ON THE POWER OF ALGEBRAIC SPECIFICATIONS 1)

J.A. Bergstra*,

M. Broy**,

J.V. Tucker***,

M. Wirsing****

ABSTRACT

We study the expressive power of different algebraic specification methods. In contrast to (nonhierarchical) initial and terminal algebra specifications which correspond to semicomputable and cosemicomputable algebras, hierarchical specifications - as e.g. in the specification language CLEAR - allow to specify hyperarithmetical algebras and are characterized by them. For partial abstract types we prove that every computable partial algebra has an equational hidden enrichment specification and discuss the power of hierarchical partial algebras. Finally we give an example of the specification of a simple nondeterministic programming language.

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

**) Technische Universität München, Institut für Informatik, Postfach 20 40 20, D-8000 München 2, FRG.

***) University of Bristol, School of Mathematics and Computer Science, University Walk, Bristol BS8 1TW, England.

****) University of Edinburgh, Dept. of Computer Science, Mayfield Road, Edinburgh EH9 3JZ, Scotland.

1) This research was supported in part by the Sonderforschungsbereich 49 - Programmiertechnik, Munich.

INTRODUCTION

In the past years several algebraic specification methods have been developed. Initial specifications (cf. [ADJ 78, BG 77, EL 80, EKTWW 80, VP 80]) can be seen as term rewrite systems and so have an immediate implementation; terminal algebras (cf. [GGM 76, Ka 80, HR 80, Wa 77]) describe the behaviour system; hierarchical specifications (cf. [BaW 81, G 75, BG 80, Lo 80]) support the modularization, and partial algebras seem well-suited to specify semantics of programming languages (cf. [Ga 80, Pa 80, KP 81]). In order to understand all these different specification methods it seems interesting to compare their expressive power and to analyse the computability of their models (cf. also [BMM 79, BMMW 79, Mj 79]). Let us always consider abstract types with conditional equations as axioms, i.e. axioms of the form $u_1=v_1 \wedge \ldots \wedge u_n=v_n \Rightarrow u=v$, and suppose first we have the usual heterogeneous total algebras as models. Then every (nonhierarchical) initial algebra I is semicomputable i.e. the equality in I is recursively enumerable. Similarly for every terminal algebra Z it is recursively enumerable whether any two terms are different in Z [BT 80]. Thus Z is cosemicomputable.

On the other hand every cosemicomputable algebra A has a terminal (hidden) enrichment specification T [BT 80] i.e. A together with some (hidden) functions is a terminal algebra of T. Every semicomputable algebra A has an initial implementation specification I [BT 79] i.e. A can be seen as a subalgebra of an initial algebra of I (cf. [EL 80, EKTWW 80]). It is still an open problem whether every semicomputable algebra has an initial hidden enrichment specification. The different problems for implementation and enrichment specifications can be seen by comparing the implementation specification of finite data structures in [AK 80] which needs only 6 equations whereas the hidden enrichment specification in [BM 80] contains more than 80 equations. But computable algebras have terminal as well as initial enrichment specifications.
A hierarchical type T - as it is possible to define in CLEAR (cf. [BG 80]), by the CANON specification method (cf. [R 80]), or in CIP-L (cf. [CIP 81]) - is a type together with a (primitive) subtype S. In every model of T a term the sort of which is a sort of S has to be interpreted by an ("old") term of S (cf. [BDPPW 79]). As long as hierarchical types are sufficiently complete ([G 75]) this extra condition does not change anything: initial models are semicomputable and terminal ones cosemicomputable.

But for types which are not sufficiently complete we will show that over computable algebras universal and existential quantifiers and even hyperarithmetical functions are definable. Every computably hyperarithmetical algebra, i.e. an algebra with a recursive domain and hyperarithmetical functions, has an (initial as well as a terminal) hierarchical enrichment specification. This implies that every hyperarithmetical algebra, i.e. an algebra where the domain may be hyperarithmetical as well, with a computable (hyperarithmetical) subtype has a hierarchical enrichment specification,too. Conversely every initial and terminal algebra of a hierarchical type is hyperarithmetical.
Although partial functions occur everywhere in program specification and partial types allow to specify programming languages independently from fixed point theory [BW 80,81], not so much work has been done in this field (apart from the deep studies of Reichel [R 79] and Andreka et al [ABN 80]). There are different possibilities to extend the equality relation to undefined terms [BW 80b]. We consider here the "existential equality" which holds for two terms if both are defined and denote the same object. Then terminal algebras are no longer cosemi-computable, but Δ_2^0. Every initial partial algebra of a (nonhierarchical) type, however, is semicomputable and every semicomputable partial algebra has a partial initial implementation specification - as announced in [H 80].

We will show that - as in the case of total algebras - every computable partial algebra has an equational partial initial (and terminal) enrichment specification. The results for arithmetical and hyperarithmetical algebras carry over, too.

Finally we consider a simple nondeterministic programming language. The equality for this language is neither recursively enumerable nor corecursively enumerable but Δ_2^0. We give first a "natural" terminal enrichment specification by partial abstract types and derive then a hierarchical enrichment specification by total abstract types.

I. TOTAL ABSTRACT TYPES

1. Specifications and their Semantics

For the definitions of S-sorted signature Σ, Σ-homomorphism, term algebra $W(\Sigma)$ and the class $ALG(\Sigma)$ of all Σ-algebras we refer e.g. to [ADJ 78] or [BDPPW 79]; for the definitions concerning formal logic and recursiveness we refer to /Sh 67/ and /Ro 67/.

For the sake of notational simplicity only we restrict our attention to single and two-sorted structures; no difficulty at all is encountered in extending the given definitions and results.

1.1 Data Structures and Abstract Types

A (data) structure is defined to be a heterogeneous algebra finitely generated by elements named in its signature Σ, a so-called (finitely generated) prime algebra. An abstract type consists of a signature Σ and a (finite) set E of conditional equations of the form

$$u_1=v_1 \wedge \ldots \wedge u_n=v_n \Rightarrow u=v$$

where u_i, v_i, u, v are Σ-terms containing free variables, i.e. they are in $W(\Sigma, x_1, \ldots, x_k)$. The class of all algebras of signature Σ satisfying the axioms E is denoted by $ALG(\Sigma,E)$.

1.2 Initial and Terminal Algebras

An initial algebra I in a class C of algebras is characterized by the fact that for every structure $B \in C$ there exists a (unique) homomorphism $\varphi: I \to B$. Let $I(\Sigma,E)$ denote an initial algebra in $ALG(\Sigma,E)$; this $I(\Sigma,E)$ always exists and is unique up to isomorphism. On the other hand, terminal algebras for $ALG(\Sigma,E)$ would always be the trivial one-point, or unit, Σ-algebra $1 \in ALG(\Sigma,E)$, where every carrier set consists of exactly one object. Instead, terminal algebra semantics turns to $ALG_0(\Sigma,E)$ with the unit algebra removed. $Z \in C_0$ is called terminal in C, if for every $B \in C_0$ there exists (at least) one homomorphism $\varphi: B \to Z$. Terminal data structures of $ALG_0(\Sigma,E)$ are denoted by $Z(\Sigma,E)$. Unfortunately, $ALG_0(\Sigma,E)$ need not always possess a terminal data structure, but when it exists it is unique (cf. [Wa 77, WB 80]).

