Initial Algebra Semantics and Continuous Algebras

J. A. GOGUEN

UCLA, Los Angeles, California

AND

J. W. THATCHER, E. G. WAGNER, AND J. B. WRIGHT

IBM Thomas J Watson Research Center, Yorktown Heights, New York

ABSTRACT Many apparently divergent approaches to specifying formal semantics of programming languages are applications of initial algebra semantics In this paper an overview of initial algebra semantics is provided The major technical feature is an initial continuous algebra which permits unified algebraic treatment of iterative and recursive semantic features in the same framework as more basic operations

KEY WORDS AND PHRASES algebraic semantics, algebras, free algebras, continuous algebras, posets, programming language semantics, flow diagrams, syntax directed translation, solutions of equations

CR CATEGORIES 5 23, 5 24

1. *Introduction*

In the past few years there has been quite a proliferation of formal semantics for programming languages, or at least of different descriptive terms, for example, operational, interpretive, fixed point, predicate calculus, denotational, algebraic, mathematical, synthesized, W-grammar, axiomatic, inherited, declarative, continuation, process, and now initial algebra semantics. Moreover, mathematical concepts, said to be deep, or strange, or new, are asserted to be relevant, for example, continuous lattices, iterative algebraic theories, infinitary logic, and bicategories This is quite perplexing. How do these things fit together, if at all? In fact, what is "syntax"; what is "semantics"?

This paper is not going to answer all these questions. But we believe the subject cannot be said to be in very good shape when such questions are ignored or glossed over, and when "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 which expose a surprising and beautiful unity in the apparent diversity of approaches.

The key concept is very simple: An algebra S is *initial* in a class $\boldsymbol{C}$ of algebras iff for every A in $\boldsymbol{C}$ there exists a unique homomorphism $h_A : S \to A$.[1]

This is a revised and expanded version of a paper entitled "Initial Algebra Semantics," which was presented at the 15th IEEE Symposium on Switching and Automata Theory, October 1974

The work of J A Goguen was partially supported by the Naropa Institute, Boulder, Colorado, and the University of Colorado, Boulder, Colorado.

Authors' addresses: J A Goguen, Computer Science Department, UCLA, Los Angeles, CA 90024, J W Thatcher, E G Wagner, and J B Wright, IBM Thomas J Watson Research Center, Yorktown Heights, NY 10598

[1] We are assuming that $\boldsymbol{C}$ comes with an "appropriate" class of homomorphisms which is closed under associative composition and includes an identity for each algebra More formally, we assume that $\boldsymbol{C}$ is a *category*. The reader seeking a more comprehensive algebraic context may want to consult [23] and/or [39]

In the cases we examine, syntax *is* an initial algebra in a class of algebras, and any other algebra A in the class is a possible *semantic domain* (or *semantic algebra*); the *semantic function* is the uniquely determined homomorphism $h_A : S \to A$, assigning a *meaning* $h_A(s)$ in A to each syntactic structure s in S. From this viewpoint it becomes clear that a major aspect of formal semantics (both practical and theoretical) is constructing *intended* semantic algebras for particular programming languages. We believe that is what Scott and Strachey [65] and their followers are doing with the tools outlined by Scott [60] and more fully developed in [62–64].

We use *abstract syntax* (cf. McCarthy [36]), but without employing any "concrete" version such as in the Vienna Definition Language [35] What is abstract about "abstract syntax" is captured by the following proposition.[2]

PROPOSITION 1 1. *If S and S' are both initial in a class $\mathbf{C}$ of algebras, then S and S' are isomorphic. If S'' is isomorphic to an initial algebra S, then S'' is also initial.*

PROOF [3] If S and S' are initial in the class $\mathbf{C}$ of algebras we have (unique) homomorphisms $h_{S'} : S \to S'$ and $h_S : S' \to S$. The composite of $h_{S'}$ followed by h_S (denoted $h_S \circ h_{S'}$) is a homomorphism from S to S. The identity function, 1_S, is also a homomorphism; so $h_S \circ h_{S'} = 1_S$ by uniqueness Similarly $h_{S'} \circ h_S = 1_{S'}$, thus h_S is an isomorphism.

If S is initial and S'' is isomorphic to S, then $h_{S''} \cdot S \to S''$ must be the isomorphism. For any algebra A in C, we have $h_A : S \to A$ so that $h_A \circ h_{S''}^{-1}$ is a homomorphism from S'' to A. If $g : S'' \to A$ is any other such homomorphism then $g \circ h_{S''} : S \to A$ so $g \circ h_{S''} = h_A$ and $g = h_A \circ h_{S''}^{-1}$. Thus S'' is initial. □

Rather than assume "that programs are 'really' abstract, hierarchically structured data objects . . . ," as in Reynolds [53], we avoid troublesome questions of definition and notation by identifying "abstract syntax" with "initial algebra"; Proposition 1.1 says abstract syntax is independent of notational variation – as it should be. For example, if Σ is a ranked alphabet,[4] define a Σ-*tree* to be an element of an initial Σ-algebra. In this way 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 connection with abstract syntax, the initial algebra approach is implicit in McCarthy's [36] proposals for a "Mathematical Science of Computation" and in Iron's pioneering paper [29] on syntax directed translation. The approach becomes more explicit at least as far back as Landin [31] and Petrone [50], as well as in McCarthy and Painter [38], Knuth [30], Burstall and Landin [10], Nivat [48], and Morris [46, 47]. Of these, Burstall and Landin first use universal algebra (a la Cohn [13]) and (implicitly) the initiality property, Morris [46, 47] brings in many-sorted algebras, which are essential for any applications of algebraic semantics to interesting programming languages.

This paper makes the initial algebra approach to semantics completely explicit, providing the necessary algebraic background and several examples. Our principal new result is the existence of initial *continuous* algebras. This extends the applicability of initial algebra semantics by combining the algebraic insights of Burstall, Landin, and Morris with the (lattice-) order-theoretic ideas of Scott and Strachey. We consider solutions of equations in continuous algebras and show, for example, how Scott's lattice of flow diagrams [61] is a special case of an initial continuous algebra.

Section 2 makes the notion of many-sorted algebra precise. That material is not mathematically new and has a survey character. However, our presentation is different

[2] In order to prove existence of initial algebras, we have to employ a concrete construction, but once done, we can forget about that construction and rely solely on Proposition 1 1

[3] This proof depends exactly on the assumptions about $\mathbf{C}$ mentioned in the footnote to the definition of "initial "

[4] A *ranked alphabet* is a family $\langle \Sigma_i \rangle_{i \in \omega}$ of disjoint sets indexed by the natural numbers Σ_k is the set of *operator symbols* of *rank k*. There are more details in Section 2 Note that the assumption that the Σ_i are disjoint is not necessary (or even desirable in general – see Thatcher [67]), we adopt it here to simplify the exposition

from the literature (we hope it is simpler) and seems to us to be of fundamental importance. A careful reading of Section 2 serves multiple purposes, because the material there can be extended to almost any initial algebra situation, including initial continuous algebras (Section 4).

Section 3 contains applications (examples) of initial many-sorted algebras. Section 4 defines "continuous algebra" and constructs initial continuous algebras. Applications of this, including Scott's lattice of flow diagrams [61], are in Section 5. Section 6 contains some questions and problems.

Some of the subject matter included here is being prepared for inclusion in the second part of the first report of the ADJ series [23] [5]

2. *Many-Sorted Algebras*

An algebra in the sense of Birkhoff [6] (see also Cohn [13] and Graetzer [27]) 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. A many-sorted algebra consists of an indexed family of sets (called carriers) and an indexed family of operations defined on Cartesian products of those sets. Generalizing to many-sorted algebras is very natural for computer science. The index set for carriers is called the set of sorts, and might be, for example, {**real**, **int**, **bool**}. An algebra A of this kind would have three carriers, $A_{\mathbf{real}}$, $A_{\mathbf{int}}$, and $A_{\mathbf{bool}}$, together with some operations such as $\uparrow : A_{\mathbf{real}} \times A_{\mathbf{int}} \to A_{\mathbf{real}}$ or $\to : A_{\mathbf{bool}} \times A_{\mathbf{real}} \times A_{\mathbf{real}} \to A_{\mathbf{real}}$, which might be exponentiation or conditional, respectively.

A finite automaton, in the sense of Rabin and Scott [52], is an algebra with two sorts: states named S and inputs named Σ, i.e. an $\{S, \Sigma\}$-sorted algebra. The transition function is an operation M of "type" $\langle S\Sigma, S\rangle$ ($M : S \times \Sigma \to S$, using the sort symbols to denote the carriers, an ambiguous but common practice). The initial state is a constant s_0 of sort S ($s_0 \in S$).

