stichting
mathematisch
centrum

MC

AFDELING INFORMATICA (DEPARTMENT OF COMPUTER SCIENCE) IW 142/80 SEPTEMBER

J.A. BERGSTRA & J.V. TUCKER

INITIAL AND FINAL ALGEBRA SEMANTICS FOR DATA TYPE SPECIFICATIONS: TWO CHARACTERISATION THEOREMS

Preprint

kruislaan 413 1098 SJ amsterdam

Printed at the Mathematical Centre, 413 Kruislaan, Amsterdam.

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

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

ACM-Computing Review-categories: 4.34

Initial and final algebra semantics for data type specifications: two characterisation theorems*

by

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

ABSTRACT

We prove that those data types which may be defined by conditional equation specifications and final algebra semantics are exactly the *cosemi-computable data types* - those data types which are effectively computable, but whose *inequality* relations are recursively enumerable. And we characterise the computable data types as those data types which may be specified by conditional equation specifications using *both* initial algebra semantics and final algebra semantics. Numerical bounds for the number of auxiliary functions and conditional equations required are included in both theorems.

KEY WORDS & PHRASES: *data types, algebraic specifications, conditional equations, initial algebra semantics, final algebra semantics, computable, semicomputable and cosemicomputable algebras*

---

* This paper is not for review as it is meant for publication elsewhere.

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

INTRODUCTION

Suppose you have it in mind to define a data type by means of a set of operators Σ whose behaviour is to be governed by a set of axioms E. Then *initial* and *final algebra semantics* represent two distinct, though natural, ways of settling upon a unique meaning for the specification (Σ,E) when the axioms E are written in certain algebraic normal forms. As its semantics, they each assign to (Σ,E) a many-sorted algebra, unique up to isomorphism, from the class $ALG(\Sigma,E)$ of all algebraic systems of signature Σ which satisfy the properties prescribed by E. Viewed from the proof theory of the axioms E, initial algebra semantics formalise the decision that two formal syntactic expressions, or terms, t,t' over Σ should be semantically equivalent if, and only if, t = t' can be *proved* from the axioms E. While final algebra semantics allows t,t' to be semantically identified as long as t = t' does not *contradict* the requirements of E, or - as one says in the terminology of model theory - t = t' is *consistent* with E.

Both techniques have been widely discussed in the literature devoted to the design of programming languages with varying degrees of exactness and approval; and it seems fair to say that most theoretical and practical work on algebraic data types can be placed in one or other of these opposing initial and final camps, usually the former. For example, looking at the origins of the algebraic specification methods, one sees that the ADJ GROUP [9,10] and ZILLES & LISKOV [16,25,26] used initial algebra semantics for their specifications, but that J.V. GUTTAG [11] probably thought in terms of final algebra semantics. (At least, GUTTAG & HORNING [12] deny they are taking initial models for their specifications and come close to an informal description of the final model strategy. Moreover, in the first paper to explicitly formulate final algebra semantics, [23], Wand argues that it is indeed the denotational semantics of Guttag's theory of specifications.) Mathematically exact declarations in favour of the far less well-understood final algebra semantics can be seen in HORNUNG & RAULEFS [13], KAMIN [14], KAPUR & SRIVAS [15], the MUNICH GROUP [8,24] and WAND [23].

The issues involved in the initial and final alternatives are many and complex; they seem to turn on independent theoretical and practical options for specific problems to do with data types. Unfortunately, no thoroughly researched comparative study of the questions involved is yet available. However it is an objective of this paper to provide some theoretical perspective for such a discussion by reporting the technical facts of life about a rather basic problem, *the adequacy problem for specification methods*, which lead to the conclusion that the theory of algebraic data types needs both the initial and the final techniques. We will prove two theorems, the First and Second Characterisation Theorems below, which characterise two kinds of effectively computable data types in terms of the initial and final algebra semantics for algebraic specifications *allowing finite sets of conditional equations only*. Before giving their statements we shall explain some background issues to do with data types which the theorems are meant to resolve; after this introduction we shall adopt an exclusively technical outlook.

Roughly speaking, a specification method M is characterised partly by syntactical properties of the specifications its uses and partly by the semantical conditions it imposes on their meanings. For example, a method M may allow specifications with equations only, or with conditional equations; it may require those sets of axioms to be finite or it may let them be recursively enumerable. Each of these four decisions yields two distinct methods depending on which of the initial and final algebra semantics is chosen. And the two ways of introducing hidden or auxiliary operators to assist in specifying data types doubles the number of methods based upon these familiar options. The adequacy problem for a particular specification method M is the informal question *Does the method* M *define all the data types one wants*? Our theorems will frame exact answers to two of three precise formulations of this question when M is assumed to use finite sets of conditional equations only and an elementary mechanism for involving hidden operators. The three versions of the adequacy question are determined by three natural and distinct kinds of effectively computable data type semantics:

Let us say that an algebra A is *effectively presented* whenever we possess an effective enumeration of its elements and we can effectively calculate its operations. Then A is said to be a *semicomputable algebra*, or a *cosemicomputable algebra*, if in addition the equality relation of A is r.e., or co-r.e., respectively. A is a *computable algebra* when equality is decidable.

Now it is obvious that an r.e. algebraic specification (Σ,E) defines a semicomputable algebra under its initial semantics: remembering the proof theoretical basis of the technique, with an r.e. set of axioms E one can simply enumerate all proofs and list the identifications $E \vdash t = t'$. It is less well known, though almost equally obvious, that the final algebra semantics of an r.e. algebraic specification defines a cosemicomputable algebra. Therefore, *if a data type can be specified both initially and finally by two r.e. sets of axioms then it must be computable*. Clearly, then, equational term rewriting systems, formal grammars, and so forth, with r.e. but not recursive word problems qualify as data types without any effectively definable final algebra specification. On the other hand in [6], we showed that the set of functions computed by LOOP-programs on the natural numbers - the primitive recursive functions - composed a data type with a finite equational specification (allowing hidden functions) under final algebra semantics, but that it does not possess an effective algebraic specification of any kind using initial algebra semantics. We will give some new examples to divide the methods in Section 2.

Concentrating on the two methods based upon finite sets of conditional equations (and allowing hidden operators), the three formal adequacy problems per method boil down to the question *Can the following known implications be reversed*?

FINITE CONDITIONAL SPECIFICATIONS + INITIAL SEMANTICS
→ SEMICOMPUTABLE DATA TYPES

FINITE CONDITIONAL SPECIFICATIONS + FINAL SEMANTICS
→ COSEMICOMPUTABLE DATA TYPES

BOTH SPECIFICATION METHODS → COMPUTABLE DATA TYPES

In Section 3, we prove that the second implication can be reversed. This argument will go quite some way toward reversing the first implication, at least far enough to prove that the third implication is an equivalence; we deal with these points in Section 4. On top of the characterisations, we are able to put numerical upper bounds for the number of auxiliary operators and the number of equations necessary to specify the cosemicomputable and computable data types:

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

1. A *is cosemicomputable.*
2. A *possesses a conditional equation specification, involving at most* 5 *hidden functions and* $15 + |\Sigma|$ *axioms, which defines* A *under its final algebra semantics.*

<u>SECOND CHARACTERISATION THEOREM</u>. *Let* A *be an algebra finitely generated by elements named in its signature* Σ. *Then the following are equivalent:*

1. A *is computable.*
2. A *possesses two conditional equation specifications, each involving at most* 5 *hidden functions and* $15 + |\Sigma|$ *axioms, such that one specification defines* A *under its initial algebra semantics while the other defines* A *under its final algebra semantics.*

This paper is the sixth in our series of mathematical studies of the power of definition and adequacy of algebraic specification methods for data type definition [2,3,4,5,6] see also [7]. Obviously, the reader is assumed familiar with the informal issues and basic algebraic machinery of algebraic specifications and their semantics. For this material ADJ[10] is essential, but the reader ought also to be experienced in following algebraic arguments as he or she will then find this paper virtually self-contained: our previous work is involved explicitly in an appeal to [5] which dispenses with finite data types, and implicitly in that *we talk about single-sorted algebras only*. Our previous articles established a standard procedure for turning single-sorted adequacy theorems into their many-sorted generalisations, and that procedure readily applies here.

1. SPECIFICATIONS AND THEIR SEMANTICS

The purpose of this first section is to describe, in a summary form, two denotational semantics for algebraic data type specifications: *initial algebra semantics* and *final algebra semantics*. Our working definitions of these two mechanisms for assigning a meaning to a specification are given as Definitions 1.5 and 1.6 below: *they, and they alone, represent what we will have in mind for initial and final algebra semantics in the technical work which follows*. By way of exposition of these two different semantics we describe them from the standpoints of category theory, logic and lastly algebra. Let us repeat that we take it for granted that the reader is well versed in the mathematical theory of data types created by the ADJ GROUP [10,21,22].