Given any data structure A of signature Σ there is a semantic mapping $v_A: W(\Sigma) \to A$ which evaluates the formal expressions over Σ as data belonging to A. v_A induces a congruence $\equiv_A$ on $W(\Sigma)$, defined by $t \equiv_A t'$ iff $v_A(t) = v_A(t'$
Often we write t^A for $v_A(t)$, $W_I(\Sigma,E)$ for $W(\Sigma)/\equiv_{I(\Sigma,E)}$ and analogously $W_Z(\Sigma,E)$. For $I(\Sigma,E)$ $t \equiv_{I(\Sigma,E)} t'$ iff $E \vdash t=t'$ is always true, whereas the terminal algebras satisfy

$$t \equiv_{Z(\Sigma,E)} t' \text{ iff } t=t' \text{ is consistent with E.}$$

Consistency simply means that there is some non-unit model $B \in ALG_0(\Sigma,E)$ where $B \models t=t'$. Hence (if Σ is one sorted) for $t,t' \in W(\Sigma)$ $Z(\Sigma,E) \models t \neq t'$ iff $E \cup \{t=t'\} \vdash \forall x,y\colon x=y$.

1.3 Hierarchical Types

A hierarchical abstract type $T = (\Sigma,E,P)$ is a type in which the type $P = (\Sigma',E')$ is designated as primitive where $\Sigma' \subseteq \Sigma$ and $E' \subseteq E$. A term $t \in W(\Sigma; x_1,\ldots,x_m)$ is called primitive; if $t=f(t_1,\ldots,t_n)$ where $f: s_1 x \ldots x s_n \to s, s \in S'$ then t is called of primitive sort. In general, there exist terms of primitive sort which are not primitive. Let A be an algebra of signature Σ and let Σ' be a signature $\Sigma' \subseteq \Sigma$. We mean by $A_{|\Sigma'}$ the Σ'-algebra whose domain is that of A and operators are those of A named in Σ', and by $\langle A\rangle_{\Sigma'}$ the Σ'-subalgebra of A generated by the constants and operators named in Σ' viz. the smallest Σ'-subalgebra of $A_{|\Sigma'}$.

Then a Σ-algebra A is hierarchical wrt Σ' iff $A_{|\Sigma'} = \langle A\rangle_{\Sigma'}$, i.e. the Σ'-reduct $A_{|\Sigma'}$ of A is a Σ'-data structure. The carrier sets s^A of A where s is a primitive sort are finitely generated by the primitive constants and operations only. The class of all Σ-algebras which are hierarchical wrt Σ' is denoted by $ALG(\Sigma,E,P)$ (and by $ALG_0(\Sigma,E,P)$ if all algebras the Σ'-reducts of which are the unit algebras are removed). Initial and terminal algebras in the category of hierarchical algebras are denoted by $I(\Sigma,E,P)$ and $Z(\Sigma,E,P)$. These initial and terminal algebras do not always exist, but if they exist, they are unique up to isomorphism. As before we denote $W(\Sigma)/\equiv_{I(\Sigma,E,P)}$ by $W_I(\Sigma,E,P)$ and $W(\Sigma)/\equiv_{Z(\Sigma,E,P)}$ by $W_Z(\Sigma,E,P)$.

1.4 Specifications with Hidden Operators

A hierarchical abstract type (Σ,E,P) where $\Sigma = (S,F)$ is said to be an (1) initial or (2) terminal conditional hierarchical (hidden) implementation specification of the algebra A with signature $\Sigma_A = (S_A, F_A)$, if $S_A = S$, $F_A \subseteq F$, and E is a finite set of conditional equations over the finite signature Σ such that

(1) $\langle W_I(\Sigma,E,P)\rangle_{\Sigma_A} \cong A$

or

(2) $\langle W_Z(\Sigma,E,P)\rangle_{\Sigma_A} \cong A$

Thus an implementation specification does not introduce new sorts but allows that A is isomorphic to a (proper) subalgebra of $W_I(\Sigma,E,P)$ or $W_Z(\Sigma,E,P)$. If A is isomorphic to $W_I(\Sigma,E,P)$ or $W_Z(\Sigma,E,P)$ itself, then (Σ,E,P) is an enrichment specification: (Σ,E,P) is called (1) initial or (2) terminal conditional hierarchical (hidden) enrichment specification of A if it is a (1) initial or (2) terminal conditional hierarchical (hidden) implementation specification and if

(1) $W_I(\Sigma,E,P)|_{\Sigma_A} \cong A$ or (2) $W_Z(\Sigma,E,P)|_{\Sigma_A} \cong A$.

If the type is not hierarchical, i.e. $P=\emptyset$, we speak of initial and terminal enrichment specifications. A specification being both initial and terminal is called monomorphic.

2. Computability of Algebras

A countable algebra A is said to have a presentation when it is given an effective coordinatisation consisting of a recursive set $\Omega \subseteq N$, a surjection $\alpha: \Omega \to A$, and, for each k-ary operation f of A, a tracking function $\bar{f}$ which commutes the following diagram

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

where $\alpha^k(x_1,\ldots,x_k) = (\alpha(x_1),\ldots,\alpha(x_k))$.

We classify the complexity of A according to the complexity of $\bar{f}$ and the complexity of the relation $\equiv_\alpha$ on Ω which is defined by

$n \equiv_\alpha m$ iff $\alpha(n) = \alpha(m)$ in A.

A is said to be computable, (semicomputable, cosemicomputable) if there exists a presentation of A such that $\equiv_\alpha$ is recursive (r.e., co-r.e.) and $\bar{f}$ is recursive (in all three cases).

A relation $P \subseteq N\times N$ is arithmetical if it has an explicit definition

$P(n,m) \Leftrightarrow Q_1x_1\ldots Q_kx_kR(n,m,x_1,\ldots,x_k)$

where R is a recursive relation and each Q_i is a quantifier $\forall$ or $\exists$.

P is $\Pi^1_1(\Sigma^1_1)$ if it is defined by

$P(n,m) \Leftrightarrow \forall\alpha\exists yR(n,m,\alpha,y)$ $(P(n,m) \Leftrightarrow \exists\alpha\forall yR(n,m,\alpha,y))$

where α is a variable for functions from N to N and R is recursive. P is hyperarithmetical iff it is both Π^1_1 and Σ^1_1. A function f is arithmetical (hyperarithmetical) iff its graph G_f is so.

Now, an algebra A is arithmetical (hyperarithmetical) if it has a presentation such that $\equiv_\alpha$ and all $\bar{f}$ are arithmetical (hyperarithmetical). A is computably arithmetical (computably hyperarithmetical), if it has an arithmetical (hyperarithmetical) presentation for which $\equiv_\alpha$ is recursive.