The definition below is equivalent to Birkhoff and Lipson's [7] "heterogeneous algebra" and therefore (according to them) is equivalent to Higgins' [28] "algebra with a scheme of operators." Birkhoff and Lipson show that the conventional theory of "universal algebra" (as in [6, 13, 27]) carries over "with undiminished force" to the theory of many-sorted algebras. In particular, the concepts of subalgebra, homomorphism, quotient, congruence relation, product, word algebras, and free algebras generalize naturally and easily. Although Birkhoff and Lipson give some computer science examples, the first explicit use for new results in computer science seems to be in Maibaum [41, 42]. The concept appears to have originated with Bénabou [5] in a category theoretic form.

Let S be a set whose elements are called *sorts*. An *S-sorted operator domain* or *signature* 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* $\langle w, s\rangle$, *arity* w, *sort* s, and *rank* $\mathbf{lg}(w)$ ($\mathbf{lg}(w)$ is the length of w, $\lambda \in S^*$ is the empty string, and $\mathbf{lg}(\lambda) = 0$). 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 for each $\sigma \in \Sigma_{w,s}$, an operation σ_A of type $\langle w, s\rangle$, i.e $\sigma_A : A_{w_1} \times \cdots \times A_{w_n} \to A_s$ where $w = w_1 \cdots w_n$ and $w_i \in S$ for $i = 1, \cdots, n$. An operation σ_A of type $\langle \lambda, s\rangle$ 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 too, the class of *many-sorted algebras*.

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

[5] The *set* of authors of this paper is referred to as "ADJ", the series referenced here is devoted to an exploration of "the junction between category theory and computer science "

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

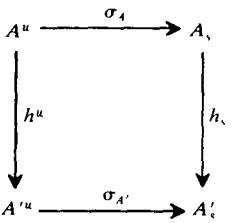
(0) if $\sigma \in \Sigma_{\lambda,s}$, then $h_s(\sigma_A) = \sigma_{A'}$;

(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

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

The utility of initial algebra semantics rests heavily on being able to handle many-sorted algebras with as much clarity and notational simplicity as the one-sorted (conventional) case. Indeed, the generalization from sets and functions to S-indexed families of sets and S-indexed families of functions is not that great a jump. If A is an S-indexed family of sets ($A = \langle A_s \rangle_{s \in S}$) and $w = s_1 \cdots s_n \in S^*$, then generalizing A^n, define A^w to be the Cartesian product $A_{s_1} \times \cdots \times A_{s_n}$. Similarly, for an S-indexed family of functions $h : A \to A'$ ($\langle h_s : A_s \to A'_s \rangle_{s \in S}$), define $h^w : A^w \to A'^w$ by $h^w(a_1, \cdots, a_n) = \langle h_{s_1}(a_1), \cdots, h_{s_n}(a_n) \rangle$, which generalizes $h^n : A^n \to A'^n$. The special case $w = \lambda$ ($n = 0$) is handled uniformly by $A^0 = A^\lambda = \{\lambda\}$, and a function σ_A with source A^λ is identified with the "constant" which is its value $\sigma_A(\lambda)$ at its single argument.[6]

Now with this notation, an *S-sorted Σ-algebra* A consists of: a carrier (ambiguously denoted) A, which is an S-indexed family of sets; and, for each operator symbol $\sigma \in \Sigma_{w,s}$, a function $\sigma_A : A^w \to A_s$. An *S-sorted Σ-homomorphism* h from A to A' is an S-indexed family of functions $\langle h_s : A_s \to A'_s \rangle$ such that $h_s(\sigma_A(a)) = \sigma_{A'}(h^w(a))$, i.e.



commutes for all $\sigma \in \Sigma_{w,s}$ and $a \in A^w$. This has now become only a minor variation on the one-sorted definition.

Letting $\mathbf{Alg}_\Sigma$ denote the class of Σ-algebras with Σ-algebra homomorphism, we state the first basic result concerning initial algebras.

PROPOSITION 2.1 *The class $\mathbf{Alg}_\Sigma$ of Σ-algebras has an initial algebra; call it T_Σ.* □

This is a well-known result, at least for the one-sorted case (see Birkhoff [6] or Cohn [13] or Graetzer [27]), and the many-sorted case is treated in Birkhoff and Lipson [7]. 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 thought of as the set of well-formed expressions (or trees) of sort s built up in the usual way (to be made precise in a moment) from the operator symbols of Σ. But we emphasize that this characterization is (mathematically) important *only* for the proof of Proposition 2.1. Once proved, we do not *need* to know how a particular initial algebra was constructed—we use only initiality. Comments about expressions (or pictures of trees) occur only to aid understanding.

Let Σ (ambiguously) denote the set of all operator symbols in the S-sorted operator domain Σ. Now let $\langle T_{\Sigma,s} \rangle_{s \in S}$ be the smallest family of sets contained in $(\Sigma \cup \{(,)\})^*$ satisfying the following two conditions (here $\{(,)\}$ is a two-element set disjoint from Σ):

(0) $\Sigma_{\lambda,s} \subseteq T_{\Sigma,s}$;

(1) if $\sigma \in \Sigma_{w,s}$, $w = s_1 \cdots s_n$, $n > 0$, and $t_i \in T_{\Sigma,s_i}$, then $\sigma(t_1 \cdots t_n) \in T_{\Sigma,s}$

Then make the family $\langle T_{\Sigma,s} \rangle$ into a Σ-algebra T_Σ by defining the operations:

(0) for $\sigma \in \Sigma_{\lambda,s}$, $\sigma_T = \sigma \in T_{\Sigma,s}$; [7]

(1) for $\sigma \in \Sigma_{w,s}$, $w = s_1 \cdots s_n$, and $t_i \in T_{\Sigma,s_i}$, $\sigma_T(t_1, \cdots, t_n) = \sigma(t_1 \cdots t_n) \in T_{\Sigma,s}$.

[6] Although this uniformity is mathematically nice, it is often more convenient to separate out the constants from the more general operators, for the most part this will be done in the sequel

[7] We are writing σ_T instead of σ_{CT_Σ}

Under the natural identification of elements of A^w with strings $a_{w_1} \cdots a_{w_n}$ such that $a_{w_i} \in A_{w_i}$, (1) immediately above can be rewritten:

for all $\sigma \in \Sigma_{w,s}$ and $t \in T_\Sigma^w$, $\sigma_T(t) = \sigma(t)$.

As an artificial example, $S = \{a, b\}$, $\Sigma_{\lambda,a} = \{x_a\}$, $\Sigma_{\lambda,b} = \{x_b\}$, $\Sigma_{ab,a} = \{f\}$, and $\Sigma_{b,b} = \{g\}$. Then $T_{\Sigma,a}$ contains terms $x_a, f(x_a x_b), f(x_a g(x_b)), f(f(x_a x_b)x_b)$, etc., while $T_{\Sigma,b}$ contains x_b, $g(x_b)$, $g(g(x_b))$, etc. Much less artificial is the correspondence, in Section 3, of any context-free grammar to an initial N-sorted algebra (where N is the set of nonterminals).

In any algebra A in the class $\mathbf{Alg}_\Sigma$, if $\sigma \in \Sigma_{w,s}$ and $a \in A^w$, then $\sigma_A(a) \in A_s$. This algebraic structure on the set T_Σ (making it the initial Σ-algebra) seems to be "synthetic syntax" [36], and σ_T is familiar as the "constructor function" denoted $\mathbf{mk} - \sigma$ in Reynolds [53].

On the other hand, "analytic syntax" [36] is based on

PROPOSITION 2.2. *For any $t \in T_{\Sigma,s}$ there exist unique $n \geq 0$, $s_1 \cdots s_n \in S^*$, $\sigma \in \Sigma_{\langle s_1 \quad s_n, s\rangle}$, and $t_i \in T_{\Sigma,s_i}$ $(1 \leq i \leq n)$ such that $t = \sigma_T(t_1, \cdots, t_n)$.* □[8]

The notation in Reynolds [53] can be generalized to many sorts as follows: For each $\sigma \in \Sigma$, $\mathbf{is} - \sigma$ tests whether its argument is of the form $\sigma_T(t_1, \cdots, t_n)$, taking values in the set {**true**, **false**}; $\mathbf{is} - s$ tests whether its argument could be of sort s; and for each pair $\langle \sigma, i\rangle$ with $1 \leq i \leq \text{rank}(\sigma)$, $s_{\langle\sigma,i\rangle}$ is the "selector function" defined iff $\mathbf{is} - \sigma$ is **true** by $s_{\langle\sigma,i\rangle}(\sigma_T(t_1, \cdots, t_n)) = t_i$. These functions are well defined by Proposition 2.2, and if $\mathbf{is} - \sigma(t)$ is **true**, then

$$\mathbf{mk} - \sigma\,(s_{\langle\sigma,1\rangle}(t), \cdots, s_{\langle\sigma,n\rangle}(t)) = t.$$

Moreover, if $\sigma \in \Sigma_{\langle s_1 \quad s_n, s\rangle}$ and if $\mathbf{is} - s_i(t_i)$ is **true** for $1 \leq i \leq n$, then

$$s_{\langle\sigma,i\rangle}(\mathbf{mk} - \sigma(t_1, \cdots, t_n)) = t_i.$$

In [53] the various selector functions are given names (e.g. ***opr***, ***opnd***, etc.), but of course this isn't necessary. In fact, as an example below illustrates, the "language" of synthetic and analytic syntax is unnecessary with the initial algebra approach to abstract syntax.

