specify that require the use of hidden machinery.

Independently of ADJ [52], we presented in [4] the algebra A_5 in Example 2.1 as an example of a data type that one wishes to define, but which needs hidden machinery. Here is the proof.

4.1 THEOREM *The algebra*

$$A = (\omega;\ 0, x+1, x^2)$$

does not possess a (FIN,EQ) *specification.*

PROOF Suppose for a contradiction that (Σ,E) is a (FIN,EQ) specification of A. We assume that E contains no trivial equations of the form t=t. Let $E = E_1 \cup E_2 \cup E_3$ where

E_1 contains the simple equations of E;

E_2 contains the equations of E of the forms

$$t_1(X) = t_2 \qquad t_2 = t_1(X) \qquad t_1(X) = t_3(Y)$$

where t_2 is simple and X,Y are free in t_1,t_3;

E_3 contains the equations of E of the form $t_1(X) = t_2(X)$ where X is free in t_1,t_2 .

First we show that $E_2 = \emptyset$. For instance, $t_1(X) = t_2$ cannot hold in A because $t_1(X)$ is interpreted in A by an injective function (because the operations of A are injective) while t_2 is interpreted as a fixed number. The case of $t_2 = t_1(X)$ is identical. Finally, $t_1(X) = t_3(Y)$ cannot hold in A because on substituting Y = 0 we obtain an equation of the previous form $t_1(X) = t_2$ which does not hold in A.

Now we show that $E_3 = \emptyset$. Actually we will show that if $A \models t_1(X) = t_2(X)$ then $t_1(X) \equiv t_2(X)$ and the equation is trivial; since we supposed E to be free of trivial equations we may conclude that $E_3 = \emptyset$.

If $A \models t_1(X) = t_2(X)$ then if $F \in \Sigma$ names $f(x) = x^2$

$$A \models t_1F(X) = t_2F(X) .$$

We will create a special representation of these terms of form tF(X) in order to prove $t_1 \equiv t_2$. Let S name $s(x) = x+1$.

Let $\Sigma' = \Sigma - \{0\}$. The terms of interest are those in $T_{\Sigma'}(F(X))$.

Let B be the following structure of infinite signature Γ

$$B = (\omega : f_0, f_1, f_2, \ldots)$$

wherein $f_i(x) = x^2 + i$; note that $f_0(x) = x^2$. This B is tailored to the semantics of $T_{\Sigma'}(F(X))$ (see Lemma 4.2). Let Γ be the signature of B with f_i named by F_i. We construct a syntactic transformation $H : T_{\Sigma'}(F(X)) \to T_\Gamma(X)$

$$H(F(X)) = F_0(X)$$

$$H(S(t)) = F_{a_0+1}(F_{a_1} \ldots F_{a_k}(X)) \text{ if } H(t) = F_{a_0}F_{a_1}\ldots F_{a_k}(X).$$

$$H(F(t)) = F_0(H(t))$$

4.2 LEMMA H *is injective and* t *and* H(t) *have identical interpretations as functions on* ω. *In particular,*

$$A \models t_1F(X) = t_2F(X) \text{ implies } B \models H(t_1F(X)) = H(t_2F(X))$$

PROOF We leave this as an exercise involving induction. □

Suppose $H(t_1F(X)) = F_{a_1}\ldots F_{a_p}(X)$ and $H(t_2F(X)) = F_{b_1}\ldots F_{b_q}(X)$. We will prove that

$$B \models F_{a_1}\ldots F_{a_p}(X) = F_{b_1}\ldots F_{b_q}(X) \Leftrightarrow p=q \text{ and } a_i=b_i \text{ for } i=1,\ldots,p\ .$$

That is, the semigroup G of functions on ω generated by the f_i under composition is a free semigroup. This done we deduce that

$$B \models H(t_1F(X)) = H(t_2F(X)) \text{ implies } H(t_1F(X)) \equiv H(t_2F(X))\ .$$

By the injectivity of H we know that $t_1F(X) \equiv t_2F(X)$. This obviously implies that $t_1 \equiv t_2$.

Suppose $B \models F_{a_1}\ldots F_{a_p}(X) = F_{b_1}\ldots F_{b_q}(X)$. If $p \neq q$ then on their interpretation as polynomials on ω, the terms on each side have different degrees, namely 2^p and 2^q respectively. Consequently, the terms cannot represent identical functions and the equation fails on B. Thus $p=q$.

We now need some special notation.

$$\sigma^i = F_{a_i}F_{a_{i+1}}\ldots F_{a_p} \qquad \tau^i = F_{b_i}F_{b_{i+1}}\ldots F_{b_p}\ .$$

$$\delta^i = \sigma^i + \tau^i \qquad \rho^i = \sigma^i - \tau^i$$

Now δ^i and ρ^i we consider as polynomials over the ring $\mathbb{Z}$ of integers. Note that

$$\deg(\sigma^i) = \deg(\delta^i) = \deg(\tau^i) = 2^{p-i+1}$$

for $i \leq p$. In this notation our equation is $B \models \sigma' = \tau'$ or equivalently $\mathbb{Z} \models \rho' = 0$.

Suppose that $\sigma' \not\equiv \tau'$. Let j be the largest index such that $a_j \neq b_j$. So $i>j$ implies $a_i = b_i$ and $\sigma^i \equiv \tau^i$. By induction on k we show that for $0 \leq k \leq j-1$ it is the case that $\mathbb{Z} \not\models \rho^{j-k} = 0$. Thus $\mathbb{Z} \not\models \rho' = 0$ which contradicts our assumption.

In the basis $k=0$ there are two cases $j=p$ and $j<p$.

If $j=p$ then $\rho^j = \sigma^j - \tau^j = (X^2 + a_p) - (X^2 + b_p) = a_p - b_p$ and by the assumption on $j=p$, $\mathbb{Z} \not\models \rho^p = 0$.

If $j<p$ then

$$\begin{aligned}
\rho^j = \sigma^j - \tau^i &= F_{a_j}\sigma^{j+1} - F_{b_j}\tau^{j+1} \\
&= ((\sigma^{j+1})^2 + a_j) - ((\tau^{j+1})^2 + b_j) \\
&= (\sigma^{j+1})^2 - (\tau^{j+1})^2 + a_j - b_j \\
&= a_j - b_j
\end{aligned}$$

because $\sigma^{j+1} = \tau^{j+1}$ by choice of j. Thus $\mathbb{Z} \not\models \rho^j = 0$.

In the induction step, let $\mathbb{Z} \not\models \rho^\ell = 0$ where $\ell = j-k$. We consider $\ell = j-(k+1) = \ell-1$.

$$\begin{aligned}
\rho^{\ell-1} = \sigma^{\ell-1} - \tau^{\ell-1} &= F_{a_{\ell-1}}\sigma^\ell - F_{b_{\ell-1}}\tau^\ell \\
&= ((\sigma^\ell)^2 + a_{\ell-1}) - ((\tau^\ell)^2 + b_{\ell-1}) \\
&= \delta^\ell \rho^\ell + a_{\ell-1} - b_{\ell-1}
\end{aligned}$$

Now $\mathbb{Z} \not\models \rho^j = 0$ implies

$$\deg(\delta^\ell \rho^\ell) \geq \deg(\delta^\ell) \geq 2^{p-\ell+1} \geq 2.$$

Hence $\deg(\rho^{\ell-1}) \geq 2$ and $\mathbb{Z} \not\models \rho^{\ell-1} = 0$.

This concludes the proof that $E_3 = \emptyset$.

Suppose $T(\Sigma, E_1) \cong A$. Let

$$E_k = \{F(S^n(0)) = S^{n^2}(0) : n \in \omega,\ n \leq k\} \quad .$$

Notice that if $i<j$ then $E_i \subset E_j$ and that A is an E_k-algebra for all k. We claim that for sufficiently large k_0, $E_{k_0} \vdash E_1$. This can be easily proved:

First define $\lambda : T(\Sigma) \to \omega$ such that

$$A \models t = S^{\lambda(t)}(0)$$

By induction, we define

$$\lambda(0) = 0$$

$$\lambda(S(t')) = \lambda(t')+1$$

$$\lambda(F(t')) = \lambda(t')^2$$

It is easy to check that this λ is uniquely determined.

<u>4.3 LEMMA</u> $E_{\lambda(t)} \vdash t = S^{\lambda(t)}(0)$

<u>PROOF</u> This is done by induction on t. The basis case t=0 is trivial. The induction step has two cases.