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

Seen from the point of view of the category $ALG^*(\Sigma,E)$, *initial algebra semantics* for algebraic specifications assigns as the meaning of (Σ,E) the initial algebra $I(\Sigma,E)$ in $ALG^*(\Sigma,E)$; this $I(\Sigma,E)$ always exists and is unique up to isomorphism. On the other hand, *final algebra semantics* would like to pick out the final object from $ALG^*(\Sigma,E)$ as the meaning of (Σ,E), but clearly this final algebra is in all cases the *trivial one-point*, or *unit*, *Σ-algebra* $1 \in ALG^*(\Sigma,E)$. (Notice 1 may not play an initial role in $ALG^*(\Sigma,E)$ because of the minimality assumption.) Instead, final algebra semantics turns to the category $ALG_0^*(\Sigma,E)$ which is simply $ALG^*(\Sigma,E)$ with the unit algebra removed. Unfortunately, $ALG_0^*(\Sigma,E)$ need not always possess a final object $F(\Sigma,E)$, but when it does this object is unique.

Because of this asymmetry, defining and using the final algebra semantics of algebraic specifications is a rather delicate matter when compared with the initial technique. Nevertheless, the technical motives behind final algebra semantics are natural enough and complement those behind initial algebra semantics. To explain these we adopt a logical point of view toward algebraic specifications from which the *raison d'être* of the semantics becomes evident.

Given any data type semantics A, a minimal algebra of finite signature Σ, consider the algebra $T(\Sigma)$ of all syntactic terms over Σ. There is an obvious semantic mapping $v_A: T(\Sigma) \to A$ which evaluates the formal expressions over Σ as data belonging to A; v_A is an epimorphism of Σ-algebras and is uniquely determined as a function by A. The congruence $\equiv_A$ induced on $T(\Sigma)$ by v_A, defined by

(1) $$t \equiv_A t' \text{ if, and only if, } v_A(t) = v_A(t') \text{ in } A,$$

for $t,t' \in T(\Sigma)$, is uniquely determined as a set by A and clearly

(2) $$A \cong T(\Sigma)/\equiv_A.$$

Combinatorially, devising a specification (Σ,E) for A amounts to devising axioms E which determine this congruence $\equiv_A$ in some precise way.

The first, and most obvious, method is to choose axioms E such that $t,t' \in T(\Sigma)$ have the same meaning in A if, and only if, one can prove that $t = t'$ from the axioms E. In the standard notation of logic, the *desired relationship* between A and E is

(3) $$A \models t = t' \text{ if, and only if, } E \vdash t = t',$$

or, equivalently,

(4) $$t \equiv_A t' \text{ if, and only if, } E \vdash t = t'.$$

This is exactly the decision made when one seeks an algebraic specification (Σ,E) and uses initial algebra semantics to define A: the equivalence

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

is always true and entails equivalence (4) when $I(\Sigma,E) \cong A$.

Final algebra semantics corresponds to a different use of the axioms in a specification (Σ,E). There one decides to assume $t,t' \in T(\Sigma)$ to have the same meaning in A if, and only if, one can assert $t = t'$ without contradicting the axioms of E:

$$t \equiv_A t' \text{ if, and only if, } t = t' \text{ is consistent with } E.$$

This notion of consistency simply means that there is some *non-unit* model

$B \in ALG^*(\Sigma,E)$ where $B \not\models t = t'$. Equivalently, the relationship desired between A and E can be expressed as follows: the congruence $\equiv_A$ has the property that *for every congruence* $\equiv$ *on* $T(\Sigma)$ *which defines a* non-unit *algebra* $T(\Sigma)/\equiv$ *in* $ALG^*(\Sigma,E)$ *we have that*

$$t \equiv t' \quad \textit{implies} \quad t \equiv_A t'.$$

As will be seen, when this relationship between $\equiv_A$ and E can be arranged we have A as the final object of $ALG_0^*(\Sigma,E)$. And it is precisely these technical observations to do with consistency which lie behind the notion of *semantic observability* much used in writings on final algebra semantics.

Now we come to our purely algebraic definitions of these semantics framed in terms of congruences on $T(\Sigma)$.

Let A be an algebra of signature Σ.

A congruence $\equiv$ on A is said to be the *unit congruence* if for any $a,b \in A$ we have $a \equiv b$; or, equivalently, if $A/\equiv$ is the unit algebra of signature Σ.

A congruence $\equiv_2$ on A is said to *extend* another congruence $\equiv_1$ on A if for any $a,b \in A$ we have $a \equiv_1 b$ implies $a \equiv_2 b$.

Let E be a set of conditional equations over Σ.

If A satisfies the axioms of E we say that A is an E-*algebra*.

A congruence $\equiv$ on algebra A is said to be an E-*congruence* if for each conditional equation in variables $X = (X_1,\ldots,X_n)$

$$t_1(X) = t_1'(X) \wedge \ldots \wedge t_k(X) = t_k'(X) \rightarrow t(X) = t'(X)$$

and for any $a \in A^n$

$$\text{if } t_1(a) \equiv t_1'(a),\ldots,t_k(a) \equiv t_k'(a) \text{ in A then } t(a) \equiv t'(a) \text{ in A.}$$

1.1 <u>LEMMA</u>. *Let* $\equiv$ *be a congruence relation on* $A \in ALG(\Sigma,E)$. *Then* $\equiv$ *is an* E-*congruence if, and only if,* $A/\equiv$ *is an* E-*algebra*.

We will now define certain least and largest E-congruences on $T(\Sigma)$ whose corresponding factor algebras will be the initial and final objects of $ALG_0^*(\Sigma,E)$ respectively. Let us consider the initial case first.

Define $\equiv_{min(E)}$ to be the intersection of all E-congruences on $T(\Sigma)$ and set $T_I(\Sigma,E) = T(\Sigma)/\equiv_{min(E)}$. It is easy to see that $\equiv_{min(E)}$ is an E-congruence and to verify that

1.2 <u>LEMMA</u>. $T_I(\Sigma,E)$ *is isomorphic to any initial object* $I(\Sigma,E)$ *of* $ALG^*(\Sigma,E)$.

Define $\equiv_{max(E)}$ to be the smallest E-congruence extending all the non-unit E-congruences on $T(\Sigma)$. Equivalently, let $\equiv_{max(E)}$ be the smallest E-congruence containing the union of all non-unit E-congruences on $T(\Sigma)$. And set $T_F(\Sigma,E) = T(\Sigma)/\equiv_{max(E)}$.

Of course we have no guarantee that $\equiv_{max(E)}$ is not the unit congruence, and that $T_F(\Sigma,E)$ is not the unit algebra, but it is easy to prove that

1.3 <u>LEMMA</u>. *Whenever* $\equiv_{max(E)}$ *is not the unit congruence,* $T_F(\Sigma,E)$ *is isomorphic to any final object* $F(\Sigma,E)$ *of* $ALG_0^*(\Sigma,E)$.

1.4 <u>OBSERVATION</u>. *For* $t,t' \in T(\Sigma)$, $t \not\equiv_{max(E)} t'$ *if, and only if, the least E-congruence extending* $\equiv_{min(E)} \cup \{t=t'\}$ *is the unit congruence.*

We can now define precisely what we mean by initial and final algebra specifications for data types.

1.5 SEMANTICS OF ALGEBRAIC SPECIFICATIONS

Let E be a set of conditional equations over the signature Σ and let A be an algebra of signature Σ.

The pair (Σ,E) is said to be a *conditional equation specification* of the algebra A with respect to (1) *initial algebra semantics* or (2) *final algebra semantics* if (1) $T_I(\Sigma,E) \cong A$ or (2) $T_F(\Sigma,E) \cong A$.

When the set of axioms E is finite we speak of *finite conditional equation specifications* with respect to these semantics.

To conclude this preparatory section, we shall explain our favoured method of involving hidden or auxiliary functions into algebraic specifications for data types.

Let A be an algebra of signature Σ_A and let Σ be a signature $\Sigma \subset \Sigma_A$. Then we mean by

$A|_\Sigma$ the Σ-algebra whose domain is that of A and whose constants and operators are those of A named in Σ: the Σ-reduct of A; and by

$\langle A \rangle_\Sigma$ the Σ-subalgebra of A generated by the constants and operators of A named in Σ *viz* the smallest Σ-subalgebra of $A|_\Sigma$.

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

1.6 ALGEBRAIC SPECIFICATIONS WITH HIDDEN OPERATORS

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

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

or

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

In this paper, all specifications involving hidden operators are made to define data types as described in Definition 1.6.

2. EFFECTIVELY COMPUTABLE ALGEBRAS

A countable algebraic system A is said to be *effectively presented* when it is given an effective coordinatisation consisting of a recursive set $\Omega \subset \omega$ and a surjection $\alpha\colon \Omega \to A$, and, for each k-ary operation σ of A, a recursive *tracking function* $\bar{\sigma}$ which commutes the following diagram

$$\begin{array}{ccc} A^k & \xrightarrow{\ \sigma\ } & A \\ {\scriptstyle \alpha^k}\uparrow & & \uparrow{\scriptstyle \alpha} \\ \Omega^k & \xrightarrow{\ \bar{\sigma}\ } & \Omega \end{array}$$

wherein $\alpha^k(x_1,\dots,x_k) = (\alpha x_1,\dots,\alpha x_k)$.