We now introduce "freely generated" Σ-algebras in order to get a concise definition of "derived operator"; for notational simplicity we consider the one-sorted case first.

Let X be a set (whose elements are called *variables*) disjoint from Σ. We form a new ranked alphabet, denoted $\Sigma(X)$, by adjoining the variables as new constant symbols: $\Sigma(X)_0 = \Sigma_0 \cup X$ and $\Sigma(X)_k = \Sigma_k$ for all $k > 0$. By Proposition 2.1, $T_{\Sigma(X)}$ is the initial $\Sigma(X)$-algebra. The trees which are in its carrier differ from those in T_Σ only in that they may have variables on the leaf nodes as well as constants from Σ_0. Now $T_{\Sigma(X)}$ has $\Sigma(X)$ as its signature, whereas we want to think of $T_{\Sigma(X)}$ as a Σ-algebra. Since the operations (and constants) named by $\Sigma(X)$ include those named by Σ, we can do this just by "forgetting" that the variables have names in the signature; that is, we define a new Σ-algebra, denoted $T_\Sigma(X)$, with carrier that of $T_{\Sigma(X)}$ and with operations those named by Σ in $T_{\Sigma(X)}$. It is the *free Σ-algebra generated by X*, in the following sense.

PROPOSITION 2.3. *If $h : X \to A$ is any function mapping X into the carrier of a Σ-algebra A, then there exists a unique Σ-homomorphism $\bar{h} : T_\Sigma(X) \to A$ extending h, i.e. $\bar{h}(x_T) = h(x)$.*[9]

PROOF. A is a Σ-algebra. Given $h : X \to A$, *make A into a $\Sigma(X)$-algebra by* having x name $h(x)$ in A ($x_A = h(x)$). By Proposition 2.1, there is a unique $\Sigma(X)$-homomorphism $\bar{h} : T_{\Sigma(X)} \to A$, and (by clause (0) of the definition of homomorphism) $\bar{h}(x_T) = x_A = h(x)$; because $\bar{h}$ is a $\Sigma(X)$-homomorphism, it is immediately a Σ-homomorphism. For the uniqueness part, if $g : T_\Sigma(X) \to A$ is a Σ-homomorphism with $g(x_T) = h(x)$, then it is also a $\Sigma(X)$-homomorphism, and thus $g = \bar{h}$. □

[8] This is familiar for well-formed expressions and/or Σ-trees, and that is sufficient proof because of Proposition 1.1.

[9] Note that x_T is whatever the zero-ary symbol $x \in X$ names in the initial algebra $T_{\Sigma(X)}$—in the above concrete construction $x_T = x$, but we can't assume that since we are depending *only* on the initiality of $T_{\Sigma(X)}$.

This construction is typical of initial algebra semantics. The phrase, "make B into a Σ-algebra by $\cdots$," is fundamental because making B into a Σ-algebra gives the unique homomorphism $h_B : T_\Sigma \to B$ by initiality. In the proof of Proposition 2.3 this was particularly easy—A was already a Σ-algebra and we made it a $\Sigma(X)$-algebra In general, if B is any set (later, any family indexed by S), we make B into a Σ-algebra by defining appropriate operations $\sigma_B : B^n \to B$ for each $\sigma \in \Sigma_n$; then (zap!) $h_B : T_\Sigma \to B$.

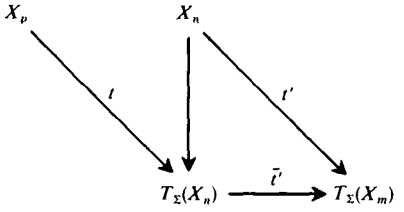
We are now in a position to define the very important notion of derived operator. The intuitive idea is that the terms in $T_\Sigma(X)$ (like polynomials in ordinary algebra) do not have values, since they contain variables, but they *do* define functions on any Σ-algebra by assigning values to the variables. Because this gives rise to new operators on Σ-algebras A compounded out of those named by Σ, the new operators are called derived operators

First, let $X_n = \{x_1, \cdots, x_n\}$, and call $t \in T_\Sigma(X_n)$ a *Σ-term in n variables* or an *n-ary Σ-term*. Now given a Σ-algebra A and $t \in T_\Sigma(X_n)$, we want to define its corresponding *derived operator* on A, $t_A : A^n \to A$. This employs Proposition 2.3 as follows: Given $\langle a_1, \cdots, a_n \rangle \in A^n$, let $a : X_n \to A$ be defined by sending $x_i \to a_i$ for $1 \leq i \leq n$; then, by Proposition 2.3, there exists a unique Σ-homomorphism $\bar{a} : T_\Sigma(X_n) \to A$ extending $a : X_n \to A$; now we define $t_A(a_1, \cdots, a_n)$ to have value $\bar{a}(t)$. Thus we are letting a vary in the expression $\bar{a}(t)$, while keeping t fixed; $\bar{a}(t)$ is the "evaluation" in A of the n-ary term t in which the variable x_i is given the value a_i in A, using the operations of the Σ-algebra A.

For the special cases when A is of the form $A = T_\Sigma(X_m)$, the above gives us a precise formulation of the operation of substitution of m-ary Σ-terms for variables. (When variables are assigned to terms, "evaluation" is just substitution.)

That is, given $t \in T_\Sigma(X_n)$ and an n-tuple of m-ary Σ-terms $t' = \langle t'_0, \cdots, t'_n \rangle \in (T_\Sigma(X_m))^n$, we again identify t' with the mapping from X_n to $T_\Sigma(X_m)$ which sends x_i to t'_i for $1 \leq i \leq n$. The result of simultaneously substituting t'_i for x_i in t for $i = 1, \cdots, n$ is precisely $\bar{t}'(t)$. We write $t \leftarrow t'$ for the substitution of (the n-tuple of m-ary Σ-terms) t' in the n-ary term t.

If we take t to be a p-tuple of n-ary Σ-terms $t = \langle t_1, \cdots, t_p \rangle \in (T_\Sigma(X_n))^p$ rather than just a single n-ary Σ-term, then we could define $t \leftarrow t'$ as $\langle t_1 \leftarrow t', \cdots, t_p \leftarrow t' \rangle$. However, by exploiting the correspondence between p-tuples $t \in (T_\Sigma(X_n))^p$ and mappings $t : X_p \to T_\Sigma(X_n)$, we can more succinctly define $t \leftarrow t'$ as $\bar{t}' \circ t$ ("$\circ$" is a function composition).

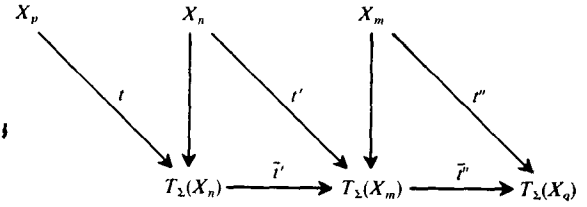


Then $t \leftarrow t' = \bar{t}' \circ t : X_p \to T_\Sigma(X_m)$ and the "ith component" of this "p-tuple" is $(t \leftarrow t')(x_i) = (\bar{t}' \circ t)(x_i) = \bar{t}'(t(x_i)) = \bar{t}'(t_i) = t_i \leftarrow t'$ just as before.