For any of the above algebras we have the following

Representation Lemma

A is isomorphic to the factor algebra $R/\equiv_\alpha$ where R is an algebra with domain N. If $\equiv_\alpha$ is recursive then it is isomorphic to an algebra with domain N provided A is infinite.

3. Characterization Results

For nonhierarchical types we only have to put together well-known results:

Let (Σ,E) be an abstract type with conditional equations as axioms. We assume for the rest of the paper that every algebra is infinite. In [BT 80a] it is proved that every finite algebra possesses an initial hidden enrichment specification which is also a final specification.

According to the characterization in I.2 the equality in the initial algebras is recursively enumerable as well as the inequality in terminal algebras. Therefore the definitions of (co-) semicomputability imply

Proposition 1

Every initial algebra of (Σ,E) is semicomputable and every terminal algebra is cosemicomputable.
If initial and terminal algebras of a type are isomorphic then their equality is recursive and we can apply the theorems in [BT 80] and [BMMW 79].

Proposition 2

For an algebra, A, finitely generated by elements named in its signature, the following statements are equivalent:

(1) A is computable.

(2) A possesses an equational enrichment specification being both initial and terminal.

(3) A possesses an isoinitial equational enrichment specification.

The algebraic notion of isoinitial algebra was introduced by [BMM 79] in order to capture the recursiveness of abstract types: an algebra A in $ALG_0(\Sigma,E)$ is isoinitial iff for all $B \in ALG_0(\Sigma,E)$ there exists an isomorphic embedding $\varphi: A \to B$.

Proof of Proposition 2

(3) $\Rightarrow$ (1): According to Proposition 1 the equality between terms without free variables of the specification type is recursive. Proposition 9 of [BMMW 79] says that the isoinitial model is computable.

(1) $\Rightarrow$ (2): By the second characterization theorem of [BT 80] we obtain an equational initial specification for A over the natural numbers N with an arithmetic containing a constant $0: \to N$ and the successor operator $S: N \to N$. In order to obtain a terminal specification we have to ensure that in all non-unit models $S^m(0) \neq S^n(0)$ holds for all $m,n \in N$ with $m \neq n$. This is equivalent to

(*') $S^m(0) = S^n(0) \Rightarrow X = Y \qquad (m \neq n)$

Now (*') can be simulated by

(*) $X = COND(DIST(0,S(X)),X,Y)$

where $COND: N^3 \to N$ and $DIST: N^2 \to N$ are two hidden functions defined by

$COND(0,Y,Z) = Z$, $COND(S(X),Y,Z) = Y$

(**) $DIST(S(X),S(Y)) = DIST(X,Y)$ $DIST(0,X) = X$
$DIST(0,X) = DIST(X,0)$

Then the initial specification turns over in a specification, being both initial and terminal.

(2) $\Rightarrow$ (3): Apply Proposition 5 and 7 of [BMMW 79]. □

For the (co-) semicomputability we apply the characterizations of [BT 79,80].

Proposition 3

Let A be an algebra finitely generated by elements named in its signature. Then the following holds:

(1) A is semicomputable iff A possesses an equational initial implementation specification.

(2) A is cosemicomputable iff A possesses a conditional terminal enrichment specification.

Now, let (Σ,E,P) be a hierarchical specification.
If the specification is sufficiently complete, i.e. for every term $t \in W(\Sigma)$ of primitive sort there exists a primitive term p such that $E \vdash t=p$ holds, then any hierarchical initial (or terminal) specification is also a nonhierarchical initial (or terminal, resp.) specification since then $ALG(\Sigma,E,P) = ALG(\Sigma,E,\emptyset)$. Therefore sufficiently complete hierarchical specifications are nothing other than the usual initial and terminal specifications. But the situation changes substantially for those types which are not sufficiently complete and admit initial or terminal extensions. Suddenly, as for optimal fixed points (cf. [MS 76]), we can specify quantifiers over the natural numbers:

Proposition 4

Let $f,g: N^n \to N$ be total recursive functions. Then there exist equational monomorphic hierarchical enrichment specifications of the natural numbers enriched by the following functions:

(a) $\exists x_1 \in N: f(x_1,\dots,x_n)>0$, i.e. of the function, $ex_f: N^{n-1} \to N$, defined by

$$ex_f(x_2,\dots,x_n) = \begin{cases} 1 & \text{if } \exists x_1 \in N: f(x_1,\dots,x_n)>0 \\ 0 & \text{if } \forall x_1 \in N: f(x_1,\dots,x_n)=0 \end{cases}$$

(b) $\neg f(x_1,\dots,x_n)>0$, i.e. of the function, not: $N^n \to N$, defined by

$$\text{not}(x_1,\dots,x_n) = \begin{cases} 1 & \text{if } f(x_1,\dots,x_n) = 0 \\ 0 & \text{if } f(x_1,\dots,x_n)>0 \end{cases}$$

(c) $f(x_1,\dots,x_n)>0 \wedge g(x_1,\dots,x_n)>0$ i.e. of the function, and: $N^n \to N$, defined by

$$\text{and}(x_1,\dots,x_n) = \begin{cases} 1 & \text{if } f(x_1,\dots,x_n)>0 \text{ and } g(x_1,\dots,x_n)>0 \\ 0 & \text{otherwise.} \end{cases}$$

Proof

Since f is total recursive the algebra N_f of the natural numbers together with the functions 0, successor, addition, multiplication, and f is computable. According to Proposition 2 N_f has an equational monomorphic enrichment specification FNAT with sort nat and the function symbols 0, S, ADD, MULT, COND, and F. We define a hierarchical type EXNAT with primitive type FNAT by introducing two new function symbols EX_F: $\underline{nat}^{n-1} \to \underline{nat}$ and EX_F': $\underline{nat}^n \to \underline{nat}$ which are specified by

$$EX_F(X_2,\dots,X_n) = COND(EX_F'(0,X_2,\dots,X_n),1,0)$$
(***)
$$EX_F'(X_1,\dots,X_n) = COND(F(X_1,\dots,X_n),1,MULT(2,EX_F'(S(X_1),X_2,\dots,X_n)))$$

EX_F' can be seen as embedding function of the function EX_F which specifies the existential formula. We show that the unique solution of the recursive equations implies:

$$(1)\quad EX_F(X_2,\dots,X_n) = \begin{cases} 0 & \text{if } \forall X_1 \in N: F(X_1,\dots,X_n)=0 \\ 1 & \text{if } \exists X_1 \in N: F(X_1,\dots,X_n) > 0 \end{cases}$$

$$(2)\quad EX_F'(X_1,\dots,X_n) = \begin{cases} 0 & \text{if } \forall X \geq X_1: F(X_1,\dots,X_n)=0 \\ 2^{X_0-X_1} & \text{if } X_0 = \mu X: X \geq X_1 \wedge F(X,X_2,\dots,X_n)>0 \end{cases}$$

where "μX" means "the least X such that".

