INITIAL ALGEBRA SEMANTICS

J. A. Goguen
Computer Science Department
UCLA
Los Angeles, California 90024

J. W. Thatcher
Mathematical Sciences Department
IBM Thomas J. Watson Research Center
Yorktown Heights, New York 10598

Abstract

Many apparently divergent approaches to specifying formal semantics for programming languages are applications of initial algebra semantics. Here we provide an overview of the concept of initial algebra semantics.

1. Introduction

There has been quite a proliferation of formal semantics for programming languages, or at least of different descriptive terms the past few years. Examples include operational, interpretive, fix-point, predicate calculus, denotational, algebraic, mathematical, synthesized, W-grammar, axiomatic, inherited, declarative, and now, initial algebra semantics. Moreover, mathematical concepts, that may be foreign to the computer scientist, are said to have bearing on the area. Examples of these are continuous lattices, iterative algebraic theories, infinitary logic and bi-categories. This is quite perplexing. What relations do all these concepts bear to one another? How can all these things fit together, or do they at all? In fact, what is "syntax"; what is "semantics"?

This paper is not going to answer all these questions, but we believe that the subject cannot be said to be in very good shape when such questions are ignored or glossed over, when the "practical" and "theoretical" approaches have so little to say to each other. In this paper we offer a unified approach to these questions with some preliminary answers. These tend to show a surprising and beautiful unity hidden in the apparent diversity of approaches.

The key to the unification lies in one of the simplest concepts of algebra: an algebra S is *initial* in a class $\underline{\underline{C}}$ of algebras iff for every A in $\underline{\underline{C}}$ there exists a unique homomorphism $h_A : S \to A$.[1]

In the cases we shall examine, syntax *is* an initial algebra in a class of algebras. Any other algebra A in the class is a candidate for a *semantic domain* (or *semantic algebra*); the *semantic function* is the uniquely determined homomorphism $h_A : S \to A$ which assigns a *meaning* $h_A(s)$ in A to each syntactic structure s in S. From this point of view it becomes clearer that a major aspect of formal semantics (both practical and theoretical) is the process of constructing the *intended* semantic algebra for particular programming languages. We argue here that that is what Scott and Strachey [54] and followers are doing using the tools outlined by Scott [50] and more fully developed in [52, 53].

Our approach uses *abstract syntax* as proposed by McCarthy [31]. But rather than employing "concrete" versions of "abstract Syntax" as in the Vienna Definition Language [29], we claim to have mathematicized (abstracted?) the concept of "abstract syntax" in the following simple Proposition.

Proposition 1.1. If S and S' are both initial in a class $\underline{\underline{C}}$ of algebras then S and S' are isomorphic and if S'' is isomorphic to an initial algebra S, then S'' is initial. □

Rather than assuming "that programs are 'really' abstract, hierarchically structured data objects ...," as Reynolds [45] does, we avoid some troublesome questions of definition and notation by identifying "abstract syntax" with "initial algebra"; Proposition 1.1 says that abstract syntax is independent of notational variation as we have always believed it should be. Illustrative of this point is the fact that for a ranked alphabet Σ, we can define a Σ-tree to be an element of an initial Σ-algebra. Thus we are not tied to any particular representation of trees (Polish-notation, infix terms, prefix parenthesized terms, functions defined on tree domains or, certain directed ordered labeled graphs). This abstract syntax for trees depends only on the essential algebraic-structural properties that characterize trees.

Because of the important tie with abstract syntax, the initial algebra approach is really implicit in McCarthy's [31] proposals for a "Mathematical Science of Computation," and in Irons' pioneering paper [24] on syntax-directed translation. The approach becomes more explicit at least as far back as Landin [26] and Petrone [42] as well as in McCarthy and Painter [33], Knuth [25], Burstall and Landin [9], Nivat [41], and Morris [40].

This paper makes the initial algebra approach to semantics completely explicit, and extends its applicability by demonstrating the existence of initial *continuous algebras*; this last is our main new result and is of considerable interest in its own right.

In the next section we make precise what kinds of algebras we are talking about -- many-sorted algebras. This material is not mathematically new, and should be viewed as survey in character. It is not brought together in this way elsewhere in the literature and seems to us to be of such fundamental importance. In studying Section 2 carefully, the reader will be serving multiple sentences concurrently because what we do there can be done in most any initial algebra situation and for initial continuous algebras (Section 4) in

[1] The reader seeking a more comprehensive understanding of the algebraic context of these ideas may want to consult [21] and/or [34].

particular. Section 3 contains applications (examples) of initial many-sorted algebras. Section 4 contains the definition of "continuous algebras" and the construction of initial continuous algebras, and we consider applications, including Scott's lattice of flow diagrams [51] in Section 5. We conclude with some questions and problems in the last section.

We individually and jointly are grateful for stimulating discussion on these matters with R. M. Burstall and R. Milner of Edinburgh University, and D. Berry, R. Pottenger, and C. Lucena of UCLA. We want especially to thank E. G. Wagner and B. Rosen for their suggestions and J. B. Wright for his continued interest, patience, and contributions which have been so very important for us. Some of the material included here is being prepared for inclusion in the second part of the first report of the ADJ series [21]. We are very appreciative of Martha Pierce's expert and expeditious typing of this manuscript.

2. Many-Sorted Algebras

An algebra in the sense of Birkhoff [4,5] and Cohn [12] is simply a set, called the carrier of the algebra, together with an indexed family of operations (functions) defined on (Cartesian powers of) that set. The generalization to many-sorted algebras is very natural for the computer scientist; a many-sorted algebra consists of an indexed family of sets (carriers) and an indexed family of operations defined on Cartesian products of those sets. The set indexing the carriers is called the set of sorts which we could take, for example, to be {real, int, bool}. An algebra of this sort would have three sets,

$A_{\underline{real}}$, $A_{\underline{int}}$, and $A_{\underline{bool}}$ together with some family of operations such as $\uparrow: A_{\underline{real}} \times A_{\underline{int}} \to A_{\underline{real}}$ or $\to: A_{\underline{bool}} \times A_{\underline{real}} \times A_{\underline{real}} \to A_{\underline{real}}$ which might be exponentiation and conditional respectively.

A familiar example of a many-sorted algebra is a finite automaton in the sense of Rabin and Scott [44] which is an {S,Σ}-sorted algebra with one operation M of "type" <SΣ,S> ($M:S \times \Sigma \longrightarrow S$, using the sorts as names for the carriers, an ambiguous but prevalent practice) which is the transition function and one constant s_0 of sort $S(s_0 \in S)$.

The definition to be given below is equivalent to that of "heterogeneous algebra" in Birkhoff and Lipson [6] and according to them, in turn equivalent to "algebra with a scheme of operators" of Higgins [23]. Birkhoff and Lipson show that the conventional theory of "universal algebra" (as in [4,5,12]) carries over "with undiminished force" to the theory of many-sorted algebras. This means, essentially, that the concepts of subalgebra, homomorphism, quotient, congruence relation,product, word algebras, and free algebras generalize naturally and easily. Although Birkhoff and Lipson use some computer science concepts as examples of many-sorted algebras the first explicit use for new results in computer science seems to be in Maibaum [37].

Let S be a set, elements of which are called sorts. An S-sorted operator domain Σ is a family $\langle\Sigma_{w,s}\rangle$ of disjoint sets indexed by $S^* \times S$. $\Sigma_{w,s}$ is the set of operator symbols of type <w,s> arity w, sort s and rank lg(w). (lg(w) is the length of w; $\lambda \in S^*$ is the empty string.) A Σ-algebra A consists of a family $\langle A_s\rangle_{s\in S}$ of sets called the carriers of A (A_s is the carrier of sort $s\in S$) and for each $\langle w,s\rangle \in S^* \times S$ and each $\sigma\in\Sigma_{w,s}$, an operation σ_A of type <w,s>, i.e., $\sigma_A : A_{w_1} \times\cdots\times A_{w_n} \longrightarrow A_s$ where $w = w_1\cdots w_n$ $(w_i \in S)$. An operation σ_A of type <λ,s> is a constant of sort s, i.e., $\sigma_A \in A_s$.

For fixed S and varying Σ we have the class of S-sorted algebras and as S varies, the class of many-sorted algebras. We will be interested in fixing S and Σ and let $\underline{\underline{\mathrm{Alg}}}_\Sigma$ denote the class of Σ-algebras.

When #S=1, we have the conventional case of (one-sorted) algebras -- here the operator domain is just as well indexed by ω = {0,1,2,···}; $\sigma\in\Sigma_n$ names an operation $\sigma_A : A^n \longrightarrow A$ in an algebra A (the single carrier and the algebra being ambiguously denoted by the same symbol).

Returning to the general case, if A and A' are both Σ-algebras, a Σ-homomorphism $h:A \longrightarrow A'$ is a family of functions $\langle h_s : A_s \longrightarrow A'_s\rangle_{s\in S}$ that preserve the operations, i.e.,

(0) if $\sigma\in\Sigma_{\lambda,s}$ then $h_s(\sigma_A) = \sigma_{A'}$.

(1) if $\sigma\in\Sigma_{s_1\cdots s_n,s}$ and $\langle a_1,\cdots,a_n\rangle \in A_{s_1} \times\cdots\times A_{s_n}$ then

$$h_s(\sigma_A(a_1,\cdots,a_n)) = \sigma_{A'}(h_{s_1}(a_1),\cdots,h_{s_n}(a_n))$$

Now that we have the class $\underline{\underline{\mathrm{Alg}}}_\Sigma$ of Σ-algebras and the concept of Σ-algebra homomorphism, we can state the first basic result concerning initial algebras.

Proposition 2.1. The class $\underline{\underline{\mathrm{Alg}}}_\Sigma$ of Σ-algebras has an initial algebra, call it T_Σ. □

This result is well known, at least for the one-sorted case, i.e., where #S=1. (See Birkhoff [4,5] or Cohn [12].) And the many-sorted case is treated in Birkhoff and Lipson [6]. T_Σ is often called the Σ-word algebra and the carriers are sometimes called the Herbrand universe for Σ. The set $T_{\Sigma,s}$ (the carrier of T_Σ of sort s) can be considered to be the set of well formed expressions (or trees) of sort s built up in the usual way from the operator symbols of Σ, but it must be emphasized that this characterization if mathematically important only for the proof of Proposition 2.1. Once proved, we don't need to know how a particular initial algebra was constructed -- we use only initiality, and comments about expressions (or pictures of trees) occur only as an aid to understanding.

In the (in any) initial algebra T_Σ in the class $\underline{\underline{\mathrm{Alg}}}_\Sigma$, if $\sigma\in\Sigma_{s_1\cdots s_n,s}$ and $t_i\in T_{\Sigma,s_i}$ $(1 \le i \le n)$ then $\sigma_T(t_1,\cdots,t_n)\in T_{\Sigma,s}$. (We write σ_T for the operation named by σ in T_Σ.) It seems that this algebraic structure is what is often referred to as "synthetic syntax." On the other hand, given $t\in T_{\Sigma,s}$ there exist unique $n \ge 0$, $\langle s_1\cdots s_n,s\rangle \in S^* \times S$, $\sigma\in\Sigma_{\langle s_1\cdots s_n,s\rangle}$ and $t_i\in T_{\Sigma,s_i}$ such that $t = \sigma_T(t_1,\cdots,t_n)$. This unique decomposition in the initial algebra coincides with "analytic syntax."

What we want to do now is give a clean definition of "derived operator" in an algebra; the intuitive idea is simple enough -- we want to define terms with variables, and corresponding to every such term is a derived operator in any Σ-algebra, A. Because it is notationally so much simpler, we will consider the one-sorted case first.