With this definition it is easy to show that substitution is associative. For given

$$t : X_p \to T_\Sigma(X_n), \quad t : X_n \to T_\Sigma(X_m), \quad \text{and} \quad t'' : X_m \to T_\Sigma(X_q),$$

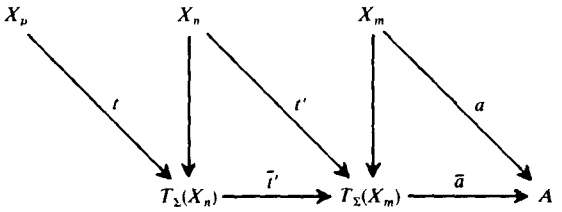
we have



$(t \leftarrow t') \leftarrow t'' = (\tilde{t}' \circ t) \leftarrow t'' = \tilde{t}'' \circ (\tilde{t}' \circ t) = (\tilde{t}'' \circ \tilde{t}') \circ t$ by definition of $\leftarrow$. But then $\tilde{t}'' \circ t'$ and $\tilde{t}'' \circ \tilde{t}'$ are both homomorphisms $T_\Sigma(X_n) \to T_\Sigma(X_q)$ extending $\tilde{t}'' \circ t'$, and so Proposition 2.3 says they are equal, whence $(t \leftarrow t') \leftarrow t'' = (\tilde{t}'' \circ \tilde{t}') \circ t = (\widetilde{\tilde{t}'' \circ t'}) \circ t = t \leftarrow (\tilde{t}'' \circ t') = t \leftarrow (t' \leftarrow t'')$ as desired. Thus we have

PROPOSITION 2.4. *Substitution is associative* For $t : X_p \to T_\Sigma(X_n)$, $t' : X_n \to T_\Sigma(X_m)$ *and* $t'' : X_m \to T_\Sigma(X_q)$, *then* $(t \leftarrow t') \leftarrow t'' = t \leftarrow (t' \leftarrow t'')$. □ [10]

Just as we can extend substitution to p-tuples, so we can extend the notion of derived operators to *p-tuples* of n-ary Σ-terms. Given $t = \langle t_1, \cdots, t_p \rangle : X_p \to T_\Sigma(X_m)$, we define $t_A : A^m \to A^p$ to be such that, for each $a \in A^m$ (i.e. $a : X_m \to A$), $t_A(a) = \bar{a} \circ t$, which, for $t = \langle t_1, \cdots, t_p \rangle$, gives $t_A(a) = \langle t(x_1)_A(a), \cdots, t(x_p)_A(a) \rangle = \langle (t_1)_A(a), \cdots, (t_p)_A(a) \rangle$. Now given



which is a trivial modification of the diagram for substitution, the proof of Proposition 2.4 is immediately modified to yield

PROPOSITION 2.5. $(t \leftarrow t')_A = t_A \circ t'_A$ (*where the right-hand side is function composition and the left-hand side is substitution*). □

Some familiar functions on trees or expressions are further examples of initial algebra semantics

Frontier. Make Σ_0^* into a Σ-algebra.

(0) For $\sigma \in \Sigma_0$, let σ be $\sigma \in \Sigma_0^*$.

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

The unique Σ-homomorphism $\boldsymbol{fr} : T_\Sigma \to \Sigma_0^*$ 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 it 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.

While the convention of using the operator symbols to denote the functions in (different) algebras can lead to some rather strange looking expressions (cf. case (0) in the example above and in the two that follow), it is a recognized convenience and avoids naming algebras in these examples and the resulting complex subscripts.

Height. Make $\omega = \{0, 1, 2, \cdots\}$[11] into a Σ-algebra:

(0) For $\sigma \in \Sigma_0$, let $\sigma = 1$.

(1) For $\sigma \in \Sigma_n$, $k_1, \cdots, k_n \in \omega$, let $\sigma(k_1, \cdots, k_n) = \max_i(k_i) + 1$.

The unique Σ-homomorphism $\boldsymbol{hg} : T_\Sigma \to \omega$ is the height function—in tree terminology, the length of the longest path from the root to a terminal node (+1).

Symbols. Make[12] $\boldsymbol{p}(\Sigma)$ into a Σ-algebra:

(0) For $\sigma \in \Sigma_0$, let $\sigma = \{\sigma\}$.

(1) For $\sigma \in \Sigma_n$, $n > 0$, and $u_1, \cdots, u_n \in \boldsymbol{p}(\Sigma)$, let $\sigma(u_1, \cdots, u_n) = \{\sigma\} \cup \bigcup_i u_i$.

The unique Σ-homomorphism $\boldsymbol{sym} : T_\Sigma \to \boldsymbol{p}(\Sigma)$ is the set of symbols that occur in an expression.

We finish off this section by showing how the important process of defining derived operators works for many-sorted algebras. Whereas in the conventional case we gener-

[10] From a categorical point of view what we have here is a category $\boldsymbol{T}_\Sigma$ (the algebraic theory [32] freely generated by Σ) with set of objects ω; morphisms from m to n are functions from X_m to $T_\Sigma(X_n)$, composition is substitution; and for each $n \in \omega$, the mapping $X_n \to T_\Sigma(X_n)$ taking x_i to x_i is the identity for n

[11] Throughout we use ω to denote the nonnegative integers.

[12] $\boldsymbol{p}(\Sigma)$ is the set of subsets of the set of operator symbols Σ ($\boldsymbol{p}$ is for "power set").

ated a free algebra from a single set, now we must have generators for each sort, i.e an S-indexed family X Given an S-sorted operator domain Σ, define an operator domain $\Sigma(X)$ by $\Sigma(X)_{\lambda,s} = X_s \cup \Sigma_{\lambda,s}$ and $\Sigma(X)_{w,s} = \Sigma_{w,s}$ for $w \neq \lambda$; that is, the generators are adjoined as symbols of arity λ and sort s. Again $T_{\Sigma(X)}$ is viewed as a Σ-algebra $T_\Sigma(X)$, and we have the "freeness" proposition proved exactly as Proposition 2 3.

PROPOSITION 2.6. *For an S-sorted operator domain Σ and an S-indexed family X, $T_\Sigma(X)$ is the Σ-algebra freely generated by X in the sense that any S-indexed family of functions, $h : X \to A$, to the carrier of a Σ-algebra A, extends uniquely to a Σ-homomorphism $\bar{h} : T_\Sigma(X) \to A$.* □

For $u = s_1 \cdots s_n \in S^*$, define X_u (the analogue of X_n is the one-sorted case) by $X_u = \{x_{1,s_1}, x_{2,s_2}, \cdots, x_{n,s_n}\}$. X_u is a set of n "sorted" or "typed" variables and can be viewed as an S-indexed family with $(X_u)_s = \{x_{i,s_i} \mid s_i = s\}$. This specifies, for each s, the countable set $\{x_{1,s}, x_{2,s}, x_{3,s}, \cdots\}$ of s-sorted variables via convenient finite subsets X_u for $u \in S^*$. For example, with $S = \{a, b\}$ again, $x_{aabab} = \{x_{1,a}, x_{2,a}, x_{4,a}, x_{3,b}, x_{5,b}\}$.

In an algebra A, a term $t \in (T_\Sigma(X_u))_s$ defines a derived operator $t_A : A^u \to A_s$ analogously to the one-sorted case: View $a \in A^u$ as an indexed family of functions $a : X_u \to A$ by $a_s(x_{i,s_i}) = a_i \in A_{s_i}$ and, using Proposition 2.6, define $t_A(a) = \bar{a}(t)$.

Again, substitution is a special case. A family of maps $t : X_u \to T_\Sigma(X_v)$ is like an n-tuple of terms in variables X_v except that $t(x_{i,s_i})$ has to be in $T_\Sigma(X_v)_{s_i}$ so that variables of sort s_i are assigned terms of sort s_i. Also given $t' : X_v \to T_\Sigma(X_w)$, S-sorted substitution is defined by $t \leftarrow t' = t \circ \bar{t'}$; $t \leftarrow t'$ is again the result of simultaneously substituting $t'_i = t'(X_{i,s_i})$ for x_{i,s_i} in each component of t. Associativity of substitution and its relationship to derived operators go through exactly as in Propositions 2.4 and 2.5.

3. *Applications of Initial Many-Sorted Algebras*

3.1 CONTEXT-FREE GRAMMARS. The most important and general example of initial many-sorted algebra semantics is semantics for context-free grammars

Let $G = \langle N, \Sigma, P\rangle$ be a content-free grammar[13] 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 $\boldsymbol{nt}(w)$ to be the string of nonterminals in w, in the same order. (More precisely, $\boldsymbol{nt} : V^* \to N^*$ is the unique extension to a monoid homomorphism of the map $V \to N^*$ which is the identity on N and takes each $\sigma \in \Sigma$ to $\lambda \in N^*$)

Now make G into an N-sorted operator domain where, for each $\langle w, A\rangle \in N^* \times N$,

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

Thus a production $A_0 \to u_0 A_1 u_1 A_2 \cdots u_{n-1} A_n u_n$ ($A_i \in N, u_i \in \Sigma^*$) 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 parse trees for derivations in G from $A \in N$.