Consider a natural number N. Then (since F is a total function) $EX_F'(M,X_2,\dots,X_n)=1$ for the least $M \geq N$ such that $F(M,X_2,\dots,X_n)>0$. If such an M exists then $EX_F'(N,X_2,\dots,X_n)$ is 2 to the power of M_0-N, where M_0 is the first such M found.

If no such M_0 can be found then 0 is the unique possible value of a fixed point for $EX_F'(N,X_2,\dots,X_n)$. The fact that 0 is a possible value can be seen by direct evaluation. Suppose that there is some other possible value V. Then V should satisfy $V=MULT(2^{X-N},EX_F'(S(X),X_2,\dots,X_n))$ for every $X \geq N$. If V>0, this cannot hold, whatever the value of $EX_F'(S(X),X_2,\dots,X_n)$ is.

Thus (1) and (2) is the unique solution of (***) for (every standard model of) the natural numbers. The hierarchical type EXNAT has (up to isomorphism) exactly one hierarchical model. Hence by definition this model is initial as well as terminal.
Analogously, NOT: $\underline{nat}^n \to \underline{nat}$ and AND: $\underline{nat} \to \underline{nat}$ can be specified by

$$NOT(X_1,\dots,X_n) = COND(F(X_1,\dots,X_n),0,1)$$
$$AND(X_1,\dots,X_n) = COND(MULT(F(X_1,\dots,X_n),G(X_1,\dots,X_n)),1,0)$$ □

Proposition 5

Any computably arithmetical algebra has an equational monomorphic hierarchical enrichment specification.

Proof

Every infinite computably arithmetical algebra is by the representation lemma isomorphic to a number algebra R with carrier set N. Every function f in R has its graph g in the arithmetical hierarchy, i.e.

$$g(x,y) \iff \exists x_1 Q_2 x_2 \dots Q_i x_i: h(x_1,\dots,x_n,x,y)>0$$
or $$g(x,y) \iff \forall x_1 Q_2 x_2 \dots Q_i x_i: h(x_1,\dots,x_n,x,y)>0$$

where $0 \le i \le n$ and h is a total recursive function. Due to Proposition 4 a little induction proves that R has the property we want. ◻

An application of a theorem of Spector (cf. [Ro 67, p.421-425]) yields the following generalization of Proposition 5 replacing computably arithmetical by computably hyperarithmetical.

<u>Proposition 6</u>

Any computably hyperarithmetical algebra has an equational monomorphic hierarchical enrichment specification.

<u>Proof</u>

According to p.425 of [Ro 67] for every hyperarithmetical relation g there exists another hyperarithmetical relation h such that the pair $\langle g,h \rangle$ is implicitly definable by an arithmetical relation r, which, due to Proposition 5, has a monomorphic hierarchical specification. Implicitly definable means that there is a first order formula over the natural number involving (apart from logical symbols $\exists, \neg, \wedge$) only the functions 0, successor, addition, multiplication, r, and g,h (as free variables) such that the graph of $\langle g,h \rangle$ is the only solution of this formula. Proposition 5 then implies the existence of the specification we want. ◻

Conversely, hyperarithmeticity is an upper bound for the complexity of hierarchical types:

<u>Proposition 7</u>

Every initial and every terminal hierarchical algebra is hyperarithmetical.

<u>Proof</u>

Let T be a hierarchical type with a nonprimitive sort $\underline{s}$, a primitive sort $\underline{p}$, function symbols $f_1, \dots, f_k$, and axioms $e_1, \dots, e_l$. Consider a two-sorted presentation of the initial algebra of T such that for every tracking function $\bar{f}: N^m \to N$ with range in $\alpha^{-1}(\underline{p})$ and all $\vec{x} \in N^m, y, z \in N$

$\bar{f}(\vec{x}) = y \wedge y \equiv_p z \Rightarrow y \le z$ (where $\equiv_p$ denotes the congruence in the primitive type)

holds. Then the graph g_f of $\bar{f}$ and its complement have explicit definitions of the form

$$(*)\begin{cases} g_f(\vec{x},y) \iff \forall \bar{f}_1 \dots \forall \bar{f}_k \, \forall eq[(\text{"eq is congruence"} \wedge \forall \vec{w}\colon e'_1 \wedge \dots \wedge e'_l) \Rightarrow eq(\bar{f}(\vec{x}),y) \wedge \forall z(y \equiv_p z \Rightarrow y \le z)] \\ \text{where } e'_i \text{ denotes the substitution of = in } e_i \text{ by eq.} \\ \neg g_f(\vec{x},y) \iff \exists z(z \not\equiv_p y \wedge g_f(\vec{x},z)) \end{cases}$$

The usual quantifier negation and quantifier contraction rules (cf. [Ro 67, p. 375]) imply that $\bar{f}$ is hyperarithmetical in $\equiv_p$. The congruence $\equiv_{\alpha,p}$ between terms of primitive sort can similarly be defined (using $eq(\bar{t},\bar{t}')$ instead of $eq(\bar{f}(\vec{x}),y) \wedge \dots$) and is thus hyperarithmetical in $\equiv_p$, too. Two terms of nonprimitive sort are congruent wrt α if their congruence is provable relative to the congruence $\equiv_{\alpha,p}$. Thus this congruence is r.e relative to $\equiv_{\alpha,p}$ and therefore hyperarithmetical in $\equiv_p$, too. Using (*) with $\equiv_\alpha$ every tracking function $\bar{f}$ (with range $\underline{s}$) is hyperarithmetical in $\equiv_\alpha$ and thus in $\equiv_p$. In a hierarchy of types there exists at least one type which is nonhierarchical and therefore its initial model is r.e.. Thus an induction shows the hyperarithmeticity of every initial hierarchical algebra. A similar argument using $\neg g_f(\vec{x},y) \iff \forall \bar{f}_1 \dots \forall \bar{f}_k \forall eq[(\text{"eq is congruence"} \wedge \forall \vec{w}\colon e'_1 \wedge \dots \wedge e'_l \wedge \bar{f}(\vec{x}) = y \wedge \forall z(y \equiv_p z \Rightarrow y \le z)) \Rightarrow \forall x',y'\colon eq(x',y'))]$ works for terminal hierarchical algebras. ◻

Propositions 6 and 7 then give the following general characterization theorem:

<u>Theorem 8</u>

Let A be a hierarchical structure which is computably hyperarithmetical for its primitive signature. Then the following two conditions are equivalent:

1. A is hyperarithmetical.
2. A has a conditional monomorphic hierarchical enrichment specification.

<u>Proof</u>

(2) ⇒ (1): Proposition 7.

(1) ⇒ (2): If A is single-sorted it suffices to apply Proposition 6.