Let X be a set (of variables) disjoint from Σ. Define the ranked alphabet $\Sigma(X)$ by $\Sigma(X)_0 = \Sigma_0 \cup X$ and $\Sigma(X)_k = \Sigma_k$ for $k > 0$. By Proposition 2.1, $T_{\Sigma(X)}$ is the initial $\Sigma(X)$-algebra. Clearly we can view $T_{\Sigma(X)}$ as a Σ-algebra, just by forgetting the new zero-ary symbols X; call this Σ-algebra $T_\Sigma(X)$.

Proposition 2.2. $T_\Sigma(X)$ is the free Σ-algebra on generators X in the sense that if $h:X \longrightarrow A$ is any function mapping X into the carrier of a Σ-algebra A, there exists a unique Σ-homomorphism $\bar{h}:T_\Sigma(X) \longrightarrow A$ that extends h in the sense that $\bar{h}(x_T) = h(x)$ for all $x \in X$.

Proof. A is a Σ-algebra. Given $h:X \longrightarrow A$, make A into a $\Sigma(X)$-algebra by having x name $h(x)$ in A $(x_A = h(x))$. Then by Proposition 2.1, there exists a unique $\Sigma(X)$-homomorphism $\bar{h}:T_{\Sigma(X)} \longrightarrow A$ and by clause (0) of the definition of homomorphism $\bar{h}(x_T) = x_A = h(x)$; obviously $\bar{h}$ is also a Σ-homomorphism. □

This construction is typical of initial algebra semantics. The phrase, "make B into a Σ-algebra by ...," is fundamental because making B into a Σ-algebra, gives the unique homomorphism $h_B:T_\Sigma \longrightarrow B$. In the proof of Proposition 2.2 it was particularly easy -- A was already a Σ-algebra and we were making it into a $\Sigma(X)$-algebra, but in general if B is any set (later family indexed by S) we can make B into a Σ-algebra by defining the appropriate operations $\sigma_B:B^n \longrightarrow B$ for each $\sigma \in \Sigma_n$; then there's that unavoidable homomorphism $h_B:T_\Sigma \longrightarrow B$.

To emphasize this point we mention another example for the one-sorted case and suggest that the reader might try others like "height" and "number of nodes" for one- and many-sorted cases.

Make Σ_0^* into a Σ-algebra by:

(0) For $\sigma \in \Sigma_0$ $\quad \sigma = \sigma \in \Sigma_0^*$

(1) For $\sigma \in \Sigma_n$, $w_1, \cdots, w_n \in \Sigma_0^*$,

$$\sigma(w_1,\cdots,w_n) = w_1 \cdots w_n .$$

The unique Σ-homomorphism $fr:T_\Sigma \longrightarrow \Sigma_0^*$ is the frontier (or yield) function.

Generally it's clear and nice to write σ_A to specify what σ names in the algebra A, but this gets tiresome and in fact is not conventional practice. For Abelian groups one doesn't see $+_G$ and 0_G, but just $+$ and 0; the symbols indeed name the functions and constants. At times it is a bit confusing, but that's what we are doing in these examples. In (0), $\sigma \in \Sigma_0$ names the string $\sigma \in \Sigma_0^*$ of length 1.

Now with a little notational dexterity, we can define derived operators. First let $X_n = \{x_1,\cdots,x_n\}$, then we can say $t \in T_\Sigma(X_n)$ is a Σ-term in n variables. For any Σ-algebra A, define $t_A:A^n \longrightarrow A$ by $t_A(a_1,\cdots,a_n) = \bar{a}(t)$ where $a:X_n \longrightarrow A$ sends $x_i \mapsto a_i$, $1 \le i \le n$, and $\bar{a}$ is the unique homomorphism guaranteed by Proposition 2.2.

As a special case we can take $A = T_\Sigma(X_m)$ and so $t_T(t_1'\cdots t_n') = \bar{t}'(t)$; here x_i, $1 \le i \le m$, is assigned t_i' by t' and $t_T(t_1'\cdots t_n')$ is the result of simultaneously substituting t_i' for x_i in t. We want to use special notation ($\leftarrow$ as a binary function) for substitution, and we want to extend it to substitution into m-tuples. Thus if we have

$$\begin{tikzcd}
X_m \arrow[dr, "t"] & X_n \arrow[d] \arrow[dr, "t'"] & X_p \arrow[d, dash] \arrow[dr, "t''"] & \\
& T_\Sigma(X_n) \arrow[r, "\bar{t}'"] & T_\Sigma(X_p) \arrow[r, "\bar{t}''"] & T_\Sigma(X_q)
\end{tikzcd}$$

we can define $t \leftarrow t'$ as $\bar{t}' \circ t$ ($f \circ g$ denotes function composition), identifying the m-tuple $\langle t_1,\cdots,t_m \rangle$ with $t:X_m \longrightarrow T_\Sigma(X_n)$. Thus $t \leftarrow t'(x_i) = (\bar{t}' \circ t)(x_i) = \bar{t}'(t(x_i))$ -- $t \leftarrow t'$ is substitution into t, componentwise. But with this extension we can show that substitution is associative because if $t'':X_p \longrightarrow T_\Sigma(X_q)$, then $(t \leftarrow t') \leftarrow t'' = (\bar{t}' \circ t) \leftarrow t'' = \bar{t}'' \circ (\bar{t}' \circ t) = (\bar{t}'' \circ \bar{t}') \circ t$. But $\bar{t}'' \circ \bar{t}'$ is a homomorphism extending $\bar{t}'' \circ t'$ so $\overline{\bar{t}'' \circ t'} = \bar{t}'' \circ \bar{t}'$. We have $(t \leftarrow t') \leftarrow t'' = (\bar{t}'' \circ \bar{t}') \circ t = \overline{\bar{t}'' \circ t'} \circ t = t \leftarrow (\bar{t}'' \circ t') = t \leftarrow (t' \leftarrow t'')$.

Proposition 2.3. Substitution is associative. For $t:X_m \longrightarrow T_\Sigma(X_n)$, $t':X_n \longrightarrow T_\Sigma(X_p)$ and $t'':X_p \longrightarrow T_\Sigma(X_q)$, $(t \leftarrow t') \leftarrow t'' = t \leftarrow (t' \leftarrow t'')$ □

An m-tuple of terms $t:X_m \longrightarrow T_\Sigma(X_n)$ defines a function $t_A:A^n \longrightarrow A^m$, so if $t':X_n \longrightarrow T_\Sigma(X_p)$, then $t_A':A^p \longrightarrow A^n$. The uniqueness of the homomorphism (initiality) also gives

Proposition 2.4. $(t \leftarrow t')_A = t_A \circ t_A'$ (where the right hand side is function composition and the left hand side is substitution) □

We conclude this section by pointing out that the same process works for many-sorted algebras. If Σ is an S-sorted operator domain and A is a Σ-algebra, we use the notation, paralleling A^n for Cartesian power, $A^w = A_{s_1} \times \cdots \times A_{s_n}$ when $w = s_1 \cdots s_n$. Also, let $X_w = \{x_{1,s_1}, x_{2,s_2}, \cdots, x_{n,s_n}\}$ be a set of variables, made into an S-indexed family by $X_s = \{x_{i,s}\}$. Then $t \in T_\Sigma(X_w)_s$ is a term of sort s in variables X_w and each such term defines a function $t_A:A^w \longrightarrow A_s$ -- a derived operator of type $\langle w,s \rangle$ on the algebra A.

3. Applications of Initial Many-sorted Algebras

Context-free Grammars.

The most important and general example comes from the notion of semantics for context-free grammars. Let $G = \langle N,\Sigma,P\rangle$ be a context-free grammar with nonterminals N, terminals, Σ ($N \cap \Sigma = \emptyset$), and productions $P \subseteq N \times (N \cup \Sigma)^+$. Let $V = N \cup \Sigma$; for any $w \in V^*$ define $nt(w)$ to be the string of nonterminals in w. Now make G into an N-sorted operator domain by

$$G_{w,A} = \{p \in P \mid p = \langle A,w'\rangle \text{ and } nt(w') = w\}.$$

Thus a production $A_0 \longrightarrow u_0A_1u_1A_2\cdots u_{n-1}A_nu_n$ ($A_i \in N, u_i \in \Sigma^*$) is an operator symbol of type $\langle A_1\cdots A_n,A_0\rangle$.

The initial G-algebra T_G has carriers $T_{G,A}$ which are derivation trees for derivations in G from A.[1]

Now the crucial point of initial algebra semantics is that any G-algebra what-so-ever (a set S_A for each nonterminal A and a function $p_S:S_{A_1}\times\cdots\times S_{A_n} \to S_{A_0}$ for each production p of type $\langle A_1\cdots A_n,A_0\rangle$) is a candidate for the semantics of the context-free language generated by G. That T_G is initial means that the unique homomorphism $h_S:T_G \longrightarrow S$ will assign "meanings" in S to all syntactically well formed phrases of the language, not just to "sentences" generated from some specified start symbol in N.

As an example, make Σ^* into a G-algebra (every carrier is Σ^*): for $p = \langle A_0,u_0A_1u_1A_2\cdots u_{n-1}A_nu_n\rangle$,

$$p(v_1,\cdots,v_n) = u_0v_1u_1v_2\cdots u_{n-1}v_nu_n.$$

Then the unique homomorphism $d:T_G \longrightarrow \Sigma^*$ yields the string derived from the derivation $t \in T_{G,A}$.[2] G might be ambiguous (d not injective) but that doesn't concern us for initial algebra semantics (the abstract syntax point of view) because the parsing problem has been factored out though, of course, not eliminated.[3]

So we see that even "string generated" is an example of initial algebra semantics for context-free grammars. Much more interesting is Knuth's [25] concept of semantics of context-free grammars which generalizes Irons' [24] ideas. In Knuth's synthesized attributes he was really doing what we are proposing here except that his definitions and notation seem to be a bit more complicated than necessary. Translating to our formulation, Knuth constructs a G-algebra S in which each carrier S_A is itself a Cartesian product of sets. The components of that product are called synthesized attributes of A. Then his "semantic rules" define (by the components) an operation of the appropriate type for each production. The semantics of the language is the unique homomorphism $h_S:T_G \longrightarrow S$.[4]

Denotational Semantics.

Scott [53] begins with the motto, "Extend BNF to semantics." In fact that is the "moral" of the previous discussion. We now know that BNF[5] yields an initial many-sorted algebra (syntax), and that semantics arises from an algebra of the same type with "meaning" the unique homomorphism. We view the principal achievement of Scott and Strachey [say 50, 53, 54] as providing mathematical tools for constructing semantic domains (many-sorted algebras with operations corresponding to the productions of a context-free grammar) so that the unique homomorphism is the <u>intended</u> (denotational) meaning of the language.

Scott and Strachey seem to say (e.g., on page 17 of PRG6 [54]) they don't care if the syntax of the language is context-free. ("The last thing we want to be dogmatic about is language.") But their semantics does depend on the context-free character of the source language, because the meaning of a phrase is a function of the meanings of the constituent phrases. ("The semantical definition is syntax directed in that it follows the same order of clauses and transforms each language construct into the intended operations on the meanings of the parts." PRG6 [54], page 12.) This is essentially saying that syntax is context-free and semantics is a homomorphism. Incidentally, our grammars need not be context-free in the strictest sense (e.g., [17]) since we permit infinite schemes of context-free rules. A specific example below will illustrate this point.

