FINAL ALGEBRA SEMANTICS AND DATA TYPE EXTENSIONS

Mitchell Wand
Computer Science Department
Indiana University
Bloomington, Indiana 47401

TECHNICAL REPORT NO. 65
FINAL ALGEBRA SEMANTICS AND
DATA TYPE EXTENSIONS

MITCHELL WAND
JULY, 1977
REVISED: JULY, 1978

to appear:
J.C.S.S.

Research reported herein was supported in part by the National Science Foundation under grant number MCS75-06678 A01.

Proposed running head - FINAL ALGEBRA SEMANTICS

Mail proofs to: Prof. Mitchell Wand
Computer Science Department
Lindley Hall 101
Indiana University
Bloomington, Indiana 47401

Abstract

We consider the problem of data type extensions. Guttag, Horowitz, and Musser have pointed out that in this situation the naive initial algebra approach requires the data type to save too much information. We formulate a category of implementations of such an extension, and we show that such a category has a final object. The resulting semantics is closer to that of Hoare, since it can be argued that an abstract data type in the sense of Hoare is a final object in the category of representations of that type. We consider as an example the specification of integer arrays, and we show that our specification yields arrays as its abstract data type. The connection with initial algebra semantics is discussed.

0. Prologue

In this paper we are concerned with the definition of new data types from old, using the viewpoint of what is called initial algebra semantics [9,10,13,14]. Before discussing the problem in detail, we summarize our interpretation of the initial algebra approach in this prologue.(*)

One wishes to specify data types axiomatically, that is, by writing down, in some logical calculus, sentences which describe those properties of the data type on which its user may rely. A program which uses a data type may then be proved correct by deducing its verification conditions from the axioms of the data type. Such a program will then work correctly with any implementation of the data type which satisfies the axioms. Thus the programmer is concerned not with single algebras, but with the class of algebras which are legal representations of the data type; the programs he writes ought to work satisfactorily regardless of which representation is used. Our first thesis, therefore, is that *a specification of a data type should present a class of algebras*. If one desired merely to construct a single algebra, then numerous mathematical techniques are available; it is the finite presentation of classes of algebras that requires formal methods.

One logical language which seems to be useful for the specification of data types is the language of generators and relations [9,10,13,16]. A presentation via generators and rela-

(*)The reader should be warned that we diverge in some details from the approach, say, of [10]. Our outlook is much closer to that of [9]; any misinterpretations are solely our responsibility, of course.

tions defines an equational class of algebras. Since one wishes to discuss connections between equational classes independent of their presentation, one introduces categories called algebraic theories [20]. An algebraic theory is a representation of its equational class, just as a Zermelo-Fraenkel ordinal is a representative of its order-isomorphism class.(*) An algebraic theory consists of equivalence classes of terms (compositions of generators), where two terms are equivalent iff their equality is deducible from the relations.(**) Of particular importance is the case where there are no relations between the generators; then the theory is called a free theory and the morphisms are just the terms. The T-algebras (or implementations, or models) of an algebraic theory T are certain functors from T to the category of sets; this picture is merely a notational variant of the conventional picture of an algebra.

The denotational semantics of a term in a T-algebra is the mapping it induces on the universe set of the algebra. This mapping is obtained by mapping the term (a morphism of a free theory F) to its equivalence class (a morphism of T), and thence, via the T-algebra (qua functor) to the desired set map (a morphism of the category of sets). We identify T-algebras with implementations, and since the functor from the free theory to T is independent of the implementation, we sometimes refer to it as

(*)Although, unlike the ordinal, an algebraic theory is not a member of the class it represents.

(**)More precisely, the morphisms of a theory are equivalence classes of (tuples of) terms.

"the semantics". These relationships are shown in Figure 0.1[*]

$$\begin{tikzcd} F \arrow[r, "semantics"] & T \arrow[d, "implementation"] \\ & Sets \end{tikzcd}$$

Figure 0.1

The fragment of algebra semantics we have described is more than an algebraicization of attribute grammars [19] with only synthesized attributes.[**] The difference is that the algebraic framework allows additional problems to be attacked:

(i) <u>equivalence</u> <u>of</u> <u>presentations</u>: given two sets of generators and relations, do they define the same class of implementations? If the generator sets are quite different, it may be difficult to state a translation theorem; it may be easier to prove the algebraic theories isomorphic. Similar questions arise with respect to simulability [8] or program trasformations [3,23].

(ii) <u>operational</u> <u>semantics</u>: given some complex term and some set of terms which we regard as known constants, a computation is a deduction (in some appropriate formal system) that the value of the complex term is always equal to the value of a particular known term in any implementation, i.e. that they are mapped by

---

(*)This is, of course, a crude notion of implementation - see (iii) below.

(**)Although synthesized attributes are enough, see, for example [4].

the semantics to the same morphism in T . It can be shown that this problem is equivalent to a word problem in a tree rewriting system [22], and then under reasonable conditions the tree re-writing system has the Church-Rosser property with various pleasant consequences.(*)

(iii) classes of implementations: An equational class is usually not quite what one wants for the class of implementations. One may desire additional closure properties (which leads to the consideration of "theories with additional structure" [5,24]) or more restricted closure properties (a situation to be considered in this paper). Rather than having a single, so-called "abstract" implementation, one always has a class of implementations, and one may pose the question of which of those implementations is "the" desired one. The conventional choice is the initial T-algebra, which has two desirable properties. First, its universe contains no values other than those required by the generators. Second, two values have the same semantics in the initial T-algebra if and only if they have the same semantics in every T-algebra. Thus no information is lost except that which is required by the relations.

Guttag et. al. [17] have suggested that in some cases the initial algebra saves too much information. It is the purpose of this paper to suggest a solution to that problem. The out-line of the paper is as follows: Section 1 presents an example

(*)See [17] for a well-illustrated discussion. Although tree rewriting systems have been the object of some study [22], their exact connection with algebraic theories has not to our knowledge been adequately explored in print.

to illustrate the problem posed by Guttag et. al. Section 2 is given over to definitions, most of which are quite standard. In Section 3, we argue that an abstract data type in the sense of [18] ought to be a *final* object in the category of data type representations. In Section 4, we present our model of data type extensions. In Section 5, we prove the main result: that the category of representations of a data type extension has a final object, which gives the final algebra semantics of the title. It is also shown that the conventional initial algebra semantics is preserved as a special case. We mention briefly an analogy between final algebra semantics and minimal realization in automata.

1. Introduction

In this section we presume a general familiarity with the mathematical structures discussed in the prologue. There are several excellent tutorials on various aspects of this material [11,12,17].