If A is two-sorted we can assume that A has two sorts A' and N. Since A is hyperarithmetical we have the following commuting diagram:

$$\begin{tikzcd} A' \times N \arrow[r, "f"] & A' & + & N \\ N \times N \arrow[u, shift right, "\alpha"] \arrow[u, shift left, "id"'] \arrow[r, "\bar{f}"] & N \arrow[u, "\alpha"] & + & N \arrow[u, "id"'] \end{tikzcd}$$

where f is in the signature of A, α induces a hyperarithmetical congruence $\equiv_\alpha$ on $N \times N$ and $\bar{f}$ is a hyperarithmetical function. Due to Proposition 6 we can specify $\equiv_\alpha$ and $\bar{f}$ by a monomorphic hierarchical enrichment of N where $\equiv_\alpha$

is simulated by EQ: $N \times N \to \{0,1\} \subseteq N$ and $\bar{f}$ is named by $\bar{F}$. Now we can extend this enrichment by a new sort A", two new functions F: $A'' \times N \to A'' + N$ and WIDE: $N \to A''$, and the axioms:

$$EQ(X,Y) = 1 \Leftrightarrow WIDE(X)=WIDE(Y)$$

$$\bar{F}(X)=Y \Leftrightarrow \begin{cases} F(WIDE(X))=WIDE(Y) & \text{if the range of } f \text{ is } A' \\ F(WIDE(X))=Y & \text{otherwise.} \end{cases}$$

This extension is the monomorphic enrichment we want. □

Since practical examples of noncomputable algebras such as programming languages have at least one computable primitive sort like natural numbers or strings the assumption of theorem 8 is not a serious restriction. If one is only interested in implementations one can remove this assumption (by a similar construction as in the previous theorem):

Corollary 9

For a hierarchical structure A the following two statements are equivalent:

1. A is hyperarithmetical
2. A possesses a conditional monomorphic hierarchical implementation specification.

II. PARTIAL ABSTRACT TYPES

4. Partial Algebras

Partial algebras are defined as total algebras except that partial functions instead of total ones may occur. The same holds for partial data structures and hierarchical partial structures. The only difference between computable (arithmetical etc.) partial algebras and computable (arithmetical etc.) total algebras is the possible use of partial tracking functions. The mapping α is total as before. The graph g of a partial function is total:

$$g(x,y) = \begin{cases} \text{true} & \text{if } f(x) = y \\ \text{false} & \text{otherwise.} \end{cases}$$

The representation lemma is exactly as before.
The evaluation function v_A: $W(\Sigma) \to A$ is a partial function. If $v_A(t)$ is defined we write $D(t)^A$ and for $v_A(t)$ we write t^A.

Partial homomorphisms will be total functions which preserve structure (for other notions of homomorphisms see [BW 80,80b]). A total mapping φ: $A \to B$ between two Σ-structures is called partial (Σ-) homomorphism if for every operation $f \in \Sigma$ and every argument $(a_1,\dots,a_k) \in A^k$:

$$D(f(a_1,\dots,a_k))^A \Rightarrow D(f(\varphi(a_1),\dots,\varphi(a_k)))^B \wedge \varphi(f^A(a_1,\dots,a_k)) = f^B(\varphi(a_1),\dots,\varphi(a_k))$$

φ is said to be a strong homomorphism if it is a homomorphism and, in addition, for all operations $f \in \Sigma$ and all arguments $(a_1,\dots,a_k)$

$$D(f(\varphi(a_1),\dots,\varphi(a_k)))^B \Rightarrow D(f(a_1,\dots,a_k))^A.$$

A partial homomorphism φ which is bijective and for which φ^{-1} is a partial homomorphism as well is called an isomorphism. A bijective partial homomorphism does not need to be an isomorphism but if φ is strong then φ is an isomorphism.

5. Equality

The equality symbol in formulas is interpreted in a two-valued classical way (cf. [R 79, ABN 80]). There are at least two possibilities to define a two-valued equality sign for partially defined terms (cf. [BW 80b]). Following here [ABN 80] we choose the "existential equality":

We define for a partial algebra A

$$A \models t=t' \text{ iff } D(t)^A \wedge D(t')^A \wedge t^A=t'^A$$

If one or both terms are undefined, then A does not satisfy t=t'. D(t) can be expressed by t=t and is used as an abbreviation for that.

"=" is not a congruence relation in the usual sense; it is not reflexive and satisfies only a weak substitution property.

"=" is characterized by the symmetry, transitivity and the properties that for all $f \in \Sigma$ and all terms $t_1,\dots,t_x$ of appropriate sorts

$$X_1=Y_1 \wedge \dots \wedge X_k=Y_k \wedge D(f(X_1,\dots,X_k)) \Rightarrow f(X_1,\dots,X_k) = f(Y_1,\dots,Y_k)$$
$$D(f(t_1,\dots,t_k)) \Rightarrow D(t_k) \wedge \dots \wedge D(t_k)$$

i.e. if X=Y and f(X) is defined then f(X)=f(Y) and f(Y) is defined; and if f(t) is defined then t is it, too. Thus the provability relation $\vdash$ is as for total algebras except that we have to take the above axioms for "=".

<u>Partial abstract types</u> are defined analogously to total ones. PALG(Σ,E) denotes the class of all partial Σ-algebras satisfying a set of axioms E. PALG(Σ,E,P) is analogously defined. Clearly the total algebras in PALG(Σ) form an equational subclass PALG(Σ,E) where $E = \{D(f(x_1,\dots,x_k)): f \in \Sigma\}$.

6. <u>Initial and Weakly Terminal Partial Algebras</u>

An <u>initial (hierarchical) partial algebra</u> is an algebra which is initial in PALG(Σ,E) (or PALG(Σ,E,P), resp.) wrt partial homomorphisms.

For every (Σ,E) an initial partial algebra $I(\Sigma,E)$ exists and is unique up to isomorphism (cf. [R 79]). The equality is defined by

$$I(\Sigma,E) \models t=t' \Leftrightarrow E \vdash D(t) \wedge D(t') \wedge t=t'$$

Therefore a term is defined in $I(\Sigma,E)$ iff its definedness is provable. The initial algebras are <u>minimally defined algebras</u>.

Furthermore, the definedness relation and equality relation of $I(\Sigma,E)$ is recursively enumerable: Every initial partial algebra is semicomputable.

For the semantics of programming languages we are particularly interested in the class MDEF(Σ,E,P) of all minimally defined algebras which is specified by

$$A \in \text{MDEF}(\Sigma,E,P) \Leftrightarrow [\ \forall t \in W(\Sigma): A \models D(t) \Leftrightarrow \forall B \in \text{PALG}(\Sigma,E,P): B \models D(t)]$$

Every partial homomorphism between two minimally defined algebras is strong. A partial data structure Z in MDEF$_0(\Sigma,E,P)$ is called <u>weakly terminal</u> if Z is terminal in MDEF$_0(\Sigma,E,P)$ wrt strong homomorphisms [BW 80b]. Unfortunately neither equality nor inequality of weakly terminal algebras need to be r.e.. They are "Δ_2^0" which means that = (and $\neq$) can be specified by a formula of the form $\forall x\, \exists y P$ as well as by $\exists x\, \forall y Q$ where P and Q are recursive relations.

Finally, initial and weakly terminal <u>partial enrichments</u> and <u>implementations</u> differ from total ones only by the use of PALG and MDEF.

