ALGEBRAICALLY SPECIFIED PROGRAMMING SYSTEMS AND HOARE'S LOGIC

J.A. Bergstra

Department of Computer Science,
University of Leiden,
Wassenaarseweg 80,
2300 RA LEIDEN,
The Netherlands.

J.V. Tucker*

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

INTRODUCTION

In this paper we will look at the structure of Hoare-like logics which are designed to prove partial correctness properties of programs belonging to algebraically specified programming systems. By an *algebraically specified programming system* we have in mind a program language possessing a selection of deterministic assignment and control constructs, and a fixed finite collection of data types defined by an algebraic specification (using initial algebra semantics). We will be interested in Hoare logics which are intrinsically defined by these languages in the sense that all assertions about the underlying data types, allowed in program correctness proofs, must be formally derivable from their algebraic specifications. Thus, viewed from the point of view of specification languages for data types, the basic question we will be exploring is *"To what extent can information about a data type and, in particular, about the computations it supports, be 'encoded' in an algebraic specification for the type?"*.

To begin with, let us recall the rôle intended for a data type specification in the construction of a programming system. A syntactic specification (Σ,E) is supposed to axiomatically characterise a data type semantics in terms of properties E of the type's primitive operators Σ. An algebraic specification, in conjunction with initial algebra semantics, achieves this in a straightforward proof-theoretical way: given syntactic expressions, or terms, t and t' over Σ then t and t' are *semantically equivalent* if, and only if, one can formally *prove* that $t = t'$ from the axioms of E. At first sight, it seems that little else beyond these *correctness of representation assertions* can be extracted from a specification by formal deductions. For example, consider the data type of natural numbers N equipped with zero, successor and predecessor. An obvious specification for N consists of the operator signature $\Sigma = \{0,\mathrm{SUCC},\mathrm{PRED}\}$ and the set E of axioms

$$\mathrm{PRED}(0) = 0 \qquad \mathrm{PRED}(\mathrm{SUCC}(X)) = X.$$

* Present address: *School of Mathematics, University of Bristol, University Walk, BRISTOL BS8 1TW, England.*

But assertions like

$$X = 0 \vee SUCC(PRED(X)) = X \text{ and } 0 \neq SUCC(0)$$

which are clearly true in the initial model N are not provable from E. In designing a Hoare logic for an algebraically specified programming system we would do well to avoid negated and disjunctive formulae altogether.

Now the programming systems we *want* to analyse are those modelled by standard while-programs computing on a single-sorted structure defined by an algebraic specification (Σ,E). Because of the special nature of assertions provable from algebraic axioms, we wish to experiment with Hoare logics based upon assertion languages consisting of *finite conjunctions of equations only.* But such a language *EL* is incompatible with the sort of boolean tests appearing in the control structures of standard while-programs. We dissolve this difficulty by applying the thesis that programming constructs should be designed with the problem of proving statements about their behaviour clearly in mind, a thesis associated with the names E.W. Dijkstra, R.W. Floyd and C.A.R. Hoare. To match the correctness proofs, which will involve equational assertions only, we design a new set of control structures, allowing only equational tests, and then derive some proof rules about their operation. This new algebraically styled programming language we call the set of *equational* while-*programs EWP*; it has essentially the same computing strength as the standard while-programs (Theorem 2.2). With these preparations, we can consider our original problem well-posed: *Can an algebraic specification for a programming language be made to axiomatise information required for correctness proofs for its programs?* We prove the following adequacy theorem for algebraic specifications and their algebraic logics for program correctness (Theorem 4.1):

THEOREM. *Let* A *be any infinite computable data type of signature* Σ. *Let* $S_1,\ldots,S_n$ *be equational* while-*programs over* Σ. *For* $i = 1,\ldots,n$ *let* p_i *and* q_i *be any precondition and postcondition for* S_i *taken from* $EL(\Sigma)$. *If the partial correctness statements* $\{p_i\}S_i\{q_i\}$ $(1 \leq i \leq n)$ *are provable in the Hoare logic for EWP which allows any true computable assertion about* A *in its correctness proofs then there exists a finite equational specification* (Σ_0,E_0), *involving at most* 6 *auxiliary operators and* 4 *equations only, such that*

(1) (Σ_0,E_0) *defines* A *under initial algebra semantics; and*

(2) *the statements* $\{p_1\}S_1\{q_1\},\ldots,\{p_n\}S_n\{q_n\}$ *can be proved in the equational Hoare logic for EWP using equational assertions from* $EL(\Sigma_0)$ *all of which are provable from the axioms of* E_0.

The existence of such a concise specification for computable data types is of interest independently of the extra proof-theoretical information it can be expected to contain (Theorem 4.2). Notice the number of equations does not even depend upon the number of operators of the data type.

To date, the general proof theory of algebraic specifications has not received the especial attention it deserves although its problematic nature is well-known:

it arises frequently in studies of the correctness of data type specifications made from Horn formulae - for example, ADJ [29], EHRIG et al. [18]; and in work on data type specification languages - for example, BURSTALL & GOGUEN [16] and GOGUEN & TARDO [20]. An attempt at a systematic treatment of the subject is contained in the interesting thesis of KAPUR [24], but a great deal more theoretical research needs to be done.

This paper is the seventh in our series on the power and adequacy of the algebraic specification methods for data types [9-14] (see also [15]); an important sequel is [7]. This paper is also related to our work with J. Tiuryn on first-order axiomatically specified programming systems [8].