Let us consider a theory of intergers, $T_{I,+}$. This theory will have one sort, denoted i, and generators as follows:

for each nonnegative integer k, a symbol $\underline{n}_k : \Lambda \to i$

$\underline{\text{undefined}}$: $\Lambda \to i$

$\underline{\text{plus}}$: $ii \to i$

subject to the identities

$\underline{\text{plus}}[\underline{n}_k, \underline{n}_p] = \underline{n}_{k+p}$ for each $k,p \epsilon \omega$ (*)

$\underline{\text{plus}}[x, \underline{\text{undefined}}] = \underline{\text{undefined}}$

$\underline{\text{plus}}[\underline{\text{undefined}}, x] = \underline{\text{undefined}}$

It is easy to show, using canonical term algebras [10], that the initial algebra of $T_{I,+}$ consists of $\{\underline{n}_k | k \epsilon \omega\} \cup \{\underline{\text{undefined}}\}$ under the usual addition. Now let us consider the theory of a data type which consists of integer-valued arrays indexed by integers (where "integers" is the type defined by $T_{I,+}$). To do this, we add a new sort (for arrays), called a, and new generators:

$\underline{\text{empty}}: \Lambda \to a$	(the empty array)
$\underline{\text{alt}}: aii \to a$	($\underline{\text{alt}}[A,j,x]$ = "A after A[j]:=x")
$\underline{\text{val}}: ai \to i$	($\underline{\text{val}}[A,j] = A[j]$)

(*)This represents a countable set of axioms.

The intended semantics of the generators (sketched above) may be captured by adding the following identities:

$$\underline{\text{val}}[\underline{\text{empty}},x] = \underline{\text{undefined}}$$
$$\underline{\text{val}}[\underline{\text{alt}}[x,\underline{n}_p,z],\underline{n}_p] = z \text{ for } p\epsilon\omega$$
$$\underline{\text{val}}[\underline{\text{alt}}[x,\underline{n}_k,z],\underline{n}_p] = \underline{\text{val}}[x,\underline{n}_p] \text{ for } k\neq p, k,p\epsilon\omega$$
$$\underline{\text{val}}[x,\underline{\text{undefined}}] = \underline{\text{undefined}}$$

We call this theory T_{ARR}.

It is clear that val, applied to an array and an integer, always reduces to an integer by applications of the identities, i.e. this set of identities is sufficiently-complete in the sense of Guttag [16]. The initial algebra of T_{ARR}, however, does not consist of the arrays we hoped to define. The initial algebra consists of two sorts. The sort S_i corresponding to i consists of $\{\underline{n}_k | k\epsilon\omega\}\cup\{\underline{\text{undefined}}\}$ as before, but the set S_a corresponding to a is defined inductively as

(i) $\underline{\text{empty}} \; \epsilon S_a$

(ii) if $x\epsilon S_a$, and $m,m'\epsilon S_i$, then $\underline{\text{alt}}[x,m,m']$ is in the set S_a

(iii) nothing else

For example $\underline{\text{alt}}[\underline{\text{alt}}[\underline{\text{empty}},\underline{n}_1,\underline{n}_1],\underline{n}_2,\underline{n}_2]$

and $\underline{\text{alt}}[\underline{\text{alt}}[\underline{\text{empty}},\underline{n}_2,\underline{n}_2],\underline{n}_1,\underline{n}_1]$

are distinct elements of the initial algebra of T_{ARR}. Here the initial algebra saves too much information: it saves not only the values in the array but also the order of all changes in the array.

One could destroy this unneeded information by adding the identities

$$\underline{alt}[\underline{alt}[x,\underline{n}_k,y],\underline{n}_k,z] = \underline{alt}[x,\underline{n}_k,z] \qquad k\in\omega$$

$$\underline{alt}[\underline{alt}[x,\underline{n}_k,y],\underline{n}_p,z] = \underline{alt}[\underline{alt}[x,\underline{n}_p,z],\underline{n}_k,y] \quad k,p\in\omega, k\neq p$$

$$\underline{alt}[x,\underline{undefined},y] = x$$

It is straightforward to see that this suppresses duplicate subscript entries and causes subscript errors on updates to be ignored. Unfortunately, adding the second axiom scheme causes the underlying operational semantics to lose the Church-Rosser property [17,22]. This is an unpleasant consequence; Guttag et.al. suggest the use of "equality interpretations" to allow information to be lost in a controlled manner.

It is the purpose of this paper to suggest another solution. We observe that the difficulty arises when we are dealing with data type <u>extensions</u>. We have "enough information" in our implementation of the extension so long as no values of the base type (e.g. integers) are merged. We wish to lose as much information as possible; therefore we are led to <u>final</u> algebras in the category of implementations which have "enough information". The main theorem of this paper shows that such final algebras exist.

2. Preliminaries

If C is a category, $C(a,b)$ denotes the set of arrows or morphisms from object a to object b. If $f \in C(a,b)$ and $g \in C(b,c)$, their composition, a member of $C(a,c)$, is denoted $g.f$. We write gf when no confusion results. If $f \in C(a,b)$ then $\mathrm{dom}(f) = a$ and $\mathrm{cod}(f) = b$. Sets will denote the category whose objects are sets and whose morphisms are the usual set-theoretic functions. Right-to-left composition (usually of functors or of functions in Sets) is written using "$\circ$": $g \circ f(x) = g(f(x))$.

If C is a category, an object a of C is initial iff for any object b of C, there is exactly one morphism in C from a to b. The object a is final iff for any object b there is exactly one morphism in C from b to a. All initial objects in a category are always isomorphic; similarly for final objects. In Sets, $\emptyset$ is initial (consider the function whose graph is empty), and any singleton set is final.

Let S be a set whose elements are called sorts. An S-sorted operator alphabet Ω is a map $\Omega : K \to S^* \times S$ for some set K[(*)]. If $s \in K$, and $\Omega s = (w,a)$, we say w is the domain of s and a is the codomain of s. If S has only one element, and $w = a^n$ (where $S = \{a\}$), we say s is n-ary; Ω is then a ranked alphabet. When no ambiguity results, we will write Ω for K and write "$s \in \Omega$". We write $\Omega(w,a)$ for $\{s \in K \mid \Omega s = (w,a)\}$.

(*)S^* denotes the free monoid generated by S.

An S-sorted algebraic theory (or just <u>theory</u>) is a category T whose objects are the elements of S^* and in which multiplication in S^* coincides with the categorical product. If T is a theory, and $f_i \epsilon T(u,w_i)$ (for $i = 1,..,n$), then the product morphism in $T(u,w_1 \ldots w_n)$ is denoted $[f_1,..,f_n]$. We write e_i for the projection morphisms. A theory-functor is a product-preserving functor between theories. If Ω is an S-sorted operator alphabet, we may construct the free theory F_Ω by the usual methods [12]; if $s \epsilon \Omega$, then $s \epsilon F_\Omega(\mathrm{dom}(s),\mathrm{cod}(s))$.

If T is an S-sorted theory, so is T^2, where $T^2(u,v) = \{(f,g) \mid f,g \epsilon T(u,v)\}$ with composition given by $(f,g)(f',g') = (ff',gg')$. An <u>equation</u> on T is an element of $T^2(w,a)$ for some $a \epsilon S$. A <u>congruence</u> on T is a subtheory R of T^2 such that for each $u,v \epsilon S^*$, $R(u,v)$ is an equivalence relation on $T(u,v)$. If R is a congruence on T, then we can form the quotient theory T/R via $T/R\,(u,v) = T(u,v)/R(u,v)$. T/R is also an S-sorted theory; it is the coequalizer of the evident diagram $R \to T^2 \rightrightarrows T$.