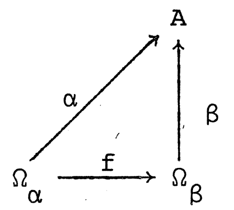
The algebra A is said to be *computable*, *semicomputable*, or *cosemicomputable*, if there exists an effective presentation $\alpha\colon \Omega \to A$ for which the relation $\equiv_\alpha$ on Ω defined by

$$n \equiv_\alpha m \quad \text{if, and only if,} \quad \alpha n = \alpha m \text{ in } A$$

is 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 [20] and, in particular, A.I. MAL'CEV [18] devoted to founding a theory of computable algebraic systems. Their special feature is that they make computability into a *finiteness condition* of algebra: an isomorphism invariant possessed of all finite structures. In the case of finitely generated algebras, the concepts enjoy a much stronger recursion-theoretic invariance property which we shall now explain.

Let α and β be effective presentations of some algebra A. Then α *recursively reduces* to β (in symbols: $\alpha \leq \beta$) if there exists a recursive function f to commute the following diagram,



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

Recursive equivalence is the fundamental identity relation between numberings of algebras and is meant to measure the uniquness of the recursion-theoretical concepts under their translation to algebraic systems.

Let R be a k-ary relation on A and let A be effectively presented by α. Then R is said to be α-computable if its preimage

$$\alpha^{-1}R = \{(x_1,\dots,x_k) \in \Omega_\alpha^k\colon (\alpha x_1,\dots,\alpha x_k) \in R\}$$

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

2.1 LEMMA. *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*). *In particular, the effectivity of a relation on an algebra is unique up to the recursive equivalence of codifications.*

The invariance property for finitely generated algebras which interests us is the existence of certain canonical effective presentations which solve the irritating problem of how to speak of a relation as being computable (say) without also having to name a coordinatisation.

Henceforth, assume A is an algebra finitely generated by elements named in its signature Σ.

Clearly, the term algebra $T(\Sigma)$ is computable under any natural gödel numbering of terms. It is easy to make a general definition of such a gödel numbering and to go on to prove that gödel numberings compose an equivalence class under recursive equivalence; so the choice of $\gamma_\alpha: \omega \to T(\Sigma)$ is unimportant. 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

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

To see that γ_A is indeed an effective coordinatisation of A one need only observe that an effective presentation for A is nothing more than an epimorphism between A and a recursive algebra of natural numbers.

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

A proof of this can be found in MAL'CEV [18]; coupled with Lemma 2.1, it has several important consequences.

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

2.4 COROLLARY. *Any two semicomputable coordinatisations of* A *are recursively equivalent.*

Let R be a recursive number algebra and α: R $\to$ A a homomorphism. Let us say that α is a *decidable, r.e.* or *co-r.e. homomorphism* accordingly as the congruence it induces on R

$$n \equiv_\alpha m \quad \text{if, and only if,} \quad \alpha n = \alpha m \text{ in } A$$

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

2.5 REPRESENTATION LEMMA. *If* A *is semicomputable, or cosemicomputable, then it can be represented as the image of a recursive number algebra* R *with domain* ω *under an r.e., or co-r.e., homomorphism* α *respectively. In particular,* A *is isomorphic to the factor algebra* $R/\equiv_\alpha$ *of* R *under the r.e., or co-r.e., congruence induced by* α. *If* A *is computable then it is isomorphic to a recursive number algebra* R *with domain* ω *providing* A *is infinite.*

What material we need from the theory of the recursive functions is elementary and is well covered by MACHTEY & YOUNG [17] with one exception: Matijacevič's Diophantine Theorem.

Let $\mathbb{Z}[X_1,\ldots,X_n]$ denote the ring of polynomials in indeterminates $X_1,\ldots,X_n$ and with integer coefficients. A set $\Omega \subset \omega^n$ is said to be *diophantine* if there exists a polynomial $p \in \mathbb{Z}[X_1,\ldots,X_n,Y_1,\ldots,Y_m]$ such that

$$(x_1,\ldots,x_n) \in \Omega \iff \exists y_1,\ldots,y_m \in \omega.[p(x_1,\ldots,x_n,y_1,\ldots,y_m) = 0]$$

Clearly, each diophantine set is recursively enumerable; the converse is a hard theorem of Y. Matijacevič:

2.6 DIOPHANTINE THEOREM. *All recursively enumerable sets are diophantine.*

The number of search variables $y_1,\ldots,y_n$ can always be chosen to be 13 or less, incidentally. A good exposition of the theorem appears in MANIN [19].

We will always use the Diophantine Theorem to obtain polynomials over the natural numbers ω rather than over $\mathbb{Z}$. We will now write down an equivalent characterisation of a diophantine set of natural numbers, one more suited to our special tasks.

Let $\omega[X_1,\ldots,X_n]$ denote the set of all polynomials having natural number coefficients and involving only addition and multiplication.

A set $\Omega \subset \omega^n$ is *ω-diophantine* if there exist p and $q \in \omega[X_1,\ldots,X_n, Y_1,\ldots,Y_m]$ such that

$$(x_1,\ldots,x_n) \in \Omega \iff \exists y_1,\ldots,y_m \in \omega.[p(x_1,\ldots,x_n,y_1,\ldots,y_m) = q(x_1,\ldots,x_n,y_1,\ldots,y_m)].$$

It is easy to check that the ω-diophantine sets are precisely the diophantine sets.

These technical preliminaries concluded, we can now turn our attention to data types and their specifications.

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

The proof of Basic Lemma 2.7 is routine and is left as an exercise for the reader. (Hint: Use Observation 1.4.) Here are examples of semicomputable and cosemicomputable algebras which are not computable.

2.8 COMBINATORY LOGIC

Consider the signature Σ consisting of constants K,S,I and a single binary operation $\cdot$. Combinatory logic can be axiomised by three equations over this Σ.

$$(K\cdot X)\cdot Y = X$$
$$((S\cdot X)\cdot Y)\cdot Z = (X\cdot Z)\cdot(Y\cdot Z)$$
$$I\cdot X = X.$$

where X,Y,Z are variables. The initial algebra of the resulting variety is known as the *term model for combinatory logic* and we denote it TMCL. Clearly, it is an algebra having a finite equational specification and it is semicomputable. It is not a computable algebra, however, because combinatory logic is a formalism strong enough to define all recursive functions; see BARENDREGT [1] for details.

2.9 POLYNOMIAL FUNCTIONS

The typical cosemicomputable algebra is a set of computable functions structured by some effective operators. For example, let A be a computable algebra of signature Σ and let $T_\Sigma[X_1,\ldots,X_n]$ be the algebra of formal polynomials in n indeterminates over Σ. Each $t \in T_\Sigma[X_1,\ldots,X_n]$ defines an n-argument *polynomial function* $A^n \to A$ which is computable. It is easy to derive an effective presentation of the Σ-algebra $PF(A^n,A)$ of all n-ary polynomial functions over A from a computable coordinatisation of $T_\Sigma[X_1,\ldots,X_n]$; and to prove that $PF(A^n,A)$ is a cosemicomputable algebra. We give an A for which $PF(A^n,A)$ is not computable when $n \geq 13$.

Let A have domain ω, constants $0,1 \in \omega$, and the operations of addition $x+y$, minus $x \dot{-} y$, multiplication $x.y$, and

$$min(x) = \begin{cases} 0 & \text{if } x = 0; \\ 1 & \text{if } x \geq 1. \end{cases}$$

Let Σ be the signature of A.

Now let Ω be any r.e. subset of ω and assume it defined by the Diophantine Theorem as

$$n \in \Omega \iff \exists y \in \omega^n.[p(n,y) = q(n,y)]$$

for $p,q \in \omega[X,Y_1,\ldots,Y_m]$. Define a family of polynomial maps $\omega^m \to \omega$ over A by

$$H_n(y) = min(p_n(y) \dot{-} q_n(y) + q_n(y) \dot{-} p_n(y)).$$

Clearly, the family $\{H_n: n \in \omega\}$ is a computable subset of $PF(A^m,A)$ and if $PF(A^m,A)$ were a computable algebra then we could decide whether or not

$$H_n = 1$$

where $1(y) = 1$ for all $y \in \omega^m$. However, it is easy to see that

$$H_n = 1 \quad \text{if, and only if,} \quad n \in \Omega.$$

Thus, choosing Ω to be r.e. and not recursive shows that $PF(A^m,A)$ cannot be computable. The reader might care to find a finite algebraic specification for $PF(A^m,A)$ as an exercise.

3. COSEMICOMPUTABLE DATA TYPES

This section is entirely given over to proving the First Characterisation Theorem stated in the Introduction. In view of Basic Lemma 2.7, we have only to prove that (1) implies (2).

Let A be a cosemicomputable algebra of signature Σ.

First of all we dispense with the relatively easy case when A is finite. In [5], we proved that any finite algebra possesses a finite conditional equation specification under initial algebra semantics which involves at most 1 auxiliary operator, 1 simple equation and 2 conditional equations. As it happens, precisely the same syntactic specification designed there for a finite algebra A also defines A under its final algebra semantics. Thus, we leave the reader to check this claim (or to devise his or her own proof of the theorem in this special case) and we move on to the considerably more difficult case when A is cosemicomputable and infinite.