We assume the reader is well versed in the theory of algebraic specifications for data types and is familiar with logics for partial correctness. The basic references for these subjects are ADJ [21] and HOARE [23], COOK [17], respectively; also the reader may care to consult the survey paper APT [1]. Knowledge of our earlier articles is desirable, but it is not strictly necessary.

Finally, we thank W.P. de Roever and K.R. Apt for focussing our attention on the proof-theoretic capacities of algebraic specifications in seminars of the Programming Language Semantics Workgroup of the Mathematical Centre and the University of Utrecht.

1. DATA TYPES

Syntactically, our programming systems are modelled by a pair $[(\Sigma,E), PROG(\Sigma)]$ consisting of an algebraic specification (Σ,E) and a set of program schemes $PROG(\Sigma)$ based upon the operator names contained in the signature Σ. Semantically, we model these languages by a pair $[A,PROG(A)]$ wherein A is an algebra of signature Σ defined by the specification (Σ,E), under initial algebra semantics, and PROG(A) is the set of all partial functions on A computable by the program schemes in $PROG(\Sigma)$ interpreted in A. The specific program schemata in which we will be interested are discussed in the next section; here we collect together some remarks about the syntax and semantics of data type specifications.

A data type will be modelled by a single-sorted algebra finitely generated by elements named in its signature. (The restriction to single-sorted structures is made for convenience in notations and to enable us to better explain the mathematical issues involved; readers acquainted with our earlier work will see immediately how to write this paper in its many-sorted generalisation.) All signatures are finite and all specifications use either *equations* or *conditional equations* as axioms. The semantics of a specification (Σ,E) will always be its *initial algebra semantics*. Thus, the unique meaning of the specification (Σ,E) is the initial algebra $I(\Sigma,E)$ of the category $ALG(\Sigma,E)$ containing all Σ-algebras satisfying the axioms of E. By $T(\Sigma,E)$ we denote the *standard term algebra construction* of $I(\Sigma,E)$; that is the factor algebra of the Σ-term algebra $T(\Sigma)$ determined by the least E-congruence on

$T(\Sigma)$.

A given algebra A has a *finite equational* (or *conditional equational*) *specification* (Σ,E) if the signature of A is Σ, E is a finite set of equations (or conditional equations) over Σ, and $A \cong T(\Sigma,E)$.

We allow hidden operators into specifications in precisely the following way.

Let A be an algebra of signature Σ_A and let Σ be a signature extended by Σ_A; that is $\Sigma \subset \Sigma_A$. Then we mean by $A|_\Sigma$ the Σ-algebra whose domain is that of A and whose operations and constants 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 operations and constants of A named in Σ *viz.* the smallest Σ-subalgebra of $A|_\Sigma$.

A given algebra A of signature Σ has a *finite equational* (or *conditional equational*) *hidden enrichment specification* (Σ_0,E_0) if $\Sigma \subset \Sigma_0$ and E is a finite set of equations (or conditional equations) over Σ such that

$$T(\Sigma_0,E_0)\big|_\Sigma = \langle T(\Sigma_0,E_0)\rangle_\Sigma \cong A$$

Finally, we formalise the concept of a computable data type using the standard definition of a *computable algebra* due to M.O. RABIN [28] and A.I. MAL'CEV [26].

An algebra A is said to be *computable* if there exists a recursive set of natural numbers Ω and a surjection $\alpha\colon \Omega \to A$ such that to each k-ary operation σ of A there corresponds a recursive *tracking function* $\bar{\sigma}\colon \omega^k \to \omega$ which commutes the following diagram,

$$\begin{tikzcd}
A^k \arrow[r, "\sigma"] & A \\
\Omega^k \arrow[u, "\alpha^k"] \arrow[r, "\bar{\sigma}"] & \Omega \arrow[u, "\alpha"']
\end{tikzcd}$$

wherein $\alpha^k(x_1,\ldots,x_k) = (\alpha x_1,\ldots,\alpha x_k)$. And, furthermore, the relation $\equiv_\alpha$, defined on Ω by $x \equiv_\alpha y$ iff $\alpha(x) = \alpha(y)$ in A, is recursive.

In this formal definition, the notion becomes a so-called *finiteness condition* of algebra: an isomorphism invariant possessed of all finite structures. Equally important is this other invariance property (MAL'CEV [26]):

If A is a finitely generated algebra computable under both $\alpha\colon \Omega_\alpha \to A$ and $\beta\colon \Omega_\beta \to A$ then α and β are *recursively equivalent* in the sense that there exist recursive functions f,g which commute the diagram:

$$\begin{tikzcd}
& A & \\
\Omega_\alpha \arrow[ur, "\alpha"] \arrow[rr, shift left, "f"] & & \Omega_\beta \arrow[ul, "\beta"'] \arrow[ll, shift left, "g"]
\end{tikzcd}$$

A corollary of this property is the following theorem.

If A is computable under coordinatisation α then a set $S \subset A^n$ is said to be (α-) *computable* if the set $\alpha^{-1}(S) = \{(x_1,\ldots,x_n) \in \Omega^n\colon (\alpha x_1,\ldots,\alpha x_n) \in S\}$ is recursive.

<u>1.1. THEOREM.</u> *Let* A *be a finitely generated computable algebra, and* $S \subset A^n$. *If* S *is computable with respect to one computable codification of* A *then it is compu-*

table with respect to every computable codification of A.

Given A computable under α then combining the associated tracking functions on the domain Ω makes up a recursive algebra of numbers from which α is an epimorphism to A. Applying the recursiveness of $\equiv_\alpha$ to this observation it is easy to prove this useful fact.