The so-called "semantic equations" describing "operations on the meanings of the parts" of productions actually describe an algebra <u>and</u> express that semantics is a homomorphism. For example, ([54], PRG6, page 27):

$$\mathcal{C}[\![\varepsilon \to \gamma_0,\gamma_1]\!] = \lambda\rho\ \underline{\text{cond}}(\mathcal{C}[\![\gamma_0]\!](\rho),\ \mathcal{C}[\![\gamma_1]\!](\rho))*\mathcal{E}[\![\varepsilon]\!].$$

Here $\mathcal{C}$ is the homomorphism component for "commands" and $\mathcal{E}$ for "Boolean expressions." The right-hand side of the equation defines an operation (in terms of <u>cond</u> and $*$) on $\Gamma^2 \times T$, where Γ is the target of $\mathcal{C}$ and T is the target of $\mathcal{E}$.

We illustrate this more clearly with a simple applicative language from Reynolds [45] which we call SAL. Let $X = \{x_1,x_2,\cdots\}$ be a set of variables. In specifying the syntax of SAL we give each rule a name so as to have mnemonics for defining a semantic algebra; those with subscript x represent families, one member for each $x \in X$. There is one nonterminal, <exp>, and all underlined symbols are terminals. Thus variables occur as terminal symbols, just as say, if does.

---

[1] Derivation trees are usually labeled by V; here they are "labeled" by productions. This minor distinction seems to be significant in applications. (See Thatcher [56].)

[2] This is actually a generalization of the frontier function defined in Section 2.

[3] As Schwartz [49] puts it, "We have sufficient confidence in our understanding of syntactic analysis, to be willing to make the outcome of syntactic analysis, namely the syntax tree representation of a program, into a standard starting point for our thinking on program semantics."

[4] We remain perplexed as to how "inherited attributes" fit in or if they should be fit in at all. The main result in Knuth's paper is the existence of an algorithm to test whether a semantic definition with both synthesized and inherited attributes is circular. We don't come close to being able to ask such a question.

[5] Backus Naur Form, a well-known and convenient formulation for context-free syntax.

(v_x) $\langle exp\rangle$:= $\underline{x}$

(cond) $\langle exp\rangle$:= $\underline{\text{if}}$ $\langle exp\rangle$ $\underline{\text{then}}$ $\langle exp\rangle$ $\underline{\text{else}}$ $\langle exp\rangle$

(apply) $\langle exp\rangle$:= $\langle exp\rangle$ $\underline{(}$ $\langle exp\rangle$ $\underline{)}$

(abs_x) $\langle exp\rangle$:= $\underline{\text{lambda}}$ $\underline{x}$ $\underline{(}$ $\langle exp\rangle$ $\underline{)}$

(let_x) $\langle exp\rangle$:= $\underline{\text{let}}$ $\underline{x}$ $\underline{\text{be}}$ $\langle exp\rangle$ $\underline{\text{in}}$ $\langle exp\rangle$

(let-rec_x) $\langle exp\rangle$:= $\underline{\text{let}}$ $\underline{\text{recursive}}$ $\underline{x}$ $\underline{\text{be}}$ $\langle exp\rangle$

From Scott [53] there is a complete lattice V of values satisfying the equation (= is really isomorphism)

$$V = I + B + [V \longrightarrow V],$$

where I and B represent respectively the integers and the Boolean values $0,1$, as lattices with $\perp$ and $\top$ adjoined, and $[V \longrightarrow V]$ is the lattice of continuous functions from V to V.

We wish variables to have values in V. Thus let E be the set of environments, i.e., of all total functions from X to V,

$$E = V^X$$

as a complete lattice with component-wise ordering, $e \sqsubseteq e'$ iff $e(x) \sqsubseteq e'(x)$ in V for all $x \in X$. Finally, meanings of expressions are continuous functions from environments to values,

$$M = [E \longrightarrow V].$$

M becomes a semantic algebra when we define a function on M (or M^2 or M^3) for each production of SAL; e.g., $\underline{\text{let}}_x$ and $\underline{\text{apply}}$ are functions $M^2 \longrightarrow M$ of rank 2; $\underline{\text{cond}}$ is rank 3; $\underline{\text{abs}}_x$ and $\underline{\text{let-rec}}_x$ are rank 1; and, $\underline{v}_x$ is rank 0, a constant.

To carry this out, we define some continuous auxiliary functions (and functionals). We are rather sketchy since this development is what Scott, Strachey and followers do anyway, and our aim here is just to show the relation to the initial algebra framework.

(3.1) $\text{access}_x : E \longrightarrow V$ is the evaluation function on V^X at $x \in X$: $\text{access}_x(e) = e(x)$.

(3.2) $c : V^3 \longrightarrow V$ is some conditional say:

$$c(v_1,v_2,v_3) = \begin{cases} v_2 & \text{if } v_1 = 0 \\ v_3 & \text{if } v_1 = 1 \\ \perp & \text{otherwise.} \end{cases}$$

(3.3) $\text{is-fun}: V \longrightarrow V$ is like a partial identity function on $V \longrightarrow V$, i.e.,

$$\text{is-fun}(v) = \begin{cases} v & \text{if } v \in [V \longrightarrow V] \\ \perp & \text{otherwise.} \end{cases}$$

(3.4) $ap: V^2 \longrightarrow V$ is application:

$$ap(v_1,v_2) = (\text{is-fun}(v_1))(v_2)$$

(3.5) $\text{assign}_x : E \times V \longrightarrow E$ is our friend the assignment operator.

$$\text{assign}_x(e,v) = e' \quad \text{where} \quad e'(y) = \begin{cases} v & \text{if } y = x \\ e(y) & \text{otherwise} \end{cases}$$

All of the above either have been shown to be continuous or can easily be shown to be such. In addition Scott [53] defines:

(3.6) $\text{abstract}: [D \times D' \longrightarrow D''] \to [D \longrightarrow [D' \longrightarrow D'']]$

by $((\text{abstract}(f))(x))(y)) = f(x,y)$ for all $f \in [D \times D' \longrightarrow D'']$, $x \in D$ and $y \in D'$.

Here we also enjoy some notational ambiguities and just write "abstract" instead of "$\text{abstract}_{D,D',D''}$"; and similarly in other cases.

(3.7) The fixed point operator $Y : [D \longrightarrow D] \longrightarrow D$, is continuous.

(3.8) If $m_i : E \longrightarrow V_i$ $(1 \le i \le k)$ are continuous then $[m_1,\cdots,m_k] : E \longrightarrow V_1 \times \cdots \times V_k$ defined by $[m_1,\cdots,m_k](e) = \langle m_1(e),\cdots,m_k(e)\rangle$ is continuous (called target-tupling).

(3.9) Composition of continuous functions is continuous.

Now we make M into a G-algebra (G is the grammar for SAL) by:

$\underline{v}_x = \text{access}_x$

$\underline{\text{cond}}(m_1,m_2,m_3) = c \circ [m_1,m_2,m_3]$

$\underline{\text{apply}}(m_1,m_2) = ap \circ [m_1,m_2]$

$\underline{\text{abs}}_x(m) = \text{abstract}(m \circ \text{assign}_x)$

$\underline{\text{let}}_x(m_1,m_2) = m_2 \circ \text{assign}_x \circ [1_E, m_1]$

$\underline{\text{let-rec}}_x(m) = Y \circ (\text{abstract}(m \circ \text{assign}_x))$

There is no claim for great gains in perspicuity from this specification of the semantic algebra, but our mode of definition does illuminate the places where semantic choices are made. It also gets rid of the need for "dragging along" the variable for environments.

Actually, these function definitions seem to make more sense as diagrams. For example

$$E \xrightarrow{[m_1,m_2,m_3]} V^3 \xrightarrow{c} V$$

defines $\underline{\text{cond}}$. For $\underline{\text{let-rec}}_x$, the most complicated, we have:

$$E \times V \xrightarrow{\text{assign}_x} E \xrightarrow{M} V.$$

So that $\text{abstract}(m \circ \text{assign}_x) : E \longrightarrow [V \longrightarrow V]$ and $Y \circ (\text{abstract}(m \circ \text{assign}_x)) : E \longrightarrow V$ because $Y : [V \longrightarrow V] \longrightarrow V$.

It is intriguing that our initial algebra semantics, the unique homomorphism from T_G (derivation trees for SAL) to M (the algebra with carrier M defined above), appears to be the same as Reynolds', which is described by means of a "meta-circular interpreter." The sense in which his semantics is "interpretive" and the connections with other semantic definitions he presents must await further investigation.

Syntax Directed Translation. We view syntax directed translation as a special case of initial algebra semantics with the "semantic algebra" nearly free. There are two problems: First, we aren't yet sure how best to treat "nondeterministic" translation and therefore don't. Second, we only formulate various concepts, giving neither new results nor simplified proofs. We hope that the reader, faced with these remarkably simple definitions, may be moved to simplify and expand the theoretical development.

Just as generalized sequential machines [17] map Σ^* to Σ'^* (for alphabets Σ and Σ'), syntax directed maps go from T_Σ to $T_{\Sigma'}$ (for operator domains Σ and Σ'). Just as one later restricts a gsm to a subset of Σ^* (e.g., a context-free language), one can later consider a syntax directed map on a subset of T_Σ, say a set of derivation trees.

Thatcher [56] shows that many prior formulations of syntax translation are special cases of a general algebraic formulation. So do we here, but with even greater simplicity, if less intuitive transparency.

Definition. A k-state root to frontier syntax map, from T_Σ into $T_{\Sigma'}$, is the unique homomorphism $h:T_\Sigma \to (T_{\Sigma'})^k$ guaranteed by initiality of T_Σ by making $(T_{\Sigma'})^k$ into a Σ-algebra. The set $[k] = \{1,2,\cdots,k\}$ is the set of states. If $h(t) = <t_1,\cdots,t_k>$ then t_i is the image of t under the syntax map from start state i. □

The standard example (c.f. Rounds [47] and Thatcher [55]) is the simple derivative. Take $\Sigma_0 = \{0,1,x\}$, $\Sigma_2 = \{+,\times\}$ and define a (2-state) syntax map (from T_Σ to T_Σ) by placing the following Σ-structure on $(T_\Sigma)^2$.

$$<t_1',t_1> + <t_2',t_2> = <t_1'+t_2', t_1+t_2>$$

$$<t_1',t_1> \times <t_2',t_2> = <(t_1'\times t_2)+(t_1\times t_2'),\ t_1\times t_2>$$

$$0 = <0,0>$$

$$1 = <0,1>$$

$$x = <1,x>$$

These "equations", especially the last three, are somewhat confusing; of course the symbols $+,\times,0,1,x$ are from Σ and, for example, the last equation says that x in T_Σ^2 names the pair $<1,x>$ where this is the pair of objects names by 1 and x in T_Σ.

Now $h:T_\Sigma \to (T_\Sigma)^2$ yields $h(t) = <t',t>$; the left member of the pair is the (unsimplified) derivative and the right member is just t again.

The definition simplifies and encompasses Thatcher's [56] "root to frontier automaton with output" (called "finite state transformation" in [55]. Intuitively if $t = \sigma(t',t'')$, t_i' is the translation of t' with root (start) state i, and t_i'' is the translation of t'' with state i, then $\sigma(<t_1',\cdots,t_k'>,<t_1'',\cdots,t_k''>)_j$ is the translation of $\sigma(t',t'')$ with state j.