Let t=S(t'). Then, by the induction hypothesis,

$$E_{\lambda(t')} \vdash t' = S^{\lambda(t')}(0)$$
$$E_{\lambda(t)} \vdash t' = S^{\lambda(t')}(0)$$
$$E_{\lambda(t)} \vdash S(t) = S(S^{\lambda(t')}(0))$$
$$E_{\lambda(t)} \vdash t = S^{\lambda(t')+1}(0)$$
$$E_{\lambda(t)} \vdash t = S^{\lambda(t)}(0)$$

Let t=F(t'). Then, by the induction hypothesis,

$$E_{\lambda(t')} \vdash t' = S^{\lambda(t')}(0)$$
$$E_{\lambda(t)} \vdash t' = S^{\lambda(t')}(0)$$
$$E_{\lambda(t)} \vdash F(t') = F(S^{\lambda(t')}(0))$$
$$E_{\lambda(t)} \vdash t = S^{\lambda(t')^2}(0)$$
$$E_{\lambda(t)} \vdash t = S^{\lambda(t)}(0)$$

□

Choose $k_0 > \max\{|e| : e \in E_1\}$. Then for any $e \equiv t = t' \in E_1$

$$E_{k_0} \vdash t = S^{\lambda(t)}(0) \text{ and } E_{k_0} \vdash t' = S^{\lambda(t')}(0)$$

Since $A \models t = t'$ we know that $\lambda(t) = \lambda(t')$ and hence that $E_{k_0} \vdash t = t'$.

Since $E_{k_0} \vdash E_1$ we know that $T(\Sigma,E_{k_0})$ is an E_1-algebra.

Since $A \cong T(\Sigma,E_1)$ is an E_{k_0}-algebra we can conclude that

$$A \cong T(\Sigma,E_1) \cong T(\Sigma,E_{k_0})$$

We contradict this statement that A is initial in $ALG(\Sigma,E_k)$ by giving an E_{k_0}-structure into which A may not be homomorphically mapped. The structure is $A_{k_0} = (\omega : 0,x+1,g)$ where $g : \omega \to \omega$ is defined by

$$g(x) = \begin{cases} x^2 & \text{if } x<k_0 \\ k_0^2 & \text{otherwise.} \end{cases}$$

Any homomorphism $\phi : A \to A_{k_0}$ must satisfy $\phi(n) = n$; notice the homomorphism property fails as follows:

$$\begin{aligned} \phi f(k_0+1) &= g\phi(k_0+1) \\ \phi((k_0+1)^2) &= g(k_0+1) \\ (k_0+1)^2 &= k_0^2 \end{aligned}$$

Thus we have shown that $T(\Sigma,E_1) \not\cong A$ and we conclude that A does not possess a (FIN,EQ) specification.

□

On adequacy grounds, to be discussed in the next section, we have the following:

<u>4.4 COROLLARY</u> (FIN,EQ) *and hence* (FIN,SEQ) *specifications are not equivalent to the infinite specification methods.*

On examining the first part of the proof, it is easy to show the following:

<u>4.5 LEMMA</u> (FIN,EQ) *specifications are not equivalent to* (FIN,SEQ) *specifications.*

Next, let us turn to (FIN,CEQ) specifications. We will now give a simple computable algebra that cannot be defined by these specifications.

The problem is alluded to in ADJ [52], but not solved. The algebra was mentioned in Majster [38] as an example of a structure without a (FIN,EQ) specification, but a proof was not provided.

Consider the following two-sorted structure C_f based on the characteristic function

$$f : \omega \to \{true, false\}$$

of a set $S_f \subset \omega$ where

$$x \in S_f \Leftrightarrow f(x) = true.$$

The structure C_f has domains ω and {true,false} linked by f:

$$C_f = (\omega; \{true,false\} : 0, x+1, true, false, f)$$

The structure C_f is uniquely determined up to isomorphism by f:

4.6 LEMMA *The following conditions are equivalent :*

(i) $f = g$;

(ii) $S_f = S_g$;

(iii) $C_f = C_g$;

(iv) $C_f \cong C_g$;

(v) *there is a homomorphism* $\phi : C_f \to C_g$.

PROOF The cycle of implications from (i) to (v) are obvious. Suppose $\phi : C_f \to C_g$ is a homomorphism. Then $\phi(n) = n$ for all $n \in \omega$ because ϕ preserves 0 and successor. Thus for all n

$$f(n) = \phi(f(n)) = g(\phi(n)) = g(n)$$

and (v) implies (i). □

Consider the following *sparsity property* on f :

For any $k \in \omega$ *there exists* $x \in \omega$ *such that* $f(x) = true$ *and* $f(x-k),\ldots,f(x-1),f(x+1),\ldots,f(x+k) = false$.

Equivalently, for the set S_f

For any $k \in \omega$ *there exists* $x \in S_f$ *such that the interval* $[x-k,x+k] \cap S_f = \{x\}$.

For example, the set $\{n^2 : n \in \omega\}$ of squares satisfies this sparsity property. Less obviously,

4.7 LEMMA *The set P of prime numbers satisfies the sparsity property.*

PROOF A significant theorem about the increasing enumeration $p_0, p_1, p_2, \ldots$ of the primes is that for any n there exist (infinitely many) i such that

$$|p_i - p_{i+1}| > n \text{ and } |p_{i+1} - p_{i+2}| > n$$

see Theorem 6.1 in Prachar [45]. □

We note that if f has the sparsity property then:

For any $k \in \omega$ *there exists* $x \in \omega$ *such that* $f(x), f(x+1), \ldots$ $f(x+k) = \text{false}$.

Or equivalently

For any $k \in \omega$ *there exists* $x \in \omega$ *such that the interval* $[x, x+k] \cap S_f = \emptyset$.

4.8 THEOREM *Let* $f : \omega \to \{\text{true}, \text{false}\}$ *satisfy the sparsity property. Then the structure* C_f *fails to possess a* (FIN,CEQ) *specification.*

PROOF Suppose for a contradiction there exists a finite conditional equation specification (Σ, E) for C_f so that $T(\Sigma, E) \cong C_f$. Let K be the class of all characteristic function structures

$$K = \{C_g : g : \omega \to \{\text{true}, \text{false}\}\} .$$

We claim that C_f is the only structure in K that satisfies all the equations in E. For suppose that $C_g \models E$ then since C_f is initial in ALG(Σ, E) there must exist a homomorphism $\psi : C_f \to C_g$. By Lemma 4.6, $C_f = C_g$.

Now define $\phi = \bigwedge_{e \in E} e$. We know that C_f is the only structure in K that satisfies ϕ. This property we will seek to contradict.

The open formula ϕ can be built up using $\neg$ and $\vee$ from equations over the two sorts of numbers and booleans. An equation $t_1 = t_2$ over booleans is equivalent to

$$(t_1 = \text{FALSE} \wedge t_2 = \text{FALSE}) \vee (t_1 = \text{TRUE} \wedge t_2 = \text{TRUE})$$

and hence we may assume that the atomic formulae of ϕ are either equations over ω, or equations over booleans having one of the forms t=FALSE or t=TRUE and that there are no variables of type booleans. The atomic formulae are therefore of the form:

$$
\begin{array}{llll}
s^n(0) = s^m(0) & & F\,s^n(0) = F\,s^m(0) \\
s^n(0) = s^m(X) & & F\,s^n(0) = F\,s^m(X) \\
s^n(X) = s^m(X) & & F\,s^n(X) = F\,s^m(X) \\
s^n(X) = s^m(Y) & & F\,s^n(X) = F\,s^m(Y)
\end{array}
$$

Let ϕ contain the numerical variables $X_1,\ldots,X_n$ and have length ℓ.

We will now construct a $g : \omega \to \{\text{true},\text{false}\}$ such that $C_g \models \phi$ and $C_f \neq C_g$.

Since f satisfies the sparsity assumption we can choose $z\in\omega$ such that $f(z) = \text{true}$ but for all $x\in[z-4\ell n,\ z+4\ell n]$ if $x\neq z$ then $f(x) = \text{false}$.

Now we define

$$
g(x) = \begin{cases} f(x) & \text{if } x\neq z \\ \text{false} & \text{if } x=z \end{cases}
$$

Thus f,g differ only at z. This has the following implications for valuations $\rho : \{X_1,\ldots,X_n\} \to \omega$ of ϕ :

4.9 LEMMA *If for each* i=1,...,n

$$|\rho(X_i) - z| > \ell$$

then

$$C_f,\rho \models \phi \quad \textit{if, and only if,} \quad C_g,\rho \models \phi \quad .$$

PROOF We prove this by induction on ϕ. The 8 cases of atomic formulae follow a similar pattern : we consider

$$\phi \equiv FS^a(X_i) = FS^b(X_j)$$

and show that

$$C_f,\rho \models \phi \qquad \text{implies} \qquad C_g,\rho \models \phi$$

$$C_f,\rho \not\models \phi \qquad \text{implies} \qquad C_g,\rho \not\models \phi$$

Now $C_f,\rho \models \phi$ entails that

$$f(s^a(\rho(X_i))) = f(s^b(\rho(X_j)))$$

where $s(x) = x+1$.