If Δ is a set of equations on T, we can construct the smallest congruence on T containing Δ as the set of theorems of a formal system E_Δ. The formal objects of E_Δ are the morphisms of T^2. We write $(f,f'):u \to v$ for $(f,f') \epsilon T^2(u,v)$, and $\vdash (f,f'):u \to v$ if (f,f') is provable in E_Δ. The axioms and rules of E_Δ are as follows:

Axioms: If $(f,f'):w \to a \in \Delta$, then $\vdash (f,f'):w \to a$ $\quad E\Delta$

For any $f \in T(u,v)$, $\vdash (f,f):u \to v$ $\quad ER$

Rules: $\dfrac{(f,g):u \to v}{(g,f):u \to v}$ ES $\qquad \dfrac{(f,g):u \to v \quad (g,h):u \to v}{(f,h):u \to v}$ ET

$$\frac{(g,g):w \to y \quad (f,f'):v \to w \quad (h,h):u \to v}{(g.f.h,g.f'.h):u \to y} \text{ EC}$$

$$\frac{(f_1,f'_1):m \to a_1,\ldots,(f_n,f'_n):m \to a_n}{([f_1,.,f_n],[f'_1,..f'_n]):m \to a_1..a_n} \text{ EP}$$

Let $E_\Delta(u,v) = \{(f,f')|\vdash(f,f'):u \to v\}$. Axiom scheme $E\Delta$ ensures that every equation in Δ is in E_Δ ; rules ER, ES, and ET ensure that each $E_\Delta(u,v)$ is an equivalence relation; rule EC closes E_Δ to a subcategory of T^2 , and rule EP closes E_Δ under the product operation of T^2 . Hence E_Δ , with composition inherited from T^2 , is the smallest congruence on T containing Δ .

A theory may be presented by (Ω,Δ) where Ω is an operator alphabet (the <u>generators</u>) and Δ is a set of equations (the <u>relations</u>). (Ω,Δ) presents the theory T where $T(u,v) = F_\Omega(u,v)/E_\Delta(u,v)$. The functor $F:F_\Omega \to T$ sending each morphism to its equivalence class is a full theory functor.

If T is an S-sorted theory, a T-algebra is a product-preserving functor $A:T \to$ <u>Sets</u> . A natural transformation $h:A \to B$ from one T-algebra to another is just a homomorphism of algebras (over Ω). The T-algebras and natural transformations form a category <u>T-Alg</u>.

If T is an S-sorted theory, the T-algebra A given by

$$A(w) = T(\Lambda,w) \qquad w \in S^*$$
$$A(f):T(\Lambda,s) \to T(\Lambda,v):g \mapsto gf \qquad f \in T(w,v)$$

is initial in T-Alg. This (when decoded) comes out to be the conventional term algebra in the case where T is a free theory; where T is not free, the carriers consist of equivalence classes (under E_Δ) of (tuples of) terms. We refer to this particular initial algebra as the canonical initial algebra. The T-algebra Z given by $Z(w) = \{1\}$ is final in T-alg. Z is the algebra whose universes consist of singleton sets for each sort (and whose operations are therefore trivial).

3. Data Type Representations

One anomalous property of the initial algebra approach is a seeming incompatibility with other notions of abstract data types e.g. [18]. If A is an initial algebra of T , and B is any other T-algebra, there is a unique morphism $A \to B$. In Hoare's version (and in related work [e.g.21]), the map runs the other way: one has the "abstraction map" A from an arbitrary implementation B to the set of "abstract values". In this section we will attempt to make some sense of these two views.

We said previously that we identify objects of the category T-Alg with implementations of the theory T . This identification is, of course, rough at best; for example, it includes the final algebra Z as a legal implementation. Even if we wish to exclude some elements of T-Alg, the class of legal implementations of T will be some subcategory K of T-Alg.

Let us imagine, therefore, that we are given a particular subcategory K of T-Alg which is known to be the category of legal implementations of T ; and let W be the "abstract data type".

In the example of Section 1, W would be given by

$W(\mathit{i}) = \{\underline{n}_k | k \epsilon \omega\} \cup \{\underline{\text{undefined}}\}$ (as before)

$W(a) = \{M | M$ is a partial function $\omega \to \omega$, of finite domain

$W(\underline{\text{val}}) = \lambda(M,j)[$if $j = \underline{\text{undefined}}$ then $\underline{\text{undefined}}$
else if M is undefined at j then $\underline{\text{undefined}}$
else $\underline{n}_{M(j)}]$

$W(\underline{\text{alt}}) = \lambda(M,j,x)[M - \{(j,y) \epsilon M\} \cup \{(j,x)\}]$

K is to the category of representations of W .

If we have a reasonable notion of "category of legal implementations", the following observations should hold:

(1) W is an object of K (A data type ought to be a legal implementation of itself)

(2) for any object A of K , there is a morphism in K from A to W (the "abstraction map")

(3) for any object A of K , there is only one morphism in K from A to W . (There is only one "reasonable" abstraction map for each data type representation A , i.e. each "concrete" value in A may reasonably represent only one "abstract" value in W .)

These observations imply that W is a final object in K , that is: <u>an abstract data type is a final object in the category of its representations</u> (where of course, "abstract data type" means abstract in the sense of [18]).

A second argument for this thesis (particularly in support of the uniqueness condition) may be made as follows: the correctness of a data type representation is proved (in [18]) relative to a particular abstraction function. Thus an implementation is a pair $(A,\mathcal{A})$ where A is a T-algebra and $\mathcal{A}$ is an abstraction map $A \to W$. This makes K a "comma category" whose objects are pairs $(A,\mathcal{A})$ as sketched above and whose morphisms $(A,\mathcal{A}) \to (B,\mathcal{B})$ are T-Alg morphisms $h:A \to B$ such that the diagram

$$\begin{tikzcd}
A \arrow[rr, "h"] \arrow[dr, "\mathcal{A}"'] & & B \arrow[dl, "\mathcal{B}"] \\
& W &
\end{tikzcd}$$

commutes. Then $(W,1)$ is a final object, since the diagram

$$\begin{tikzcd}
A \arrow[rr, "h"] \arrow[dr, "\mathcal{A}"'] & & W \arrow[dl, "1"] \\
& W &
\end{tikzcd}$$

commutes iff $h=\mathcal{A}$.

4. Data Type Extensions

Guttag [16] has suggested concentration on the issue of data type extensions--that is, the process of adding new types to existing type structures. In the example of Section 1, we extended $T_{I,+}$ to T_{ARR}. This extension is presented by adding new generators and relations to the generators and relations in the presentation of $T_{I,+}$. A presentation of a data type extension, then, might be a 4-tuple $(\Omega_0,\Delta_0,\Omega_1,\Delta_1)$ where (Ω_0,Δ_0) is a presentation of a base theory T_0 (e.g. $T_{I,+}$), and Ω_1 and Δ_1 are new generators and relations to be "added". The theory T_1 of old and new data types is (roughly) $F_{(\Omega_0\cup\Omega_1)}/(\Delta_0\cup\Delta_1)$.

What we are trying to present is a functor $T_0 \to T_1$; that is, we are trying to specify both the new theory T_1 and its relation to the base theory T_0. What requirements should be placed on this functor? Clearly, it should be product-preserving. One might require sorts of T_0 to be mapped to sorts in T_1, but for our purposes this is unnecessary. One would be upset if the additional identities in T_1 caused values in T_0 to merge (e.g. if in T_{ARR} we could conclude that $\underline{n}_2 = \underline{n}_3$). For this purpose we could ask that the functor be faithful.