If $k = 1$ and the Σ-structure on $T_{\Sigma'}$ is uniform in the sense that for each $\sigma\epsilon\Sigma_n$ there is a term t_σ in $T_{\Sigma'}(X_n)$ such that for all $t_i\epsilon T_{\Sigma'}$, $\sigma(t_1',\cdots,t_n') = t_\sigma\leftarrow<t_1',\cdots,t_n'>$, then the resulting syntax map is called a homomorphism, generalizing the homomorphisms of ordinary formal language theory. If, in addition, each t_σ is linear (no repetitions of variables), then the resulting maps include the "syntax directed translations" of Lewis and Stearnes [28] and the syntax directed translation schemes of Aho and Ullman [1]. If $k \geq 1$ and the Σ-structure on $(T_{\Sigma'})^k$ is uniform in the above sense, our syntax maps include the generalized syntax directed translation schemes of Aho and Ullman [2].

Outside tree automata theory (i.e., outside Baker [3], Magidor and Moran [35], and Thatcher [56], etc.) we don't seem to find frontier to root translation as much as root to frontier maps, except when they coincide; $k = 1$. But a frontier to root translation (from T_Σ to $T_{\Sigma'}$) is determined by giving $[k] \times T_{\Sigma'}$, a Σ-structure ($[k]$ is again the set of states); then the unique homomorphism $h:T_\Sigma \to [k] \times T_{\Sigma'}$ yields a pair $<i,t'>$, where i is the terminating state (which could be designated final or nonfinal) and t' is the translated expression.

For example, a tree automaton (Doner [14], Thatcher and Wright [57]) with transition function $M:\Sigma \times [k]^n \to [k]$ is imitated as a frontier to root translation by making $[k] \times T_\Sigma$ into a Σ-algebra with $\sigma(<i_1,t_1>,\cdots,<i_n,t_n>) = <M(\sigma,i_1,\cdots,i_n),\ \sigma(t_1\cdots t_n)>$; the second component is the identity.

Much further investigation is needed in relating frontier to root and root to frontier translations, and finding realistic applications.

Systems of Equations. If A is a Σ-algebra we make $\mathcal{P}A$ ($\mathcal{P}A$ is the set of subsets of A) into a Σ-algebra with the "subset construction:"

(0) $\sigma_{\mathcal{P}A} = \{\sigma_A\}$, for $\sigma\epsilon\Sigma_0$

(1) For $n>0$, $\sigma\epsilon\Sigma_n$ and $u_1,\cdots,u_n\epsilon\mathcal{P}A$

$$\sigma_{\mathcal{P}A}(u_1,\cdots,u_n) = \{\sigma_A(t_1,\cdots,t_n) \mid t_i\epsilon u_i, 1\leq i\leq n\}$$

Associated with each $t\epsilon T_\Sigma(x_n)$ is a derived operator $t_{\mathcal{P}A}:(\mathcal{P}A)^n \to \mathcal{P}A$; extending this idea to sets of terms, if $r \subseteq T_\Sigma(x_n)$, define $r_{\mathcal{P}A}(u_1,\cdots,u_n) = \bigcup\{t_{\mathcal{P}A}(u_1,\cdots,u_n) \mid t\epsilon r\}$. $E:X_n \to \mathcal{P}T_\Sigma(X_n)$ is called a system of equations which determines a function $E_{\mathcal{P}A}:(\mathcal{P}A)^n \to (\mathcal{P}A)^n$, by $E_{\mathcal{P}A} = [E(x_1)_{\mathcal{P}A},\cdots,E(x_n)_{\mathcal{P}A}]$ (target-tupling), called the system function by Mezei and Wright [39] who show that $E_{\mathcal{P}A}$ is ω-continuous, i.e., preserves least upper bounds of chains in $(\mathcal{P}A)^n$. Therefore $E_{\mathcal{P}A}$ has a minimum fixed point, called its solution say $<s_1,\cdots,s_n>\epsilon(\mathcal{P}A)^n$; in fact, $<s_1,\cdots,s_n> = \bigcup E^n_{\mathcal{P}A}(\emptyset,\cdots,\emptyset)$. A subset of A is equational iff it is a component of a solution of a finite system of equations. The entire theory of equational sets (c.f. Mezei and Wright [39], Eilenberg and Wright [15], Blikle [7], and Nivat [41]) flows from these definitions, from initiality (to get derived operators) and the Tarski fixed-point theorem (to get solutions).

Note that equational subsets of A were equational elements of the algebra $\mathcal{P}A$. This suggests that we consider Σ-algebras whose carriers are ω-complete upper semi-lattices (least upper bounds of finite sets and ω-chains exist) and whose operations are ω-continuous (preserve least upper bounds of ω-chains). For such an algebra A, when E is finite, the system function E_A can be defined as above and the solution (minimum fixed point) is an n-tuple in A; $a \epsilon A$ is equational just in case it is a component of the solution of a finite system of equations. This is the direction we are heading in Section 4.

4. Continuous Algebras

First, some preliminaries on posets, in particular the poset of partial functions. The terms "complete" and "continuous" are getting confusing. Here we use a terminology that eliminates the existing confusion and facilitates general theorems about varied forms of completeness and continuity.

Following Scott, we use $\sqsubseteq$ for the order in a poset P, $\sqcup$ for binary (or finite) least upper bounds, and $\bigsqcup$ for least upper bounds of arbitrary sets. We assume all posets have a minimum element denoted $\bot$ ("bottom"), while $\top$ ("top") denotes the maximum element if it exists. We use the symbols $\sqcup$ and $\bigsqcup$, etc. to modify "complete" and "continuous", and as such they describe subsets of P. For example, a $\bigsqcup$-complete (read "mug-complete") poset has least upper bounds of all subsets of P and thus is a "complete lattice"; a $\bigsqcup$-continuous function preserves all least upper bounds that exist. In tabular form:

$\sqcup$	finite subsets of P
ω	ω-chains in P
ℓ	linearly ordered subsets of P (chains)
Δ	directed subsets of P
$\bigsqcup$	all subsets.

For each symbol X above, a function $f:P \longrightarrow P'$ is X-continuous iff it preserves all least upper bounds of X-sets that exist in P; a poset P is X-complete iff all X-sets have least upper bounds in P, a poset is $\dot{X}$-complete iff all least upper bounds of bounded X-sets exist in P. P is X-bounded iff every X-set has an upper bound in P; and a subset S of P is X-bounded [X-complete] in P iff every X-set in S has a [least] upper bound in P.

Examples of the use of these conventions are:

(4.1) A function $f:P \longrightarrow P'$ is $\dot{X}$-continuous iff it is X-continuous.

(4.2) A poset is ℓ-complete iff it is Δ-complete (c.f. [12] and [38]).

(4.3) Let $P^\top$ be the poset P with $\top$ adjoined, (i.e. $\top \notin P$ and $p \sqsubset \top$ for all $p \epsilon P$). Then $P^\top$ is $\bigsqcup$-complete (a complete lattice) iff P is $\dot{\bigsqcup}$-complete; and, $P^\top$ is $\sqcup$-complete (an upper-semi lattice) iff P is $\dot{\sqcup}$-complete. (This was observed by J. B. Wright as we were considering the connections between the present approach and Scott's lattice orientation.)

(4.4) Our modifier symbols are ordered ($\omega \subseteq \ell \subseteq \Delta \subseteq \bigsqcup$); so P $\bigsqcup$-complete implies P Δ-complete implies P ℓ-complete implies P ω-complete.

(4.5) $S \subseteq P$ is directed iff S is $\sqcup$-bounded (which) implies $S \neq \emptyset$.

(4.6) For each of our X's, f X-continuous implies f monotonic, because $p_0 \sqsubseteq p_1$ in P implies $\{p_0,p_1\}$ is an X-set having a least upper bound (namely p_1). Thus $f(p_0 \sqcup p_1) = f(p_1) = f(p_0) \sqcup f(p_1)$ so $f(p_0) \sqsubseteq f(p_1)$.

For sets A,B, let $[A \mathrel{-\!\circ\!\!\rightarrow} B]$ be the poset of partial functions from A to B. Since we have fixed source and target, the elements of $[A \mathrel{-\!\circ\!\!\rightarrow} B]$ correspond to functional subsets of $A \times B$, i.e., if $\langle a,b\rangle \epsilon f$ and $\langle a,b'\rangle \epsilon f$, then $b = b'$. The order relation on $[A \mathrel{-\!\circ\!\!\rightarrow} B]$ is simply set inclusion, and least upper bound (when it exists) is set union; it exists iff the set union is a function. For[1] $f:A \mathrel{-\!\circ\!\!\rightarrow} B$ and $C \subseteq A$ $f\rceil C = \{\langle a,b\rangle \mid a\epsilon C \text{ and } \langle a,b\rangle \epsilon f\}$ is the restriction of f to C; $def(f) = \{a \mid \langle a,b\rangle \epsilon f\}$ is the domain of definition of f. The following are well know facts about $[A \mathrel{-\!\circ\!\!\rightarrow} B]$:

(4.7) $[A \mathrel{-\!\circ\!\!\rightarrow} B]$ is Δ-complete and $\dot{\bigsqcup}$-complete.

(4.8) If $\langle \alpha_i \rangle_{i<\omega}$ is a chain of subsets of A such that $\bigcup \alpha_i \supseteq def(f)$ then $\bigsqcup f\rceil\alpha_i = f$.

We discuss initial one-sorted "continuous algebras" in detail first; the definition for the many-sorted case will be given later.

Let $\langle \Sigma_i \rangle_{i\epsilon\omega}$ be a one-sorted operator domain (i.e., ranked alphabet). A Σ-tree is a partial function $t:\omega^* \mathrel{-\!\circ\!\!\rightarrow} \Sigma$ such that, for all $u\epsilon\omega^*$ and $i\epsilon\omega$,

(a) $ui\epsilon\, def(t)$ implies $u\epsilon\, def(t)$;

(b) $ui\epsilon\, def(t)$ implies $t(u)\epsilon\, \Sigma_n$ and $i<n$.

(Note that $ui\epsilon\, def(t)$ does not imply $uk\epsilon\, def(t)$ for any other k.)

Let CT_Σ denote the set of Σ-trees, and let F_Σ denote the subset of CT_Σ containing the finite Σ-trees ($def(t)$ is finite) and let $\bot$ (in both these sets) be the totally undefined function. The set union of functions satisfying (a) and (b) will also satisfy (a) and (b); so least upper bounds in $[\omega^* \mathrel{-\!\circ\!\!\rightarrow} \Sigma]$ are least upper bounds in CT_Σ. Thus from (4.7):

Proposition 4.1. CT_Σ is a Δ-complete and $\dot{\bigsqcup}$-complete poset. □

Let $\omega^{(n)}$ be the subset of ω^* containing strings of length less than n; then $\bigcup_n \omega^{(n)} = \omega^*$. And taking $t^{(n)} = t\rceil\omega^{(n)}$, then $t = \bigsqcup t^{(n)}$. By condition (b), each $t^{(n)}$ is finite (even though $\omega^{(n)}$ is infinite).

Proposition 4.2. For any $t\epsilon CT_\Sigma$, $t = \bigsqcup t^{(n)}$ and $t^{(n)} \epsilon F_\Sigma$. □

Now make CT_Σ into a Σ-algebra by:

(0) For $\sigma\epsilon\Sigma_0$, let $\sigma_{CT} = \{\langle\lambda,\sigma\rangle\}$

(1) For $\sigma\epsilon\Sigma_n$, $n>0$ and $t_1,\cdots,t_n \epsilon CT_\Sigma$, let

$$\sigma_{CT}(t_1,\cdots,t_n) = \{\langle\lambda,\sigma\rangle\} \cup \bigcup_{i<n}\{\langle iu,\sigma'\rangle \mid \langle u,\sigma'\rangle \epsilon t_{i+1}\}$$

[1] $f:A \mathrel{-\!\circ\!\!\rightarrow} B$ designates a function A to B with a possible "hole".