As $|\rho(X_i)-z| > \ell$ and $a<\ell$, and $|\rho(X_j)-z| > \ell$ and $b<\ell$,

$$s^a(\rho(X_i)) \neq z \text{ and } s^b(\rho(X_j)) \neq z \,.$$

Hence,

$$\begin{aligned} g\, s^a\,(\rho(X_i)) &= f\, s^a(\rho(X_i)) && \text{by definition of } g \\ &= f\, s^b(\rho(X_j)) && \text{by equation} \\ &= g\, s^b(\rho(X_j)) \end{aligned}$$

and $C_g,\rho \models \phi$. The second argument is similar.

The induction steps for $\vee$ and $-$ are easy.

□

Since $f \neq g$, we have that $C_f \neq C_g$ and $C_g \not\models \phi$ (remember the assumptions on ϕ). Thus, there exists a valuation $\sigma : \{X_1,\ldots,X_n\} \to \omega$ such that $C_g,\sigma \models -\phi$. In view of the difference between f and g, we may expect the elements $V = \{\sigma(X_1),\ldots,\sigma(X_n)\}$ to be "near" z. We will construct another valuation τ that mainly coincides with σ but which changes values in a "small" interval around z to larger values in a "small" interval higher up such that

$$C_g,\sigma \models -\phi \quad \text{implies} \quad C_g,\tau \models -\phi$$

and Lemma 4.9 can be applied to τ to yield

$$C_g,\tau \models -\phi \quad \text{implies} \quad C_f,\tau \models -\phi$$

This done we note that $C_f \not\models \phi$, which is a contradiction.

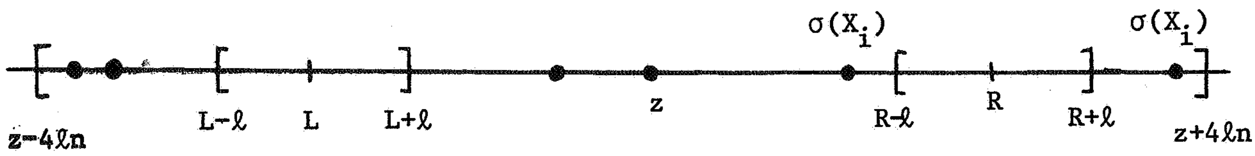
To construct τ we first find two numbers L,R such that

(i) $L \in [z-4\ell n, z]$ $\qquad$ $R \in [z,\, z+4\ell n]$

(ii) $z-4\ell n < L-\ell$ $\qquad$ $L+\ell < z$

(iii) $z < R-\ell$ $\qquad$ $R+\ell < z+4\ell n$

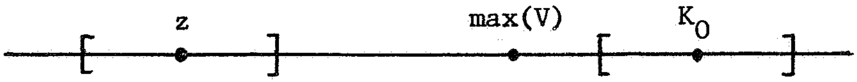
(iv) $[L-\ell,\, L+\ell] \cap V = \emptyset$ $\qquad$ $[R-\ell,\, R+\ell] \cap V = \emptyset$



Suppose no such L at the centre of an interval of length ℓ existed then condition (iv) implies there must be $4\ell n/2\ell = 2n$ elements of V within $[z-4\ell n, z]$.

By sparsity, there is a number $K_0 > \max(V) + 4\ell n + z$ such that the interval $[K_0-4\ell n, K_0+4\ell n]$ does not contain elements x with $f(x) =$ true nor elements of V. Set $d = K_0 - z$. Then define valuation

$$\tau(X_i) = \begin{cases} \sigma(X_i)+d & \text{if } \sigma(X_i) \in [L,R] \\ \sigma(X_i) & \text{if } \sigma(X_i) \notin [L,R] \end{cases}$$



4.10 LEMMA *For any open formula ψ of length $\le \ell$ and with variables among $X_1,\ldots,X_n$ we have*

$$A_g,\sigma \models \psi \textit{ if, and only if, } A_g,\tau \models \psi$$

PROOF This is shown by induction on the structure of ψ. The basis case divides into subcases determined by the atomic formulae.

Consider $S^a(X_i) = S^b(X_j)$. If $\sigma(X_i)$ and $\sigma(X_i)$ are outside $[L,R]$ then σ and τ agree on X_i and X_j and we are done:

$$A_g,\sigma \models S^a(X_i) = S^b(X_j) \text{ and}$$
$$A_g,\tau \models S^a(X_i) = S^b(X_j) .$$

In the case $\sigma(X_i) \in [L,R]$ and $\sigma(X_j) \notin [L,R]$ it is the case that

$$A_g,\sigma \not\models S^a(X_i) = S^b(X_j) \text{ and}$$
$$A_g,\sigma \not\models S^a(X_i) = S^b(X_j)$$

To see this note that $\sigma(X_j) \notin [L-\ell, R+\ell]$ and so $|\sigma(X_i) - \sigma(X_j)| > \ell$. But $A_f,\sigma \models S^a(X_i) = S^b(X_j)$ implies

$$|\sigma(X_i) - \sigma(X_j)| \le a-b \le a+b .$$

Since $a+b < \ell$ this equation cannot hold.

Concerning $A_g,\tau \not\models S^a(X_i) = S^b(X_j)$. We note that $|\tau(X_i) - \tau(X_j)| > \ell$ because $\tau(X_i) = \sigma(X_i)$ and

$$\tau(X_j) = \sigma(X_j)+d = \sigma(X_j) + \max(V) + 4\ell n+z > \sigma(X_i) + \ell$$

By the same reasoning we can deduce the equation does not hold.

The other cases of atomic formulae follow similarly and the induction steps are obvious. □

We have shown that for the constructed τ

$$C_g,\sigma \models -\phi \quad \text{implies} \quad C_g,\tau \models -\phi$$

Since $|\tau(X_i) - z| > \ell$ for each i we can apply the Lemma 4.9 to conclude

$$C_g,\tau \models -\phi \quad \text{implies} \quad C_f,\tau \models -\phi$$

which is the desired contradiction, as explained earlier. □

Again, on adequacy grounds, we can deduce the following improvement to Corollary 4.4.

4.11 COROLLARY (FIN,CEQ) *specifications are not equivalent to the infinite r.e. specification methods.*

The non-equivalence with the infinite recursive specification methods follows in Section 6.

The fact that (FIN,CEQ) and (FIN,EQ) specifications are not equivalent was established in ADJ [52] using a rather simple, if artificial, type. In Bergstra and Meyer [2] a natural example of a data type of *sets-of-integers* is shown to have a (FIN,CEQ) specification, but not to have a (FIN,EQ) specification.

5. ADEQUACY AND COMPLETENESS THEOREMS FOR SPECIFICATIONS WITH HIDDEN MECHANISMS

In this section we will prove that the (FIN,EQ,HE) specifications are adequate for the computable data types and that the (FIN,EQ,HES) specifications are complete for the semicomputable data types. We will use some fairly elementary results from the theory of recursive functions and present proofs in some detail in order to establish properly the translation of ideas of computability to ideas of algebra.

As with the situation concerning hidden functions, outlined in the previous section, there have been some observations concerning effective calculability and the power of methods already. In Guttag [21] and Guttag and Horning [22], the definition of the partial recursive functions by

Herbrand-Gödel-Kleene equations is claimed to establish adequacy for their methods. However a considerable amount of work, particularly on the technical foundations of their specification methods, is necessary to establish that fact. A puzzle arises in their claim, however : the semantics of Herbrand-Gödel-Kleene equations is that of an operational rewrite rule system and hence ought naturally lead to an initial algebra semantics for equations; but Guttag and Horning deny such a semantics is intended for their methods.

In Majster [38], a similar sentiment concerning Herbrand-Gödel-Kleene computability and finite equations and hidden functions is expressed. Again considerable work is required to develop the initial algebra semantics of specifications for partial types, which Majster's interpretation clearly involves, and to develop the necessary computability theory for data types as we have here, in the case of types with total operations. Later in our series [1] we considered computable data types with partial functions.

For simplicity, the theorems will be proved in the case of single-sorted data types only. The many-sorted generalisations are indeed true, but we prefer to follow the usual practice of our series of leaving the generalisation to the reader. However, in [11] it was expedient to give an account of an interesting relationship between the single-sorted and many-sorted cases of computable data types which can be of help here.

5.1 THEOREM *Let* A *be a single-sorted minimal algebra of signature* Σ. *If* A *is computable then* A *has a* (FIN,EQ,HE) *specification.*

PROOF The case that A is finite is accounted for by Proposition 1.12. Suppose A is infinite.