Regarding the generality of this situation, we have just seen that any context-free grammar gives rise to an initial N-sorted algebra. Conversely (because we allow infinite grammars), if Σ is any S-sorted operator domain, then there is a context-free grammar $\hat{\Sigma}$ which gives us the initial Σ-algebra back again under the construction above. In particular, take

$$\hat{\Sigma} = \langle S, \Sigma \cup \{(,)\}, P\rangle \text{ with } P = \{\langle s, \sigma(s_1 \cdots s_n)\rangle \mid \sigma \in \Sigma_{s_1 \cdots s_n, s}\}.$$

Then one can check that, under the identification of the operator symbol $\langle s, \sigma(s_1 \cdots s_n)\rangle$ in $\hat{\Sigma}$ with σ in Σ, $T_{\hat{\Sigma}}$ and T_Σ are isomorphic.

The impact of initial algebra semantics here is that any G-algebra whatsoever (a set S_A for each nonterminal A and a function $p_S : S_{A_1} \times \cdots \times S_{A_n} \to S_A$ for each production p

[13] Usually N, Σ, and P are finite (see [19]) We don't have (or want) that restriction here We believe that the context-free grammar for the abstract syntax of many programming languages will have an infinite number of terminals, nonterminals, and productions We don't consider the effective presentation of such systems here (see van Wijngaarden [77]), but this must be carefully thought out

of type $\langle A_1 \cdots A_n, A\rangle$) provides a semantics for the context-free language generated by G. T_G, being initial, gives the unique homomorphism $h_S : T_G \to S$ which assigns "meanings" in S to all syntactically well-formed phrases of the language (not just to the "sentences" generated from some specified start symbol in N).

For example, we make Σ^* into a G-algebra (with every carrier Σ^*) by letting $p(v_1, \cdots, v_n) = u_0v_1u_1v_2 \cdots u_{n-1}v_nu_n$ for $p = \langle A_0, u_0A_1u_1A_2 \cdots u_{n-1}A_nu_n\rangle$. Then the unique homomorphism $d : T_G \to \Sigma^*$ assigns to a derivation $t \in T_{G,A}$ the string which is derived.[14] Note that $d : T_{\hat{\Sigma}} \to (\Sigma \cup \{(,)\})^*$ gives exactly the well-formed Σ-expressions corresponding to parse trees in $T_{\hat{\Sigma}}$; this is the basis for proving the isomorphism of $T_{\hat{\Sigma}}$ and T_Σ.

G might be ambiguous (d not injective), but that is not a problem since initial algebra semantics (the abstract syntax point of view) factors out the parsing problem (without, of course, eliminating it).[15]

Thus even "string generated" is an initial algebra semantics for context-free grammars. Knuth's [30] synthesized attributes (following Irons's [29] ideas) really do what we describe here, but Knuth's definitions and notation seem more complex than is necessary. In 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 Knuth's "semantic rules" define (by the components) operations of the appropriate type for each production. Of course, semantics of the language is the unique homomorphism $h_s \cdot T_G \to S$.[16]

3.2. DENOTATIONAL SEMANTICS Scott [63] begins with the motto "Extend BNF to semantics." In fact, that is the "moral" of the previous discussion. We now know that BNF[17] yields an initial many-sorted algebra, that semantics arises from an algebra of the same type, and that "meaning" is the unique homomorphism. We view the principal achievement of Scott and Strachey (say [60, 64, 65]) 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 *intended* (denotational) meaning of the language.

Scott and Strachey seem to say (e.g. [65, p. 17]) that 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 its constituent phrases. (From [65, p. 12], "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.") This essentially says that syntax is context-free and semantics is a homomorphism.[18]

In general, the "semantic equations" (typical of [65, 26], etc.) define the meaning of a syntactic construct C as a function F_C of the meanings of the components to that construct, and in so doing the semantic equations describe an algebra (the function F_C is the operation corresponding to the syntactic construct C) *and* say that semantics is a

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

[15] As Schwartz [59] 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 program, into a standard starting point for our thinking on program semantics."

[16] Knuth's [30] inherited attributes are not treated in this paper, but Goguen and Zamfir (unpublished) have suggested treating them as variables (that is, new uninterpreted constants) in free algebras whose operator domain contains, in addition to these variable symbols for inherited attributes, all the operations and constants which are employed in semantic definitions. Burstall (personal communication) has suggested using $[I \to S]$ as the carrier of a generalized semantic algebra. These issues will hopefully be explored elsewhere. On the other hand, Knuth's [30] main result, an algorithm testing for circularity of a semantic definition involving both inherited and synthesized attributes, lies outside the domain of interest of this paper.

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

[18] The homomorphic character of the semantic function must have been clear to Scott. In fact, he uses the phrase "continuous algebraic homomorphism" [61, p. 46] for the semantic function on the lattice of flow diagrams (See Section 5.3).

homomorphism. Usually semantic equations give the meaning with respect to an "environment," for example [65, p. 27],

$$\mathscr{C}[\![\epsilon \to \gamma_0, \gamma_1]\!](\rho) = cond(\mathscr{C}[\![\gamma_0]\!](\rho), \mathscr{C}[\![\gamma_1]\!](\rho)) * \mathscr{E}[\![\epsilon]\!],$$

where ρ is an environment; $\mathscr{C}$ is the semantic function for commands with target, say, Γ; and $\mathscr{E}$ is the semantic function for expressions with target $T = \{\textbf{true}, \textbf{false}\}$. The initial algebra "translation" of that semantic equation abstracts the environment variable to get

$$\mathscr{C}[\![\epsilon \to \gamma_0, \gamma_1]\!] = \lambda\rho\ cond(\mathscr{C}[\![\gamma_0]\!](\rho), \mathscr{C}[\![\gamma_1]\!](\rho)) * \mathscr{E}[\![\epsilon]\!].$$

Now this has exactly the form of the homomorphism condition,

$$\mathscr{C}[\![\ \to (\epsilon, \gamma_0, \gamma_1)]\!] = F_{\to}(\mathscr{E}[\![\epsilon]\!], \mathscr{C}[\![\gamma_0]\!], \mathscr{C}[\![\gamma_1]\!]),$$

and the operation (in the semantic target algebra) is $F_{\to} : T \times \Gamma^2 \to \Gamma$, defined for *all* $\bar{\epsilon} \in T$ and $\bar{\gamma}_0, \bar{\gamma}_1 \in \Gamma$ in terms of *cond*, *, and abstraction:

$$F_{\to}(\bar{\epsilon}, \bar{\gamma}_0, \bar{\gamma}_1) = \lambda\rho\ cond(\bar{\gamma}_0(\rho), \bar{\gamma}_1(\rho)) * \bar{\epsilon}.$$

The point is that one need only define the target algebra; initiality yields the semantic homomorphism.

We illustrate this in more detail with a *s*imple *a*pplicative *l*anguage which we call SAL, drawn (and modified) from Reynolds [53].[19] Let $X = \{x_1, x_2, \cdots\}$ be a set of *variables* (identifiers). In specifying SAL syntax we give each rule a name in order 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 boldfaced symbols are terminals. Thus variables occur as terminal symbols, just as, say, **if** does. The (context-free) productions of SAL are the following[20]:

(v_x)	⟨exp⟩	::=	x
(cond)	⟨exp⟩	::=	**if** ⟨exp⟩ **then** ⟨exp⟩ **else** ⟨exp⟩
(apply)	⟨exp⟩	::=	⟨exp⟩ **(**⟨exp⟩**)**
(abs_x)	⟨exp⟩	::=	**lambda** x **(**⟨exp⟩**)**
(let_x)	⟨exp⟩	::=	**let** x **be** ⟨exp⟩ **in** ⟨exp⟩
($let\text{-}rec_x$)	⟨exp⟩	::=	**let recursive** x **be** ⟨exp⟩ **in** ⟨exp⟩

From Scott [63] there is a complete lattice[21] V of *values* satisfying the isomorphism,

$$V \cong I + B + [V \to V], \qquad (*)$$

where I and B represent respectively the integers and the Boolean values $\{\textbf{true}, \textbf{false}\}$ as lattices with $\bot$ and $\top$ adjoined, and $[V \to V]$ is the complete lattice of continuous functions from V to V. The +-operation on the right-hand side of (*) is the "coalesced lattice sum"[22] (disjoint union with minimum (and maximum) elements identified). For $X \in \{I, B, [V \to V]\}$, let j_X be the injection $j_X : X \to I + B + [V \to V]$, and let ϕ be the isomorphism $\phi : V \cong I + B + [V \to V]$. Then V is equipped with injection $\iota_X : X \to V$ ($\iota_X = \phi^{-1} \circ j_X$) and also "projections" $\pi_x : V \to X$ such that

$$\pi_X(V) = \begin{cases} \phi(V) & \text{if } \phi(V) \in X, \\ \bot & \text{otherwise} \end{cases}$$