If $t_1,\cdots,t_n \in CT_\Sigma$ then $\sigma_{CT}(t_1,\cdots,t_n) \in CT_\Sigma$; in fact if the t_i's are finite then so is $\sigma_{CT}(t_1,\cdots,t_n)$. So we have given F_Σ Σ-structure as well.

The ordering on CT_Σ is related to this algebraic structure in the following way.

Proposition 4.3. In CT_Σ, $t \sqsubseteq t'$ iff $t = \perp$ or $t = t' = \sigma_{CT} \in \Sigma_0$ or there exist $\sigma \in \Sigma_n$, $t_1,\cdots,t_n$ and $t'_1,\cdots,t'_n$ such that $t = \sigma_{CT}(t_1,\cdots,t_n)$, $t' = \sigma_{CT}(t'_1,\cdots,t'_n)$ and $t_i \sqsubseteq t'_i$, $1 \le i \le n$. □

Proposition 4.4. F_Σ is the free Σ-algebra on one generator, $\perp$, i.e., it is an initial $\Sigma(\perp)$-algebra. □

For a proof, either show there is a unique $\Sigma(\perp)$-homomorphism $h_A: F_\Sigma \longrightarrow A$ for any other $\Sigma(\perp)$-algebra A, or show that F_Σ is isomorphic to (say) the usual algebra of $\Sigma(\perp)$-expressions and then use Proposition 1.1. Either way the result depends on the uniqueness of decomposition in F_Σ; either $t = \perp$, $t \in \Sigma_0$ or there exist unique $n>0$, $\sigma \in \Sigma_n$ and $t_1,\cdots,t_n \in F_\Sigma$ such that $t = \sigma_{CT}(t_1,\cdots,t_n)$.

Proposition 4.5. The operations of CT_Σ are $\bigsqcup$-continuous. □

Because everything in the proof of 4.5 is finite when restricted to F_Σ, the same proof yields:

Proposition 4.6. The operations of F_Σ are $\bigsqcup$-continuous. □

Let an ordered Σ-algebra be a Σ-algebra whose carrier is a poset with $\perp$ and whose operations are monotonic. A homomorphism of ordered Σ-algebras is a monotonic Σ-homomorphism preserving $\perp$. Let $\underline{\underline{Palg}}_\Sigma$ be the class of ordered Σ-algebras. Since the operations of F_Σ are monotonic (Propositions 4.6 and (4.6)), F_Σ is an ordered Σ-algebra. In fact,

Proposition 4.7. F_Σ is initial in $\underline{\underline{Palg}}_\Sigma$. □

A Σ-algebra is ω-continuous iff its carrier is ω-complete with $\perp$ and its operations are ω-continuous. By Propositions 4.1 and 4.5 CT_Σ is an ω-continuous algebra.

Theorem 4.8. CT_Σ is initial in the class of ω-continuous algebras with Δ-continuous Σ-homomorphisms. □

The main idea of the proof of Theorem 4.8 is that every ω-continuous Σ-algebra A is in $\underline{\underline{Palg}}_\Sigma$ so there is a unique homomorphism $h_0: F_\Sigma \longrightarrow A$ which has a unique extension using Proposition 4.2: $\overline{h}_0(t) = \bigsqcup h_0(t^{(n)})$ which is proved to be a homomorphism and to be Δ-continuous.

We say an algebra is Δ-continuous iff its carrier is Δ-complete and its operations are Δ-continuous. Let $\underline{\underline{\Delta alg}}_\Sigma$ be the class of Δ-continuous Σ-algebras with Δ-continuous Σ-homomorphisms. Since CT_Σ is in this class which is a subclass of that described in Theorem 4.8 we have

Corollary 4.9. CT_Σ is initial in class $\underline{\underline{\Delta alg}}_\Sigma$ of Δ-continuous Σ-algebras with Δ-continuous homomorphisms. □

There are confusingly many initiality results for CT_Σ. For example, the following was proved in collaboration J. B. Wright.

Theorem 4.10. CT_Σ is initial in the class of Σ-algebras with ω-complete carriers, operations that are both ω-continuous and $\sqcup$-continuous and homomorphisms that are $\bigsqcup$-continuous Σ-homomorphisms. □

Since the following is a subclass of that in Theorem 4.10 and contains CT_Σ we have

Corollary 4.11. CT_Σ is initial in the class of Σ-algebras with Δ-complete carriers and $\bigsqcup$-continuous operations and homomorphisms. □

Having established the existence of an initial (say ω-) continuous algebra, we can, as in Section 2, consider its derived operators. Let A be an ω-continuous algebra and $a: X_n \longrightarrow A$, then $\overline{a}: CT_\Sigma(X_n) \longrightarrow A$ is the unique homomorphism determined by making A into a continuous $\Sigma(X_n)$-algebra. As before $t_A(a) = \overline{a}(t)$. One can check that if $a \sqsubseteq a'$ $(a(x_j) \sqsubseteq a'(x_j)$, $1 \le j \le n)$ then $\overline{a} \sqsubseteq \overline{a}'$ $(\overline{a}(t) \sqsubseteq \overline{a}'(t)$, $t \in CT_\Sigma(X_n))$.

Proposition 4.12. For each n, each $t \in T_\Sigma(X_n)$ and each ω-continuous Σ-algebra A, $t_A: A^n \longrightarrow A$ is ω-continuous. □

As in Section 2, substitution is defined as a special case of derived operators, and it is associative; in fact substitution is itself Δ-continuous, and this depends only on initiality of CT_Σ. (Details will appear elsewhere.)

The promised S-sorted initial continuous Σ-algebra is a simple modification of the one-sorted case. A Σ-tree of sort $s \in S$ is a partial function $t: \omega^* \longrightarrow \Sigma$ such that

(0) If $\lambda \in$ def(t) then $t(\lambda)$ has sort s.

(1) If $w \in \omega^*$, $i \in \omega$ and $wi \in$ def(t) then

  (a) $w \in$ def(t)

  (b) t(w) has arity $s_1 \cdots s_n$ with $i<n$ and

  (c) t(wi) has sort s_i.

Then CT_Σ has carriers $CT_{\Sigma,s}$ consisting of all s-sorted Σ-trees (including $\perp$). It gets Σ-structure just as in the one-sorted case and we have

Proposition 4.13. CT_Σ is initial in the class of ω-continuous Σ-algebras with Δ-continuous Σ-homomorphisms. □

5. Applications of Initial Continuous Algebras

The initiality of CT_Σ has important consequences in the theory of computation. As the initiality of T_Σ clarifies and simplifies what we usually think of as tree manipulation, CT_Σ gives us sure mathematical footing for dealing with infinite trees which arise in the semantics of flow diagrams, recursive schemes, and data structures.

CT_Σ has advantages both in its abstract character (simply an initial algebra in the class of, say Δ-continuous Σ-algebras) and in concreteness (we know one representative of CT_Σ as (infinite) Σ-trees). The abstract formulation makes it easy to define derived operations and solve equations in Δ-continuous Σ-algebras, in fact, this is easier than the conventional case where the subset construction is required. The concrete representation gives a clear understanding of what kind of objects make up CT_Σ, for example, our formulation for the lattice of flow diagrams is not only much simpler, but intuitively more convincing.

We know of three papers containing general results that we believe to be intimately related to ours on initial continuous algebras: Wand's [62] free μ-clones, Nivat's [41] algebra of schematic languages, and Bloom and Elgot's [8] free iterative algebraic theories. Each of these has a markedly different approach, supporting, at least for us, the contention that these general ideas are truly significant for the theory of computation.

Because of the radically different approaches, because the first two references are somewhat informal or sketchy (admittedly in Nivat's case) and because initiality of CT_Σ is new, we are not able, at this time, to completely describe the connections between the various results. As we procede in this section we try to point out where the connections are to be made.

As far as we know, our main technical results (Theorems 4.8 and 4.10) are new, but independent of the novelty of those theorems, we hope the reader will find that the development is much crisper when viewed as initial algebra semantics.

In our discussion and examples below we focus on the class $\underline{\Delta alg}_\Sigma$ of Δ-continuous Σ-algebras with Δ-continuous homomorphisms, that is, we rely only on Corollary 4.9, stating that CT_Σ is initial in $\underline{\Delta alg}_\Sigma$. What we say applies to the other classes of algebras mentioned in Theorems 4.8, 4.10 and Corollary 4.11, but at this time we don't see the importance of those applications.

Systems of (Regular) Equations.

In Section 3 we showed how initial algebra semantics was involved in solving equations expressed in the initial many-sorted Σ-algebra. That pervasive phrase, "make pA into a Σ-algebra by..." was used to define the derived operator $t_{pA}:(pA)^n \longrightarrow (pA)$ for each $t \epsilon T_\Sigma(X_n)$. A rather unnatural step follows: extending derived operators to sets of terms by taking unions of values. For algebras in $\underline{\Delta alg}_\Sigma$, this isn't necessary; they have order structure so equations expressed in CT_Σ can be solved directly in any Δ-continuous Σ-algebra A.

A <u>system of</u> n <u>equations expressed in</u> CT_Σ is a function $E:X_n \longrightarrow CT_\Sigma(X_n)$. This simple definition replaces the informal one: "...a sequence or set of the form $x_i = t_i$ where t_i is a term in n variables, $1 \le i \le n$." To read our system that way, write $x_i = E(x_i)$; $E(x_i)$ is the <u>right-hand side</u> of the i^{th} equation.

For any Δ-continuous algebra A, $E_A:A^n \longrightarrow A^n$ is the derived operator of E over A; recall from Section 2 that $(E_A(a))_i = E(x_i)_A(a) = \overline{a}(E(x_i))$ for all $a \epsilon A^n$ and $i \le n$. By Proposition 4.12, each component of E_A $(E(x_i)_A:A^n \longrightarrow A)$ is Δ-continuous and so is their target-tuple, $E_A = [E(x_1)_A,\cdots,E(x_n)_A]:A^n \longrightarrow A^n$, by (3.8).

Thus E_A has a minimum fixed-point which we denote by $|E_A| \epsilon A^n$ and call the <u>solution</u> of E over A. We know $|E_A| = \bigsqcup_{k \epsilon \omega} E_A^k(\perp,\cdots,\perp)$; $E_A(|E_A|) = |E_A|$; and if $E_A(a) = a$ then $|E_A| \sqsubseteq a$ for all $a \epsilon A^n$. For the special case $A = CT_\Sigma$, write $|E|$ for the solution of E over CT_Σ.

Familiar results from the theory of equational sets follow neatly in this generalized setting. Recall $h_A:CT_\Sigma \longrightarrow A$ is the unique Σ-homomorphism from CT_Σ to the Δ-continuous Σ-algebra A. Then write $|E|_A$ for $h_A|E|$

<u>Proposition 5.1</u>. For any system $E:X_n \longrightarrow CT_\Sigma(X_n)$ of equations and any Δ-continuous Σ-algebra A, $|E|_A = |E_A|$. $\square$

This result, familiar from Mezei and Wright [39, Theorem 5.5] says that one can solve equations in the initial algebra and interpret the solution, or interpret the equations and obtain a solution. Either way gives the same answer.

Call a system $E:X_n \longrightarrow CT_\Sigma(X_n)$, <u>ideal</u> (c.f. Elgot [16]) iff each $E(x_i)$ is nontrivial, i.e., for some m, $\sigma \epsilon \Sigma_m$ and $t_1,\cdots,t_m \epsilon CT_\Sigma(X_n)$, $E(x_i) = \sigma_{CT}(t_1,\cdots,t_m)$. The following result was obtained in collaboration with J. B. Wright.