By the Representation Lemma 3.5, A is isomorphic to a recursive algebra R of numbers, say

$$R = (\omega;\ c_1,\ldots,c_n,\ f_1,\ldots,f_m)$$

where the $c_i \in \omega$ and the f_i are recursive functions on ω. R is minimal, of course. We will show that R has a (FIN,EQ,HE) specification by constructing an algebra R' having a (FIN,EQ) specification and such that

$$R'|_\Sigma = \langle R'\rangle_\Sigma \cong R\ .$$

First we will prove the following technical fact :

5.2 LEMMA *Let* $g_1,\ldots,g_m$ *be primitive recursive functions and let* $\lambda_1,\ldots,\lambda_\ell$ *be the functions appearing in their explicit definitions. Then the algebra*

$$B = (\omega;\ 0, x+1, \lambda_1,\ldots,\lambda_\ell,\ f_1,\ldots,f_m)$$

has a (FIN,EQ) *specification.*

PROOF Without loss of generality, we can assume that the operations of B are ordered in a list

$$0, x+1, \theta_1,\ldots,\theta_{\ell+m}$$

so that any function is to the right of all those functions appearing in its explicit definition.

Define a sequence of algebras

$$B_0 = (\omega;\ 0, x+1)$$

$$B_{n+1} = (A_n,\ \theta_{n+1})$$

for $n=0,\ldots,\ell+m$. We will prove that each B_n has a (FIN,EQ) specification and so, in particular, $A=B_{\ell+m}$ has such a specification.

The base of the sequence is obvious : let $\Sigma_0 = \{0,S\}$ and $E_0=\emptyset$ so $B_0 \cong T(\Sigma_0)$.

Assume that B_n has a (FIN,EQ) specification (Σ_n,E_n) so that $B_n = T(\Sigma_n,E_n)$, and consider B_{n+1}. By the construction of the list, the new function θ_{n+1} is either a projection function, or is defined by composition or primitive recursion from earlier θ_i,θ_j with $i,j < n+1$. These three cases are treated in like manner so we will write out the case of primitive recursion, only.

Suppose

$$\theta_{n+1}(0,x_1,\ldots,x_k) = \theta_i(x_1,\ldots,x_k)$$

$$\theta_{n+1}(y+1,x_1,\ldots,x_k) = \theta_j(y,x_1,\ldots,x_k,\ \theta_{n+1}(y,x_1,\ldots,x_k))\ .$$

Then set $\Sigma_{n+1} = \Sigma_n \cup \{\underline{\theta}_{n+1}\}$ and E_{n+1} to be E_n with these equations adjoined

$$\underline{\theta}_{n+1}(0,X_1,\ldots,X_k) = \underline{\theta}_i(X_1,\ldots,X_k)$$

$$\underline{\theta}_{n+1}(S(Y),X_1,\ldots,X_k) = \underline{\theta}_j(Y,X_1,\ldots,X_k,\ \underline{\theta}_{n+1}(Y,X_1,\ldots,X_k))\ .$$

Clearly, (Σ_{n+1},E_{n+1}) is a (FIN,EQ) specification so we must show that $T(\Sigma_{n+1},E_{n+1}) \cong B_{n+1}$. We use Lemma 2.9. We know that $B_{n+1}|_{\Sigma_n} = B_n$ and that $B_n \cong T(\Sigma_n,E_n)$ so we must verify that

$$T(\Sigma_{n+1},E_{n+1})|_{\Sigma_n} \cong T(\Sigma_n,E_n)$$

to apply Lemma 2.9. For this we can use Lemma 2.8.

Consider $T = \{S^r(0) : r\epsilon\omega\}$. Now T is a transversal for $T(\Sigma_n,E_n)$ because

$$T(\Sigma_n,E_n)|_{\Sigma_0} \cong B_n|_{\Sigma_0} = B_0 \cong T(\Sigma_0) \ .$$

Condition (i) of Lemma 2.8 is fulfilled by E_{n+1} because B_{n+1} is an E_{n+1} algebra. Since condition (ii) is automatic, we are left with condition (iii). This condition is checked by considering

$$\underline{\theta}_{n+1}(S^r(0),\ S^{r_1}(0),\ldots,S^{r_k}(0))$$

and showing that it is E_{n+1} equivalent to an element of T going by the equations for $\underline{\theta}_{n+1}$ to elements of $T(\Sigma_n)$ in which T is an $E_n \subset E_{n+1}$ transversal.

□

We will now construct R' from R. Let $f : \omega^k \to \omega$ be a recursive function. Then the graph of f

$$\mathrm{graph}(f) = \{(x_1,\ldots,x_k,f(x_1,\ldots,x_k)) : x_1,\ldots,x_k\epsilon\omega\}$$

is recursively enumerable. Since every r.e. set has a primitive recursive enumeration, let $h_1,\ldots,h_k$, $g : \omega \to \omega$ be primitive recursive functions enumerating graph(f). Thus,

$$\mathrm{graph}(f) = \{(h_1(z),\ldots,h_k(z),g(z)) : z \in \omega\}$$

and, in particular, for all $z\epsilon\omega$

$$f(h_1(z),\ldots,h_k(z)) = g(z) \ .$$

Now for each k_j-ary recursive operation f_j of R choose primitive recursive functions

$$h_1^j,\ldots,h_{k_j}^j \text{ and } g^j$$

that enumerate graph(f_j) as above. Let $\{\lambda_{ij}\}$ and $\{\mu_j\}$ be the *lists* of functions making up the explicit definitions of the h_i^j and g^j, respectively. Define

$$R' = (\omega;\ 0,x+1,\{\lambda_{ij}\},\{\mu_j\},\ h_1^j,\ldots,h_{k_j}^j,g^j,f_j,c_1,\ldots,c_n) \quad 1\le j\le m,\ 1\le i\le k_j$$

Clearly, $R'|_\Sigma = \langle R'\rangle_\Sigma = R$. We have to show that R' has a (FIN,EQ) specification.

First set

$$R_0' = (\omega;\ 0, x+1, \{\lambda_{ij}\}, \{\mu_j\},\ h_1^j, \ldots, h_{k_j}^j, g^j)_{1\le j\le m,\ \ 1\le i\le k_j}$$

and let its signature be Σ_0'. Then

$$R|_{\Sigma_0'} = \langle R\rangle_{\Sigma_0'} = R_0'$$

and by Lemma 5.2, R_0' has a (FIN,EQ) specification (Σ_0', E_0').

We now define a specification for R'. Let Σ' be the signature of R' so that $\Sigma' = \Sigma_0' \cup \Sigma$. Let E' be E_0' with the following equations added :

for each constant $\underline{c}_j \in \Sigma$, $\underline{c}_j = S^{c_j}(0)$

for each operation $\underline{f}_j \in \Sigma$,

$$\underline{f}_j(\underline{h}_1^j(X), \ldots, \underline{h}_k^j(X)) = \underline{g}^j(X) \ .$$

The pair (Σ', E') is a (FIN,EQ) specification so we must verify that $T(\Sigma', E') \cong R'$. This is done by Lemma 2.9.

Clearly, R' is an E'-algebra so all that remains is the hypothesis $T(\Sigma', E')|_{\Sigma_0'} \cong T(\Sigma_0', E_0')$. For this we look to Lemma 2.8.

Consider $T = \{S^r(0) : r \in \omega\}$. That T is a transversal for $T(\Sigma_0', E_0')$ follows from the fact that

$$T(\Sigma_0', E_0')|_{\{0,S\}} \cong R_0'|_{\{0,S\}} \cong T(\Sigma_0) \ .$$

Conditions (i) and (ii) of Lemma 2.8 are true of T by inspection of E, which leaves condition (iii). So consider the term

$$\underline{f}(S^{r_1}(0), \ldots, S^{r_k}(0))$$

The isomorphism between $T(\Sigma_0', E_0')$ and R_0' implies there is an $S^z(0)$ such that

$$S^{r_i}(0) \equiv_{E_0'} \underline{h}_i S^z(0) \qquad \text{for } 1 \le i \le k \ .$$

Thus,

$$\underline{f}(S^{r_1}(0), \ldots, S^{r_k}(0)) \equiv_{E'} \underline{f}(\underline{h}_1 S^z(0), \ldots, h_k S^z(0))$$

$$\equiv_{E'} \underline{g} S^z(0) \ .$$

Since $\underline{g}S^z(0) \in T(\Sigma_0')$ and T is an E_0'-transversal,

$$\underline{g}S^z(0) \equiv_{E_0'} S^i(0)$$

for some i whence the condition follows as $\equiv_{E_0'} \subset \equiv_{E'}$. □

Next we turn to the only completeness theorem we know for the semicomputable data types that concerns finite specifications. The use of hidden sorts we first saw in Subrahmanyam [49].