Guttag [16] proposed a new condition for data type extensions. He suggested that a presentation of a data type extension was "sufficiently-complete" iff any term in $F_{\Omega_0\cup\Omega_1}(\Lambda,a)$, where a is a sort in T_0, is reducible via identities in $\Delta_0\cup\Delta_1$ to a term in T_0. The appropriate condition on the functor is Λ-fullness, which is defined as follows:

<u>Definition</u> Let T_0 be an S-sorted theory, and let C be any category. A functor $i:T_0 \to C$ is <u>Λ-full</u> (respectively <u>Λ-faithful</u>) iff for every $a \in S$, the function $T_0(\Lambda,a) \to C(i(\Lambda),i(a))$ given by $f \mapsto i(f)$ is surjective (resp., injective).

If a data type extension functor is Λ-full, then no "new" values of the old types will be present in the initial algebra of T_1. Note that the functor $T_{I,+} \to T_{ARR}$ is Λ-full but not full; the term

$$\underline{val}[\underline{alt}[\underline{empty},x_1,x_2],x_3]$$

is not equivalent to any morphism of $T_{I,+}$.

<u>Definition</u> A <u>data type extension</u> is a functor $i:T_0 \to T_1$ where T_0 and T_1 are algebraic theories, and i is product-preserving, Λ-full, and Λ-faithful.

Given a data type extension i, what is its category of representations? Clearly it should be a subcategory of <u>T_1-Alg</u>. Thus a typical representation of i is shown in Figure 4.1.(*)

$$\begin{tikzcd}
T_0 \arrow[r, "i"] & T_1 \arrow[d, "A"] \\
 & \text{Sets}
\end{tikzcd}$$

Figure 4.1

Having imposed the Λ-faithfulness condition on i to ensure that T_1 does not merge values in T_0, we would not

(*)Since i itself uniquely determines T_0 and T_1, we say "implementation of i" rather than "implementation of T_1 relative to T_0 via i" or the like.

like this information to be lost by A . Therefore we require $A \circ i$ to be Λ-faithful. For example, if i is the data type extension $T_{I,+} \to T_{ARR}$, the T_{ARR}-algebra $W: T_{ARR} \to Sets$ (defined in Section 3) is not Λ-faithful (it merges just those array values in $T_{ARR}(\Lambda, a)$ which we felt deserved merging), but $W \circ i$ is Λ-faithful (the integers don't get merged).

We impose a second condition on implementations: a "reachability" condition, which means that an implementation of i has no values except required by T_1.

Definition If $i: T_0 \to T_1$ is a data type extension, the category $K_{\underline{i}}$ of implementations of $\underline{i}$ is the full subcategory of $\underline{T_1\text{-Alg}}$ consisting of functors $A: T_1 \to Sets$ such that

(1) $A \circ i$ is Λ-faithful

and (2) for each object w of T_1, the map $\eta_w^A: T_1(\Lambda, w) \to A(w)$ given by $f \mapsto Af()$, is surjective.

Condition (2) is worthy of more explanation for the non-initiate. If $f \in T_1(\Lambda, w)$, then $Af \in Sets(A(\Lambda), A(w))$. Thus Af is a function of no arguments, yielding a value in $A(w)$. Thus $Af()$, being the application of Af to a string of no arguments, evaluates to this value. Another condition equivalent to condition (2) is that for each w , the map $T_1(\Lambda, w) \to Sets(A(\Lambda), A(w))$ given by $f \mapsto Af$ is surjective.

Proposition 1. Let T be any theory. A T-algebra $A: T \to Sets$ is initial iff for every object w of T , η_w^A is bijective.

Proof. In the canonical initial T-algebra C , $\eta_w^C(f) = f$, so η_w^C is bijective. Since all initial T-algebras are isomorphic, η_w^B is bijective for every initial T-algebra B.

If A is any T-algebra and η_w^A is bijective, $\eta_w^C \circ (\eta_w^A)^{-1}$ is an isomorphism between A and the canonical initial T-algebra C. ■

<u>Proposition 2</u>. If $i: T_0 \to T_1$ is a data type extension, and A is an object of K_i, then $A \circ i: T_0 \to \text{Sets}$ is an initial T_0-algebra.

<u>Proof</u>. $A \circ i$ is Λ-faithful, so for each object w of T_0, $\eta_w^{A \circ i}$ is injective. η_w^A is surjective, so $\eta_w^{A \circ i} = \eta_w^A \circ i$ is surjective as well. Hence, by Proposition 1, $A \circ i$ is initial. ■

This proposition establishes that in the terminology of [10] the sorts of T_0 are "protected" in T_1. The map $A \mapsto A \circ i$ ("composition with i") is the forgetful functor $\underline{T}_1\text{-}\underline{\text{Alg}} \to \underline{T}_0\text{-}\underline{\text{Alg}}$ mentioned in [10, Definition 9].

<u>Proposition 3</u>. Let $i: T_0 \to T_1$ be a theory-functor. Then i is a data type extension iff for every object A of K_i, $A \circ i$ is an initial T_0-algebra.

<u>Proof</u>. The "only-if" was shown in the previous proposition. For the reverse direction, let A be an initial T_1-algebra. $\eta_w^{A \circ i}$ is a bijection by hypothesis, and η_w^A is a bijection by Proposition 1. For any $f \in T_0(\Lambda, w)$, $\eta_w^{A \circ i}(f) = \eta_w^A(i(f))$. Hence $i(f) = (\eta_w^A)^{-1} \circ \eta_w^{A \circ i}(f)$. So i restricted to $T_0(\Lambda, w)$ is a bijection, and i is Λ-full and Λ-faithful. ■

5. Results

We have now returned to the situation we found in Section 3: we have a category K_i of data type representations. Can we find an abstract data type in K_i ? Our main theorem gives an affirmative answer:

Main Theorem. If $i:T_0 \to T_1$ is a data type extension, then K_i has a final object.

We begin with a characterization of the objects of K_i, given by Theorem 1.

Definition. Let $i:T_0 \to T_1$ be a data type extension. A congruence Q on T_1 is i-faithful iff the composite $T_0 \xrightarrow{i} T_1 \to T_1/Q$ is Λ-faithful.

Lemma 1. Let $i:T_0 \to T_1$ be a data type extension. A congruence Q on T_1 is i-faithful iff for every pair of morphisms $f,g \in T_0$ with domain Λ , if $(i(f),i(g)) \in Q$, then $f = g$. ■

Theorem 1. Let $i:T_0 \to T_1$ be a data type extension. A product-preserving functor $A:T_1 \to \text{Sets}$ is an object of K_i iff there exists an i-faithful congruence Q on T_1 such that the diagram

$$\begin{tikzcd}
T_1 \arrow[r] \arrow[dr, "A"'] & T_1/Q \arrow[d, "B"] \\
& \text{Sets}
\end{tikzcd}$$

commutes, where B is an initial algebra of T_1/Q .

Proof. (<=): Given an i-faithful congruence Q , we merely observe that B is faithful.