[19] Our language differs substantially from Reynolds's in that his employs a call-by-value interpretation of application

[20] The fact that this is an ambiguous grammar is irrelevant for our purposes

[21] We discuss completeness, continuity, and alternatives to lattices in Section 4 For now, a *complete* lattice is a partially ordered set where every subset has a least upper bound, a function is *continuous* iff it preserves least upper bounds of directed subsets, and D is *directed* iff every finite subset of D has an upper bound in D The minimum and maximum elements of a complete lattice are denoted $\bot$ and $\top$, respectively

[22] For lattices (and analogously for posets, with only $\bot$), the *coalesced sum* of two lattices L_1 and L_2 consists of the disjoint union (say) $L_1 \times \{1\} \cup L_2 \times \{2\}$ with $\langle \bot, 1\rangle$ and $\langle \top, 1\rangle$ identified respectively with $\langle \bot, 2\rangle$, $\langle \top, 2\rangle$ under the obvious ordering The *separated sum* is the disjoint union with new $\bot$ and $\top$ adjoined, $\bot \subseteq \langle l, \iota\rangle \subseteq \top$ for all $l \in L_\iota$ and $\iota \in \{1, 2\}$.

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 componentwise 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, $M = [E \to V]$.

M becomes a semantic algebra when we define an operation on M for each production of SAL; let_x, $let\text{-}rec_x$, and *apply* are of rank 2, functions $M^2 \to M$; *cond* is rank 3; abs_x is rank 1; and v_x is rank 0, a constant.

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

(3.1) $access_x : E \to V$ is the evaluation function on V^X at $x \in X$: $access_x(e) = e(x)$.

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

$$c(v_1, v_2, v_3) = \begin{cases} \top & \text{if } v_1 = \top, \\ v_2 & \text{if } v_1 = \textbf{true}, \\ v_3 & \text{if } v_1 = \textbf{false}, \\ \bot & \text{otherwise.} \end{cases}$$

(3.3) $ap : V^2 \to V$ is application: $ap(v_1, v_2) = (\pi_{[V \to V]}(v_1))(v_2)$, where $\pi_{[V \to V]}$ is the indicated projection; see discussion following (*) above.

(3.4) $assign_x : E \times V \to E$ is our friend the assignment operator:

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

All of the above have either been shown to be continuous or can easily be shown to be such. In addition, Scott [63] *defines*

(3.5) $abstract : [D \times D' \to D''] \to [D \to [D' \to D'']]$ by $((abstract(f))(x))(y) = f(x,y)$ for all $f \in [D \times D' \to D'']$, $x \in D$, and $y \in D'$.

Here we too enjoy some notational ambiguity by writing "*abstract*" instead of "$abstract_{D,D',D''}$"; and similarly in other cases.

(3.6) The least fixed point operator $Y : [D \to D] \to D$ is continuous.

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

(3.8) For continuous functions $f : D \to D'$ and $f' : D' \to D''$, the composite $f' \circ f$ is continuous; and so is composition itself, $\circ : ([D' \to D''] \times [D \to D']) \to [D \to D'']$.

Now make $M = [V^X \to V]$ into a SAL-algebra by giving an appropriate operation for each production of SAL:

$v_x = access_x$

$cond(m_1, m_2, m_3) = c \circ [m_1, m_2, m_3]$

$apply(m_1, m_2) = ap \circ [m_1, m_2]$

$abs_x(m) = \iota_{[V \to V]} \circ abstract(m \circ assign_x)$

$let_x(m_1, m_2) = m_2 \circ assign_x \circ [1_E, m_1]$

$let\text{-}rec_x(m_1, m_2) = let_x(y \circ abstract(m_1 \circ assign_x), m_2)$

Actually the definitions of most of these operations seem to be clearer as diagrams, since the source and target of each function is explicit. "Abstraction" is regarded as a way of getting one function from another The function "assumed" (or given) appears on the top, the function "deduced" appears below it, and the process is represented by a line between them; thus

$$\frac{D \times D' \xrightarrow{f} D''}{D \xrightarrow{g} [D' \to D'']}$$

for $g = abstract(f)$. Then the first five definitions are:

(1) $v_x = E \xrightarrow{access} V$

(2) $cond(m_1, m_2, m_3) = E \xrightarrow{[m_1, m_2, m_3]_x} V^3 \xrightarrow{c} V$

(3) $apply(m_1, m_2) = E \xrightarrow{[m_1, m_2]} V^2 \xrightarrow{ap} V$

(4) $abs_x(m) = \dfrac{E \times V \xrightarrow{assign_x} E \xrightarrow{m} V}{E \to [V \to V] \xrightarrow{\psi(1 \to 1)} V}$

(5) $let_x(m_1, m_2) = E \xrightarrow{[1_E, m_1]} E \times V \xrightarrow{assign_x} E \xrightarrow{m_2} V$

With M a SAL-algebra, the unique homomorphism $h_M : T_G \to M$ is a semantics for SAL. There is no claim for great gains in perspicuity, but our mode of definition does illuminate the places where semantic choices are made. It is through the introduction of standard mathematical functions—composition, tupling, injection, and the like—that these choices are clarified, and this process has the additional benefit of getting rid of the need for "dragging along" the variable for environments.

Given the particular conditional and application functions ((3.2) and (3.3) respectively), then the functions *cond* and *apply* ((2) and (3) respectively) on the semantic carrier M are essentially the only choices possible; this differs from Reynold's meta-circular interpreters in that he presumes a "defining language" in which the interpreters are written and the semantic choices have not been completely made for that meta-language. In the semantic carrier $E \to V$, there is simply no way to make a distinction between order of evaluation of operator and operand in (2), for example. If expressions had side effects and $E \to E \times V$ were taken as carrier, then the two orders of evaluation would be clearly represented in

$$E \xrightarrow{m_2} E \times V \xrightarrow{m_1 \times 1_V} E \times V \times V \xrightarrow{1_E \times ap} E \times V$$

and

$$E \xrightarrow{m_1} E \times V \xrightarrow{m_2 \times 1_V} E \times V \times V \xrightarrow{1_E \times ap'} E \times V,$$

where $ap'(v_1, v_2) = ap(v_2, v_1) = v_2(v_1)$ and $f \times g(a, b) = \langle f(a), g(b) \rangle$

3.3. Syntax Directed Translation. The notion of syntax directed translation pioneered by Irons [29] has been mathematically treated by several authors [1–3, 30, 35, 50, 63, 73]. We can view these technical interpretations of syntax directed translation as special cases 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 [19] 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 generalized sequential machine 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 or parse trees

Thatcher [69] 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 3.1 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 after making $(T_{\Sigma'})^k$ into a Σ-algebra. The set $[k] = \{1, 2, \cdots, k\}$ is the set of *states*. If $h(t) = \langle t_1, \cdots, t_k \rangle$, then t_i is the image of t under the syntax map from *start state* i. □

The standard example (cf. Rounds [57], Thatcher [69]) is polynomial 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 $(T_\Sigma)^2$:

$$\langle t_1', t_1\rangle + \langle t_2', t_2\rangle = \langle t_1' + t_2', t_1 + t_2\rangle$$
$$\langle t_1', t_1\rangle \times \langle t_2', t_2\rangle = \langle (t_1' \times t_2) + (t_1 \times t_2'), t_1 \times t_2\rangle$$
$$0 = \langle 0, 0\rangle$$
$$1 = \langle 0, 1\rangle$$
$$x = \langle 1, x\rangle$$

These "equations," especially the last three, may be somewhat confusing; of course the symbols $+$, $\times$, 0, 1, x are from Σ and, for example, the last equation says that x in Σ names the pair $\langle 1, x\rangle$ in $(T_\Sigma)^2$ of objects named by 1 and x (each in T_Σ).

Now $h : T_\Sigma \to (T_\Sigma)^2$ yields $h(t) = \langle t', t\rangle$; 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 [69] "root to frontier automaton with output" (called "finite state transformation" in [68]). Intuitively, if $t = \sigma(t', t'')$, t_ι' is the translation of t' from root (start) state ι, and t_ι'' is the translation of t'' from state ι, then $\sigma(\langle t_1', \cdots, t_k'\rangle, \langle t_1'', \cdots, t_k''\rangle)_\jmath$ is the translation of $\sigma(t', t'')$ from state $\jmath$.