5.3 THEOREM *Let* A *be a single-sorted minimal algebra of signature* Σ. *If* A *is semicomputable then* A *has a* (FIN,EQ,HES) *specification.*

PROOF We consider the case when A is infinite. Since A is semicomputable we can choose a recursive algebra

$$R = (\omega;\ c_1,\ldots,c_n,\ f_1,\ldots,f_m)$$

of numbers and a canonical codification $\gamma_A : R \to A$ with $\equiv_{\gamma_A}$ r.e. (by the Representation Lemma 3.11). Recall from Section 3 that γ_A is factored thus :

$$\begin{tikzcd} T(\Sigma) \arrow[r, "v"] & A \\ R \arrow[u, "\gamma_*"] \arrow[ur, "\gamma_A"'] & \end{tikzcd}$$

and that $S_A = \{\gamma_*(i) = \gamma_*(j) : (i,j) \in \equiv_{\gamma_A}\}$ is the set of all simple equations true in A.

Since $\equiv_{\gamma_A}$ is r.e. we can choose primitive recursive functions g,h to enumerate it so that $\equiv_{\gamma_A} = \{(g(z),\ h(z)) : z\in\omega\}$ and hence

$$S_A = \{\gamma_*(g(z)) = \gamma_*(h(z)) : z\in\omega\} .$$

Adjoin the functions to R to make (R,g,h) denoted R_0, with signature Σ_0.

Consider next the new two-sorted structure of signature Γ

$$B = (A,\omega;\ a_1,\ldots,a_n,\ \sigma_1,\ldots,\sigma_m,\ c_1,\ldots,c_n,\ f_1,\ldots,f_m,g,h,\gamma)$$

Clearly, $B|_\Sigma = A$ with nat the sort name for ω and nat $\notin \Sigma$. We shall prove the theorem by showing that B has a (FIN,EQ,HE) specification.

Since R_0 is computable it has a (FIN,EQ,HE) specification by Theorem 5.1. More precisely, from the argument of Theorem 5.1, there is a new recursive number algebra R_0' and a (FIN,EQ) specification (Σ_0', E_0') such that

$$T(\Sigma_0', E_0') \cong R_0'$$

$$R_0'|_{\Sigma_0} = \langle R_0' \rangle_{\Sigma_0} = R_0$$

$$R_0'|_{\{0,S\}} = (\omega;\ 0, x+1)$$

Define B' to be B with all the new operations added to R_0 to make R_0'; let B' have signature Γ'. Clearly $B'|_\Gamma = B$.

Now $\Sigma \subset \Gamma'$, $\Sigma_0 \subset \Sigma_0' \subset \Gamma'$ and $\{0,S\} \subset \Gamma'$.

We show that B' has a (FIN,EQ) specification (Γ', F).

Define F to be E_0' together with the following equations over Γ' :

$$\underline{\gamma}(\underline{c}_i) = \underline{a}_i$$

$$\underline{\gamma}(\underline{f}_i(X_1, \ldots, X_{k_i})) = \underline{\sigma}(\underline{\gamma}(X_1), \ldots, \underline{\gamma}(X_{k_i}))$$

$$\gamma\underline{g}(X) = \gamma\underline{h}(X)$$

Clearly, (Γ', F) is a (FIN,EQ) specification. To prove that

$$T(\Gamma', F) \cong B'$$

we take two steps.

First we claim $T(\Gamma', F)$ is an S_A-algebra so that $T(\Gamma', F) \cong T(\Gamma', F \cup S_A)$.

Secondly, we claim $B' \cong T(\Gamma', F \cup S_A)$.

Consider the second claim first. B' is an $(F \cup S_A)$-algebra so, by initiality and the fact that B' is minimal, there is a unique epimorphism $\phi : T(\Gamma', F \cup S_A) \to B'$. To check that ϕ is injective we split ϕ into ϕ_1, ϕ_2

$$\phi_1 = \phi \restriction T(\Gamma', F \cup S_A)|_\Sigma \quad \text{and} \quad \phi_2 = \phi \restriction T(\Gamma', F \cup S_A)|_{\Sigma_0'}$$

Now $T(\Gamma', F \cup S_A)|_\Sigma$ is an S_A-algebra and

$$\phi_1 : T(\Gamma', F \cup S_A)|_\Sigma \to B'|_\Sigma \quad .$$

Now $B'|_{\Sigma} = A \cong T(\Sigma,S_A)$. Hence $T(\Gamma', F \cup S_A)|_{\Sigma}$ is initial for S_A-algebras and

$$T(\Gamma, F \cup S_A)|_{\Sigma} \cong B'|_{\Sigma}$$

by ϕ. The injectivity of ϕ_2 follows the same lines.

Finally, consider the first claim. Observe $\{S^r(0) : r\epsilon\omega\}$ is a transversal for $T(\Sigma_0',E_0')$ so that

$$\underline{g}(S^r(0)) \equiv_{E_0'} S^{f(r)}(0)$$

$$\underline{h}(S^r(0)) \equiv_{E_0'} S^{g(r)}(0)$$

since $T(\Sigma_0',E_0')|_{\Sigma_0} \cong R_0$. Moreover one may now use the equations given for F to show that

$$\underline{\gamma}(S^r(0)) \equiv_F \gamma_*(r)$$

by induction on the complexity of terms. From these observations,

$$\begin{aligned} \gamma_*(g(z)) &\equiv_F \underline{\gamma}(S^{g(z)}(0)) \\ &\equiv_F \underline{\gamma}(\underline{g}S^z(0)) \\ &\equiv_F \underline{\gamma}(\underline{h}S^z(0)) \\ &\equiv_F \underline{\gamma}(S^{h(z)}(0)) \\ &\equiv_F \gamma_*(h(z)) \end{aligned}$$

whence $T(\Gamma',F)$ is an S_A-algebra.

□

With these results we have almost completed our work on Figure B in Section 2.

6. COMPLETING THE CLASSIFICATION

Some four results are necessary to complete the analysis of specification methods, without hidden machinery, represented in Figure A; and one result is outstanding for Figure B. We begin with completeness issues.

6.1 PROPOSITION *Let* A *be a semicomputable minimal algebra of signature* Σ. *Then* A *has a* (REC,CEQ) *specification.*

PROOF By the Completeness Lemma 3.12, A has an (RE,SEQ) specification (Σ,E). Let E be enumerated by f so that $f(i) = e_i$. Let c be a constant symbol for Σ and define E_c to be the set of all conditional equations of the form

$$\underbrace{c = c \wedge \ldots \wedge c = c}_{i\text{-times}} \to e_i$$

for $i \in \omega$. Clearly,

$$T(\Sigma,E_c) = T(\Sigma,E) \cong A$$

But E_c is a recursive set of axioms, for given any conditional equation $e \equiv e_1 \wedge \ldots \wedge e_n \to e'$ one first decides whether or not the e_j are c=c : if not then $e \notin E_c$. If the e_j are c=c then one computes $f(n) = e_n$ and checks whether or not e_n is e'.

□

6.2 PROPOSITION *Let* A *be a semicomputable minimal algebra of signature* Σ. *Then* A *has a* (REC,SEQ,HE) *specification.*

PROOF By the Completeness Lemma 3.12, A has an (RE,SEQ) specification (Σ,E). Let $E = \bigcup_s E_s$ is the set of equations of sort $s \in \Sigma$. Let f_s and g_s be recursive functions that enumerate E_s so that

$$E_s = \{f_s(i) = g_s(i) : i \in \omega\} .$$

Now for each sort s we adjoin to Σ a new function symbol I_s to make a new signature Σ'. We define E'_s to be E_s with the following simple equations adjoined

$$I_s(t) = t \qquad \text{for each } t \in T(\Sigma)$$

$$\underbrace{I_s \ldots I_s}_{i \text{ times}}(f_s(i)) = g_s(i) \quad \text{for each } i \in \omega$$

Now $E' = \bigcup_s E'_s$ is a recursive set of simple equations over Σ', by reasoning analogous to that in the previous result. Clearly,

$$T(\Sigma',E')|_\Sigma \cong T(\Sigma,E) \cong A.$$

□

From this result we can obtain a simple counter-example to the converse of 3.13.