<u>Proposition 5.2</u>. Solutions of ideal systems of equations over CT_Σ are <u>unique</u>. $\square$

This rather surprising result seems to say the order relation on CT_Σ is somewhat immaterial once used to obtain solutions, suggesting a strong connection with Elgot's [16] iterative algebraic theories, to be mentioned again below.

A system E of equations is <u>finite</u> if each right-hand side is finite, i.e., $E:X_n \longrightarrow F_\Sigma(X_n)$. Now E_A can be defined over any ordered algebra A (Proposition 4.7) and $a \epsilon A^n$ is the <u>solution</u> of E over A if a is the minimum fixed point of E_A.

An ordered algebra A is <u>equationally complete</u> if every <u>finite</u> system of equations has a solution over A. Obviously, every Δ-continuous (even ω-continuous) algebra is equationally complete -- in fact, complete with respect to arbitrary systems of equations -- but an ordered Σ-algebra can be equationally complete without being ω-complete as the following important case illustrates.

Let R_Σ denote the set of equational elements of CT_Σ, i.e., $R_\Sigma = \{|E|_i \; | E:X_n \longrightarrow F_\Sigma(X_n)$ and $1 \le i \le n\}$.

The "R" in R_Σ stands for "recognizable" or "regular" as applied to sets of finite trees in Doner

[14] and Thatcher and Wright [57]. (Scott [51] calls F_Σ rational, R_Σ algebraic and $CT_\Sigma - R_\Sigma$ transcendental for the appropriate Σ discussed below.) The following two propositions are indicative of the connection.

Proposition 5.3. For each $t \in R_\Sigma$ there exists a recognizable subset R_t of F_Σ such that $t = \bigsqcup R_t$. □

Proposition 5.4. In the concrete construction of CT_Σ, if $t \in R_\Sigma$ then for each $\sigma \in \Sigma$ $t^{-1}(\sigma) \subseteq \omega^*$ is a regular subset of $\{0,1,\cdots,k\}^*$ for some k. □

The recognizable subsets of F_Σ (viewed as the initial $\Sigma(\bot)$ algebra) which are bounded (with respect to the order on F_Σ) properly include the schematic languages of Nivat [41] defined by systems of equation (in the sense of Section 3) where each right-hand side is of the form $\{t,\bot\}$ with $t \in F_\Sigma(X_n)$. Corresponding to each schematic language $L \subseteq F_\Sigma$ is $\bigsqcup L \in R_\Sigma$, and every element of R_Σ is obtainable this way.

Proposition 5.5. R_Σ is the smallest equationally complete subalgebra of CT_Σ. □

By definition, R_Σ is equationally complete; showing it is a subalgebra is essentially what Nivat [41] does when he proves his set of schematic languages forms a Σ-algebra.

The Lattice of Flow Diagrams.

Although we can't top Scott's prose, we can offer simple and concise mathematical alternatives to his lattice of flow diagrams [51]. Comparison is only slightly encumbered by the fact that Scott uses lattices ($\bigsqcup$-complete posets) whereas we prefer Δ-complete (or even ω-complete) posets. Gordon [22] calls Δ-complete posets "semi-domains" and argues the advantages of semi-domains over complete lattices in the theory of computation; calling them "complete posets" (and appealing to 4.2), Lewis and Rosen [27, Part 2] do the same. But it matters little here since any CT_Σ is $\bigsqcup^{\bullet}$-complete (Proposition 4.1) and $CT_\Sigma^\top$ is a complete lattice (4.3). Thus we can pretend that Scott was actually using Δ-complete posets.

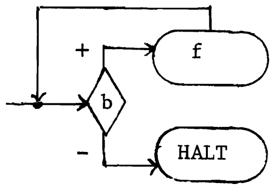
We describe three alternatives for Scott's lattice of flow diagrams; each is simply CT_Σ for a natural choice of Σ; the first two are different in interesting ways -- the third differs from the second trivially and is included for completeness, while providing an example of an initial many-sorted Δ-continuous Σ-algebra.

For all three, we emphasize that the algebraic approach (as opposed to the lattice approach) leads directly to algebras of flow diagrams, rather than ordered sets of flow diagrams. (Nivat [41] stresses this point and it seems clear that Scott also saw this as important [51, Section 4].) It is hoped that this avenue will put us in a better position to tackle some of the questions concerning equivalence of flow diagrams raised by Scott [51, Sections 8 and 9].

Recall from [51], I is a function symbol denoting the identity operation; F is a set of function symbols (operations occurring in boxes of a flow diagram); and B is a set of predicate symbols (occurring as tests).

Let $\Sigma = \langle\{I\}, F, B\rangle$ ($\Sigma_0 = \{I\}$, $\Sigma_1 = F$, $\Sigma_2 = B$, $\Sigma_k = \emptyset$, $k > 2$). $CT_\Sigma^\top$ is a lattice of flow diagrams, i.e., every (nonrecursive) flow diagram is represented by its unfolding in $CT_\Sigma^\top$ but it doesn't quite have either the algebraic, or order-structure of Scott's lattice. It is interesting in that the equational elements (R_Σ) represent the flow diagrams. The informal idea, to be developed elsewhere, is to take a variable x_i for each node i of a flow diagram and an equation $x_i = b(x_j,x_k)$, $x_i = f(x_j)$ or $x_i = I$ if node i is a b-test, an f application or a halt, respectively.

For example, the while loop is the solution to the system $\langle x_1 = b(x_2,x_3),\ x_2 = f(x_1),\ x_3 = I\rangle$ which can be collapsed to $x_1 = b(f(x_1), I))$ and written $x_1 = (b \rightarrow (f; x_1), I)$ by Scott.



The element of R_Σ which is the component corresponding to the start node is the unfolded version of the given flow diagram. Conversely, any finite system of equations or CT_Σ can be reduced to "primitive" equations (much as in Mezei and Wright [39] and, closer to the current context, Wand [62, Theorem 2]) of the above form, from which a standard flow diagram can be retrieved.

Wand [62] takes Scott's construction "trivally modified" as a starting point calling $\underline{E}(\Sigma)$ the lattice of flow diagrams with operator symbols Σ. He says this follows Elgot's [16] idea of handling tests (rank 2 or greater) and operations (rank 1) uniformly. As far as we can tell, his $\underline{E}(\Sigma)$ must be $CT_\Sigma^\top$; the "finite diagrams", $F_\Sigma(X_n)$ with "return codes" $\{x_1,\cdots,x_n\}$ (also paralleling Elgot [16] because of the n "exits"), and $R_\Sigma(X_n)$ (equational elements of $CT_{\Sigma(X_n)}$) the "loop-representable" flow diagrams. All this is quite informed in [62] and misses a subtle but important distinction brought out by looking at a second formulation for the lattice of flow diagrams.

One can define context-free sets over initial algebras (Rounds [47]) and this applieds to initial continuous algebras as well (see below). Alternatively, as Maibaum [36,37], Turner [58], and Wand [63] have observed, one can make the function symbols zero-ary and introduce a symbol for composition. Then the context-free sets become equational in this modified algebra.

In that spirit, let $\Sigma' = \langle F \cup \{I\}, \emptyset, B \cup \{;\}\rangle$. Again every completely defined tree program (over F and B) is represented in $CT_{\Sigma'}$. This time the algebraic structure corresponds to what Scott sought: every element of $CT_{\Sigma'}$ is either I, an element of F, d;d' or b(d,d') (written $b \rightarrow d,d'$), for diagrams d and d'.

But in this algebra, the equational elements correspond to unfoldings of recursive flow diagrams. For example, the solution to the equation $x_1 = b(f;(x_1;f'),I)$, which seemed to mystify Scott (Figure 21 [51]), is the unfolding of the recursive flow diagram x_1 : If b then (f;call x_1;f') else halt.

Nivat's [41] program schemes over $<F \cup \{I\}, B>$ can be reformulated in this setting, being systems of equations $E:X_n \longrightarrow F_{\Sigma'}(X_n)$ which are primitive (much as mentioned above: each $E(x_i)$ is $f\epsilon F$, I, $(\alpha_1;\alpha_2)$, or $(b \longrightarrow \alpha_1,\alpha_2)$, $\alpha_i \in (F \cup \{I\} \cup X_n)$ and connected ($x_1 < x_i$ for $1 < i \le n$, where $<$ is the transitive closure of x_i "calls" x_j, i.e., x_j occurs in $E(x_i)$).

The program schemes form a Σ'-algebra say $PS_{\Sigma'}$: one defines $E;E'$ and $b \longrightarrow E,E'$ as needed in the proof of Proposition 5.5. The function $E \longmapsto |E|_1$ is a homomorphism from $PS_{\Sigma'}$ to $R_{\Sigma'}$, but of course many program schemes give rise to the same "unfolding" in R_Σ.

For our last formulation of a lattice of flow diagrams, let Σ'' be a {b,d}-sorted operator domain with $\Sigma''_{<\lambda,b>} = B$, $\Sigma''_{<\lambda,d>} = \{I\} \cup F$, $\Sigma''_{<dd,d>} = \{;\}$ and $\Sigma''_{<bdd,d>} = \{\to\}$. Then $CT_{\Sigma'',b}$ contains only $B \cup \{\bot\}$, but $CT^T_{\Sigma'',d}$ is isomorphic (in terms of order and algebra) to Scott's lattice E of flow-diagrams. (This corresponds to what would be done in [36, 37, 58, 63], making all the function symbols of $<\{I\}, F, B>$ into zero-ary symbols; the "composition symbols" are ";" and "$\to$".) But all that has been added are diagrams such as $(\bot \longrightarrow d,d')$ which seem to be of little interest at this point. Because there are no operations of sort b, equations in $CT_{\Sigma''}$ yield the same solutions as those in the previous formulation.

Semantics of Flow Diagrams.

Theorem 4.8 says that any Δ-continuous Σ-algebra is a possible semantic algebra for CT_Σ. We look at the second formulation for the lattice of flow diagrams, $\Sigma' = <F \cup \{I\}, \emptyset, B \cup \{;\}>$, and take, as Scott does, the Δ-complete (4.7) poset $[S \mathrel{-\!\circ\!\!\to} S]$ of "partial state transformations" as carrier of a semantic algebra ambiguously denoted S. Given (Scott's [51]) interpretations of the function symbols $\mathscr{F}:F \longrightarrow [S \mathrel{-\!\circ\!\!\to} S]$ and symbols, $\mathscr{B}:B \longrightarrow [S \mathrel{-\!\circ\!\!\to} \{0,1\}]$. Make $[S \mathrel{-\!\circ\!\!\to} S]$ into a Σ'-algebra by: $I_S = 1_S$; $f_S = \mathscr{F}(f)$; $b_S(\sigma,\sigma') = c\circ[\mathscr{B}(b),\sigma,\sigma']$; and $;_S(\sigma,\sigma') = \sigma'\circ\sigma$. These operations ($b_S$ and $;_S$) are Δ-continuous (c.f., 3.2, 3.8, 3.9): initiality of $CT_{\Sigma'}$ gives a unique homomorphism $\mathscr{V}_S:CT_{\Sigma'} \longrightarrow S$ which is Scott's [51] semantics of flow diagrams. Note, however, that $\mathscr{F}$ and $\mathscr{B}$ uniquely determine $\mathscr{V}_S$ only insofar as the intended meanings of tests and composition are fixed.