We divide the proof of this case into two parts. First, we will frame an auxiliary hypothesis H and prove the First Characterisation Theorem for any infinite cosemicomputable algebra satisfying the extra condition H. This done, we will then prove that, indeed, *every* infinite cosemicomputable algebra satisfies our hypothesis H.

3.1 PARTITION HYPOTHESIS

Let A be effectively presented by $\alpha\colon \Omega_\alpha \to A$. By an *α-computable partition* we mean a family $V = \{V_i : i \in \omega\}$ of non-empty subsets of A such that

(i) $V_i \cap V_j = \emptyset$ if $i \neq j$.

(ii) $\bigcup_{i\in\omega} V_i = A$.

(iii) The V_i are α-computable subsets of A uniformly in i; that is, the function $M_V\colon \omega \times \Omega_\alpha \to \{0,1\}$ defined by

$$M_V(i,n) = \begin{cases} 0 & \text{if } \alpha n \in V_i; \\ 1 & \text{if } \alpha n \notin V_i; \end{cases}$$

is recursive.

Thanks to Lemma 2.1 and the Reduction Lemma 2.2 we need not be careful about the coding α to which an α-computable partition is tied. And as our hypothesis H we may take the statement that A *possesses a computable partition.*

3.2 THE PROOF FOR A INFINITE, COSEMICOMPUTABLE, AND POSSESSING A COMPUTABLE PARTITION

Let A be the image of recursive number algebra R, with domain ω, under co-r.e. homomorphism $\alpha\colon R \to A$ (Representation Lemma 2.5) and assume A has a computable partition with respect to this α. In outline, our plan is to add to R a constant and some 5 functions to make a new recursive number algebra R_0 such that

(a) $R_0|_\Sigma = \langle R_0\rangle_\Sigma = R$;

(b) $\equiv_\alpha$ is a congruence on R_0.

In consequence,

$$R_0/{\equiv_\alpha}|_\Sigma = \langle R_0/{\equiv_\alpha}\rangle_\Sigma = R/{\equiv_\alpha} \cong A.$$

For $R_0/{\equiv_\alpha}$ we will make a conditional equation specification (Σ_0, E_α) which defines it under final algebra semantics and which satisfies the required boundedness conditions. The first four new functions are designed to simulate arithmetic on R and, in particular, to respect the congruence $\equiv_\alpha$ on R. This latter condition will mean that the new functions will induce an arithmetic on $R_0/\equiv_\alpha$. With arithmetic internalised in this way, the fifth function will internalise an entirely new coding of R whose domain is the arithmetic on R. And, because the fifth function respects the congruence $\equiv_\alpha$, this coding may be internalised to $R_0/\equiv_\alpha$. This done we are able to systematically specify the coding, the recursive functions of R and the congruence $\equiv_\alpha$ itself, all by means of the Diophantine Theorem 2.6. So much for the informal description: first we built the arithmetic in R from the hypothesis of A having an α-computable partition.

Let $V = \{V_i : i \in \omega\}$ be an α-computable partition of A. Now V determines an α-computable equivalence relation $\equiv_V$ on A whose equivalence classes are the V_i. In particular, the factor set $A/\equiv_V$ is a computable set under coordinatisation $\alpha(V)\colon R \to A/\equiv_V$ defined by $\alpha(V)(n) = [\alpha n]$. This set supports a natural arithmetic based upon using V_0 as zero, and taking

$$V_i \to V_{i+1} \quad \text{as successor}$$
$$(V_i, V_j) \to V_{i+j} \quad \text{as addition}$$
$$(V_i, V_j) \to V_{i.j} \quad \text{as multiplication}$$

and this *partition arithmetic* on $A/\equiv_V$ is what we propose to model in R by special tracking functions for $\equiv_{\alpha(V)}$.

The first four functions are determined by choosing a recursive transversal for $\equiv_{\alpha(V)}$ on R.

Let $t\colon \omega \to R$ be a recursive function enumerating the following "least code" transversal $T(V,R)$ for $\equiv_{\alpha(V)}$,

$$t(i) = (\mu z \in R)[\alpha z \in V_i].$$

Thus $T(V,R) = \text{im}(t)$ and, obviously, it is a recursive set such that $T(V,A) = \{\alpha n : n \in T(V,R)\}$ is an α-computable transversal for $\equiv_V$ on A.

We also define the recursive function $d\colon R \to \omega$ by

$$d(n) = (\mu z \epsilon \omega)[\alpha n \epsilon V_z]$$

which gives the location of n, and so of αn, in the equivalence classes of $\equiv_{\alpha(V)}$, and $\equiv_V$.

The new operators required on R to make R_0 are

Projection	$proj_R(x) = t(d(X))$
Zero	$O_R = t(0)$
Successor	$succ_R(x) = t(d(x)+1)$
Addition	$add_R(x,y) = t(d(x)+d(y))$
Multiplication	$mult_R(x,y) = t(d(x).d(y))$

The reader should pause to become familiar with the effects of these functions. Notice, for example, that by the guiding principles of their design, these operators make an algebra

$$(T(V,R);\ O_R,\ succ_R,\ add_R,\ mult_R)$$

which is recursive and is isomorphic to the arithmetic we described on $A/\equiv_V$ under the mapping $\alpha(V) : T(V,R) \rightarrow A/\equiv_V$. The role of $proj_R$ is solely to *internalise* this *transversal arithmetic* within R. Notice, too, that what the partition property provides is this: because $\equiv_V$ is a coarser equivalence relation than equality in A, the relation $\equiv_{\alpha(V)}$ is coarser than $\equiv_\alpha$ on R with the result that each of the four new maps respect $\equiv_\alpha$ in a particularly strong way:

$$x \equiv_\alpha x' \text{ implies } proj_R(x) = proj_R(x')$$
$$x \equiv_\alpha x' \text{ implies } succ_R(x) = succ_R(x')$$

and similarly for add_R and $mult_R$. Thus, $\equiv_\alpha$ remains a congruence on the algebra R when these functions are added.

Let $\Sigma_{arith} = \{0, SUCC, ADD, MULT\}$ denote the signature of the transversal arithmetic.

The fifth function required is there to code R by our transveral arithmetic. Choose any recursive bijection $enum_{V,R}: T(V,R) \rightarrow R$. This bijective renumbering of R we refer to as the *transversal coding*, but it should be thought of strictly in terms of the arithmetical structure of T(V,R) and divorced from its original connections with α-codes. This can be made visible in our notations. Observe that the arithmetical structure of T(V,R) entails we may write the set as a list without repetitions

$$T(V,R) = \{succ_R^n(0_R) : n \in \omega\}$$

and, moreover, implicit in our view of the transversal coding is this composition

$$\omega \xrightarrow{\lambda n.succ_R^n(0_R)} T(V,R) \xrightarrow{enum_{V,R}}$$

Still, the transversal coding must be internalised and this means it must be defined outside T(V,R). Thus, we take as our last function in the construction of R_0 from R.

$$enum_R(n) = enum_{V,R}(proj_R(n)).$$

Again, we see that the partition yields

$$x \equiv_\alpha x' \quad \text{implies} \quad enum_R(x) = enum_R(x')$$

and so we know that $\equiv_\alpha$ is a congruence on R_0. Thus, given (a) and (b) we concentrate on the problem of specifying $R_0/\equiv_\alpha$ by conditional equations (without hidden functions and using 15 + $|\Sigma|$ formulae). This task we divide into the problems of specifying R_0 and then pressing on to specify $R_0/\equiv_\alpha$. As it turns out, the first job will be to give a specification (Σ_0, E_0), involving no hidden functions and 14 axioms, which defines R_0 by means of *initial algebra semantics*. Whence one more axiom e_α added to E_0 will make a specification (Σ_0, E_α) which completes the proof of the theorem in Case 3.2 (the reader curious about this arrangement is invited to read the proof of Lemma 3.7 first). Here is the specification (Σ_0, E_0) for R_0.

The first 10 equations specify the transversal arithmetic.