1.2. LEMMA. *Every computable algebra* A *is isomorphic to a recursive number algebra* Ω *whose domain is the set of natural numbers,* ω, *if* A *is infinite, or else is the set of the first* m *natural numbers,* ω_m, *if* A *is finite of cardinality* m.

A reference for the elementary theory of the recursive functions is MACHTEY & YOUNG [25]; however, our main tool is the Diophantine Theorem which we now state (a good account of this subject is contained in MANIN [27]).

Let $\mathbb{Z}[X_1,\ldots,X_n]$ denote the ring of polynomials with integer coefficients in indeterminates $X_1,\ldots,X_n$. A set $\Omega \subset \omega^k$ is said to be *diophantine* if there exists a polynomial $p \in \mathbb{Z}[X_1,\ldots,X_k, Y_1,\ldots,Y_\ell]$ such that for $x = (x_1,\ldots,x_k) \in \omega^k$ and $y = (y_1,\ldots,y_\ell) \in \omega^\ell$

$$x \in \Omega \Longleftrightarrow \exists y \in \omega^\ell.[p(x,y) = 0].$$

Equivalently, a diophantine set Ω can be defined by asking for polynomials $p,q \in \omega[X_1,\ldots,X_k, Y_1,\ldots,Y_\ell]$, the semiring of polynomials with natural number coefficients in the indeterminates $X_1,\ldots,X_k$, $Y_1,\ldots,Y_\ell$, such that for $x = (x_1,\ldots,x_k) \in \omega^k$ and $y = (y_1,\ldots,y_\ell) \in \omega^\ell$,

$$x \in \Omega \Longleftrightarrow \exists y \in \omega^\ell.\ [p(x,y) = q(x,y)].$$

Clearly, each diophantine set is recursively enumerable; the converse is due to Y. Matijacevič:

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

2. WHILE-PROGRAMS

Let Σ be a signature and let $WP = WP(\Sigma)$ denote the class of standard while-programs over Σ. For the semantics of WP we leave the reader free to choose any sensible account of while-program computations applicable to an arbitrary Σ-structure A, from the graph-theoretical semantics of GREIBACH [22] to the denotational semantics of DE BAKKER [6]. For the purposes at hand, perhaps a naive operational view would be best [30], but the reader's choice can hardly be problematical.

The class of equational while-programs $EWP = EWP(\Sigma)$ represents a modified program formulae, one designed to avoid the use of negations and disjunctions because the Hoare logics we have in mind to service algebraic specifications are proof systems based upon equational first-order formulae. The class EWP is inductively defined from assignment statements by means of composition, guarded conditionals and the while-construct augmented by an algebraic assertion as a correctness check:

ASSIGNMENT.	For X a program variable and t a polynomial expression over Σ we may form an *assignment statement* X : = t .
COMPOSITION.	For S_1 and S_2 equational <u>while</u>-programs we may form their *composition* $S_1; S_2$.
GUARDED CONDITIONALS.	For t_i and t_i' $(1 \leq i \leq k)$ polynomial expressions over Σ and S_i $(1 \leq i \leq k)$ equational <u>while</u>-programs we may form the *guarded conditional* $(t_1 = t_1' \rightarrow S_1 \square \ldots \square t_k = t_k' \rightarrow S_k)$.
ITERATION WITH CORRECTNESS CHECK.	For t, t', r, s polynomial expressions over Σ and S an equational <u>while</u>-program then we may form the *guarded iteration* <u>while</u> t = t' <u>do</u> S <u>od</u> <u>now</u> <u>check</u> r = s <u>won</u>.

It is quite adequate for the technical work to follow to give an informal description of the semantics of equational <u>while</u>-program computations. The semantics of the assignment statements and composition operations are handled in the usual way (of the reader's chosen semantics). For the conditional operator and the iteration operator the reader must formalise the following naive operational meanings for these constructs:

An execution of the guarded conditional operator on initial state σ involves an arbitrary but effectively computable choice of one index $1 \leq i \leq k$ for which $t_i = t_i'$ is true of σ after which S_i is executed on σ. The execution results in a divergent computation whenever none of the guards $t_i = t_i'$ is true. Thus, we require a deterministic implementation of the command's usual non-deterministic semantics.

An execution of the iteration construct on initial state σ corresponds to the usual execution of the <u>while</u>-construct except that for termination executing the embedded <u>while</u>-construct on σ must lead to a final state for which r = s holds true.

For A any Σ-structure, let $\mathcal{WP}(A)$ and $\mathcal{EWP}(A)$ denote the sets of all partial functions on A computable by the programs of $\mathcal{WP}$ and $\mathcal{EWP}$ respectively. We conclude this section with a comparison of the computing powers of these two classes of programs (Theorem 2.2).

First of all, let $\mathcal{WP}_0 = \mathcal{WP}_0(\Sigma)$ be the class of all those standard <u>while</u>-programs which involve boolean tests in their conditional and <u>while</u>-constructs only of the forms

$$t = t \quad \text{or} \quad t \neq t'$$

for t, t' polynomial expressions over Σ. Let $\mathcal{WP}_0(A)$ be the set of all functions on Σ-structure A computable by programs from $\mathcal{WP}_0$. The proofs of the following facts are routine exercises

<u>2.1. LEMMA</u>. *For any Σ-structure* A, $\mathcal{WP}_0(A) = \mathcal{WP}(A)$.

<u>2.2. THEOREM</u>. *Let* A *be any structure. Then* $\mathcal{EWP}(A) \subset \mathcal{WP}(A)$. *If* A *possesses*

constants T,F *and a binary operator*

$$E(a,b) = \begin{cases} T & \text{if } a = b \\ F & \text{if } a \neq b \end{cases}$$

then $EWP(A) = WP(A)$.