(=>): Given an object A of K_1, let $Q(w,v) = \{(f,g) \mid f,g \in T_1(w,v)$ and $A(f) = A(g)\}$. Q is the usual kernel congruence. To show that Q is i-faithful, let $f,g \in T_0(\Lambda,w)$ for some w, and let $(i(f),i(g)) \in Q(\Lambda,w)$. Then $A(i(f)) = A(i(g))$. Since $A \circ i$ is Λ-faithful, we conclude $f = g$. Hence Q is i-faithful by Lemma 1. Let $[f]$ denote the equivalence class of f modulo Q.

Let j denote the functor $T_1 \to T_1/Q$. Let B denote the T_1/Q-algebra given by

$$B(w) = A(w)$$
$$B([f]) = A(f) \qquad f \in T_1(w,v)$$

The second portion of the definition is independent of representatives by the construction of Q. Hence $A = B \circ j$.

It remains to show that B is an initial algebra of T_1/Q. Let C denote the canonical initial algebra of T_1/Q. We claim that B and C are isomorphic in $\underline{(T_1/Q)\text{-Alg}}$. We must give for each object w of T_1/Q, a bijective map $\xi_w : B(w) \to C(w)$ such that ξ is a natural transformation $B \to C$. In order to do this, let η_w denote the surjective map $T_1(\Lambda,w) \to A(w)$ whose existence is guaranteed by the definition of K_1. Given $x \in B(w) = A(w)$, let $\eta_w^{-1}(x)$ denote any $f \in T_1(\Lambda,w)$ such that $A(f) = x$. Now let $\xi_w(x) = j(\eta_w^{-1}(x))$. The value of $\xi_w(x)$ is independent of the choice of η_w^{-1} by construction of Q. It is easy to see that ξ_w is bijective. To show that ξ is a natural transformation, we must show that for any $[f] \in T_1/Q(w,v)$, $C([f]) \circ \xi_w = \xi_v \circ B([f])$. Let $x \in B(w) = A(w)$. Then

$$\begin{aligned} C([f])\circ\xi_w(x) &= (C([f]))(j(g)) \quad \text{where } \eta_w(g) = x \\ &= (C([f]))([g]) \\ &= [gf] \end{aligned}$$

and

$$\begin{aligned} \xi_w\circ B([f])(x) &= \xi_w\circ A(f)\ (x) \\ &= \xi_w\circ A(f)\circ A(g)\ () \quad \text{since } x = \eta_w(g) = (Ag)(), \\ &= \xi_w\circ A(gf)\ () \\ &= \xi_w\circ\eta_w(gf) \\ &= j\circ\eta_w^{-1}\circ\eta_w(gf) \\ &= j(gf) \\ &= [gf]\ . \ \blacksquare \end{aligned}$$

<u>Lemma 2</u>. Let $i:T_0 \to T_1$ be a data type extension, let A,B be objects of K_i, and for each object w of T_1, let ξ_w be a map $A(w) \to B(w)$. Then ξ is a morphism of K_i iff for each w, $\xi_w \circ \eta_w^A = \eta_w^B$.

<u>Proof</u>. Let ξ be any natural transformation from A to B. Then for any $f \in T_1(\Lambda,w)$ following diagram commutes:

$$\begin{tikzcd}
A(\Lambda) \arrow[r, "!"] \arrow[d, "Af"'] & B(\Lambda) \arrow[d, "Bf"] \\
A(w) \arrow[r, "\xi_w"'] & B(w)
\end{tikzcd}$$

Notice that since $A(\Lambda)$ and $B(\Lambda)$ are singleton sets, the topmost arrow is unique. Chasing the unique member of $A(\Lambda)$ around the diagram, we conclude that $\xi_w(\eta_w^A(f)) = \eta_w^B(f)$ for every f. Hence $\xi_w \circ \eta_w^A = \eta_w^B$.

For the "if" portion, assume that for each w, $\xi_w \circ \eta_w^A = \eta_w^B$. By surjectivity of η_w^A, it will suffice to show $\xi_v \circ Af \circ \eta_w^A = Bf \circ \xi_w \circ \eta_w^A$ (See Figure 5.1).

$$\begin{tikzcd}
T(\Lambda,w) \arrow[dr, "\eta_w^A"'] & & \\
& A(w) \arrow[r, "\xi_w"] \arrow[d, "A(f)"'] & B(w) \arrow[d, "B(g)"] \\
& A(v) \arrow[r, "\xi_v"'] & B(v)
\end{tikzcd}$$

Figure 5.1

We chase an element g of $T(\Lambda,w)$ around the diagram as follows:

$$\begin{aligned}
\xi_v \circ Af \circ \eta^A_w(g) &= \xi_v \circ Af \circ Ag\,() && (\text{Def'n of } \eta^A_w)\\
&= \xi_v \circ A(g.f)\,() && (A \text{ is a functor})\\
&= \xi_v \circ \eta^A_v(g.f) && (\text{Def'n of } \eta^A_v)\\
&= \eta^B_v(g.f) && (\xi_v \circ \eta^A_v = \eta^B_v)\\
&= B(g.f)\,() && (\text{Def'n of } \eta^B_v)\\
&= Bf \circ Bg\,() && (B \text{ is a functor})\\
&= Bf \circ \eta^B_w(g) && (\text{Def'n of } \eta^B_w)\\
&= Bf \circ \xi_w \circ \eta^A_w(g) && (\eta^B_w = \xi_w \circ \eta^A_w).
\end{aligned}$$

So ξ is a natural transformation. ■

Lemma 3. If $i:T_0 \to T_1$ is a data type extension, and A,B are objects of K_i, then there is at most one morphism from A to B in K_i.

Proof. Let ξ,ξ' be two natural transformations from A to B. Then $\xi_w \circ \eta^A_w = \eta^B_w = \xi'_w \circ \eta^A_w$. By surjectivity of η^A_w, $\xi_w = \xi'_w$. ■

We now need a theorem about least upper bounds of sets of congruences.

Theorem 2. Let T be an S-sorted theory. Let the congruences on T be ordered by inclusion. Let $\mathcal{Q}$ be any set of congruences on T. Then $\mathcal{Q}$ has a least upper bound, denoted $\bigvee\mathcal{Q}$, characterized as follows: If $f,g \in T(w,v)$, then $(f,g) \in \bigvee\mathcal{Q}$ iff there exist $f_0,\ldots,f_n \in T(w,v)$ such that

(i) $f_0 = f$

(ii) $f_n = g$

(iii) for each i , $0 \leq i < n$, there exists $Q \in \mathcal{Q}$ such that $(f_i, f_{i+1}) \in Q(w,v)$.

Proof. Let $Z(w,v) = \{(f,g) | (f,g) \in Q(w,v)$ for some $Q \in \mathcal{Q}\}$. We claim that E_Z is the desired least upper bound of $\mathcal{Q}$. If $Q \in \mathcal{Q}$, then $Q \subseteq Z \subseteq E_Z$. If R is a congruence and $Q \subseteq R$ for each $Q \in \mathcal{Q}$, then $Z \subseteq R$, so $E_Z \subseteq E_R = R$. So E_Z is the least upper bound.