If $k = 1$ and the Σ-structure on $T_{\Sigma'}$ is *uniform* in the sense that for each $\sigma \in \Sigma_n$ there is a term t_σ in $T_{\Sigma'}(X_n)$ such that for all $t' \in T_{\Sigma'}$, $\sigma(t_1', \cdots, t_n') = t_\sigma \leftarrow \langle t_1', \cdots, t_n'\rangle$, 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 Stearns [34] 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].

The other case of syntax directed translation we want to consider is frontier to root translation. Outside tree automata theory (i.e. outside Baker [3], Magidor and Moran [40], Thatcher [69], 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 the definition of finite state frontier to root syntax map is remarkably simple and particularly interesting in that it is obtained by replacing "kth power" in Definition 3.1 by "product with k." (More technically, we replace the "k-fold product of $T_{\Sigma'}$" with its category theoretic dual, the "k-fold coproduct of $T_{\Sigma'}$.")

Definition 3 2. A *k-state frontier to root syntax map* from T_Σ to $T_{\Sigma'}$ is the unique homomorphism $h : T_\Sigma \to [k] \times T_{\Sigma'}$ guaranteed by initiality of T_Σ after making $[k] \times T_{\Sigma'}$ into a Σ-algebra. The set $[k] = \{1, 2, \cdots, k\}$ is the set of *states*. If $h(t) = \langle \iota, t'\rangle$, then t' is the image of t and the *terminating state* is ι. □

For example, a tree automaton (Doner [15], Thatcher and Wright [70]) with transition function $M : \Sigma \times [k]^n \to [k]$ is imitated as a frontier to root syntax map by making $[k] \times T_\Sigma$ into a Σ-algebra with $\sigma(\langle i_1, t_1\rangle, \cdots, \langle i_n, t_n\rangle) = \langle M(\sigma, i_1, \cdots, i_n), \sigma(t_1 \cdots t_n)\rangle$, the second component is the identity.

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

(0) For $\sigma \in \Sigma_0$, $\sigma_{\boldsymbol{p}A} = \{\sigma_A\}$.

(1) For $n > 0$, $\sigma \in \Sigma_n$, and $u_1, \cdots, u_n \in \boldsymbol{p}A$,

$$\sigma_{\boldsymbol{p}A}(u_1, \cdots, u_n) = \{\sigma_A(t_1, \cdots, t_n) \mid t_\iota \in u_\iota, 1 \leq \iota \leq n\}.$$

Associated with each $t \in T_\Sigma(X_n)$ is a derived operator $t_{\boldsymbol{p}A} : (\boldsymbol{p}A)^n \to \boldsymbol{p}A$; extending this idea to sets of terms, if $r \subseteq \boldsymbol{p}T_\Sigma(x_n)$, define

$$r_{\boldsymbol{p}A}(u_1, \cdots, u_n) = \cup\{t_{\boldsymbol{p}A}(u_1, \cdots, u_n) \mid t \in r\}.$$

$E : X_n \to \boldsymbol{p}T_\Sigma(X_n)$ is called a *system of equations* (in n variables) which determines a function $E_{\boldsymbol{p}A} : (\boldsymbol{p}A)^n \to (\boldsymbol{p}A)^n$, by $E_{\boldsymbol{p}A} = [E(x_1)_{\boldsymbol{p}A}, \cdots, E(x_n)_{\boldsymbol{p}A}]$ (target-tupling), called the *system function* by Mezei and Wright [45], who show that $E_{\boldsymbol{p}A}$ is ω-continuous, i.e. least upper bounds of ω-chains in $(\boldsymbol{p}A)^n$ are preserved. Therefore $E_{\boldsymbol{p}A}$ has a minimum fixed point, called its *solution*, say $\langle s_1, \cdots, s_n\rangle \in (\boldsymbol{p}A)^n$; in fact,

$$\langle s_1, \cdots, s_n\rangle = \cup_k E_{\boldsymbol{p}A}^k(\varnothing, \cdots, \varnothing).$$

A subset of A is *equational* iff it is a component of a solution of a system of equations. The entire theory of equational sets (cf. Mezei and Wright [45], Eilenberg and Wright [16], Blikle [8], and Nivat [48]) flows from these definitions, from initiality (to get derived operators) and from the fixed-point theorem (to get solutions).

4 *Continuous Algebras*

We begin with some preliminaries on partially ordered sets (posets) and the poset of partial functions in particular. Ordered structures have played an important role in many different approaches to the semantics of programming; see Bekić [4], Gordon [26], Manna [43], Nivat [48], Park [49], Reynolds [53, 54], Scott [61–64], Scott and Strachey [65], Wagner [73, 74], and Wand [75], among others. This diversity of effort has resulted in a diversity of terminology, leading to a certain amount of confusion – heightened by the fact that the various key words have been used by different authors for different concepts. "Complete" sometimes refers to the existence of least upper bounds of arbitrary sets [62], or of directed sets [26]; and "continuous" modifies lattices [62], algebraic theories [11], categories [23], algebras here, and most often, functions Scott [64] and Reynolds [53, 54] speak of "domains" as complete, countably based, continuous lattices, but then Gordon [26] argues that "semidomains" are more appropriate than "domains," where his domains are complete lattices and his semidomains are directed-set-complete posets.

In addition, it is not clear that there is just one "right" order structure for semantics. Thus in this paper we suggest a terminology and mathematical development which hopefully eliminates confusion and permits the expression of general theorems about varied forms of completeness and continuity. Our presentation is admittedly biased toward posets over lattices,[23] based in part on the sometimes greater complexity of the structures which the lattice approach generates, as well as upon the difficulty of giving concrete computational interpretations to these additional elements when they arise. At first sight it might seem that there is no real difficulty, because one can merely add enough "overdefined" elements to a sufficiently complete poset to get a complete lattice (and often one element "$\top$" will do; see (4.3) below). But more sophisticated constructions, when constrained to start with lattices, may produce far more than one additional element. For example, coproducts (over index set I) produce free (generated by I) Boolean algebras of overdefined elements, and our initial continuous algebra construction produces an infinite and unruly collection of objects involving substitution instances of $\top$ in infinite trees (see discussion after Proposition 4.2)

The reader should bear in mind, however, that the algebraic approach does not impose any choice between (say) directed complete posets and lattices. It is the marriage between the algebraic and the order theoretic approaches that is important.

Following Scott, we use $\sqsubseteq$ for the order in a poset P, $\sqcup$ (read "cup") for binary (or finite) least upper bounds, and $\bigsqcup$ (read "mug") 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. A subset S of P is *directed* iff every finite subset of S has an upper bound in S A function $f : P \to P'$ from a poset P to a poset P' is *monotonic* iff for all $p_0 \sqsubseteq p_1$ in P, $f(p_0) \sqsubseteq f(p_1)$ in P'.

In order to be able to introduce concepts such as "complete" and "continuous" in a uniform manner, it is convenient to introduce the following notation. We say a subset S of P is a

$\sqcup$-set	if S if finite nonempty,
ω-set	if S is an ω-chain,
ℓ-set	if S is nonempty, and linearly ordered (i.e. a chain),
Δ-set	if S is directed,
$\bigsqcup$-set	if S is nonempty

[23] For further discussions of the relative merits of lattices versus posets see after Corollary 4 12 and also Gordon [26] and Lewis and Rosen [33, Pt 2]

For each symbol $Z \in \{\sqcup, \omega, \ell, \Delta, \dot{\sqcup}\}$, a function $f : P \to P'$ is *Z-continuous* iff it preserves all least upper bounds (that exist) of Z-sets in P; a poset P is *Z-complete* iff all Z-sets have least upper bounds in P. Each of these symbols can be modified by a "dot" to indicate boundedness: A *Ż-set* is a Z-set with an upper bound in P and, e.g., a poset is *Ż*-complete iff each bounded Z-set has a least upper bound in P. P is *Z-bounded* iff every Z-set has an upper bound in P (is a *Ż*-set). Finally we need the modifier "strict": P is *strict* iff P contains a minimum element $\perp$ (we only consider strict posets here): $f : P \to P'$ is *strict* iff it preserves $\perp$.

Examples of the use of these conventions are:

(4.1) A function $f : P \to P'$ is Z-continuous iff it is Z-continuous.

(4.2) A poset is ℓ-complete iff it is Δ-complete (cf. [13] and [44]).

(4.3) Let $P^\top$ be the poset P with $\top$ adjoined (i.e. $\top \notin P$ and $p \sqsubseteq \top$ for all $p \in P$). Then $P^\top$ is $\dot{\sqcup}$-complete (a complete lattice) iff P is $\dot{\sqcup}$-complete; and $P^\top$ is $\sqcup$-complete (an upper-semilattice) iff P is $\sqcup$-complete.