6.3 COROLLARY *There are* (REC,SEQ) *specifications that define non-computable algebras.*

PROOF Choose A to be semicomputable, but not computable, and apply the above constructions to it. The algebra $T(\Sigma',E')$ is not computable. □

Next we will prove that the (REC,EQ) specifications are not complete. Let Σ be the following signature:

sorts	nat	s
constants	$0 : \text{nat}$	
functions	$S : \text{nat} \to \text{nat}$	
	$F : \text{nat} \to s$	
	$G : \text{nat} \to s$	

Let $W \subset \omega$ and define

$$E_W = \{F(S^n(0)) = G(S^n(0)) : n \in W\}$$

Set $A_W = T(\Sigma, E_W)$. We have axiomatised the equality of functions on a given set of numbers:

6.4 LEMMA *The congruence* $\equiv_{E_W}$ *is the set*

$$\{(FS^n(0),\ GS^n(0)) : n \in W\} \cup \{(GS^n(0),\ FS^n(0)) : n \in W\}$$
$$\cup \{(t,t) : t \in T(\Sigma)\}$$

Thus, for $n \in \omega$,

$$n \in W \textit{ if, and only if, } FS^n(0) \equiv_{E_W} GS^n(0)\ .$$

In consequence, for $W,\ Z \subset \omega$

$$A_W \cong A_Z \textit{ if, and only if, } W = Z.$$

PROOF Let $\equiv$ denote the set defined above. It is easy to show that $\equiv$ is a congruence. Since $T(\Sigma)/\equiv$ satisfies E_W we note that $\equiv$ is an E_W-congruence and that $\equiv_{E_W} \subset \equiv$, since $\equiv_{E_W}$ is the least E_W-congruence. Conversely, it is easy to check that $\equiv \subset \equiv_{E_W}$. The other properties are immediate. □

Notice that the equivalence classes of $\equiv_{E_W}$ are all finite and contain either one or two elements.

6.5 THEOREM *Let W be an r.e. non-recursive set. Then the algebra* A_W *is semicomputable but it fails to possess a* (REC,EQ) *specification.*

PROOF The algebra is clearly semicomputable on account of Lemma 3.12: the congruence $\equiv_{E_W}$ is an r.e. set of simple equations that specifies A_W.

Suppose for a contradiction that there was a specification (Σ,E) with E a recursive set of non-trivial equations such that $T(\Sigma,E) \cong A_W$ i.e. $\equiv_E$ is $\equiv_{E_W}$. Let $E = E_1 \cup E_2$ where E_1 is the subset of all simple equations in E. We first show that $E_2 = \phi$.

Let $t_1=t_2$ be a non-trivial element of E_2. Depending upon the occurrences of free variables in the equation, there are three possibilities: an equation of one of the following forms is valid in A_W:

$$t_1 = t_2(X) \qquad t_1(X) = t_2(X) \qquad t_1(X) = t_2(Y)$$

(remember the operations in Σ are unary).

If $t_1(X) = t_2(Y)$ is valid then setting Y=0 gives that $\{t_1(r) : r\epsilon T(\Sigma)\}$ is a subset of $[t_2(0)]_E$ which contradicts the finiteness of the equivalence classes. Thus no such equations are in E.

If $t_1(X) = t_2(X)$ is valid then choose $n\epsilon W$ and substitute $FS^n(0)$

$$t_1FS^n(0) \equiv_E t_2FS^n(0) \equiv_E t_1GS^n(0) \equiv_E t_2GS^n(0)$$

this yields an equivalence class with four elements which is not possible. (Note if $W=\emptyset$ then it is recursive.)

Finally, if $t_1=t_2(X)$ is valid then again the set $[t_1]_E$ must contain the infinite set $\{t_2(r) : r\epsilon T(\Sigma)\}$, which is not possible.

Thus, $T(\Sigma,E_1) \cong A_W$ and the simple equations in E_1 must have the form

$$FS^n(0) = GS^n(0) \quad \text{or} \quad GS^n(0) = FS^n(0)$$

with $n\epsilon W$. Without loss of generality we may take

$$E_1 = \{FS^n(0) = GS^n(0) : n\epsilon Z \subset W\}.$$

But this means that $T(\Sigma,E_1)$ is A_Z (by definition). Since Z is recursive and W is not recursive we know that $Z \neq W$ and, by Lemma 6.4, that $A_Z \not\cong A_W$. This contradiction of $T(\Sigma,E_1) \cong A_W$ completes the proof. □

We will now prove that the (FIN,CEQ) specifications are not comparable with the (REC,EQ) specifications (recall Theorem 4.8).

6.6 THEOREM *There is a semicomputable algebra* A *that possesses a* (FIN,CEQ) *specification but fails to possess a* (REC,EQ) *specification.*

PROOF The construction of the algebra A is complicated and involves certain constructions made earlier. First, let W be an r.e. non-recursive set and let A_W be the two-sorted algebra made for Theorem 6.5 : $A_W = T(\Sigma, E_W)$.

Let $h : \omega \to \omega$ be a recursive function that enumerates W and define the single-sorted structure

$$B_W = (\omega, 0, x+1, h)$$

with signature Σ_1. By Theorem 5.1, since B_W is computable, there exists a (FIN,EQ,HE) specification (Σ_2, E_2) such that

$$T(\Sigma_2,E_2)|_{\Sigma_1} = \langle T(\Sigma_2,E_2)\rangle_{\Sigma_1} \cong B_W .$$

Let $C_W = T(\Sigma_2,E_2)$. Thus C_W is a computable algebra with a (FIN,EQ) specification.

We will join the independent structures A_W and C_W by means of a map $\phi : A_W \to C_W$ that identifies their independent copies of ω. This results in the structure D_W which is the algebra required in the theorem. In constructing D_W we will work with specifications.

First we assume that $\Sigma \cap \Sigma_2 = \emptyset$. Let N and $\hat{N}$ denote the copies of the natural numbers in A_W and C_W respectively.

Let D_W^- be the join $[A_W, C_W]$ of the two structures; by the Join Lemma 1.13,

$$D_W^- = T(\Sigma \cup \Sigma_2, E_W \cup E_2) .$$

To identify N and $\hat{N}$ we take transversals

$$\{S^n(0) : n\in\omega \text{ and } \{\hat{S}^n(\hat{0}) : n\in\omega\}$$

for N and $\hat{N}$ and define map $\phi : N \to \hat{N}$ by

$$\phi([S^n(0)]) = [\hat{S}^n(\hat{0})].$$

The map ϕ is added to D_W^- as a new operation to make $D_{W,\phi}^-$. By the Function Lemma 1.16, this algebra is axiomatised by adding a new function symbol Φ to $\Sigma \cup \Sigma_2$ and equations

$$E_\phi = \{\Phi(S^n(0)) = \hat{S}^n(\hat{0}) : n\in\omega\}$$

to $E_W \cup E_2$. For $\Sigma_3 = \Sigma \cup \Sigma_2 \cup \{\Phi\}$ and $E_3 = E_W \cup E_2 \cup E_\phi$ we have $D_{W,\phi}^- \cong T(\Sigma_3,E_3)$.

We take $D_W = T(\Sigma_3,E_3)$. We claim that D_W is a structure satisfying the properties of the theorem.

6.7 LEMMA D_W *possesses a* (FIN,CEQ) *specification.*

PROOF The infinite set E_3 of specifying equations for D_W is made up of infinitely many axioms E_W for A_W, finitely many axioms E_2 for C_W and infinitely many axioms E_ϕ for the linking of N, $\hat{N}$. Leaving E_2 alone, we make new sets

$$E_\phi^0 = \{\Phi(0) = \hat{0}$$

$$\Phi(S(X)) = \hat{S}(\Phi(X))\}$$

$$E_W^0 = \{\Phi(X) = H(Y) \to F(X) = G(X)\}$$

and define $E_3^0 = E_W^0 \cup E_2 \cup E_\phi^0$. We claim that $D_W \cong T(\Sigma_3,E_3^0)$.

We will prove this by means of the Refinement Lemma 1.15. This requires us to know the following three conditions :

(a) $E_3^0 \vdash E_3$

(b) $D_W \cong T(\Sigma_3,E_3)$

(c) $D_W \models E_3^0$

Of course, (b) holds by definition. We consider (c).

Now $D_W \models E_2$ because of (b) and $D_W \models E_\phi^0$ by inspection. Consider $D \models E_W^0$.

Let σ be any valuation of the free variables of the equation in E_W^0; say $\sigma(X) = [S^n(0)]$, $\sigma(Y) = [\hat{S}^m(\hat{0})]$. And suppose $D_W,\sigma \models \Phi(X) = H(Y)$. Then

$$D_W \models \Phi(S^n(0)) = H(\hat{S}^m(\hat{0})) \qquad \text{by inspection}$$

$$D_W = \hat{S}^n(\hat{0}) = H(\hat{S}^m(\hat{0})) \qquad \text{by } D_W \models E_\phi \ .$$

By the construction of (the component algebra B_W in) D_W, it may be checked that

$$D_W \models H(\hat{S}^m(0)) = \hat{S}^{h(m)}(\hat{0})$$

and hence $n=h(m)$ and $n\in W$. We must now verify that

$$D_W,\sigma \models F(X) = G(X) \text{ i.e. } D_W \models F(S^n(0)) = G(S^n(0)) \ .$$

This is true by virtue of E_W in the specification of D_W. This concludes our check of (c).