It remains to show that $(f,g) \in E_Z$ iff there exists a sequence $f_0, .., f_n$ as specified. If the sequence exists, then $(f,g) \in E_Z$ by repeated application of transitivity (rule ET). We will next show that if $(f,g) \in E_Z$, then the sequence exists. The proof is by induction on derivations in the formal system E_Z . If $(f,g) \in E_Z$ via an axiom, then it is easy to see that the required sequence exists. For each rule we will have an induction step of the form: "if sequences exist for the hypotheses of the rule, then a sequence exists for the conclusion of the rule." (In the following steps, we write "$f_0, .., f_n$ is a sequence for (f,g)" to mean $f_0, .., f_n$ satisfies condition (iii) of the theorem, $f_0 = f$ and $f_n = g$). Names of quantities in the rules are taken from Section 2.

(ES): If $f_0, f_1, .., f_n$ is a sequence for (f,g) , then $f_n, f_{n-1}, .., f_0$ is a sequence for (g,f) .

(ET): If $f_0, .., f_n$ is a sequence for (f,g) , and $g_0, .., g_m$ is a sequence for (g,h) , then $f_0, .., f_n = g_0, g_1, .., g_m$ is a sequence for (f,h) .

(EC): if $f_0,..,f_n$ is a sequence for (f,f') , then for each i , $0\leq i<n$, there exists $Q\epsilon\mathcal{Q}$ such that $(f_i,f_{i+1})\epsilon Q$. Since Q is a congruence, $(gf_ih,gf_{i+1}h)\epsilon Q$ as well. Hence $gf_0h,..,gf_nh$ is a sequence for $(gfh,gf'h)$.

(EP): For $1\leq i\leq n$, let $f_{i0},..,f_{ip_i}$ be a sequence for (f_i,f_i') . Let $P(k) = \sum_{i=1}^{k-1} p_i$. We construct a sequence g_j $(0\leq j\leq P(n+1))$ for $([f_1,..,f_n],[f_1',..,f_n'])$ by specifying the projections of the g_j :

$$e_kg_j = \begin{cases} f_k & j<P(k) \\ f_{ki} & P(k)\leq j<P(k+1);\ i = j-P(k) \\ f_k' & j\leq P(k+1) \end{cases}$$

The effect of this construction is to create a sequence which changes one component at a time:

$[f_{10},f_2,..,f_n],[f_{11},f_2,..,f_n],..,[f_1',f_2,..,f_n],[f_1',f_{21},..f_n],..$ etc. As in the argument for rule EC, for each step in the sequence, (g_j,g_{j+1}) is in some $Q\epsilon\mathcal{Q}$ because the pair of components which change is in some congruence Q , so the whole step is in Q by applying rule EP for Q . Verification of the details is left to the diligent reader. ■

Note: Theorem 2 is a variation of a theorem well-known for single-sorted algebras [15,Lemma 10.2]. An alternate proof could be obtained by proving the theorem for many-sorted algebras in general [2], and then observing that an S-sorted theory is itself an $S^*\times S^*$-sorted algebra [1].

Theorem 3. Let $i:T_0\to T_1$ be a data type extension, and let

$\mathcal{Q}$ be a set of i-faithful congruences on T_1 . Then $\bigvee\mathcal{Q}$ is i-faithful.

<u>Proof</u>. Let $f,g\in T_0(\Lambda,w)$, with $(i(f),i(g))\in\bigvee\mathcal{Q}$. Then by Theorem 2 there exist $f_0,..,f_n\in T_1(\Lambda,w)$ such that $f_0 = i(f)$, $f_n = i(g)$, and for each j , $0\le j<n$, $(f_j,f_{j+1})\in Q$ for some $Q\in\mathcal{Q}$. Since i is a data type extension, it is Λ-full, so for each f_j there exists $g_j\in T_0(\Lambda,w)$ such that $f_j = i(g_j)$. By Lemma 1, $g_j = g_{j+1}$. Hence $f_j = f_{j+1}$, and $f_0 = f_n$. Since i is Λ-faithful, $f = g$. By Lemma 1, this establishes that $\bigvee\mathcal{Q}$ is i-faithful. ■

We are now ready to prove the main theorem.

<u>Proof of the main theorem</u>. Let $\mathcal{Q}$ be the set of all i-faithful congruences on T_1 . Let $\hat{Q} = \bigvee\mathcal{Q}$, let $\hat{T} = T_1/\hat{Q}$, and let $\hat{j}$ be the quotient functor $T_1\to\hat{T}$. Let W be the canonical initial algebra of $\hat{T}$, and let $C = W\circ\hat{j}:T_1\to\mathrm{Sets}$. We claim that C is a final object of K_i . By Theorem 3, $\hat{Q}$ is i-faithful, so by Theorem 1, C is an object of K_i .

Now let A be any object of K_i . By Theorem 1, there exists an i-faithful congruence Q on T_1 , with quotient functor $j:T_1\to T_1/Q$, and an initial algebra B of T_1/Q such that $A = B\circ j$. Since Q is i-faithful, we have a theory-functor $k:(T_1/Q)\to(T_1/\hat{Q}) = \hat{T}$ such that $\hat{j} = k\circ j$.

By Lemma 2, we need only show that for each object w of T_1 , η^C_w factors through η^A_w . We claim that $\eta^C_w = k\circ(\eta^B_w)^{-1}\circ\eta^A_w$ Since B is an initial T_1/Q-algebra, η^B_w is a bijection, so $(\eta^B_w)^{-1}$ is well defined. So, if $f\in T_1(\Lambda,w)$, then

$$
\begin{aligned}
&k\circ(\eta_w^B)^{-1}\circ\eta_w^A(f) = \\
&\quad = k\circ(\eta_w^B)^{-1}(Af)() && (\text{Definition of } \eta_w^A) \\
&\quad = k\circ j(f) && (*) \\
&\quad = \hat{j}(f) && (\hat{j} = k\circ j) \\
&\quad = \eta_w^C(f) && (\text{Definition of } C)
\end{aligned}
$$

As justification for the step marked (*), we calculate:

$$\eta_w^B(j(f)) = B(j(f))() = Af()$$

Hence $\xi_w = k\circ(\eta_w^B)^{-1}$ is the required natural transformation. Uniqueness is guaranteed by Lemma 3. ■

6. Example

In this section we will complete our consideration of the array example.

Proposition 3. Let i be the data type extension $T_{I,+} \to T_{ARR}$. Then the T_{ARR}-algebra W, defined in Section 3, is a final object of K_i.

Proof: First, it is straightforward to check that W is an object of K_i. Furthermore, if $f \epsilon T_{ARR}(\Lambda, a)$, it is easy to show from the axioms for T_{ARR} that the partial function $W(f)$ is defined at just those integers j such that in T_{ARR}, $\underline{val}[f,\underline{n}_j] = \underline{n}_k$ for some k, and that for any other j, $\underline{val}[f,\underline{n}_j]$ = undefined.