3. HOARE LOGICS FOR EQUATIONAL WHILE-PROGRAMS

Having settled on the programming formalism EWP for operating with algebraically specified data types, it remains for us to provide it with the two Hoare logics for proving partial correctness properties for its computations. The first Hoare logic $HL(EL(\Sigma),EO(E))$ has an algebraic form and is designed for use with algebraically specified programming systems $[(\Sigma,E),EWP(\Sigma)]$. Its principal characteristics are an equational assertion language $EL(\Sigma)$ and an oracle $EO(E)$ for the Rule of Consequence which consists of those equational assertions *provable* from the data type specification (Σ,E).

The second Hoare logic $HL(CL(A),CO(A))$ is made to model a Hoare logic whose assertion language $CL(A)$ defines precisely the decidable assertions about a computable data type A and has as oracle the set of all decidable assertions $CO(A)$ *true* of A.

We shall define both these Hoare logics as particular instances of a general description of Hoare logics for EWP. This general format is made inside the infinitary language $L_{\omega_1,\omega} = L_{\omega_1,\omega}(\Sigma)$ based upon the signature Σ as this language is sufficiently expressive to faithfully represent $CO(A)$ whereas first-order logic is not. (In this use of $L_{\omega_1,\omega}$ to circumvent expressibility problems in the logic of program correctness we follow ENGELER [19] and BACK [4,5]).

Let L be a sublanguage of $L_{\omega_1,\omega}(\Sigma)$ by which we mean L is a set of infinitary formulae closed under finite conjunctions and substitutions. The basic syntactic object of a Hoare-like logic with assertion language L is the *L-asserted program*. This is an expression of the form $\{p\}S\{q\}$ where S is a program and $p,q \in L$ and, in this paper, the finitely many free variables of S, p, q coincide.

Let $O \subset L \times L$ such that if $(\alpha,\beta) \in O$ then the formulae α and β have the same finite set of free variables.

The *Hoare logic* $HL(L,O)$ for EWP based upon *assertion language* L *and oracle* O is defined as the set of all asserted programs $\{\alpha\}S\{\beta\}$ for $\alpha,\beta \in L$ and $S \in EWP$ generated by the following axioms and proof rules: let $S,S_1,\dots,S_k \in EWP$; $p,q,p_1,q_1,r \in L$; and let $t,t',t_1,t_1',\dots,t_k,t_k',s,s'$ be polynomial expressions over Σ.

1. ASSIGNMENT AXIOM SCHEME. The asserted program

$$\{p[t/X]\}\ X := t\{p\}$$

is an axiom where $p[t/X]$ stands for the result of substituting the expression t for free occurrences of variable X in p.

2. COMPOSITION RULE. $$\frac{\{p\}S_1\{r\},\{r\}S_2\{q\}}{\{p\}S_1;S_2\{q\}}$$

3. GUARDED CONDITION RULE.
$$\frac{\{p\wedge t_1=t_1'\}S_1\{q\},\ldots,\{p\wedge t_k=t_k'\}S_k\{q\}}{\{p\}(t_1=t_1'\rightarrow S_1\Box\ldots\Box t_k=t_k'\rightarrow S_k)\{q\}}$$

4. ITERATION RULE.
$$\frac{\{p\wedge t=t'\}S\{p\}}{\{p\}\ \underline{\text{while}}\ t=t'\ \underline{\text{do}}\ S\ \underline{\text{od}}\ \underline{\text{now}}\ \underline{\text{check}}\ s=s'\ \underline{\text{won}}\ \{p\wedge s=s'\}}$$

5. CONSEQUENCE RULE. $$\frac{(p,p_1)\in\mathcal{O},\{p_1\}S\{q_1\},\ (q_1,q)\in\mathcal{O}}{\{p\}S\{q\}}\ .$$

Notice that all proofs in HL(L,$\mathcal{O}$) are finitely long.

The semantics of HL(L,$\mathcal{O}$) is simply that of the partial program correctness semantics for asserted programs derived from the standard satisfaction semantics of the infinitary formulae of the assertion language. Thus, a given asserted program {p}S{q}, with S, p, q having n free variables, is said to be *valid* over a Σ-structure A if for $\sigma \in$ States(A), whenever $A \models p(\sigma)$ then either $S(\sigma)$ converges and $A \models q(S(\sigma))$ or else $S(\sigma)$ diverges. We shall abbreviate validity by $A \models \{p\}S\{q\}$.

The *partial correctness theory* of *EWP* in language *L* over Σ-structure A is defined by

$$PC(L,A) = \{\{p\}S\{q\}: A \models \{p\}S\{q\} \text{ for } S \in EWP,\ p,q \in L\}.$$

A Hoare logic HL(L,$\mathcal{O}$) is said to be *sound* for structure A if HL(L,$\mathcal{O}$) ⊂ PC(L,A) The oracle $\mathcal{O}$ is said to be *valid* over a structure A if for any $(p,q) \in \mathcal{O}$, with p,q having n free variables, and for any $a \in A^n$, $A \models p(a) \rightarrow q(a)$.

3.1 SOUNDNESS THEOREM. *Let* HL(L,$\mathcal{O}$) *be a Hoare logic and* A *any* Σ*-structure. If the oracle* $\mathcal{O}$ *is valid for* A *then the Hoare logic* HL(L,$\mathcal{O}$) *is sound.*

The proof of Theorem 3.1 we leave as an easy exercise for the reader and his or her semantics for *EWP*. The following observation is obvious.