To get the intended semantics of flow diagrams when expressed in CT_Σ ($\Sigma = <\{I\}, F, B>$) it appears automatic that we appeal to continuations introduced by C. Wadsworth and L. Morris (reported in Reynolds [45]). Taking the same carrier for the semantic algebra, operations ($f\epsilon F$) are of type $[S \mathrel{-\!\circ\!\!\to} S] \longrightarrow [S \mathrel{-\!\circ\!\!\to} S]$, i.e., they are continuations. The extent to which continuations are generally treated from an initial algebra point of view and the relationship between continuation semantics for CT_Σ and the semantics described above for $CT_{\Sigma'}$, are interesting subjects that must await further investigation.

Systems of (Regular) Equations with Parameters.

Completely parallel to equational elements of an algebra, we can define equational functions much in the spirit of Wagner [60,61] and Wand [62].

A function[1] $E:X_n \longrightarrow CT_{\Sigma(X_p)}(X_n)$ is a system of n-equations with p parameters, $\{x_1,\cdots,x_p\}$. For any algebra A in $\underline{\Delta alg}_\Sigma$ and for each $a\epsilon A^p$, make A into an $\Sigma(X_p)$-algebra A(a) by having x_i name a_i. The derived operator over A(a) is then $E_{A(a)}:A^n \longrightarrow A^n$ which has a minimum fixed point, $|E_{A(a)}|$. The function $|E_A|:A^p \longrightarrow A^n$ determined by system E is defined by $|E_A|(a) = |E_{A(a)}|$.

The equation of Proposition 5.1 holds true for equational functions as well. For E as given above, $|E|:X_n \longrightarrow CT_{\Sigma(X_p)}$. (Solving in $CT_{\Sigma(X_p)}$) so $|E|_A:A^p \longrightarrow A^n$ and we have

Proposition 5.1'. For a system of equations $E:X_n \longrightarrow CT_{\Sigma(X_p)}(X_n)$ with p parameters and any Δ-continuous Σ-algebra A, $|E_A| = |E|_A:A^p \longrightarrow A^n$. □

Solving finite systems of equations with parameters, $E:X_n \longrightarrow F_{\Sigma(X_p)}(X_n)$ over $CT_{\Sigma(X_p)}$ yields $|E|\epsilon R^n_{\Sigma(X_p)}$ by definition. But as sets $R_{\Sigma(X_p)} = R_\Sigma(X_p)$, thus we can talk about solving systems of equations whose right-hand sides are in $R_{\Sigma(X_p)}$; these give nothing new (Nivat [41] proves this result; within his schematic language formulation, it is a consequence of the substitution theorem for context-free languages [17]).

Proposition 5.6. Systems of equations whose right-hand sides are in $R_\Sigma(X_p)$ have solutions in R_Σ. □

We believe the principal connections with Wand [62] and Bloom and Elgot [8] are to be found here. First we conjecture that $<R_\Sigma(X_p)>_{p\epsilon\omega}$ is the free μ-clone generated by Σ, the existence of which is Wand's principal result. Next, let $\overline{R}_\Sigma(X_p)$ be the set of maximal elements of $R_\Sigma(X_p)$. (In the concrete construction of $CT_{\Sigma(X_p)}$, t is maximal iff $t(u)\epsilon\Sigma_k$ implies $ui\epsilon\ def(t)$, $0 \le i < k$.) The algebraic structure of $<\overline{R}_\Sigma(X_p)>_{p\epsilon\omega}$ under substitution ($t:X_n \longrightarrow R_\Sigma(X_p)$, $t':X_p \longrightarrow R_\Sigma(X_q)$; $t\leftarrow t':X_n \longrightarrow R_\Sigma(X_q)$; see Section 2) gives rise to an algebraic theory (c.f. Elgot [16]). Propositions 5.2 and 5.6 say this algebraic theory is iterative which means that every ideal system of

[1] X_p and X_n are, technically, assumed to be disjoint copies of the respective sets of variables.

equations $E:X_n \longrightarrow \bar{R}_{\Sigma(X_p)}(X_n)$ has a unique solution $|E|:X_n \longrightarrow \bar{R}_{\Sigma}(X_p)$ (called $E^{\dagger}$ in [16]) in the sense that $E \leftarrow |E| = |E|$ (this is the definition of fixed point in CT_{Σ} since $E_{CT}(|E|) = E \leftarrow |E|$). We conjecture (with J. B. Wright) that this algebraic theory is the free iterative theory generated by Σ, the existence of which is the principal result of Bloom and Elgot [8].

Systems of Context-Free Equations (Polyadic Recursion).

Whereas for regular equations, the variables (nonterminals) are zero-ary, in the context-free case they are aribrary. For both, initial algebra semantics permits a crisp formulation. Fix a ranked alphabet Σ of "given" function symbols. Let[1] $Z = \langle Z_n \rangle_{n \in \omega}$ be a ranked alphabet of function variables ("new function symbols" or "nonterminals") disjoint from Σ, so $\Sigma \cup Z$ is the ranked alphabet with $(\Sigma \cup Z)_n = \Sigma_n \cup Z_n$.

A recursive definition R is a family of functions $\langle R_n : Z_n \longrightarrow F_{\Sigma \cup Z}(X_n) \rangle_{n \in \omega}$ assigning to each n-ary function variable f, a finite term in Σ, Z and variables, $\{x_1, \cdots, x_n\}$. This simple definition replaces various forms that occur in the literature. For example, Rosen [46] would, for each $f \in Z_n$, write $f(x_1, \cdots, x_n) \leftarrow t$ where $\{x_1, \cdots, x_n\}$ are called formal parameters and t is a tree composed of new and given function symbols and the formal parameters.

Let A be any Δ-continuous Σ-algebra. For any family $\alpha = \langle \alpha_n : Z_n \longrightarrow [A^n \longrightarrow A] \rangle_{n \in \omega}$ of assignments of Δ-continuous functions to function symbols of Z, make A into a Δ-continuous $(\Sigma \cup Z)$-algebra by having $f \in Z_n$ name $\alpha_n(f): A^n \longrightarrow A$ and call that $(\Sigma \cup Z)$-algebra, $A(\alpha)$. Then for each $t \in F_{\Sigma \cup Z}(X_n)$, $t_{A(\alpha)}: A^n \longrightarrow A$ is the derived operator in the algebra $A(\alpha)$ and is Δ-continuous by Proposition 4.12. Now we interpret R by $R_A(\alpha) = R(f)_{A(\alpha)}$, i.e., $R_{A,n}: [Z_n \longrightarrow [A_n \longrightarrow A]] \longrightarrow [Z_n \longrightarrow [A_n \longrightarrow A]]$ and we will just write $R_A: \langle Z_n \longrightarrow [A_n \longrightarrow A] \rangle \longrightarrow \langle Z_n \longrightarrow [A_n \longrightarrow A] \rangle$ for this family of functions. R_A is called the interpretation of R in A, analogous to the system function of Mezei and Wright [39].

This definition closely parallels that for the derived operator E_A associated with a system of regular equations; indeed, corresponding to Proposition 4.12 we have

Proposition 5.7. For any recursive definition $R = \langle R_n : Z_n \longrightarrow F_{\Sigma \cup Z}(X_n) \rangle$, and any Δ-continuous Σ-algebra A, $R_A: \langle Z_n \longrightarrow [A^n \longrightarrow A] \rangle \longrightarrow \langle Z_n \longrightarrow [A^n \longrightarrow A] \rangle$ is Δ-continuous. □

Thus R_A has a minimum fixed point denoted $|R_A| = \langle |R_A|_n : Z_n \longrightarrow [A^n \longrightarrow A] \rangle$ called the solution of R over A and again $|R_A| = \bigsqcup_{k \in \omega} R_A(\perp)$ (Where $\perp : Z_n \longrightarrow [A^n \longrightarrow A]$ assigns to each function variable the constant function $\perp : A^n \longrightarrow A$); $R_A(|R_A|) = |R_A|$; and, if $R_A(\alpha) = \alpha$ then $|R_A| \sqsubseteq \alpha$ (which means for each $z \in Z_n$ $|R_A|_n(z) \sqsubseteq \alpha(z)$, i.e., for all $a \in A^n$, $|R_A|_n(z)(a) \sqsubseteq \alpha(z)(a)$).

As with regular equations with or without parameters, recursive definitions can be solved in CT itself. This is, in effect, one mathematization of the "copy rule". An $\alpha = \langle \alpha_n : Z_n \longrightarrow CT_{\Sigma}(X_n) \rangle$ assigns to each n-ary function variable a (possibly infinite) Σ-tree with n-variables. Then CT_{Σ} is made into a $(\Sigma \cup Z)$-algebra through substitution: For $z \in Z_n$, $z_{CT}(t_1, \cdots, t_n) = \alpha(z) \leftarrow \langle t_1, \cdots, t_n \rangle = \alpha(z)_{CT}(t_1, \cdots, t_n)$. This differs from what we did above; we are only assigning derived operators to the function variables, rather than arbitrary Δ-continuous functions $CT^n_{\Sigma} \longrightarrow CT_{\Sigma}$. Then $R_{CT}: \langle Z_n \longrightarrow CT_{\Sigma}(X_n) \rangle \longrightarrow \langle Z_n \longrightarrow CT_{\Sigma}(X_n) \rangle$ and R_{CT} is also Δ-continuous and has a fixed-point solution over CT_{Σ} which we denote without the CT-subscript as before, $|R|$; $|R|_n : Z_n \longrightarrow CT_{\Sigma}(X_n)$. $|R|_n$ associates with each n-ary function variable f, a possibly infinite Σ-tree called the "complete expansion" by Rosen. (See the factorial example as Figure 1 in [46].)

Again, like regular equations and regular equations with parameters, given R and a Δ-continuous Σ-algebra A, we can interpret R and solve, giving $|R_A|$; or solve and interpret giving $|R|_A$ $(|R|_{A,n}(z) = |R|_n(z)_A)$ -- the answer is the same either way and the proof here is essentially the same as in the previous two cases, depending on the uniqueness part of the initiality of CT_{Σ}.

Proposition 5.1". For any recursive definition $R = \langle R_n : Z_n \longrightarrow F_{\Sigma \cup Z}(X_n) \rangle$ and Δ-continuous Σ-algebra A, $|R|_A = |R_A|$. □

Throughout this Section, Δ-continuous Σ-algebras A are playing the role of "interpretations" (as in [13,46] for example) and $|R_A|$ is the "value" of the recursive definition with respect to that interpretation. Designating some function variable to be the one defined by a recursive definition is simply selecting a component of the definition and the value. If we have two recursive definitions, R, R' with some common $z \in Z_n$ among their function variables such that $|R|_n(z) = |R'|_n(z)$ ("tree-equivalent", by Rosen [46]) then Proposition 5.1" (similarly for 5.1', 5.1) says that the function variable z gets the same value with respect to any interpretation, A: $|R_A|_n(z) = |R|_n(z)_A = |R'|_n(z)_A = |R'_A|_n(z)$, i.e., "tree equivalence" implies equivalence under all interpretation ("strong equivalence"). This is Theorem 3.3 of [46] and appears to be Theorem 1 of [13] although reference [59] is not available to us at this time.[2]

[1] We would like to have used F for the function variables but that notation conflicts with the notation for the finite elements of CT_{Σ}.

[2] Thanks to B. Rosen for pointing out these connections.

6. Conclusion

In Sections 3 and 5 we have surveyed how initial algebra semantics works in a number of areas. The applicability of initial continuous algebras even exceeds our expectations. We have only skimmed the surface of what might be called the equational theory of continuous algebras and hope to fully develop it elsewhere. The following questions and problem areas are among those we would like to answer or see answered.