7. <u>Characterization Results for Partial Algebras</u>

Let (Σ,E) be a partial abstract type with conditional equations as axioms.

<u>Proposition 10</u>

Every initial algebra of (Σ,E) is semicomputable and every weakly terminal algebra is Δ_2^0.

<u>Proof</u>

We only have to show that the inequality of a weakly terminal algebra Z is Δ_2^0; then the equality is Δ_2^0, too. Now $Z \models t\neq t' \Leftrightarrow (E \nvdash D(t) \vee (E \nvdash D(t')) \vee (E \cup \{t=t'\} \vdash x=y)$

Each of these three ... $\vdash$ ... is r.e. and therefore can be written as $\exists zP$ with P recursive. Thus

$$Z \models t\neq t' \Leftrightarrow \neg\exists zP_1 \vee \neg\exists zP_2 \vee \exists zP_3 \Leftrightarrow (\forall z\forall z': \neg P_1 z \vee \neg P_2 z') \vee \exists zP_3$$
$$\Leftrightarrow \forall zP' \vee \exists z''Q' \text{ (by quantifier contraction)}$$
$$\Leftrightarrow \begin{cases} \forall z\exists z''(P' \vee Q') \text{ as well as} \\ \exists z''\forall z(P' \vee Q') \end{cases}$$

The nondeterministic language in III shows that Δ_2^0 formulas are really needed to describe weakly terminal algebras.

For computable partial algebras we obtain the following specification theorem.

<u>Theorem 11</u>

Let A be an (infinite) computable partial Σ-algebra. Then (1) has an equational initial partial enrichment specification (Σ_0,E_0) involving at most $11+|\Sigma|$ equations and at most 5 auxiliary (hidden) functions. (2) A has an equational weakly terminal partial enrichment specification involving two more functions and five more axioms.

<u>Proof</u>

The first step of the proof is to find a structure A_0, with domain N, and signature $\Sigma_0 \supseteq \Sigma$ such that $A_{0|\Sigma} \cong \langle A_0\rangle_\Sigma \cong A$ which contains besides the functions named in Σ the following functions, named in $\Sigma_0 - \Sigma$:

x+1 named S, x+y named ADD, x×y named MULT and two functions g and h, named G and H, and defined respectively by:

$g(x,y) = P_x(y)$, where (P_i), $i \in N$, is an effective enumeration of the primitive recursive functions.
$h(x,y,z) = z$, if $x=y$, and $h(x,y,z) = 0$, otherwise. Finally Σ_0 contains a constant $\underline{0}$ for 0.

Then we list all equations E_0 that provide an initial algebra specification (Σ_0,E_0) of A_0, thus obtaining a hidden function specification of A. Simultaneously we present some comment that should replace a formal proof of $W_I(\Sigma_0,E_0) \cong A_0$. Such a proof could be given using traversals like in Bergstra and Tucker [BT 80b].

(1) $\underline{0} = \underline{0}$ this equations ensures $D(\underline{0})$.
(2) $S(X) = S(X)$ this ensures $D(S^n(\underline{0}))$ for $n \in N$.
(3) - (7): $ADD(X,0) = X$; $ADD(X,S(Y)) = S(ADD(X,Y))$.
$MULT(X,\underline{0}) = \underline{0}$; $MULT(X,S(Y)) = ADD(X,MULT(X,Y))$
(8) $H(X,Y,Z) = H(Y,X,Z)$
(9) $H(X,X,Z) = Z$
(10) $H(X,ADD(X,S(Y)),Z) = \underline{0}$

Equations (3) - (10) ensure that ADD, MULT and H work correctly on all arguments of the form $S^n(\underline{0})$.

For G(X,Y), choose, using the diophantine theorem of Y. Matijasevic (cf. [Ma 77]) two polynomials, in 0,1,+,•, and indeterminates $X,Y,Z,Z_1,\dots,Z_t$, $\underline{p}$ and $\underline{q}$ such that for all a,b,c

$$g(a,b) = c \iff \exists d_1\dots d_t \in N:\ p(a,b,c,d_1,\dots,d_t) = q(a,b,c,d_1,\dots,d_t)$$

Let P,Q be the formal versions of these polynomials over Σ_0. Then equation (11) is as follows:

$$H(P(X,Y,Z,Z_1,\dots,Z_t),Q(X,Y,Z,Z_1,\dots,Z_t),Z) = H(P(X,Y,Z,Z_1,\dots,Z_t),Q(X,Y,Z,Z_1,\dots,Z_t),G(X,Y)).$$

This equation is valid in A_0. Now suppose $g(a,b)=c$, then for some $d_1,\dots,d_t$ (1)....(10) imply $P(S^a(\underline{0}),S^b(\underline{0}),S^c(\underline{0}),S^{d_1}(\underline{0}),\dots,S^{d_t}(\underline{0})) = Q(S^a(\underline{0}),S^b(\underline{0}),S^c(\underline{0}),S^{d_1}(\underline{0}),\dots S^{d_t}(\underline{0}))$.

Thus, one finds from (11) (cf. a similar argument in [BT 80b]): $S^c(\underline{0}) = G(S^a(\underline{0}),S^b(\underline{0}))$.

Finally, for each $f \in \Sigma$ let W_f be $\{(a_1,\dots,a_{k+1}) \in N^{k+1}:\ A_0 \models f(S^{a_1}(\underline{0}),\dots,S^{a_k}(\underline{0})) = S^{a_{k+1}}(\underline{0})\}$

If $W_f = \emptyset$ then no equation for f is added. If it is nonempty then, because W_f is recursively enumerable there are primitive recursive functions $g_1,\dots,g_k$, g_{k+1} such that $W_f = \{(g_1(b),\dots,g_{k+1}(b)): b \in N\}$. Now choose numbers $n_1..n_{k+1}$ such that for all m, $i \leq k$ $g(n_i,m) = g_i(m)$ and define equation (12^f) for f as follows:

(12) $f(G(S^{n_1}(\underline{0}),X),\dots,G(S^{n_k}(\underline{0}),X)) = G(S^{n_{k+1}}(\underline{0}),X)$

The same construction as in the proof of Proposition 2, (1) ⇒ (2), gives an equational weakly terminal specification with two further hidden functions and five other equations. □

According to Kaphengst's result (announced in [H 80]) semicomputable partial structures can be exactly characterized by implementations:

A partial structure A is semicomputable iff A possesses an initial partial implementation specification.

As in the case of total algebras a hierarchical partial specification (Σ,E,P) is nothing other than a partial specification if (Σ,E,P) is <u>partially sufficiently complete</u>, i.e. for every term $t \in W(\Sigma)$ of prim. sort <u>such that D(t) is provable</u> there exists a primitive term p with $E \vdash t=p$ ([BW 80,80b]). For partial types not being partially sufficiently complete we can go up in the (hyper)arithmetical hierarchy as before. We only note that every hierarchical initial partial algebra is π^1_1 and that every weakly terminal algebra is Δ^1_2. Since the graph of a partial function is total, Proposition 6 of I.3 applies directly to partial types.