Projection $\qquad PROJ(0) = 0 \qquad (1)$

$PROJ(SUCC(X)) = SUCC(PROJ(X)) \qquad (2)$

$PROJ(X) = PROJ(PROJ(X)) \qquad (3)$

Successor	$SUCC(X) = SUCC(PROJ(X))$	(4)
Addition	$ADD(X,O) = PROJ(X)$	(5)
	$ADD(X,SUCC(Y)) = SUCC(ADD(X,Y))$	(6)
	$ADD(X,Y) = ADD(PROJ(X),PROJ(Y))$	(7)
Multiplication	$MULT(X,O) = 0$	(8)
	$MULT(X,SUCC(Y)) = ADD(MULT(X,Y),X)$	(9)
	$MULT(X,Y) = MULT(PROJ(X),PROJ(Y))$	(10)

Next we construct 3 formulae to specify the transversal coding of R. Consider these two sets designed to recover T(V,R) from that coding. (We drop the subscript R from the operations of R_0.)

$$J_1 = \{(n,m) \in \omega\times\omega: enum(succ^n(O_R)) \in T(V,R) \ \&\ enum(succ^n(O_R)) = succ^m(O_R)\}$$

$$J_2 = \{(n,m) \in \omega\times\omega: enum(succ^n(O_R)) \notin T(V,R) \ \&\ proj(enum(succ^n(O_R))) = enum(succ^m(O_R))\}.$$

Our hypotheses imply that both sets are r.e. subsets of $\omega\times\omega$ and hence, by the Diophantine Theorem, there are polynomials p_1,q_1 and p_2,q_2, in $2+k(1)$ and $2+k(2)$ variables respectively, such that

$$(n,m) \in J_1 \iff \exists z \in \omega^{k(1)}.[p_1(n,m,z) = q_1(n,m,z)]$$

$$(n,m) \in J_2 \iff \exists z \in \omega^{k(2)}.[p_2(n,m,z) = q_2(n,m,z)].$$

Let P_1,Q_1 and P_2,Q_2 be formal polynomials over Σ_{arith} corresponding to p_1, q_1 and p_2,q_2 respectively. Then our enumeration axioms are

$$P_1(X,Y,Z_1,\ldots,Z_{k(1)}) = Q_1(X,Y,Z_1,\ldots,Z_{k(1)}) \to ENUM(PROJ(X)) = PROJ(Y) \quad (11)$$

$$P_2(X,Y,Z_1,\ldots,Z_{k(2)}) = Q_2(X,Y,Z_1,\ldots,Z_{k(2)}) \to PROJ(ENUM(PROJ(X))) = ENUM(PROJ(Y)) \quad (12)$$

$$ENUM(X) = ENUM(PROJ(X)) \quad (13)$$

It now remains to add axioms to specify the new constant O and the original constants and operations of R. We need one formula in each case and this will make the total $|E_0| = 14 + |\Sigma|$.

Let $c \in R$ be a constant named by $\underline{c} \in \{0\} \cup \Sigma$. To this c there corresponds a unique $n \in \omega$ such that $c = enum(succ^n(0_R))$: assign the identification

$$\underline{c} = ENUM(SUCC^n(0)).$$

Let $f: R^k \to R$ be an operation named by $\underline{f} \in \Sigma$. Consider the graph of f translated to the transversal coding

$$G(f) = \{(n(1),\dots,n(k),m): \underline{f}(enum(succ^{n(1)}(0_R)),\dots,enum(succ^{n(k)}(0_R))) = enum(succ^m(0_R))\}.$$

Our hypotheses imply G(f) is an r.e. set and again we define it by means of the Diophantine Theorem. Let p_f, q_f be polynomials in $k+1+k(f)$ variables such that

$$(n(1),\dots,n(k),m) \in G(f) \iff \exists z \in \omega^{k(f)}.[p_f(n(1),\dots,n(k),m,z) = q_f(n(1),\dots,n(k),m,z)].$$

And choosing formal polynomials P_f, Q_f over Σ_{arith} corresponding to p_f, q_f we assign the axiom

$$P_f(X_1,\dots,X_k,Y,Z_1,\dots,Z_{k(f)}) = Q_f(X_1,\dots,X_k,Y,Z_1,\dots,Z_{k(f)}) \to \underline{f}(ENUM(PROJ(X_1)),\dots,ENUM(PROJ(X_k))) = ENUM(PROJ(Y))$$

3.3 <u>LEMMA</u>. *The specification* (Σ_0, E_0) *defines* R_0 *with respect to initial algebra semantics:*

$$T_I(\Sigma_0, E_0) \cong R_0.$$

<u>PROOF</u>. First we picture R_0 through the transversal coding

$$R_0 = \{enum(succ^n(0)): n \in \omega\}.$$

Remembering that

$$\omega \xrightarrow{\lambda n.\mathit{succ}^n(0_R)} T(V,R) \xrightarrow{\mathit{enum}} R$$

is bijective, we define ϕ: $R_0 \rightarrow T(\Sigma_0, E_0)$ by

$$\phi(\mathit{enum}(\mathit{succ}^n(0_R))) = [\mathit{ENUM}(\mathit{SUCC}^n(0_R))].$$

We write $\equiv_{E_0}$ as $\equiv$ and denote the equivalence class of $t \in T(\Sigma_0)$ under $\equiv$ by $[t]$. To show that ϕ is bijective is to prove that

$$T = \{\mathit{ENUM}(\mathit{SUCC}^n(0)) : n \in \omega\}$$

is a transversal for $\equiv$. To show ϕ is a homomorphism will be an easy exercise afterwards.

Consider T as a transversal. It is easy to check that no distinct elements of T are equivalent under $\equiv$ because they denote different elements of R_0 and R_0 is an E_0-algebra. Thus, we have to prove that each $t \in T(\Sigma_0)$ is E_0-equivalent to some member of T and this is done by induction on term complexity.

The basis is obvious thanks to the identifications assigned to the constants of Σ_0.

Assume, as induction hypothesis, that all subterms of $t \in T(\Sigma_0)$ are E_0-equivalent to some element of T. We have to consider each situation corresponding to the leading function symbol of t:

$$\mathit{PROJ},\ \mathit{SUCC},\ \mathit{ADD},\ \mathit{MULT},\ \mathit{ENUM},\ \underline{f} \in \Sigma$$

CASE 1: $t = \mathit{PROJ}(s)$

By the induction hypothesis $s \equiv \mathit{ENUM}(\mathit{SUCC}^n(0))$.

Subcase 1.1. If $\mathit{enum}(\mathit{succ}^n(0_R)) \in T(V,R)$ then $\mathit{PROJ}(s) \equiv \mathit{ENUM}(\mathit{SUCC}^n(0))$

Subcase 1.2. If $\mathit{enum}(\mathit{succ}^n(0_R)) \notin T(V,R)$ then $\mathit{PROJ}(s) \equiv \mathit{ENUM}(\mathit{SUCC}^m(0))$ for $(n,m) \in J_2$.

PROOF OF SUBCASE 1.1. This first subcase is quite involved as it introduces techniques and lemmata of use throughout the proof of Lemma 3.2; we shall write out its argument in detail. The bulk of the work lies in showing this important fact:

3.4 LEMMA. *Let* $enum(succ^n(0_R)) = succ^m(0_R)$. *Then* $ENUM(SUCC^n(0)) \equiv SUCC^m(0)$.

Assume we have done this. Thus, immediately we know that for $(n,m) \in J_1$

$$PROJ(ENUM(SUCC^n(0))) \equiv PROJ(SUCC^m(0)).$$

A little lemma already required in the proof of Lemma 3.4 is this:

3.5 LEMMA. *For any* $k \in \omega$, $PROJ(SUCC^k(0)) \equiv SUCC^k(0)$.

PROOF. This is an easy induction on k whose basis is covered by equation (1) and whose induction step is covered by equation (2). Q.E.D.

Applying Lemma 3.5 we can deduce that

$$\begin{aligned} PROJ(ENUM(SUCC^n(0))) &\equiv SUCC^m(0) \\ &\equiv ENUM(SUCC^n(0)) \end{aligned}$$

the latter step using Lemma 3.4 again. This is what is required for Subcase 1.1.

Consider the proof of Lemma 3.4. We must use equation (11) which means we must verify the premiss that there exist $t_1,\ldots,t_{k(1)} \in T(\Sigma_0)$ for which

$$\begin{aligned} P_1(SUCC^n(0),SUCC^m(0),t_1,\ldots,t_{k(1)}) \equiv Q_2(&SUCC^n(0),SUCC^m(0), \\ &t_1,\ldots,t_{k(1)}). \end{aligned}$$

From this premiss we can conclude, directly, that

$$ENUM(PROJ(SUCC^n(0))) \equiv PROJ(SUCC^m(0)).$$

By Lemma 3.5, the Lemma 3.4 follows.

So consider the premiss. Since $(n,m) \in J_1$ we know there exists $z = (z(1),\ldots,z(k(1))) \in \omega^{k(1)}$ such that $p_1(n,m,z) = q_1(n,m,z)$. We claim the premiss is true on choosing $t_i = \mathit{SUCC}^{z(i)}(0)$, $1 \le i \le k(1)$. This follows from another invaluable general lemma:

3.6 LEMMA. *Let* $p(x_1,\ldots,x_k)$ *be any polynomial over* ω *and let* $P(X_1,\ldots,X_k)$ *be its formal translation to a polynomial over* Σ_{arith}. *Then for any* $n(1),\ldots,n(k) \in \omega$

$$P(\mathit{SUCC}^{n(1)}(0),\ldots,\mathit{SUCC}^{n(k)}(0)) \equiv \mathit{SUCC}^{p(n(1),\ldots,n(k))}(0).$$

PROOF. This is done by a straightforward induction on the complexity of the polynomial $P(X_1,\ldots,X_k)$. The basis case, where $P(X_1,\ldots,X_k) = 0$ or $P(X_1,\ldots,X_k) = X_i$ for $1 \le i \le k$, is immediate. The induction step divides into 3 cases determined by the leading operator symbol of $P(X_1,\ldots,X_k)$. When this is *SUCC* the induction step is immediate. When it is *ADD* one requires an easy induction on m to prove that

$$\mathit{ADD}(\mathit{SUCC}^{n}(0),\mathit{SUCC}^{m}(0)) \equiv \mathit{SUCC}^{n+m}(0).$$

The basis of this induction will use equation (5) and Lemma 3.5; the induction step will use equation (6). When the leading operator symbol is *MULT* one has to prove

$$\mathit{MUTL}(\mathit{SUCC}^{n}(0),\mathit{SUCC}^{m}(0)) \equiv \mathit{SUCC}^{n.m}(0)$$

by induction on m. Here the basis is covered by equation (8); and the induction step is covered by equation (9) together with the previously completed case of addition. Q.E.D.