(A) What are the exact relationships between our R_Σ, defined as equational elements of CT_Σ, Wand's free μ-clones and Bloom and Elgot's, free iterative theories?

(B) How is the conventional theory of equational sets (e.g., Mezei and Wright [39]) retrieved within the (simpler) continuous algebra setting?

(C) If every finite system of equations has a solution in an ordered (not necessarily ω-complete) algebra, does it follow that $|E_A| = \bigsqcup E_A^k(\perp)$ anyway?

(D) How does the work of Maibaum [37], Turner [53] and Wand [63] (which identify context-free sets over one algebra with recognizable sets in another) connect to the relationship between the two ranked alphabets Σ and Σ' in the discussion of the lattice of flow diagrams or more generally between the solution of regular equations and the solution of context-free equations?

(E) Gordon [22] proves the equivalence of a denotational (Scott like) semantics and an interpretive semantics for pure LISP. The proof is very long. The question is, why? Our initial algebra formulation for denotational semantics (Section 3) is significantly different from that used before. "Structural induction" and "finite approximations" are proof techniques used by Gordon and are implicit in initial continuous algebras. Could these be put together to give a more workable semantics for "real" programming languages?

(F) We know T_Σ is initial in the class of Σ-algebras and CT_Σ is initial in the class of Δ-continuous Σ-algebras -- and these results bear fruit. On either side of CT, we ask the question: in what class of algebras is R_Σ (and $\overline{R}_\Sigma$) initial and in what class of algebras is Nivat's PS_Σ, initial? The latter is essentially the question posed in Burstall and Thatcher [10].

(G) Chandra [11] uses infinite tree flow diagrams which are outside R_Σ and contained in CT_Σ (for appropriate Σ); a clean mathematical semantics is determined by picking appropriate Δ-continuous Σ-algebras. But can the translations from finite counter schema, finite stack schema, etc. into infinite tree diagrams be made more mathematical? Most important, how are his "meanings" defined from an initial algebra point of view?

(H) Can the questions raised by Scott in [51] be answered in the initial continuous algebra setting; should they?

These are just a few questions; if the reader has been faced with infinite structures relating to semantics of programs or data structures, then many more specific questions may be suggested. We hope this paper will stimulate such questions and that the initial algebra viewpoint will be fruitful.

REFERENCES

[1] Aho, A.V. and Ullman, J.D., "Properties of syntax directed translations," J. Comput. System Sci. 3 (1969) 319-334.

[2] Aho, A.V. and Ullman, J.D., "Translations of a context-free grammar," Information and Control 19 (1971) 439-475.

[3] Baker, B.S., "Tree transductions and families of tree languages," Harvard University, Center for Research in Computing Technology Technical Report TR9-73 (1973).

[4] Birkhoff, G., "Structure of abstract algebras," Proc. Cambridge Philosophical Society 31 (1938) 433-454.

[5] Birkhoff, G., Lattice Theory, 3rd ed., American Math. Soc., Providence, R. I. (1967).

[6] Birkhoff, G. and Lipson, J.D., "Heterogeneous algebras," J. Combinatorial Theory 8 (1970) 115-133.

[7] Blikle, A., "Equational languages," Information and Control 21 (1972) 134-147.

[8] Bloom, S.L., and Elgot, C.C., "The existence and construction of free iterative theories," IBM Research Report RC-4937 (1974).

[9] Burstall, R.M., and Landin, P.J., "Programs and their proofs: an algebraic approach," Machine Intelligence 4 (Eds. B. Meltzer and D. Michie) Edinburgh University Press (1969) 17-43.

[10] Burstall, R.M. and Thatcher, J.W., "The algebraic theory of recursive program schemes," Category Theory Applied to Computation and Control, U.Mass. (1974) 154-160.

[11] Chandra, A.K., "Degrees of translatability and canonical forms of program schemes: Part I," Proc. Sixth Ann. ACM Symp. on Theory of Computing, Seattle (1974) 1-12.

[12] Cohn, P.M., Universal Algebra, Harper and Row, New York (1965).

[13] Courcelle, B. and Vuillemin, J., "Semantics and axiomatics of a simple recursive language," Proc. Sixth Ann. ACM Symp. on Theory of Computing, Seattle (1974) 13-26.

[14] Doner, J.E., "Tree acceptors and some of their applications," J.Comput. System Sci. 4 (1970) 406-451.

[15] Eilenberg, S. and Wright, J.B., "Automata in general algebras," Information and Control 11 (1967) 452-470.

[16] Elgot, C.C., "Monadic computation and iterative algebraic theories," IBM Research Report, RC-4564 (1973).

[17] Ginsburg, S., The Mathematical Theory of Context-Free Languages, McGraw-Hill, New York (1962).

[18] Goguen, J.A., "On homomorphisms, correctness, termination, unfoldments and equivalence of flow diagram programs," Proc. 13th IEEE Conference on Switching and Automata Theory (1972) 52-60. Also to appear, JCSS.

[19] Goguen, J.A., "Semantics of computation," Category Theory Applied to Computation and Control, U.Mass. (1974) 234-249.

[20] Goguen, J.A. and Thatcher, J.W., "Initial algebra semantics," IBM Research Report, to appear (1974).

[21] Goguen, J.A. Thatcher, J.W., Wagner, E.G., and Wright, J.B., "A junction between computer science and category theory," I,II Basic definitions and examples; III,IV Algebraic theories and structures; V Initial factorizations and dummy variables. IBM Research Reports: I Part 1, RC-4526 (1973); others to appear.

[22] Gordon, M., "Models of pure LISP," Ph.D. Thesis, Edinburgh University (1973).

[23] Higgins, P.J., "Algebras with a schema of operators," Math. Nachr. 27 (1963) 115-132.

[24] Irons, E.T., "A syntax directed compiler for ALGOL 60," Comm. ACM 4 (1961) 51-55.

[25] Knuth, D.E., "Semantics of context-free languages," Math. Systems Theory 2 (1968) 127-145.

[26] Landin, P.J., "The mechanical evaluation of expressions," Comput. J 6 (1964) 308-320.

[27] Lewis, C.H. and Rosen, B.K., "Recursively defined data types, Part 1," Proc. ACM Symposium on Principles of Programming Languages (1973) 125-138. Part 2, IBM Research Report RC-4713 (1974).

[28] Lewis, P.M. and Stearns, R.E., "Syntax directed transduction," J. Assoc. Comput. Mach. 15 (1968) 464-488.

[29] Lucas, P., Lauer, P., and Stigleitner, H., "Method and notation for the formal definition of programming languages," Technical Report TR 25.087, IBM Laboratory, Vienna (1968).

[30] Lucena, C.J., "Towards a methodology for the synthesis of reliable programs," UCLA Internal Memo. 128 (1974).

[31] McCarthy, J., "Towards a mathematical science of computation," Proc. IFIP Congress (1962) 21-28.

[32] McCarthy, J., "A formal description of a subset of ALGOL," in "Formal Language Description Languages for Computer Programming," Proc. IFIP Working Conference 1964 (T. B. Steel, Jr., Ed.) North Holland Publishing Co., Amsterdam (1966) 1-12.

[33] McCarthy, J. and Painter, J., "Correctness of a compiler for arithmetic expressions," in "Mathematical Aspects of Computer Science," Proc. of Symposia in Applied Mathematics 19 (J.T.Schwartz, ed.) American Math. Soc., Providence, R.I. (1967) 33-41.

[34] Mac Lane, S., Category Theory for the Working Mathematician, Springer-Verlag, (1971).

[35] Magidor, M. and Moran, G., "Finite automata over finite trees," Technical Report 30, Hebrew Univ., Jerusalem, Israel (1969).

[36] Maibaum, T.S.E., "The characterization of the derivation trees of context-free sets of terms as regular sets," Proc. IEEE Conference on Switching and Automata Theory (1972) 224-230.

[37] Maibaum, T.S.E., "Generalized grammars and homomorphic images of regular sets," University of Waterloo, Research Report, CS-73-30 (1973).

[38] Markowsky, G., "Notes on posets, lattices, and categories with applications to computation," IBM Research Report to appear (1974).

[39] Mezei, J. and Wright, J.B., "Algebraic automata and context-free sets," Information and Control 11 (1967) 3-29.

[40] Morris, F.L., "Advice on structuring compilers and proving them correct," Proc. Symposium on Principles of Programming Languages, Boston (1973).

[41] Nivat, M., "Languages algebraic sur le magna libre et semantique des schemas de programme," in Automata, Languages and Programming (M. Nivat, ed.) North Holland (1972) 293-308.

[42] Petrone, L., "Syntactic mappings of context-free languages," Proc. IFIP Congress 2 (1965) 590-591.

[43] Pottenger, R., Masters Thesis, UCLA (1974).

[44] Rabin, M.P. and Scott, D., "Finite automata and their decision problems," IBM J. Res. Develop. 3 (1959) 114-125.

[45] Reynolds, J.C., "Definitional interpreters for higher-order programming languages," Proc. ACM National Conference (1972) 717-740.

[46] Rosen, B.K., "Program equivalence and context-free grammars," Proc., 13th Annual IEEE Symposium on Switching and Automata Theory (1972) 7-18. Revised as IBM Research Report RC-4822 (1974).

[47] Rounds, W. C., "Mappings and grammars on trees," Math. Systems Theory 4 (1970) 257-287.

[48] Schützenberger, M.P., "Context-free languages and push down automata," Information and Control 6 (1963) 246-264.

[49] Schwartz, J.T., "Semantic definition methods," Formal Semantics of Programming Languages (R.Rustin, ed.) Prentice Hall (1972) 1-23.

[50] Scott, D., "Outline of a mathematical theory of computation," Proc. 4th Annual Princeton Conference on Information Sciences and Systems (1970) 169-176.

[51] Scott, D., "The lattice of flow diagrams," Semantics of Algorithmic Languages (E. Engeler, ed.) Springer Lecture Notes in Mathematics 182 (1971) 311-366.

[52] Scott, D., "Continuous lattices," Oxford University Computing Laboratory Technical Monograph PRG 7 (1971).

[53] Scott, D., "Data types as lattices," unpublished notes, Amsterdam (1972).

[54] Scott, D. and Strachey, C., "Towards a mathematical semantics for computer languages," Proc. Symposium on Computers and Automata, Polytechnic Institute of Brooklyn 21, (1971) Oxford University Computing Laboratory Technical Monograph, PRG 6.

[55] Thatcher, J.W., "Generalized^2 sequential machines," J. Comput. System Sci. 4 (1970) 339-367.

[56] Thatcher, J.W., "Tree automata: An informal survey," in Currents in Computing, Prentice-Hall (1973) 143-172.

[57] Thatcher, J.W. and Wright, J.B., "Generalized finite automata theory with an application to a decision problem of second-order logic," J. Comput. System Sci. 4 (1968) 57-81.

[58] Turner, R., Doctoral Dissertation, University of London (1973).

[59] Vuillemin, J., "Syntaxe, Semantique et Axiomatique d'un Langage de Programmation Simple," These d'Etat (to appear).

[60] Wagner, E.G., "An algebraic theory of recursive definitions and recursive languages," Proc. Third Annual ACM Symposium on Theory of Computing (1971) 12-23.

[61] Wagner, E.G., "Languages for defining sets in arbitrary algebras," Proc. Eleventh Annual IEEE Symposium on Switching and Automata Theory (1971) 192-201.

[62] Wand, M., "A concrete approach to abstract recursive definitions," Automata Languages and Programming (M. Nivat, ed.) (1972) 331-341.

[63] Wand, M., "An algebraic formulation of the Chomsky hierarchy," Category Theory in Computation and Control, U.Mass. (1974) 216-221.