<u>Corollary 12</u>

Any computably hyperarithmetical partial algebra has an equational hierarchical partial enrichment specification which is both initial and weakly terminal.

III. <u>A NON-DETERMINISTIC PROGRAMMING LANGUAGE</u>

Finally we give an example of the specification of a Dijkstra-style nondeterministic programming language (cf. [Di 76]). As semantics we choose the so-called "angelic nondeterminism" as it is used e.g. for nondeterministic automata (for a comparison of different semantics of nondeterminism cf. [HP 79, BW 81]). We define a hierarchical partial abstract type comprising the following primitive sorts:

$\underline{\text{dom}}$, the sort of semantic objects including the truth values tt and ff

$\underline{\text{id}}$, the sort of identifiers for programming variables

$\underline{\text{state}}$, the sort of states. By $\sigma[d/v]$ we denote the substitution of $d \in \underline{\text{dom}}$ in state σ for the value of the variable v.

$\underline{\text{exp}}$, the sort of expressions built by 0,S,ADD and identifiers of $\underline{\text{id}}$ together with a total evaluation function eval: $\underline{\text{exp}} \times \underline{\text{state}} \to \underline{\text{dom}}$

$\underline{\text{bexp}}$, the sort of boolean expression with evaluation function beval.

The described type is computable. Hence we can assume a computable monomorphic total enrichment specification for it.

As only nonprimitive sort we introduce the sort $\underline{\text{nd}}$ together with the "constructor functions":

NOP: $\to \underline{\text{nd}}$ ASSIGN: $\underline{\text{id}} \times \underline{\text{exp}} \to \underline{\text{nd}}$ written as ":="

SEMI: $\underline{\text{nd}} \times \underline{\text{nd}} \to \underline{\text{nd}}$ written as ";"   DO: $(\underline{\text{bexp}} \times \underline{\text{nd}})^2 \to \underline{\text{nd}}$ written as "$\underline{\text{DO}}\ldots \,\square\, \ldots\underline{\text{OD}}$"

As "semantic functions" we use stops: $\underline{\text{nd}} \times \underline{\text{state}} \to \{tt,ff\}$ and exec: $\underline{\text{nd}} \times \underline{\text{state}}^2 \to \{tt,ff\}$ with the intended meaning

stops(S,σ) = tt iff at least one execution of S starting in state σ leads to a result

exec(S,σ,σ') = tt iff there exists a computation step sequence of S beginning in state σ and ending in state σ'.

Hence in "angelic nondeterminism" the execution of a statement S in state σ does not terminate only if exec(S,σ,σ') is false for all σ'. This makes the angelic nondeterminism noncontinuous wrt the Egli-Milner-ordering which is used to describe least fixed points of nondeterministic programs. We specify this variant of nondeterminism as follows: Let DO denote $\underline{\text{DO}}\ C \to S \,\square\, C' \to S'\ \underline{\text{OD}}$

(1) $\underline{\text{DO}}\ C \to S \,\square\, C' \to S'\ \underline{\text{OD}} = \underline{\text{DO}}\ C' \to S' \,\square\, C \to S\ \underline{\text{OD}} \wedge D(S;S')$

(2) $\text{exec}(\text{NOP},\sigma,\sigma) = tt$

(3) $\text{eval}(E,\sigma) = d \Rightarrow \text{exec}(V:=E,\sigma,\sigma[d/V]) = tt$

(4) $\text{beval}(C,\sigma) = tt \wedge \text{exec}(S;\text{DO},\sigma,\sigma') = tt \Rightarrow \text{exec}(\text{DO},\sigma,\sigma') = tt$

(5) $\text{beval}(C,\sigma) = ff \wedge \text{beval}(C',\sigma) = ff \Rightarrow \text{exec}(\text{DO},\sigma,\sigma) = tt$

(6) $\text{exec}(S,\sigma,\sigma') = tt \wedge \text{exec}(S',\sigma',\sigma'') = tt \Rightarrow \text{exec}(S;S',\sigma,\sigma'') = tt$

(7) $\text{exec}(S,\sigma,\sigma') = tt \Rightarrow \text{stops}(S,\sigma) = tt$

Let us call this specification AN. According to (1) - (3) every nondeterministic statement is (syntactically) defined in every model whereas stops and exec may be partial functions. AN is partially sufficiently complete. It has an initial partial algebra I the equality of which in $\underline{\text{nd}}$ is determined by $I \models S=S' \iff (1) \vdash S=S'$. AN has a weakly terminal algebra Z describing exactly the mathematical semantics of statements:

$Z \models S=S'$ iff for all states σ and σ':

$$(\text{AN} \vdash \text{stops}(S,\sigma) = tt \iff \text{AN} \vdash \text{stops}(S',\sigma) = tt) \text{ and } (\text{AN} \vdash \text{exec}(S,\sigma,\sigma') = tt \iff \text{AN} \vdash \text{exec}(S',\sigma,\sigma') = tt)$$

Now, the relation "AN $\vdash$ ..." is r.e; thus, there exist recursive relations T_1 and T_2 such that

$$(*)\quad \begin{aligned} Z \models S=S' \text{ iff } \forall\sigma,\sigma' [(\exists z: T_1(z,\ulcorner\text{stops}\ldots\urcorner,S,\sigma) &\iff \exists z': T_1(z',\ulcorner\text{stops}\ldots\urcorner,S,\sigma)) \\ \wedge (\exists z: T_2(z,\ulcorner\text{exec}\ldots\urcorner,S,\sigma,\sigma') &\iff \exists z': T_2(z',\ulcorner\text{exec}\ldots\urcorner,S,\sigma,\sigma'))] \end{aligned}$$

By the methods of theorem 8 we codify "$\equiv_Z$" into $\underline{\text{dom}}$ and introduce the function WIDE: $\underline{\text{dom}} \to \underline{\text{nd}}$ and operations $\overline{\text{NOP}}$, $\overline{\text{ASSIGN}}$ $\overline{\text{SEMI}}$, and $\overline{\text{DO}}$ which simulate the nondeterministic statements on $\underline{\text{dom}}$. Finally the partial function stops is specified (using their graph over the functions $\overline{\text{NOP}}$, $\overline{\text{ASSIGN}}$,...:) such that $\overline{\text{stops}}(S,\sigma) = tt$, if $Z \models \text{stops}(S,\sigma) = tt$, and $\overline{\text{stops}}(S,\sigma) = ff$, otherwise. We proceed analogously for exec. The axioms mentioned at the end of the proof of theorem 8 conclude our specification by a hierarchical monomorphic total enrichment. (*) is not the best possible description of the equality in Z. Indeed, it is co-r.e since according to Proposition 10 $Z \models S \neq S' \iff E \nvdash D(S) \vee E \nvdash D(S') \vee (E \cup \{S=S'\} \vdash x=y \iff E \cup \{S=S'\} \vdash x=y$ (because of $E \vdash D(S) \wedge E \vdash D(S')$). However, if we change our type slightly by replacing axiom (1) by $\text{beval}(C',\sigma) = tt \wedge \text{exec}(S';\text{DO},\sigma,\sigma') \Rightarrow \text{exec}(\text{DO},\sigma,\sigma')$ then nondeterministic statements without any terminating execution sequence are undefined and the equality in the new weakly terminal model is exactly Δ_2^0.