3.2 FINITENESS LEMMA. *Suppose* HL(L,$\mathcal{O}$) ⊢ {p}S{q} *for* p,q ∈ *L and* S ∈ *EWP*. *If* $O_{p,q} \subset \mathcal{O}$ *is the set of all oracle assertions appearing in some proof of* {p}S{q} *then* HL(L,$O_{p,q}$) ⊢ {p}S{q}.

3.3 EQUATIONAL HOARE LOGIC. Given an algebraic specification (Σ,E) we assign to it an *equational Hoare logic* HL(EL(Σ),$E\mathcal{O}$(E)) defined by taking the assertion language L to be the set EL(Σ) of all *finite conjunctions of equations* over Σ and taking as the oracle $\mathcal{O}$ the set $E\mathcal{O}$(E) of all pairs of finite conjunctions of equations $(p,q) \in EL(\Sigma) \times EL(\Sigma)$ such that

$$E \vdash p \rightarrow q.$$

Thus, HL(EL(Σ),$E\mathcal{O}$(E)) is an entirely syntactical construction and

$$HL(EL(\Sigma),\ E\mathcal{O}(E)) \vdash \{p\}S\{q\}$$

tells us that the pre- and post- conditions p and q are finite conjunctions of equations defining a partial correctness statement provable from equational information derivable from the axioms E.

3.4. COMPUTABLE HOARE LOGIC. Given a computable data type A of signature Σ we assign to it a *Hoare logic of computable assertions* $HL(CL(A),CO(A))$ defined by taking the assertion language L to be the set $CL(A)$ of all infinitary formulae $p \in L_{\omega_1,\omega}$ such that the set $\{a \in A^n : A \models p(a)\}$ is computable. Notice this $CL(A)$ is an absolutely well-defined construction thanks to Theorem 1.1. As an oracle O we take the set $CO(A)$ of *all* pairs of infinitary formulae $(p,q) \in CL(A) \times CL(A)$ such that $A \models p \to q$. Thus, $HL(CL(A),CO(A))$ is a semantical construction and

$$HL(CL(A),CO(A)) \vdash \{p\}S\{q\}$$

tells us that the pre- and post- conditions are decidable predicates defining a partial correctness statement *using true computable intermediate assertions only*: See APT, BERGSTRA & MEERTENS [3] and APT [2] for a discussion of this hybrid type of Hoare logic and its mathematical structure.

3.5. BASIC OBSERVATIONS. *For any computable data type* A *of signature* Σ *, each computable subset* $S \subset A^n$ *is definable in* $CL(A)$. *Clearly,* $EL(\Sigma) \subset CL(A)$.

4. THE ADEQUACY THEOREM

4.1. THEOREM. *Let* A *be an infinite computable data type of signature* Σ. *Suppose that*

$$HL(CL(A),CO(A)) \vdash \{p_i\}S_i\{q_i\}$$

wherein $S_i \in EWP(\Sigma)$ and $p_i,q_i \in EL(\Sigma)$ *for* $i = 1,\dots,n$. *Then there exists an equational specification* (Σ_0,E_0), *with* $\Sigma_0 - \Sigma$ *containing at most 5 new function symbols and 1 constant and with* E_0 *containing 4 equations over* Σ_0, *such that*

(1) *under its initial algebra semantics* (Σ_0,E_0) *defines* A *as a hidden enrichment specification, and*

(2) $HL(EL(\Sigma_0),EO(E_0)) \vdash \{p_i\}S_i\{q_i\}$ *for* $i = 1,\dots,n$.

PROOF. We will divide the proof into two largely independent blocks. First of all, let A be isomorphic to a recursive number algebra R with domain ω (Lemma 1.2). We will make a new recursive number algebra R_H, of signature Σ_H, such that $R_H|_\Sigma = \langle R_H\rangle_\Sigma = R$. And we will make a set of conditional equations E_H, which are true of R_H,

$$R_H \models E_H$$

and for which

$$HL(EL(\Sigma_H),EO(E_H)) \vdash \{p_i\}S_i\{q_i\} \quad (1 \le i \le n).$$

The second block is the proof of the following general specification cum compression theorem.

4.2. SPECIFICATION THEOREM. *Let* A *be an infinite computable algebra finitely generated by elements named in its signature* Σ. *Then there exists a specification* (Σ_0,E_0), *in which* Σ_0 *extends* Σ *by 5 new function symbols and 1 new constant and*

E_0 *contains only 4 equations over* Σ_0, *such that* (Σ_0, E_0) *defines* A *as a hidden enrichment specification under its initial algebra semantics.*

Moreover, for any finite set E *of conditional axioms over* Σ *satisfied by* A, (Σ_0, E_0) *can be chosen so that each axiom of* E *is formally provable by the rules of first-order logic from* E_0.

Our theorem now follows immediately from these two blocks. In the Specification Theorem 4.2, take $A = R_H$ as the algebra to be specified and take $E = E_H$ as the axioms to be compressed. The specification (Σ_0, E_0) specifies R_H and since E_0 proves E we know that E_0 proves the $\{p_i\}S_i\{q_i\}$ in the equational Hoare logic over Σ_H. To obtain the result of our main theorem we recover R from R_H and check the numerical bounds claimed; these latter tasks are trivial, of course. Consider the part of the proof devoted to the Hoare logics involved.