PROOF OF SUBCASE 1.2. Given the pattern of reasoning in Subcase 1.1, this subcase can be completed quite concisely. Let $\mathit{proj}(\mathit{enum}(\mathit{succ}^{n}(0_R))) = \mathit{enum}(\mathit{succ}^{m}(0_R))$ so that $(n,m) \in J_2$. We shall prove that

$$\mathit{PROJ}(\mathit{ENUM}(\mathit{SUCC}^{n}(0))) \equiv \mathit{ENUM}(\mathit{SUCC}^{m}(0))$$

by using equation (12). Thanks to Lemma 3.5, it is enough to verify the premiss of (12) that there exist $t_1,\dots,t_{k(2)} \in T(\Sigma_0)$ such that

$$P_2(SUCC^n(0), SUCC^m(0), t_1,\dots,t_{k(2)})$$
$$\equiv Q_2(SUCC^n(0), SUCC^m(0), t_1,\dots,t_{k(2)}).$$

Since $(n,m) \in J_2$, there exists $z = (z(1),\dots,z(k(2))) \in \omega^{k(2)}$ such that $p_2(n,m,z) = q_2(n,m,z)$. Taking $t_i = SUCC^{z(i)}(0)$ and applying Lemma 3.6 the premiss is true.

This first case provides two evidently important identities: Lemma 3.4 and the statement of Subcase 1.2:

$$(n,m) \in J_1 \text{ if, and only if, } ENUM(SUCC^n(0)) \equiv SUCC^m(0)$$

$$(n,m) \in J_2 \text{ if, and only if, } PROJ(ENUM(SUCC^n(0)) \equiv ENUM(SUCC^m(0))$$

From these we can deduce for $enum(succ^n(0_R)) \notin T(V,R)$

$$PROJ(ENUM(SUCC^n(0))) \equiv SUCC^m(0), \text{ if and only if, } \exists z.[(n,z) \in J_2 \ \& \ (z,m) \in J_1]$$

and taken together we have the means to access the algebraic specification's model of the transversal arithmetic. The next three cases $t = SUCC(s)$, $t = ADD(s_1,s_2)$ and $t = MULT(s_1,s_2)$ are routine to check.

CASE 2: $t = SUCC(s)$

By the induction hypothesis $s \equiv ENUM(SUCC^n(0))$

Subcase 2.1. If $enum(succ^n(0_R)) \in T(V,R)$ then $SUCC(s) \equiv ENUM(SUCC^m(0))$ for $(n,z) \in J_1$ & $(z+1,m) \in J_1$.

Subcase 2.2. If $enum(succ^n(0_R)) \notin T(V,R)$ then $SUCC(s) \equiv ENUM(SUCC^m(0))$ for $(n,z) \in J_2$ and $(z,w),(w+1,m) \in J_1$.

Consider $enum(succ^n(0_R)) \in T(V,R)$. Then Lemma 3.4 says that

$$SUCC(ENUM(SUCC^n(0))) \equiv SUCC(SUCC^z(0))$$
$$\equiv SUCC^{z+1}(0) \qquad \text{for } (n,z) \in J_1.$$

To make a reduction to an element of T, we have only to prefix an *ENUM* to the right-hand side by applying Lemma 3.4 again: $SUCC^{z+1}(0) \equiv ENUM(SUCC^{z}(0))$ for $(z+1,m) \in J_1$.

Consider $enum(succ^{n}(0_R)) \notin T(V,R)$. Then equation (4) and Subcase 1.2 allows us to write

$$SUCC(ENUM(SUCC^{n}(0))) \equiv SUCC(PROJ(ENUM(SUCC^{n}(0))))$$
$$\equiv SUCC(ENUM(SUCC^{z}(0))) \text{ for } (n,z) \in J_2.$$

But $enum(succ^{z}(0_R)) \in T(V,R)$ so the right-hand side is covered by Subcase 2.1. Thus

$$SUCC(ENUM(SUCC^{z}(0))) \equiv ENUM(SUCC^{m}(0)) \text{ for } (z,w) \in J_1 \text{ and } (w+1,m) \in J_1.$$

The two other arithmetical cases follow the same pattern: equations (7) and (10) guarantee that the identities of Lemma 3.4 and Subcase 1.2 can reduce the subterms to numerals. Lemma 3.4 gives the numeral which is E_0-equivalent to t. And the prefixing of an *ENUM*, to complete the reduction of t to an element of T, is again done by Lemma 3.4. We omit the details leaving them as a straightforward, if tedious, exercise for the reader.

CASE 5: $t = ENUM(s)$

By the induction step $s \equiv ENUM(SUCC^{n}(0))$.

Subcase 5.1. If $enum(succ(0_R)) \in T(V,R)$ then $ENUM(s) \equiv ENUM(SUCC^{m}(0))$ for $(n,m) \in J_1$.

Subcase 5.2. If $enum(succ(0_R)) \notin T(V,R)$ then $ENUM(s) \equiv ENUM(SUCC^{m}(0))$ for $(n,z) \in J_2$ and $(z,m) \in J_1$.

Subcase 5.1 is immediate from Lemma 3.4 which says that $ENUM(SUCC^{n}(0)) \equiv SUCC^{m}(0)$ for $(n,m) \in J_1$.

In Subcase 5.2, we may use equation (12) and Subcase 1.2 to write

$$ENUM(ENUM(SUCC^{n}(0))) \equiv ENUM(PROJ(ENUM(SUCC^{n}(0))))$$
$$\equiv ENUM(ENUM(SUCC^{z}(0))) \text{ for } (n,z) \in J_2.$$

But $enum(succ^{z}(O_R)) \in T(V,R)$ so we are in the situation of Subcase 5.1 again.

CASE 6: $t = \underline{f}(s_1,\ldots,s_k)$

By the induction hypothesis $s_i \equiv ENUM(SUCC^{n(i)}(0))$, $1 \le i \le k$. We claim that

$$\underline{f}(s_1,\ldots,s_k) \equiv ENUM(SUCC^{m}(0)) \quad \text{for } (n(1),\ldots,n(k),m) \in G(f).$$

Now, given $n = (n(1),\ldots,n(k)) \in \omega^k$ and m with $(n,m) \in G(f)$, we can choose $z = (z(1),\ldots,z(k(f)))$ such that $p_f(n,m,z) = q_f(n,m,z)$. Substituting $SUCC^{n(i)}(0)$, $SUCC^{m}(0)$ and $SUCC^{z(i)}(0)$ into the premiss of equation (13) we can (via Lemma 3.6) detach the identity

$$\underline{f}(ENUM(PROJ(SUCC^{n(1)}(0)),\ldots,ENUM(PROJ(SUCC^{n(k)}(0)))) \equiv ENUM(PROJ(SUCC^{m}(0))).$$

By Lemma 3.5 this reduces to our claim.

To complete the proof of Lemma 3.3 we have now to verify that ϕ: $R_0 \to T(\Sigma_0,E_0)$ is a homomorphism. Each constant and each operation of R_0 must be considered, but we will write out only one case which is entirely typical. We will now show that for any $x \in R_0$,

$$\phi(enum(x)) = ENUM(\phi(x)).$$

Write $x = enum(succ^{n}(O_R))$.

If $enum(succ^{n}(O_R)) \in T(V,R)$ then

$$\begin{aligned}
\phi(enum(enum(succ^{n}(O_R)))) &= \phi(enum(succ^{m}(O_R))) \text{ for } (n,m) \in J_1; \\
&= [ENUM(SUCC^{m}(0))] \text{ by definition of } \phi; \\
&= [ENUM(ENUM(SUCC^{n}(0)))] \text{ by Subcase 5.1}; \\
&= ENUM[ENUM(SUCC^{n}(0))] \text{ by definition of } ENUM; \\
&= ENUM[\phi(enum(succ^{n}(O_R)))].
\end{aligned}$$

If $enum(succ^{n}(O_R)) \notin T(V,R)$ then

$$\begin{aligned}\phi(enum(enum(succ^n(0_R)))) &= \phi(enum(proj(enum(succ^n(0_R))))) \\ &= \phi(enum(enum(succ^z(0_R)))) \text{ for } (n,z) \in J_2; \\ &= \phi(enum(succ^m(0_R))) \text{ for } (z,m) \in J_1; \\ &= [ENUM(SUCC^m(0))] \text{ by definition of } \phi; \\ &= [ENUM(ENUM(SUCC^n(0)))] \text{ by Subcase 5.2;} \\ &= ENUM[ENUM(SUCC^n(0))] \text{ by definition of } ENUM; \\ &= ENUM(\phi(enum(succ^n(0_R)))).\end{aligned}$$

This concludes the proof of Lemma 3.3.