Consider (a). Since $E_2 \subset E_3^0$ we have that $E_3^0 \vdash E_2$. To show that $E_3^0 \vdash E_\phi$ is a matter of showing that $E_\phi^0 \vdash E_\phi$ by induction. Thus it remains to show $E_3^0 \vdash E_W$.

Let $F(S^n(0)) = G(S^n(0)) \in E_W$. Then $n \in W$ and there is $m \in \omega$ such that $h(m) = n$. We have

$$B_W \models \hat{S}^n(\hat{0}) = H(\hat{S}^m(\hat{0}))$$

$$C_W \models \hat{S}^n(\hat{0}) = H(\hat{S}^m(\hat{0}))$$

and since $C_W = T(\Sigma_2, E_2)$, by initiality (Provability Criterion 1.5)

$$E_2 \vdash \hat{S}^n(\hat{0}) = H(\hat{S}^m(\hat{0})) \ .$$

Now $E_\phi \vdash \Phi(S^n(0)) = \hat{S}^n(\hat{0})$, and since $E_3^0 \vdash E_\phi$

$$E_3^0 \vdash \Phi(S^n(0)) = H(\hat{S}^m(0))$$

Applying the axiom of E_W^0 we obtain

$$E_3^0 \vdash F\, S^n(0) = G\, S^n(0)$$

on substitution $S^n(0)$ for X and $\hat{S}^m(\hat{0})$ for Y.

This concludes the proof of (a) and, by the Refinement Lemma 1.15, the proof that $D_W \cong T(\Sigma_3, E_3^0)$. □

<u>6.8 LEMMA</u> D_W *does not possess a* (REC,EQ) *specification.*

<u>PROOF</u> Consider the relationship between D_W and A_W. Clearly $D_W|_\Sigma = A_W$ and each function symbol of $\Sigma_3 - \Sigma$ has codomain sort in $\Sigma_3 - \Sigma$.

<u>6.9 LEMMA</u> *Let* A *and* B *be arbitrary algebras of signatures* Σ *and* Σ'. *Suppose that* $\Sigma \subset \Sigma'$ *and* $B|_\Sigma \cong A$. *Suppose that each function symbol of* $\Sigma' - \Sigma$ *has codomain sort in* $\Sigma' - \Sigma$. *Then for any equational specification* (Σ', E') *we have*

$$B \cong T(\Sigma', E') \textit{ implies } A \cong T(\Sigma, E)$$

where $E = E' \cap L(\Sigma)$ *and* $L(\Sigma)$ *is the first-order language over* Σ. *Thus if* B *has a* (*,EQ) *specification then* A *has a* (*,EQ) *specification.*

Now if we assume that D_W has a (REC,EQ) specification then, by Lemma 6.9, A_W has a (REC,EQ) specification : this is not the case because of Theorem 6.5.

PROOF OF LEMMA 6.9 Let $B \cong T(\Sigma',E')$. Construct B^* a homomorphic image of B that is made by collapsing all domains in B named in $\Sigma'-\Sigma$ to a singleton set : we take $B^* = B/\equiv_{E^*}$ where

$$E^* = \{X_s = Y_s : s \text{ sort in } \Sigma'-\Sigma\} .$$

By the fact that the operators of $\Sigma'-\Sigma$ have codomains in $\Sigma'-\Sigma$ we have that

$$B^*|_\Sigma \cong A .$$

And that $B^* = T(\Sigma', E' \cup E^*)$, by the Factor Lemma 1.14.

Now take $E = E' \cap L(\Sigma)$ i.e. E is the set of equations involving operators from Σ only. Notice that $E \cup E^* \vdash E' \cup E^*$ because $E^* \vdash E'-E$. By the Refinement Lemma 1.15,

$$T(\Sigma', E \cup E^*) \cong B^* .$$

We will now show that $T(\Sigma,E) \cong A$. Clearly $A \models E$ because $B \models E'$ and $B|_\Sigma \cong A$. Thus we must show that A is initial in $ALG(\Sigma,E)$. Suppose $C \in ALG(\Sigma,E)$; we must construct a homomorphism $A \to C$. First we enrich C to a Σ'-structure C' by adding singleton domains for the new sorts and the uniquely determined operators having codomains among the new sorts. Note that $C' \models E \cup E^*$. By the initiality of B^* for $ALG(\Sigma', E \cup E^*)$ there is a homomorphism $\phi : B^* \to C'$. On restricting our interest to Σ we find that ϕ induces a homomorphism $A \cong B|_\Sigma \to C$. This concludes the proof of Lemma 6.9 and that of Theorem 6.6. □

Finally, in Figure A, we must separate the simple equations from the equations :

6.10 THEOREM *There is an algebra* A *that possesses a* (FIN,EQ) *specification but fails to possess a* (FIN,SEQ) *specification.*

PROOF Let Σ be the following signature

sorts	nat
constants	0 : nat
functions	S : nat $\to$ nat
	P : nat $\to$ nat

and let E be the set containing the equations

$$P(0) = 0 \qquad PS(X) = X \ .$$

Clearly, $T(\Sigma,E)$ is the structure

$$A = (\{0,1,\ldots\};\ 0,\ x+1,\ x-1)$$

and A has a (FIN,EQ) specification.

Suppose for a contradiction that A has a (FIN,SEQ) specification (Σ,E_0). Define for $k\in\omega$

$$E_k = \{P(0) = 0\} \cup \{PS^{n+1}(0) = S^n(0) : n \leq k\} \quad .$$

<u>6.11 LEMMA</u> *Let* e *be a simple equation over* Σ. *If* $|e| < k$ *and* $A \models e$ *then* $E_k \vdash e$.

<u>PROOF</u> By induction on the structure of e. □

Let $k_0 = \max\{|e| : e\in E_0\}$; remember E_0 is finite. Then, by Lemma 6.11, $E_{k_0} \vdash E_0$. Since $A \models E_{k_0}$ we have that $A \cong T(\Sigma,E_{k_0})$ by the Refinement Lemma 1.15. To this statement we obtain a contradiction.

Let $B = (\omega : 0,\ x+1,\ s,\ p)$ where $p : \omega \to \omega$ is defined by

$$p(n) = \begin{cases} 0 & \text{if } n = 0 \\ n-1 & \text{if } 0 < n \leq k_0 \\ 0 & \text{if } n > k_0 \end{cases}$$

Now $B \models E_{k_0}$ and so, by the initiality of $A \cong T(\Sigma,E_{k_0})$, there exists a homomorphism $\phi : A \to B$. But we can calculate $\phi(1)$ in two ways :

$$\phi(1) = \phi(s(0)) = s\phi(0) = s(0) = 1$$

$$\begin{aligned}\phi(1) &= \phi(p^{k_0}\ s^{k_0}(1)) \\ &= p^{k_0}\phi(s^{k_0}(1)) \\ &= p^{k_0}(k_0+1) \\ &= 0\end{aligned}$$

This is a contradiction. □

7. CONCLUDING REMARKS

The classification programme can be extended to other specification methods : closely related are algebraic specifications equipped with final algebra semantics; specifications that allow forms of negation under both initial and final algebra semantics; specifications possessing stronger properties such as associated rewrite rule systems that are confluent and noetherian, or such as ω-completeness; specifications that allow partial operations under initial and final algebra semantics. In each of these cases there is much work to be done for the mathematics of the methods is not as simple as that of the cases considered here (see, for example, Heering [23], Broy and Wirsing [15]). In addition, the classification programme could include techniques such as those in Klaeren [32] and Loeckx [35].

We have taken some steps in these directions in our series [3-11], particularly focussing on the subject of adequacy and completeness theorems for (FIN,EQ,HE) and (FIN,CEQ,HE) specifications under initial and final algebra semantics. Those of our results that are improvements of Theorem 5.1 are proved using substantially harder arguments (involving the Diophantine Theorem for r.e. sets); thus the virtue of Theorem 5.1 is its use of basic facts about computability theory. In the case of semicomputable algebras, Theorem 5.3 is both simple and the *only* completeness theorem for the finite specifications under initial algebra semantics that is known. We feel that the solution of Open Problem 3.15 will be an important step forward.

REFERENCES

[1] J.A. Bergstra, M. Broy, M. Wirsing, J.V. Tucker, *On the power of algebraic specifications*, in J. Gruska and M. Chytil (eds.) *Mathematical foundations of computer science 1981, Proceedings Strbske Pleso, Czechoslovakia*, Springer Lecture Notes in Computer Science 118, Springer-Verlag, Berlin, 1981.