For notational convenience we take n = 1. Suppose that $HL(CL(R), CO(R)) \vdash \{p\}S\{q\}$ wherein $p,q \in CL(R)$ are conjunctions of equations. Let P be a proof of this fact in the Hoare logic and let $\{{}^1P, \ldots, {}^{\ell}P\}$ be a list of all the formulae of $CL(R)$ occurring in P. Let $X_1, \ldots, X_k$ be a list of all the free variables mentioned in the formulae of P. Now, each formula ${}^iP$ arising in the proof P can be assumed to be factorised into the form

$$ {}^iP = \bigwedge_{j=1}^{a(i)} {}^iP_j $$

where ${}^iP_j$ is either an equation over Σ or is some formula of $CL(R)$ that is neither an equation, nor a conjunction of two other formulae of $CL(R)$. We shall transform P into a proof $\phi(P)$ is an equational Hoare logic and we propose to do this by replacing these latter complex subformulae of the ${}^iP$ with equations over a signature Σ_H extending Σ; thus, ${}^iP$ is turned into a formula $\phi({}^iP)$ which is a finite conjunction of equations over Σ_H. Replacing each occurrence of ${}^iP$ in P by the formula $\phi({}^iP)$ results in a syntactical object $\phi(P)$ which *looks* like a proof of $\{p\}S\{q\}$ in an equational Hoare logic over Σ_H. What remains is the task of finding an oracle to define a Hoare logic in which $\phi(P)$ is indeed such a proof. And, of course, we have to show that the oracle can be specified by a finite set of conditional axioms.

The formal rôle of the algebra R_H is to prove the consistency of these syntactic manoeuvres and to act as a template for the second half of the proof which applies the Specification Theorem 4.2. But it seems best to introduce R_H straightaway to explain the idea behind our choice of Σ_H.

For each $1 \le i \le \ell$, let $I_i \subset \{1, \ldots, a(i)\}$ denote the set of indices for those subformulae ${}^iP_j$ of ${}^iP$ which are not equations. To define R_H we add to R the numbers $0,1,2 \in R$ as distinguished constants and also these two functions

$$ \underline{succ}(x) = x+1 $$

$$ \underline{sat}(x_1, \ldots, x_k, i, j) = \begin{cases} 1 & \text{if } 0 \le i \le \ell,\ j \in I_i \text{ and } A \models {}^iP_j(x_1, \ldots, x_k); \\ 2 & \text{if } 0 \le i \le \ell,\ j \in I_i \text{ and } A \not\models {}^iP_j(x_1, \ldots, x_k); \\ 0 & \text{otherwise.} \end{cases} $$

Since each ${}^{i}P_j$ defines a computable predicate on R, the function <u>sat</u> is recursive. Let the signature of R_H be $\Sigma_H = \Sigma \cup \{0, \underline{TRUE}, \underline{FALSE}, \underline{SUCC}, \underline{SAT}\}$.

The syntactic transformation of the proof P into $\phi(P)$ proceeds as follows. Given the formula ${}^{i}P$ of P, we leave alone all those components ${}^{i}P_j$ which are already equations over Σ, and we replace each ${}^{i}P_j$ which is not by

$$\underline{SAT}(X_1,\ldots,X_k,\underline{SUCC}^i(0),\underline{SUCC}^j(0)) = \underline{TRUE}$$

which is an equation over Σ_H. The resulting formula $\phi({}^{i}P)$ is a finite conjunction of equations over Σ_H as expected; and therefore, replacing every occurrence of every ${}^{i}P$ in the proof P produces $\phi(P)$ which *could* be a proof of the asserted program $\{p\}S\{q\}$ is an equational Hoare logic over Σ_H. To define that Hoare logic we must inspect the oracle axioms appearing in the proof P.

Let $Q = \{Q_1 \to Q_1', \ldots, Q_t \to Q_t'\}$ be a list of every use of the oracle $CO(R)$ in the proof P. By the Finiteness Lemma 3.2,

$$HL(CL(R),Q) \vdash \{p\}S\{q\}.$$

Since each Q_i and Q_i', for $1 \le i \le t$, are some ${}^{\lambda}P$ and ${}^{\mu}P$ we can define

$$\phi(Q) = \{\phi(Q_1) \to \phi(Q_1'),\ldots,\phi(Q_t) \to \phi(Q_t')\}.$$

A trivial induction on proof structure allows us to conclude that

$$HL(EL(\Sigma_H),\phi(Q)) \vdash \{p\}S\{q\}.$$

Thus to complete this stage of the argument we have only to get the oracle $\phi(Q)$ specified by a set E_H of conditional equations over Σ_H. Now remember that each formula $\phi(Q_i) \to \phi(Q_i')$ is *almost* a conditional equation: the deviation is that $\phi(Q_i')$ is a conjunction of equations over Σ_H. The following lemma shows how to unpick the conjunctions of $\phi(Q_i')$ to form a set of conditional equations E_H; its proof is a simple logical exercise.

<u>4.3 LEMMA</u>. *Let* Γ *be any signature and let* $\{r_i(X) = r_i'(X)\colon 1 \le i \le n\}$ *and* $\{s_j(X) = s_j'(X)\colon 1 \le j \le m\}$ *be two sets of equations over* Γ *in a list of variables X. Then for any formula* $\Phi \in L(\Gamma)$ *the following are equivalent:*

1. $$\left\{\bigwedge_{i=1}^{n} r_i(X) = r_i'(X) \to \bigwedge_{j=1}^{m} s_j(X) = s_j'(X)\right\} \vdash \Phi$$

2. $$\left\{\bigwedge_{i=1}^{n} r_i(X) = r_i'(X) \to s_j(X) = s_j'(X)\colon 1 \le j \le m\right\} \vdash \Phi.$$

<u>PROOF OF THE SPECIFICATION THEOREM</u>. First, let A be infinite and isomorphic with a recursive number algebra R whose domain is ω (Lemma 1.2). We add the following constants and operations to R to make a new recursive number algebra R_ω

$$0,\ x+1,\ x+y,\ x.y$$

(If R contains any of these functions beforehand then some of this list is redundant, of course: R_H already possesses zero and the successor function remember.)