Finally, we shall make one new axiom e_α which when added to E_0 forms $E_\alpha = E_0 \cup \{e_\alpha\}$ and completes a final algebra specification for $R_0/\equiv_\alpha$. Translating $\equiv_\alpha$ into the transversal coding we get

$$J_\alpha = \{(n,m) \in \omega\times\omega : enum(succ^n(0_R)) \not\equiv_\alpha enum(succ^m(0_R))\}.$$

By our hypothesis, this is an r.e. set and so we can define it, via the Diophantine Theorem, as

$$\{(n,m) \in \omega\times\omega : \exists z \in \omega^{k(\alpha)}.[p_\alpha(n,m,z) = q_\alpha(n,m,z)]\}.$$

Taking P_α, Q_α as formal translations of p_α, q_α we set e_α to be the formula

$$P_\alpha(X,Y,Z_1,\ldots,Z_{k(\alpha)}) = Q_\alpha(X,Y,Z_1,\ldots,Z_{k(\alpha)}) \wedge ENUM(PROJ(X)) = ENUM(PROJ(Y)) \rightarrow U = V.$$

3.7 <u>LEMMA</u>. *The specification* (Σ_0, E_α) *defines* $R_0/\equiv_\alpha$ *with respect to final algebra semantics:*

$$T_F(\Sigma, E_\alpha) \cong R_0/\equiv_\alpha.$$

<u>PROOF</u>. We prove the representation as follows. Let ψ be the unique semantic evaluation epimorphism $T(\Sigma_0) \rightarrow R_0/\equiv_\alpha$ so that $T(\Sigma_0)/\equiv_\psi$ is isomorphic to $R_0/\equiv_\alpha$. We will show that $\equiv_\psi$ is a maximal E_α-congruence on $T(\Sigma_0)$ whence it will follow that $\equiv_\psi$ is $\equiv_{\max(E_\alpha)}$ and

$$T(\Sigma_0)/\equiv_{max(E_\alpha)} = T_F(\Sigma_0,E_\alpha) \cong R_0/\equiv_\alpha .$$

It is a routine matter to check that $\equiv_\psi$ is non-unit and is, itself, an E_α-congruence. Consider its maximality. We have to show that if $\equiv$ is any non-unit E_α-congruence then $\equiv$ is a subcongruence of $\equiv_\psi$. Contrapositively, we shall argue that if $\equiv$ is an E_α-congruence which is not a subcongruence of $\equiv_\psi$ then $\equiv$ is the unit congruence.

This is done by finding terms $t,t' \in T(\Sigma_0)$ such that

(i) there exists $s = (s_1,\ldots,s_{k(\alpha)}) \in T(\Sigma_0)^{k(\alpha)}$ for which $P_\alpha(t,t',s) \equiv Q_\alpha(t,t',s)$; and

(ii) $ENUM(t) \equiv ENUM(t')$

because then we may apply conditional equation e_α to deduce $\equiv$ is unit. We have to get these terms from R_0, of course.

Using the assumption that $\equiv$ is an E_α-congruence and the initiality of R_0 for E_0-algebras (Lemma 3.3), we can define an epimorphism $\phi\colon R_0 \to T(\Sigma_0)/\equiv$ which translates $\equiv$ into the numerical congruence $\equiv_\phi$ since $R_0/\equiv_\phi$ is isomorphic with $T(\Sigma_0)/\equiv$. It is easy to see that our hypothesis $\equiv \not\subset \equiv_\psi$ means that $\equiv_\phi \not\subset \equiv_\alpha$ in R_0.

Thus, we choose $enum(succ^n(0_R))$, $enum(succ^m(0_R)) \in R_0$ such that

$$enum(succ^n(0_R)) \equiv_\phi enum(succ^m(0_R)) \text{ but } enum(succ^n(0_R)) \not\equiv_\alpha enum(succ^m(0_R)).$$

By the diophantine definition of $\not\equiv_\alpha$ there exist $z = (z(1),\ldots,z(k(\alpha))) \in \omega^{k(\alpha)}$ for which $p_\alpha(n,m,z) = q_\alpha(n,m,z)$. We set $t = SUCC^n(0)$, $t' = SUCC^m(0)$ and $s_i = SUCC^{z(i)}(0)$ for $1 \le i \le k(\alpha)$. Now condition (i) follows from Lemma 3.6, and condition (ii) from our choice of n,m and, in both cases, the initiality of R_0. Q.E.D.

This concludes the proof of Case 3.2.

3.8 PROPOSITION. *Every infinite cosemicomputable algebra possesses a computable partition.*

PROOF. Thanks to the Representation Lemma 2.5, this proposition follows from this next statement whose proof is an exercise in Recursive Function Theory:

3.9 LEMMA. *Let $\equiv$ be a co-r.e. equivalence relation on ω having infinitely many equivalence classes. Then there is a family $V = \{V_i : i \in \omega\}$ of non-empty disjoint subsets of ω such that*

(1) $\bigcup_{i\in\omega} V_i = \omega$;
(2) $n \in V_i$ *is recursive uniformly in* i;
(3) *if* $n \equiv m$ *and* $m \in V_i$ *then* $n \in V_i$.

PROOF. We will describe an effective procedure which constructs the family V in stages. These stages we index by natural numbers. At each even stage $s = 2n$ we will have started the building of $V_0,\ldots,V_{n-1}$, but no other members of V. Our task at this stage will be to give V_n its first element. At each odd stage $s = 2n+1$ we will ensure that n, itself, belongs to one of $V_0,\ldots,V_{n-1}$. Thus at the beginning of each stage s we will have made only finite parts of $V_0,\ldots,V_{n-1}$ and nothing else. Let V_i^s denote the status of V_i at the beginning of stage s.

Even from this outline it is clear that conditions (1) and (2) will hold for V. By construction,

$$n \in V_i \iff i \leq 2n \quad \& \quad n \in V_i^{2n}$$

and we will know that every n is assigned sooner or later at an odd stage. Condition (3) will be routine to check after we have described the procedure. We formalise an enumeration of $\not\equiv$ by

$$n \not\equiv m \quad \text{if, and only if,} \quad \exists k.R(k,n,m)$$

for some recursive predicate R.

Stage $s = 2n$.

Now $V_0^{s-1},\ldots,V_{n-1}^{s-1}$ are non-empty, but $V_n^{s-1} = \emptyset$. We want to name the first element of V_n. We enumerate the finite set $V^{s-1} = V_0^{s-1} \cup\ldots\cup V_{n-1}^{s-1}$ searching for some $z \in \omega$ such that for all $m \in V^{s-1}$, $z \not\equiv m$.

Such an element z will exist because $\omega/\equiv$ is infinite. This z is put into V_n with the result that at the conclusion of this stage

$$V_i^s = V_i^{s-1} \quad \text{for } 0 \le i \le n-1 \quad \text{and } V_n^s = \{z\}.$$

Stage s = 2n+1

Again $V_0^{s-1},\dots,V_{n-1}^{s-1}$ are non-empty but we are concerned only with the number n. First, we recursively decide whether $n \in V^{s-1} = V_0^{s-1} \cup \dots \cup V_{n-1}^{s-1}$. If this is so we are done and at the conclusion of this test $V_i^s = V_i^{s-1}$ for $0 \le i \le n-1$.

Assume $n \in V^{s-1}$. Now we will put this n in some V_i, $1 \le i \le n-1$. By searching sufficiently far out in the enumeration of $\not\equiv$ it is possible to find some k_0 and an $1 \le i \le n-1$ such that for every $j \ne i$, and $0 \le j \le n-1$, and for every $m \in V_j^{s-1}$ there is a $k < k_0$ for which R(k,m,n) is true. That is we will come across a V_i^{s-1} for which we can verify that $n \not\equiv m$ for $m \in V^s - V_i^{s-1}$. We put $n \in V_i^{s-1}$. Thus, at the end of this case of stage s = 2n+1

$$V_j^s = V_j^{s-1}, \quad \text{for } j \ne i \text{ and } 1 \le j \le n-1, \quad \text{and } V_i^s = V_t^{s-1} \cup \{n\}.$$

This construction proves Lemma 3.9 and so concludes the proofs of Proposition 3.8, and of our main theorem. Q.E.D.

4. SEMICOMPUTABLE DATA TYPES

Our characterisation theorem for cosemicomputable data types focusses attention on a question we noticed and left open in the first paper of our series [2] (see also [7]). We shall reformulate it now as an opinion:

4.1 CONJECTURE. *Let* A *be an algebra finitely generated by elements named in its signature* Σ. *Then there exist* $N \in \omega$ *and* $M = M(|\Sigma|) \in \omega$ *such that the following are equivalent:*

1. A *is semicomputable.*
2. A *possesses a conditional equation specification, involving at most* N *hidden functions and* M *conditional equations, which defines* A *as a*

hidden enrichment under its initial algebra semantics. Moreover, we expect that $N \leq 6$ *and* $M \leq 20 + \rceil\Sigma\rceil$.