ACKNOWLEDGEMENT Thanks go to Eleanor Kerse for her excellent typing, for saving the paper from mail damage, and her patience in waiting for the manuscripts of this paper.

REFERENCES

[ADJ 78] J.A. Goguen, J.W. Thatcher, E.G. Wagner: An initial algebra approach to the specification, correctness and implementation of abstract data types. In: Current Trends in Programming Methodology IV. Prentice Hall /1978/, 80-144.

[ANS 80] H. Andreka, B. Burmeister, I. Nemeti: Quasivarieties of partial algebras - a unifying approach towards a two-valued model theory for partial algebras. TH Darmstadt, FB Math. Preprint Nr. 557.

[AK 80] A. Arnold, M. Karpinski: An easy improvement of Bergstra-Meyer's 81 equations bound for data specifications. EATCS Bull. 12, /1980/.

[BaW 81] F.L. Bauer, H. Wössner: Algorithmic language and program development. Berlin: Springer /1981/, to appear.

[BM 80] J. Bergstra, J. Meyer: On bounds for the specification of finite monoids by means of equations using only unary hidden functions. Univ. of Leiden, Rep. 80-11, /1980/.

[BT 79] J. Bergstra, J. Tucker: Algebraic specifications of computable and semicomputable data structures. Math. Centre Amsterdam Rep. IW 115,/1979/.

[BT 80] J. Bergstra, J. Tucker: Initial and final algebra semantics for data type specifications: Two characterization theorems. Math. Centre Amsterdam Rep. IW 142, /1980/.

[BT 80a] J. Bergstra, J. Tucker: On bounds for the specification of finite data types by means of equations and conditional equations, Math. Centre Amsterdam Rep. IW 131, /1980/.

[BT 80b] J. Bergstra, J. Tucker: Equational specifications for computable data types: six hidden functions suffice and other sufficiency bounds, Math. Centre Amsterdam Rep. IW 128, /1980/.

[BMM 79] A. Bertoni, G. Mauri, P. Miglioli: A characterization of abstract data types as model-theoretic invariants, 6th ICALP, LNCS 71, 26-37, /1979/.

[BMMW 79] A. Bertoni, G. Mauri, P. Miglioli, M. Wirsing: On different approaches to abstract data types and the existence of recursive models. EATCS Bull. 9, 47-57, /1979/.

[BDPPW 79] M. Broy, W. Dosch, H. Partsch, P. Pepper, M. Wirsing: Existential quantifiers in abstract data types. 6th ICALP, LNCS 71, 73-87, /1979/.

[BW 80b] M. Broy, M. Wirsing: Initial versus terminal algebra semantics for partially defined abstract types. TU München, TUM-I8018, /1980/.

[BW 81] M. Broy, M. Wirsing: On the algebraic specification of nondeterministic programming languages. 6th CAAP, Genova, /1981/.

[BG 77] R.M. Burstall, J.A. Goguen: Putting theories together to make specifications. Proc. Int. Conf. A.I.,/1977/.

[BG 80] R.M. Burstall, J.A. Goguen: The semantics of CLEAR: a specification language. Proc. C'hagen Winter Schl. on Abstract Software Specifications, /1980/.

[CIP 81] Report on the wide spectrum language CIP-L, TU München, to appear.

[DI 76] E.W. Dijkstra: A discipline of programming. Prentice Hall, Englewood Cliffs, /1976/.

[EL 80] H.D. Ehrich, U. Lipeck: Proving implementations correct - two alternative approaches. IFIP Congress 80.

[EKTWW 80] H. Ehrig, H.J. Kreowski, J.W. Thatcher, E.G. Wagner, J.B. Wright: Parameterized data types in algebraic specification languages. 7th ICALP, LNCS 85, 157-168, /1980/.

[Ga 80] M.C. Gaudel: Generation et preuve de compilateurs basées sur une semantique formelle des langages de programmation. These d'Etat, Nancy, /1980/.

[GGM 76] V. Giarratana, F. Gimona, U. Montanari: Observability concepts in abstract data type specifications. 5th MFCS, LNCS 45, 576-587, /1976/.

[G 75] J.V. Guttag: The specification and application to programming of abstract data types. Ph.D. thesis, Univ. of Toronto, /1975/.

[HP 79] M.C. Hennessy, G.D. Plotkin: Full abstraction for a simple parallel programming language. MFCS 79,Olomouc.

[HR 80] G. Hornung, P. Raulefs: Terminal algebra semantics and retractions for abstract data types. 7th ICALP, LNCS 85, 310-323, /1980/.

[H 80] U. Hupbach: Abstract implementation of abstract data type. 9th MFCS, Rydzyna, LNCS 88, 291-304, /1980/.

[Ka 80] S. Kamin: Final data type specifications: a new data type specification method. 7th POPL, Las Vegas,/1979/.

[HKP 81] H. Klaeren, H. Petzsch: The development of an interpreter by means of algebraic software specifications, FPC 81, Peniscola, LNCS 107, 335-346.

[Lo 80] J. Loeckx: Algorithmic specifications of abstract data types. Univ. Saarbrücken, Rep. A 80/12/

[Mj 79] M. Majster: Data types, abstract data types and their specification problems. TCS 8, 89-127, /1979/.

[MS 76] Z. Manna, A. Shamir: The theoretical aspects of the optimal fixed point. Siam J. of Comp. 5, /1976/.

[Mn 77] Y. Manin: A course in mathematical logic, Springer-Verlag, New York, /1977/.

]Pa 60] C. Pair: Types abstraits et semantique algébrique des langages de programmation. Centre Recherche Informatique de Nancy, 80-R-011, /1980/.

[R 79] H. Reichel: Theorie der Aequoide, Dissertation B. Humboldt-Universität Berlin, /1979/.

[R 80] H. Reichel: Initially restricting algebraic theories. 9th MFCS, Rydzyna, LNCS 88, 504-514.

[R 67] H. Rogers Jr.: Theory of recursive functions and effective computability. McGraw-Hill: New York, /1967/.

[Sh 67] J. Shoenfield: Mathematical logic. Addison-Wesley, Reading, /1967/.

[VP 79] P.A. Veloso, T.H. Pequeno: Don't write more axioms than you have to. Univ. Rio de Janeiro, Monografias en Ciencia do Computacao No. 10, /1979/.

[Wa 77] M. Wand: Final algebra semantics and data type extensions. Indiana University TR65, /1978/.

[WB 80] M. Wirsing, M. Broy: Abstract data types as lattices of finitely generated models. 9th MFCS, LNCS 88, 673-685, /1980/.