Next, let k denote the maximum number of conjunctions occurring in the premisses of the conditional axioms in E, or let k = 1 if E contains only equations. Without loss of generality, we can assume every conditional equation of E has k conjunctions in their premisses by padding with trivially valid equations X = X. Thus, each conditional equation in E has the form

$$t_1 = t_1' \wedge \ldots \wedge t_k = t_k' \rightarrow t = t'.$$

We now define two more recursive functions which must be added to R_ω.

$$d(x,y,z) = \begin{cases} 0 & \text{if } x=y \text{ and } z=0; \\ 1 & \text{otherwise} \end{cases}$$

$$h(x_1,y_1,\ldots,x_k,y_k,z) = \begin{cases} z & \text{if } \wedge_{i=1}^{k} x_i = y_i; \\ 0 & \text{otherwise.} \end{cases}$$

Let R_0 be the result of adding these 5 functions and 1 constant to R. Clearly, $R_0|_\Sigma = \langle R_0 \rangle_\Sigma = R$. Let $\Sigma_0 = \Sigma \cup \{0,SUCC,ADD,MULT,D,H\}$ be the signature of R_0. We shall construct a specification (Σ_0,E_0) which encorporates the conditional equations E, specifies R_0 under its initial algebra semantics and uses only 4 equations. This construction proceeds in several stages the first of which ends with a conditional specification of R_0.

<u>4.4. LEMMA</u>. R_0 *possesses an initial algebra specification* (Σ_0,E_1) *in which* E_1 *contains at most* $6 + |\Sigma|$ *conditional equations each one of which has at most* 1 *premiss.*

<u>PROOF</u>. The equations for the arithmetic are

$ADD(X,0) = X;$ $\quad$ $MULT(X,0) = 0$

$ADD(X,SUCC(Y)) = SUCC(ADD(X,Y));$ $\quad$ $MULT(X,SUCC(Y)) = ADD(MULT(X,Y),X)$

For each constant $\underline{c} \in \Sigma$ naming number $c \in R$ take the identification

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

For each function symbol $\underline{f} \in \Sigma \cup \{D,H\}$ naming function $f\colon \omega^n \rightarrow \omega$ which is either an operator of R, or is d or h we construct a conditional equation as follows. Consider the graph of f,

$$G(f) = \{(x_1,\ldots,x_n,y) \in \omega^{n+1}\colon f(x_1,\ldots,x_n) = y\}.$$

This is an r.e. set and so, by the Diophantine Theorem, there exist polynomials p_f and q_f from $\omega[X,Y,Z] = \omega[X_1,\ldots,X_n,Y,Z\ ,\ldots,Z_m]$ such that

$$G(f) = \{(x,y) \in \omega^n \times \omega\colon \exists z \in \omega^m . p_f(x,y,z) = q_f(x,y,z)\}.$$

Let P_f and Q_f be formal translations of p_f and q_f to polynomials over {0,SUCC,ADD, MULT}. For the function symbol $\underline{f}$ we assign the conditional equation

$$P_f(X,Y,Z) = Q_f(X,Y,Z) \rightarrow \underline{f}(X) = Y.$$

This completes the definition of E .

The proof that $T(\Sigma_0,E_1) \cong R_0$ begins by defining $\phi\colon R_0 \rightarrow T(\Sigma_0,E_1)$ by

$$\phi(n) = [SUCC^n(0)]$$

where $[SUCC^n(0)]$ is the equivalence class of terms in $T(\Sigma_0)$ which are E_1-equivalent to $SUCC^n(0)$. This map ϕ is the required isomorphism. The proof is a routine exercise and is, in fact, a simplified version of the corresponding proof in [11]. We take the liberty of omitting it. Q.E.D.

Now we must absorb the conditional equations of E. Take the conditional equations of E_1 and pad out their premisses to contain k conjunctions of equations, if necessary. (Here it is important that $k \geq 1$.) This done, set $E_2 = E \cup E_1$.

We will now describe a transformation of the set of *conditional equations* E_2 to a set of *equations* E_3 satisfying these three conditions:

1. $|E_3| = |E_2| + 1$
2. $E_3 \vdash E_2$
3. $R_0 \models E_3$.

The technique is quite general and we will use it again in a moment.

The first and "extra" equation in E_3 is simply

$$H(X_1,X_1,\ldots,X_k,X_k,Z) = Z$$

The rest are made to correspond to the conditional equations of E_2.

For each conditional equation

$$t_1 = t_1' \wedge \ldots \wedge t_k = t_k' \rightarrow t = t'$$

in E_2 we write the equation

$$H(t_1,t_1',\ldots,t_k,t_k',t) = H(t_1,t_1',\ldots,t_k,t_k',t')$$

This is all of E_3. Condition (1) is obvious and the arguments for properties (2) and (3) are straightforward logical exercises.

Now we are going to transform back the set of equations E_3 into a set E_4 of conditional equations! However, this E_4 will contain only 3 conditional equations, consistent with R_0, and be able to formally prove the equations of E_3. The first two elements of E_4 are

$$D(X,Y,Z) = 0 \rightarrow X = Y \text{ and } D(X,Y,Z) = 0 \rightarrow Z = 0.$$

Let $E_3 = \{r_1=s_1,\ldots,r_\ell=s_\ell\}$ for $\ell = 6 + |\Sigma| + |E| + 1$. From this list we inductively define the following polynomials

$$D_1 = D(r_1,s_1,0) \qquad D_{i+1} = D(r_i,s_i,D_i)$$

for $i = 1,\ldots,\ell-1$ and take the equation

$$D_\ell = 0,$$

to make E_4. Again the properties we claimed for E_4 are routine matters to verify.