Since (2) implies (1) by Basic Lemma 2.7, the conjecture is the statement that (1) implies (2). Actually, we did not ask for bounds in [2], but we do so here although the unbounded adequacy problem remains open. Until the conjecture is settled, the precise numerical values of the bounds are of secondary importance, of course.

The theoretical importance of a confirmation of the conjecture is evident. First, semicomputable data types abound and one simply wants an adequacy theorem for them (one sharper than the result we proved in [2], certainly). And, secondly, if Conjecture 4.1 could be turned into a theorem then it would completely resolve the debate between the advocates of initial and final algebra semantics for specifications, at least for *theoria* if not for *praxis*. It seems hard to imagine a more elegant state of affairs than that depicted in the Venn diagram of Figure 4.1.

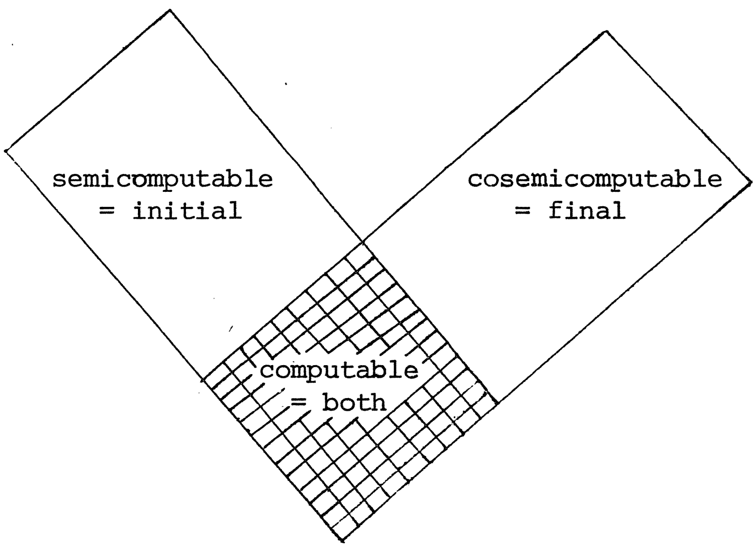


Figure 4.1

We will conclude this paper by explaining the extent to which its methods fail to establish our conjecture.

Assuming A to be semicomputable, we can first of all dispense with the finite case because we proved the existence of a bounded conditional equational specification for it in [5] (1 hidden function, 1 identification and 2 conditional equations are sufficient for *any* finite data type!) Now, if A is infinite then it turns out that a small adaptation to the proof of Proposition 3.2 will settle Conjecture 4.1 under the hypothesis that A has a computable partition. Let us explain this.

The first change in the proof of Proposition 3.2 is made at the relatively late stage of the construction of the last axiom e_α from a diophantine definition of the r.e. set J_α. As A is semicomputable we want to consider the complement of J_α instead: since

$$\neg J_\alpha = \{(n,m) \in \omega\times\omega : \text{enum}(\text{succ}^n(O_R)) \equiv_\alpha \text{enum}(\text{succ}^m(O_R))\}$$

is r.e. we can define it, via the Diophantine Theorem, as

$$\{(n,m) \in \omega\times\omega : \exists z \in \omega^{k(\alpha)}.[p_\alpha(n,m,z) = q_\alpha(n,m,z)]\}$$

for (new) polynomials p_α, q_α. Taking P_α, Q_α as formal versions of p_α, q_α we take, as the new e_α, the axiom

$$P_\alpha(X,Y,Z_1,\ldots,Z_{k(\alpha)}) = Q_\alpha(X,Y,Z_1,\ldots,Z_{k(\alpha)}) \\ \to \text{ENUM}(\text{PROJ}(X)) = \text{ENUM}(\text{PROJ}(Y)).$$

The redefined specification (Σ_0, E_α) specifies $R/\equiv_\alpha$ under its initial algebra semantics: a fact which can be readily verified and is much easier than Lemma 3.7. Thus, we know this next fact which imporves our earlier bounded adequacy theorem for computable data types in [4], and obtains for us the Second Characterisation Theorem stated in the Introduction.

4.2 THEOREM. *Let* A *be an infinite semicomputable algebra, finitely generated by elements named in its signature. If* A *has a computable partition then* A *possesses a conditional equation specification, involving* 5 *hidden functions and* 15 + $|\Sigma|$ *conditional equations, which defines* A *as a hidden enrichment under its initial algebra semantics.*

Unfortunately our strategy for the semicomputable case breaks down at the last minute:

4.3 THEOREM. *There exists a finitely generated semicomputable algebra (having an initial algebra specification without hidden functions and with only 3 equations!) which does not possess a computable partition.*

The algebra in question is that in Example 2.8 and Theorem 4.3 is merely a rephrasing of Scott's Theorem about the term model of combinatory logic: Scott has shown that one cannot even computably partition TMCL into *two* sets, see BARENDREGHT [1], Theorem 2.21.

REFERENCES

[1] BARENDREGT, H.P., *The type free lambda calculus,* in J.K. Barwise, *Handbook of mathematical logic,* North-Holland, Amsterdam, 1977, 1091-1132.

[2] BERGSTRA, J.A. & J.V. TUCKER, *Algebraic specifications of computable and semicomputable data structures,* Mathematical Centre, Department of Computer Science Research Report IW 115, Amsterdam, 1979.

[3] ———— , *A characterisation of computable data types by means of a finite, equational specification method,* in J.W. de Bakker & J. van Leeuwen (eds.) *Automata, languages and programming, Seventh colloquium, Noordwijkerhout, 1980,* Springer-Verlag, Berlin, 1980, 76-90.

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

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

[6] ―――――― , *A natural data type with a finite equational final semantics specification but no effective equational initial semantics specification,* Mathematical Centre, Department of Computer Science Research Report IW 133, Amsterdam, 1980.

[7] ―――――― , *On the adequacy of finite equational methods for data type specification,* ACM-SIGPLAN Notices 14 (11) (1979) 13-18.

[8] BROY, M., W. DOSCH, H. PARTSCH, P. PEPPER & 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.

[9] GOGUEN, J.A., J.W. THATCHER, E.G. WAGNER & J.B. WRIGHT, *Abstract data types as initial algebras and correctness of data representations,* in *Proceedings ACM Conference on Computer Graphics, Pattern Recognition and Data Structure,* ACM, New York, 1975, 89-93.

[10] GOGUEN, J.A., J.W. THATCHER & E.G. WAGNER, *An initial algebra approach to the specification, correctness and implementation of abstract data types,* in R.T. Yeh (ed.) *Current trends in programming methodology* IV, *Data structuring,* Prentice-Hall, Engelwood Cliffs, New Jersey, 1978, 80-149.

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

[12] GUTTAG, J.V. & J.J. HORNING, *The algebraic specification of abstract data types,* Acta Informatica 10 (1978) 27-52.

[13] HORNUNG, G. & P. RAULEFS, *Terminal algebra semantics and retractions for abstract data types,* in J.W. de Bakker & J. van Leeuwen, *Automata, languages and programming, Seveth colloquium, Noordwijkerhout* 1980, Springer-Verlag, Berlin, 1980, 310-325.

[14] KAMIN, S., *Final data type specifications: a new data type specification method,* in *Seventh ACM Principles of Programming Languages Conference,* Las Vegas, ACM, 1980, 131-138.

[15] KAPUR, D. & M.K. SRIVAS, *Expressiveness of the operation set of a data abstraction,* in *Seventh ACM Principles of Programming Languages Conference,* Las Vegas, ACM, 1980.

[16] LISKOV, B. & S. ZILLES, *Specification techniques for data abstractions,* IEEE Transactions on Software Engineering 1 (1975) 7-19.

[17] MACHTEY, M. & P. YOUNG, *An introduction to the general theory of algorithms,* North-Holland, New York, 1978.

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

[19] MANIN, Y., *A course in mathematical logic,* Springer-Verlag, New York, 1977.

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

[21] THATCHER, J.W., E.G. WAGNER & J.B. WRIGHT, *Specification of abstract data types using conditional axioms,* IBM Research Report RC 6214, Yorktown Heights, 1979.

[22] ———— , *Data type specification: parametrization and the power of specification techniques,* IBM Research Report RC 7757, Yorktown Heights, 1979.

[23] WAND, M., *Final algebra semantics and data type extensions,* J. Computer and Systems Sciences 19 (1979), 27-44.

[24] WIRSING, M. & M. BROY, *Abstract data types as lattices of finitely generated models,* Mathematical foundations of computer science, Eighth Symposium Rydzyna 1980, Springer-Verlag, Berlin, 1980.

[25] ZILLES, S., *Algebraic specification of data types,* Project MAC Progress Report 11, M.I.T., Cambridge, 1974.

[26] ———— , *An introduction to data algebras,* working paper, IBM Research Laboratory, San Jose, California, 1975.