Now, let A be any object of K_i. By Lemma 2, to get a morphism $\xi : A \to W$, we must show that η_w^W factors through η_w^A. Since the η's are in the category of sets, we need only show that for all $f,g \epsilon T_1(\Lambda, w)$, if $\eta_w^A(f) = \eta_w^A(g)$, then $\eta_w^W(f) = \eta_w^W(g)$. Because A and W preserve products, it is enough to prove this for the case where w is a single sort. The integer sorts of all the algebras in K_i are isomorphic (they are initial algebras of $T_{I,+}$ by Proposition 2), so the only interesting case is where $w = a$ (the array sort). In the following, we will write η^X for η_a^X.

Let $f,g \epsilon T_{ARR}(\Lambda, a)$. We will show that if $\eta^W(f) \neq \eta^W(g)$, and $\eta^A(f) = \eta^A(g)$, then $A \circ i$ is not Λ-faithful. There are two ways in which $\eta^W(f)$ and $\eta^W(g)$ could be unequal. They may have different domains, or they may have different values at some point of their domain.

If they have different domains, then without loss of generality there exists some $\underline{n}_p \epsilon T_{ARR}(\Lambda,\mathcal{i})$ such that in T_{ARR}, $\underline{\text{val}}[f,\underline{n}_p] = \underline{\text{undefined}}$ and $\underline{\text{val}}[g,\underline{n}_p] = \underline{n}_k$ for some k. If $\eta^A(f) = \eta^A(g)$, then $\eta^A(\underline{\text{undefined}}) = A(\underline{\text{val}})[\eta^A(f),\eta^A(\underline{n}_p)] = A(\underline{\text{val}})[\eta^A(g),\eta^A(\underline{n}_p)] = \eta^A(\underline{n}_k)$. So $A \circ i$ is not Λ-faithful.

Similarly, if they are unequal at some point in their domain, there exists some $\underline{n}_p \epsilon T_{ARR}(\Lambda,\mathcal{i})$ such that in T_{ARR}, $\underline{\text{val}}[f,\underline{n}_p] = \underline{n}_j$ and $\underline{\text{val}}[g,\underline{n}_p] = \underline{n}_k$, for $k \neq j$. If $\eta^A(f) = \eta^A(g)$, it then follows that $\eta^A(\underline{n}_j) = \eta^A(\underline{n}_k)$, again showing that $A \circ i$ is not Λ-faithful. ■

We have considered initial and final algebras in K_i. Are there any interesting intermediate cases? The answer is yes. Add, for example, to the axioms for T_{ARR} the following axioms:

$$\underline{\text{alt}}[\underline{\text{alt}}[x,\underline{n}_k,y],\underline{n}_k,z] = \underline{\text{alt}}[x,\underline{n}_k,z] \qquad k\epsilon\omega \& k\neq 7 \& k \neq 9$$

$$\underline{\text{alt}}[\underline{\text{alt}}[x,\underline{n}_k,y],\underline{n}_p,z] = \underline{\text{alt}}[\underline{\text{alt}}[x,\underline{n}_p,z],\underline{n}_k,y] \qquad k,p\epsilon\omega, k\neq p$$

$$\underline{\text{alt}}[x,\underline{\text{undefined}},y] = x$$

The resulting initial algebra suppresses most of the order information but preserves "traces" of all values assigned to location 7 or to location 9. A single trace showing how these assignments were interleaved may be obtained by putting similar restrictions on the second axiom scheme.

7. Final vs. Initial Algebra Semantics

The final algebra constructed in the main theorem is an initial algebra of $T_1/\hat{Q}$. Why then, do we distinguish "final algebra semantics" from "initial algebra semantics"? The answer lies in the primacy of specification.

We believe that a program should interact with a data type only through its specifications. Thus a specification, which is a formal object in some logical calculus, must present a class of algebras, namely, the class of implementations of the data type.

As we argued in Section 3, a theory of data type representations should make the "true" data type a final object in its category of representations. Methodologically, final algebra semantics is more desirable in this regard.

Methodological considerations aside, it may be that T_0 and T_1 have tractable presentations, but $T_1/\hat{Q}$ does not. In our case, T_0 and T_1 had Church-Rosser presentations, but the obvious presentation of $T_1/\hat{Q}$ was not Church-Rosser. We leave it open whether there exist finitely presentable T_0 and T_1 such that $T_1/\hat{Q}$ is not finitely presentable.

In any case, "final algebra semantics" should be regarded as an extension, rather than a competitor, of initial algebra semantics. If i is the identity functor $T_1 \to T_1$, then K_i consists entirely of initial T_1-algebras. Furthermore, if T_1 and $T_1/\hat{Q}$ happen to be equal, then initial and final algebras coincide again. For example, take T_0 to be $T_{I,+}$ as before and let T_1 be given by adding a new sort s ("string of integers") with $T_1(\Lambda, s)=\omega^*$ and operations

$$\underline{sel}_k : s \to i \quad k\epsilon\omega$$

which select the k-th integer from a string (or give <u>undefined</u> if the string is too short). Now, any i-faithful congruence on T_1 must be the equality relation (for if not, assume α and β are two distinct congruent strings. Since they are distinct, they must differ at some position (say the j-th position). Then

$$\underline{n}_k = \underline{sel}_j \alpha \equiv \underline{sel}_j \beta = \underline{n}_p$$

for $k \neq p$, violating i-faithfulness.) So again K_i consists of initial T_1-algebras, but the presentation of K_i by i also specifies the relation of T_0 to T_1.

8. Concluding Remarks

The situation discussed in this paper is reminiscent of categories of automata, constrained by the requirement that some external behavior be maintained. In our case the "external behavior" is the behavior which is reflected in the sorts of T_0 ; hence the condition that $A \circ i$ be Λ-faithful. One has initial realizations and, if one imposes a reachability condition, one has a minimal realization which is a final object [7]. We, too, have an initial realization (an initial T_1-algebra), and a final or minimal realization whose existence is our main result.

Similar remarks are echoed in [6]. Our notion of extension includes all of [10, Def. 9] including enrichment. It also allows the possibility that a single sort in T_0 is mapped to a tuple of sorts in T_1 . We regard this paper as complementary to [10], which seems to be devoted to the problems of specifying T_1 (which is no small task!).

Another echo deserving of mention is that of data structure selection and optimization. The initial implementation is a very crude data structure, consisting solely of trees (see e.g. [17]). In our example, arrays are represented as lists of subscript-value pairs (without even deleting updated entries!). By looking at the required updates and addresses, the data structure implementing the data type may be optimized until no redundant information is stored. In our example, arrays turn out to be optimal in this sense.

We leave open the question of formulating a "behavior" functor adjoint to "minimal realization" [7]; such a development might shed some light on the distinction between data structures and data types. Another extension could involve types with type parameters.

References

1. Benabou, Jean, Structures Algebriques dan les Categoriés, Cahiers de Topologie et Géométrie Différentielle 10:1 (1968), 1-126.

2. Birkhoff, G. and Lipson, D., Heterogeneous Algebras, J. Combinatorial Theory 8 (1970), 115-133.

3. Burstall, R. M., and Darlington, John, A Transformation System for Developing Recursive Programs, J. ACM 24 (1977), 44-67.

4. Chirica, L. M., and Martin, D. F., An Algebraic Formulation of Knuthian Semantics, "Proc. 17th IEEE Symp. on Foundations of Computing," 1976.