The final stage is an application of our technique which turns conditional equations into equations. This produces a set of equations E_5 such that

4. $|E_5| = |E_4| + 1 = 4$
5. $E_5 \vdash E_4$
6. $R_0 \models E_5$.

This E_5 is the set of equations E_0 required for the statement of the Specification Theorem 4.2.

To see that E_0 proves the given conditional equations, recall the chain

$$E_0 = E_5 \vdash E_4 \vdash E_3 \vdash E_2 = E \cup E_1.$$

And that (Σ_0,E_0) specifies R_0 follows from this chain, condition (6) and Lemma 4.4.

Q.E.D.

We are indebted to Ms. Rita Martin (School of Mathematics, University of Bristol) for making this fine typescript.

REFERENCES

[1] APT, K.R., *Ten years of Hoare's logic, a survey* in F.V. JENSEN, B.H. MAYOH and K.K. MØLLER (eds.), *Proceedings from 5th Scandinavian Logic Symposium*, Aalborg University Press, Aalborg, 1979, 1-44. (A second edition of this paper will appear in ACM Transactions on Programming Languages and Systems.)

[2] ———— , *Recursive assertions and parallel programs*, Preprint Erasmus University, Rotterdam, 1979.

[3] APT, K.R., J.A. BERGSTRA & L.G.L.T. MEERTENS, *Recursive assertions are not enough - or are they?*, Theoretical Computer Science 8 (1979), 73-87.

[4] BACK, R.J.R., *Correctness preserving program refinements: proof theory and applications*, Mathematical Centre Tracts 131 (1980), Amsterdam.

[5] ————, *Proving total correctness of nondeterministic programs in infinitary logic*, to appear in Acta Informatica.

[6] DE BAKKER, J.W., *Mathematical theory of program correctness*, Prentice-Hall International, London, 1980.

[7] BERGSTRA, J.A., & J. TERLOUW, *A completeness result for algebraic Hoare logics*, University of Leiden, Department of Computer Science Report 81-09, Leiden 1981.

[8] BERGSTRA, J.A., J. TIURYN & J.V. TUCKER, *Floyd's principle correctness theories and program equivalence*, Mathematical Centre, Department of Computer Science Research Report IW 145, Amsterdam, 1980. (To appear in Theoretical Computer Science.)

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

[10] ——— & ——— , *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, 7th colloquium, Noordwijkerhout, 1980*, Springer-Verlag, Berlin, 1980, 76-90.

[11] ——— & ——— , *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.

[12] ——— & ——— , *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.

[13] ——— & ——— , *A natural data type with a finite equational final semantics specification but no effective equational initial semantics specification*, Bulletin European Association for Theoretical Computer Science, 11 (1980), 22-33.

[14] ——— & ——— , *Initial and final algebra semantics for data type specifications: two characterisation theorems*, Mathematical Centre, Department of Computer Science Research Report IW 142, Amsterdam, 1980.

[15] ——— & ——— , *On the adequacy of finite equational methods for data type specifications*, ACM-SIGPLAN Notices 14 (11) (1979) 13-18.

[16] BURSTALL, R.M. & J.A. GOGUEN, *Putting theories together to make specifications*, Proceedings 5th International Joint Conference on Artificial Intelligence, Cambridge, Mass., 1977, 1045-1058.

[17] COOK, S.A., *Soundness and completeness of an axiom system for program verification*, SIAM J. Computing 7 (1978) 70-90.

[18] EHRIG, H., H.-J. KREOWSKI, J.W. THATCHER, E. WAGNER, J.B. WRIGHT, *Parameterized data types in algebraic specification languages*, in J.W. DE BAKKER and J. VAN LEEUWEN (eds.), *Automata, languages and programming, 7th Colloquium, Noordwijkerhout*, 1980, Springer-Verlag, Berlin, 1980, 157-168.

[19] ENGELER, E., *Algorithmic logic*, in J.W. DE BAKKER (ed.), *Foundations of Computer Science*, Mathematical Centre Tracts 63 (1975) 57-85.

[20] GOGUEN, J.A. & J.J. TARDO, *An introduction to OBJ: a language for writing and testing formal algebraic program specifications*, Proceedings IEEE Conference on Specifications of Reliable Software, Cambridge, Mass., 1979, 170-189.

[21] 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.

[22] GREIBACH, S.A., *Theory of program structures: schemes, semantics, verification*, Springer-Verlag, Berlin, 1975.

[23] HOARE, C.A.R., *An axiomatic basis for computer programming*, Communications Association Computing Machinery 12 (1969) 576-583.

[24] KAPUR, D., *Towards a theory for abstract data types*, M.I.T. Laboratory for Computer Science Research Report TR-237, Boston, 1980.

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

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

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

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

[29] THATCHER, J.W., E.G. WAGNER & J.B. WRIGHT, *Data type specification: parameterizations and the power of specification techniques*, IBM-T.J. Watson Research Center Report RC 7757, Yorktown Heights, 1979.

[30] TUCKER, J.V., *Computing in algebraic systems*, in F.R. DRAKE and S.S. WAINER (eds.), *Recursion theory, its generalisations and applications*, Cambridge University Press, Cambridge, 1980, 215-235.