[2] J.A. Bergstra and J-J Ch Meyer, *On specifying sets of integers*, Elektron. Informationsverarb. u Kybernet. 20 (1984), 531-541.

[3] J.A. Bergstra and J.V. Tucker, *On the adequacy of finite equational methods for data type specifications*, ACM-Sigplan Notices, 14(11) (1979), 13-18.

[4] ______, *Algebraic specifications of computable and semicomputable data structures*, Department of Computer Science Research Report IW 115, Mathematical Centre, Amsterdam, 1979.

[5] ______, *A characterisation of computable data types by means of a finite, equational specification method,* in J.W. de Bakker and J. van Leeuwen (eds.) *Automata, Languages and Programming, Seventh Colloquium, Noordwijkerhout, 1980,* Springer-Verlag, Berlin, 1980, 76-90.

[6] ______, *Equational specifications for computable data types : six hidden functions suffice and other sufficiency bounds,* Department Computer Science Research Report IW 128, Mathematical Centre, Amsterdam, 1980.

[7] ______, *On bounds for the specification of finite data types by means of equations and conditional equations,* Department Computer Science Research Report IW 131, Mathematical Centre, Amsterdam, 1980.

[8] ______, *A natural data type with a finite equational final semantics specification but no effective equational initial semantics specification,* Bull. EATCS, 11 (1980), 23-33.

[9] ______, *Algebraically specified programming systems and Hoare's logic,* in S. Even and O. Kariv (eds) *Automata, Languages and programming, Eighth Colloquium, Acre, 1981,* Springer Lecture Notes in Computer Science 115, Springer-Verlag, 1981, 348-362.

[10] ______, *Initial and final algebra semantics for data type specifications: two characterisation theorems,* SIAM Journal on Computing 12 (1983) 366-387.

[11] ______, *The completeness of the algebraic specification methods for data types,* Information and Control, 54 (1982) 186-200.

[12] ______, *Expressiveness and the completeness of Hoare's logic,* Journal of Computer and System Sciences, 25 (1982) 267-284.

[13] ______, *The axiomatic semantics of programs based on Hoare's logic,* Acta Informatica 21 (1984) 293-320.

[14] M. Broy, W. Dosch, H. Parsch, P. Pepper and M. Wirsing, *Existential quantifiers in abstract data types,* in H. Maurer (ed.) *Automata, Languages and Programming, Sixth Colloquium, Graz, 1980,* Springer-Verlag, Berlin, 1979, 72-87.

[15] M. Broy and M. Wirsing, *Partial abstract types,* Acta Informatica 18 (1982) 47-64.

[16] P.M. Cohn, *Universal algebra,* Harper and Row, New York, 1965.

[17] J.A. Goguen, J.W. Thatcher, E.G. Wagner and J.B. Wright, *Abstract data types as initial algebras and correctness of data representations,* in Proc. ACM Conference on Computer Graphics, Pattern Recognition and Data Structure, ACM, New York, 1975, 89-93.

[18] J.A. Goguen, J.W. Thatcher and E.G. Wagner, *An initial algebra approach to the specification, correctness and implementation of abstract data types,* R.T. Yeh, (ed.) *Current Trends in Programming Methodology, IV, Data Structuring,* Prentice-Hall, Englewood Cliffs, NJ, 1978, 80-149.

[19] G. Grätzer, *Universal algebra,* Van Nostrand, Princeton, 1968.

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

[21] J.V. Guttag, *The specification and application to programming of abstract data types,* Ph.D. thesis, Department Computer Science, University of Toronto, Toronto, 1975.

[22] J.V. Guttag and J.J. Horning, *The algebraic specification of abstract data types,* Acta Informatica, 10(1978),27-52.

[23] J. Heering, *Partial evaluation and ω-completeness of algebraic specifications,* CWI Report CS-R8501, Centre for Mathematics and Computer Science, Amsterdam, 1983.

[24] P.N. Hilfinger, Correspondence from members, ACM-Sigplan Notices 13, January 1978, 11-12.

[25] C.A.R. Hoare, *An axiomatic basis for computer programming,* Communications ACM, 12 (1969) 576-583.

[26] ______, *Proof of correctness of data representations,* Acta Informatica, 1 (1972) 271-281.

[27] ______, *Notes on data structuring,* in O.J. Dahl, E.W. Dijkstra and C.A.R. Hoare (eds.) *Structured programming,* Academic Press, London, 1972.

[28] D.W. Jones, *A note on some limits of the algebraic specification methods,* ACM-Sigplan Notices 13, April 1978.

[29] B. Kutzler and F. Lichtenberger, *Bibliography on abstract data types,* Springer-Verlag, Berlin 1983.

[30] S. Kamin, *Some defintions for algebraic data type specifications,* ACM-Sigplan Notices 14, March 1979, 28-37.

[31] D. Kapur, *Specifications of Majster's traversable stack and Veloso's traversable stack,* ACM-Sigplan Notices, 14, May 1979, 46-53.

[32] H. Klaeren, *A constructive method for abstract algebraic software specification,* Theoretical Computer Science, 30 (1984) 139-204.

[33] G. Lallement, *Semigroups and combinatorial applications,* J. Wiley & Sons, 1978.

[34] B. Liskov and S. Zilles, *Specification techniques for data abstractions,* IEEE Transactions on Software Engineering SE-1, 1975, 7-19.

[35] J. Loeckx, *Algorithmic specifications of data types,* in S. Even and O. Kariv (eds) *Automata, Languages and programming, Eighth Colloqium, Acre, 1981,* Springer Lecture Notes in Computer Science 115, Springer-Verlag, 1981, 129-147.

[36] R. Lyndon and P. Schupp, *Combinatorial group theory,* Springer-Verlag, Berlin, 1977.

[37] M.E. Majster, *Limits of the "algebraic" specification of abstract data types,* ACM-Sigplan Notices 12, October 1977, 37-42.

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

[39] A.I. Mal'cev, *Constructive algebras, I.,* Russian Mathematical Surveys 16 (1961) 77-129.

[40] ______, *Algorithms and recursive functions,* Wolters-Noordhoff, Groningen, 1970.

[41] ______, *Algebraic systems,* Springer-Verlag, Berlin, 1973.

[42] J. Meseguer and J. Goguen, *Initiality, induction and computability* in M. Nivat and J. Reynolds (eds) *Algebraic methods in semantics,* Cambridge University Press, 1985.

[43] D.L. Parnas, *On the criteria to be used in decomposing systems into modules,* Communications ACM, 15 (1972) 1053-1058.

[44] ______, *A technique for software module specification with examples,* Communications ACM, 15 (1972) 330-336.

[45] K. Prachar, *Primzahlverteilung,* Springer-Verlag, Berlin, 1957.

[46] M.O. Rabin, *Computable algebra, general theory and the theory of computable fields,* Transactions American Mathematical Society, 98 (1960) 341-360.

[47] H. Rogers, *Theory of recursive functions and effective computability,* McGraw-Hill, New York, 1967.

[48] J.J. Rotman, *Theory of groups,* Allyn and Bacon, Boston, 1973.

[49] P.A. Subrahmanyam, *On a finite axiomatisation of the data type* L, ACM-Sigplan Notices 13, April 1978, 80-84.

[50] V. Stoltenberg-Hansen and J.V. Tucker, *Computable algebra. An Introduction to computable rings and fields,* in preparation.

[51] J.W. Thatcher, E.G. Wagner and J.B. Wright, *Specification of abstract data types using conditional axioms,* IBM Research Report RC 6214, Yorktown Heights, NY, 1979.

[52] ______, *Data type specifications : parametrization and the power of specification techniques,* IBM Research Report RC 7757, Yorktown Heights, NY, 1979.

[53] J.V. Tucker and J.I. Zucker, *Program correctness over abstract data types, with error-state semantics,* Research Monograph, in preparation.

[54] Veloso P.A.S., Traversable stack with fewer errors, ACM-Sigplan Notices 14 (1979), 55-59.

[55] M. Wand, *Final algebra semantics and data type extensions,* Journal Computing Systems Science, 19 (1979) 27-44.

[56] M. Wirsing and M. Broy, *Abstract data types as lattices of finitely generated models,* Mathematical Foundations of Computer Science, Eighth Symposium, Rydzyna 1980, Springer-Verlag, Berlin, 1980.

[57] A. van Wijngaarden, *Numerical analysis as an independent science,* BIT 6 (1966) 66-81.

[58] S. Zilles, *An introduction to data algebras,* Working Paper, IBM Research Laboratory, San Jose, California 1975.