5. Elgot, C. C., Monadic Computation and Iterative Algebraic Theories, "Proceedings of the Logic Colloquium," (Bristol, 1973), (H. E. Rose and J. C. Shepherdson, eds.), North-Holland, Amsterdam, 1975, pp. 175-230.

6. Giarratana, V., Gimona, F., and Montanari, U., Observability Concepts in Abstract Data Type Specifications, "Mathematical Foundations of Computer Science 1976," (A. Mazurkiewicz, ed.) Springer Lecture Notes in Computer Science, vol. 45 (1976), 576-587.

7. Goguen, J. A., Realization is Universal, Math. Sys. Th. 6 (1972), 359-374.

8. Goguen, J. A., On Homomorphisms, Correctness, Termination, Unfoldments, and Equivalence of Flow Diagram Programs, J. Comp. & Sys. Sci. 8 (1974), 333-365.

9. Goguen, J. A., Correctness and Equivalence of Data Types, "Mathematical Systems Theory," (Udine, 1975), (G. Marchesihi & S. K. Mitter, eds.), Lecture Notes in Economics and Mathematical Systems, vol. 131, Springer, 1976, pp. 352-358.

10. Goguen, J. A., Thatcher, J. W., and Wagner, E. G., An Initial Algebra Approach to the Specification, Correctness, and Implementation of Abstract Data Types, to appear in "Current Trends in Programming Methodology, IV, Data Structuring", (R. Yeh, ed.), Prentice-Hall, New Jersey, 1978.

11. Goguen, J. A., Thatcher, J. W., Wagner, E. G., and Wright, J. B., A Junction between Computer Science and Category Theory: I, Basic Definitions and Examples, (Part 1), IBM Research Report RC 4526, 1973, (Part 2), IBM Research Report RC 5908, 1976.

12. Goguen, J. A., Thatcher, J. W., Wagner, E. G., and Wright, J. B., An Introduction to Categories, Algebraic Theories, and Algebras, IBM Research Report RC 5369, April, 1975.

13. Goguen, J. A., Thatcher, J. W., Wagner, E. G., and Wright, J. B., Abstract Data Types as Initial Algebras and Correctness of Data Representations, "Proc. ACM Conf. on Computer Graphics, Pattern Recognition, and Data Structures", May, 1975, pp. 89-93.

14. Goguen, J. A., Thatcher, J. W., Wagner, E. G., and Wright, J. B., Initial Algebra Semantics and Continuous Algebras, J. ACM 24 (1977), 68-95.

15. Grätzer, George, "Universal Algebra", Van Nostrand, Princeton, 1968.

16. Guttag, J. V., The Specification and Application to Programming of Abstract Data Types, Computer System Research Report CSRG-59, University of Toronto, Department of Computer Science, 1975.

17. Guttag, J. V., Horowitz, E., and Musser, D. R., Abstract Data Types and Software Validation, University of Southern California Information Sciences Institute Research Report ISI/RR-76-48 (August, 1976).

18. Hoare, C. A. R., Proving Correctness of Data Representations, Acta Informatica 1 (1972), 271-281.

19. Knuth, D. E., Semantics of Context-Free Languages, Math. Sys. Th. 2 (1968), 127-145; correction, 5 (1971), 95-96.

20. Lawvere, F. W., Functorial Semantics of Algebraic Theories, Proc. NAS USA 50 (1963), 869-872.

21. Robinson, L., and Levitt, K. N., Proof Techniques for Hierarchically Structured Programs, Comm. ACM 20:4 (April, 1977), 271-283.

22. Rosen, B. K., Tree Manipulating Systems and Church-Rosser Theorems, J. ACM 20 (1973), 160-187.

23. Wand, M., Continuation-Based Program Transformation Strategies, Technical Report No. 61, Indiana University Computer Science Department, 1977.

24. Wright, J. B., Thatcher, J. W., Wagner, E. G., and Goguen, J. A., Rational Algebraic Theories and Fixed Point Solutions, "Proc. 17th IEEE Symp. on Foundations of Computing", 1976.

12. Goguen, J. A., Thatcher, J. W., Wagner, E. G., and Wright, J. B., An Introduction to Categories, Algebraic Theories, and Algebras, IBM Research Report RC 5369, April, 1975.

13. Goguen, J. A., Thatcher, J. W., Wagner, E. G., and Wright, J. B., Abstract Data Types as Initial Algebras and Correctness of Data Representations, "Proc. ACM Conf. on Computer Graphics, Pattern Recognition, and Data Structures", May, 1975, pp. 89-93.

14. Goguen, J. A., Thatcher, J. W., Wagner, E. G., and Wright, J. B., Initial Algebra Semantics and Continuous Algebras, J. ACM 24 (1977), 68-95.

15. Grätzer, George, "Universal Algebra", Van Nostrand, Princeton, 1968.

16. Guttag, J. V., The Specification and Application to Programming of Abstract Data Types, Computer System Research Report CSRG-59, University of Toronto, Department of Computer Science, 1975.

17. Guttag, J. V., Horowitz, E., and Musser, D. R., Abstract Data Types and Software Validation, University of Southern California Information Sciences Institute Research Report ISI/RR-76-48 (August, 1976).

18. Hoare, C. A. R., Proving Correctness of Data Representations, Acta Informatica 1 (1972), 271-281.

19. Knuth, D. E., Semantics of Context-Free Languages, Math. Sys. Th. 2 (1968), 127-145; correction, 5 (1971), 95-96.

20. Lawvere, F. W., Functorial Semantics of Algebraic Theories, Proc. NAS USA 50 (1963), 869-872.

21. Robinson, L., and Levitt, K. N., Proof Techniques for Hierarchically Structured Programs, Comm. ACM 20:4 (April, 1977), 271-283.

22. Rosen, B. K., Tree Manipulating Systems and Church-Rosser Theorems, J. ACM 20 (1973), 160-187.

23. Wand, M., Continuation-Based Program Transformation Strategies, Technical Report No. 61, Indiana University Computer Science Department, 1977.

24. Wright, J. B., Thatcher, J. W., Wagner, E. G., and Goguen, J. A., Rational Algebraic Theories and Fixed Point Solutions, "Proc. 17th IEEE Symp. on Foundations of Computing", 1976.

LIST OF SYMBOLS

≠	AMS #10	≥	AMS #29
'	AMS #61	Σ	AMS #69
→	AMS #97	×	AMS #1
[	AMS #137	ϵ , ε	AMS #53
]	AMS #138	^	circumflex
{	AMS #139	=>	AMS #112
}	AMS #140	ω , ω	l.c. omega
∘	AMS #80	↦	Barred arrow
Ø	AMS #150	/	AMS #143
*	AMS #131		
$\mid$	AMS #145		
⊂	AMS #45		
Δ	AMS #92		
⊢	AMS #85		
Λ	AMS #68		
=	AMS #8		
∪	AMS #58		
<=	AMS #113		
■	AMS #96		
•	AMS #78		
∨	AMS #67		
≤	AMS #28		
<	AMS #24		
⊆	AMS #47		

Set underlined symbols in boldface e.g. $\underline{\text{undefined}}$, $\underline{\text{plus}}$, $\underline{n}_k$

Set script symbols in script e.g. $\mathscr{i}$, $\mathscr{a}$