(4.4) Some of the modifier symbols are ordered ($\omega \subseteq \ell \subseteq \Delta \subseteq \dot{\sqcup}$), so P $\dot{\sqcup}$-complete implies P Δ-complete implies P ℓ-complete implies P ω-complete.

(4.5) $S \subseteq P$ is directed iff S is nonempty and $\sqcup$-bounded.

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

(4.7) Least upper bounds of Z-sets work by components on Cartesian products, i.e.

$$\bigsqcup_{i \in I} \langle p_{1,i}, \cdots, p_{k,i} \rangle = \langle \bigsqcup_{i \in I} p_{1,i}, \cdots, \bigsqcup_{i \in I} p_{k,i} \rangle$$

for any Z-set $\{\langle p_{1,i}, \cdots, p_{k,i} \rangle\}_{i \in I}$ in $P_1 \times \cdots \times P_k$

(4.8) For $Z \in \{\omega, \ell, \Delta\}$, $f : P_1 \times \cdots \times P_k \to R$ is Z-continuous iff f is Z-continuous componentwise, i.e. iff

$$f(\bigsqcup_{i_1 \in I_1} p_{1,i_1}, \cdots, \bigsqcup_{i_k \in I_k} p_{k,i_k}) = \bigsqcup_{\langle i_1, \ldots, i_k \rangle \in I_1 \times \cdots \times I_k} f(p_{1,i_1}, \cdots, p_{k,i_k}).$$

For sets A, B, let $[A \rightharpoonup B]$ be the poset of *partial* functions[24] from A to B. Since we have fixed source and target, the elements of $[A \rightharpoonup B]$ correspond to *functional* subsets of $A \times B$; i.e. if $\langle a, b \rangle \in f$ and $\langle a, b' \rangle \in f$, then $b = b'$. The order relation on $[A \rightharpoonup B]$ is simply set inclusion, and the least upper bound (when it exists) is set union; it exists iff the set union is a function For $f : A \rightharpoonup B$ and $C \subseteq A$, $f \restriction C = \{\langle a, b \rangle \mid a \in C$ and $\langle a, b \rangle \in f\}$ is the *restriction* of f to C; $def(f) = \{a \mid \langle a, b \rangle \in f\}$ is the *domain of definition* of f. The following are well-known facts about $[A \rightharpoonup B]$.

(4.9) $[A \rightharpoonup B]$ is strict Δ-complete and $\dot{\sqcup}$-complete.

(4.10) If $f \sqsubseteq \bigsqcup_{i \in I} f_i$ in $[A \rightharpoonup B]$ where $def(f)$ is finite and $\{f_i\}_{i \in I}$ is directed, then $f \sqsubseteq f_j$ for some $j \in I$. This says the finite partial functions are *Δ-compact*

So much for preliminaries We now turn to the definition of "Σ-trees" which provide the carriers of initial "continuous algebras" The idea is based on the well-known device of representing a Σ-tree as a function defined on a prefix closed subset of ω^* (called a tree domain; see [67]) taking values in Σ In Figure 1(a) ($\Sigma_0 = \{b, x_0, x_1\}$, $\Sigma_2 = \{f\}$, and $\Sigma_i = \varnothing$ for $i \neq 0, 2$}), the tree (expression $f(f(f(x_0x_1)b)x_0)$) is represented by the function $t : \{\lambda, 0, 1, 00, 01, 000, 001\} \to \{f, b, x_0, x_1\}$ where $\lambda, 0, 00 \mapsto f$; $1,000 \mapsto x_0$; $01 \mapsto b$; and $001 \mapsto x_1$. The first generalization considers the representation as a partial function on ω^* for which the domain of definition has the tree domain property. This subsumes the previous formulation and immediately includes infinite trees. Figure 1(b) is such, represented by $t : \omega^* \rightharpoonup \{f, b\}$ with $0^* \mapsto f$; $0^*1 \mapsto b$; and t undefined otherwise. (This example solves the equation $x = f(x, b)$; see Section 5.1.) Finally we allow trees to be partial, the effect being that leaves can be unlabeled (or labeled with $\perp$; see Proposition 4.4). The third tree in the sequence (chain) of Figure 2 is the partial function $t : \omega^* \rightharpoonup$

[24] F $A \rightharpoonup B$ designates a function A to B with a possible "hole"

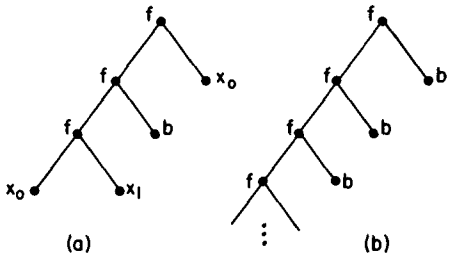


FIG 1

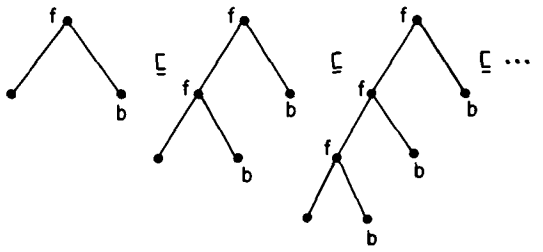


FIG 2

$\{f, b\}$ with $\lambda, 0, 00 \mapsto f$; $1, 01, 001 \mapsto b$, and undefined otherwise. The natural order relation on these trees is the one obtained from $[\omega^* \multimap \{b, f\}]$, and the least upper bound of the chain of Figure 2 is the tree of Figure 1(b).

We now give the precise definition of Σ-tree in the simpler one-sorted case and consider many-sorted operator domains at the end of this section

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

(a) $ui \in def(t)$ implies $u \in def(t)$;

(b) $ui \in def(t)$ implies $t(u) \in \Sigma_n$ and $i < n$ for some $n > 0$. (Note that $ui \in def(t)$ does *not* imply $uk \in def(t)$ for any other k.)

Let CT_Σ denote the set of Σ-trees; and let F_Σ denote the subset of all finite Σ-trees (those trees t for which $def(t)$ is finite) The ordering on $[\omega^* \multimap \Sigma]$ induces an ordering $\sqsubseteq$ on CT_Σ (and F_Σ). If a set of partial functions satisfying (a) and (b) has a least upper bound (set union) f in $[\omega^* \multimap \Sigma]$, then f also satisfies (a) and (b); so least upper bounds in $[\omega^* \multimap \Sigma]$ are least upper bounds in CT_Σ Thus, from (4.9),

PROPOSITION 4.1 *CT_Σ is a strict Δ-complete and $\dot{\sqcup}$-complete poset.* □

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

PROPOSITION 4.2 *For any $t \in CT_\Sigma$, $t = \bigsqcup_{n \in \omega} t^{(n)}$ and $t^{(n)} \in F_\Sigma$.* □

Now make CT_Σ into a Σ-algebra.

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

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

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

If $t_1, \cdots, t_n \in CT_\Sigma$ and $\sigma \in \Sigma_n$, then $\sigma_{CT}(t_1, \cdots, t_n) \in CT_\Sigma$; in fact, if the t_i are finite, then so is $\sigma_{CT}(t_1, \cdots, t_n)$. Thus we have also given Σ-structure to F_Σ

The ordering on CT_Σ is related to this algebraic structure as follows

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

PROOF. Both directions are quite clear from the definitions, especially sufficiency For necessity, if $def(t) = \varnothing$, then $t = \perp$. If $def(t) \neq \varnothing$, then $\lambda \in def(t)$ and $t(\lambda) = t'(\lambda)$;

so if $t(\lambda) \in \Sigma_n$, define $t_i = \{\langle u, \sigma' \rangle \mid \langle iu, \sigma' \rangle \in t\}$ and $t'_i = \{\langle u, \sigma' \rangle \mid \langle iu, \sigma' \rangle \in t'\}$. By the definition of operations σ_{CT} and the order relation on $[\omega^* \nrightarrow \Sigma]$, $t = \sigma_{CT}(t_1, \cdots, t_n)$, $t' = \sigma_{CT}(t'_1, \cdots, t'_n)$, and $t_i \sqsubseteq t'_i$. □

PROPOSITION 4.4. *F_Σ is the free Σ-algebra on one generator $\perp$; i.e. it is an intial $\Sigma(\perp)$-algebra.* □

We won't burden the reader with a proof of Proposition 4.4 here because it is fairly obvious and a good exercise. It can be done in two ways: either show there is a unique $\Sigma(\perp)$-homomorphism $h_A : F_\Sigma \to A$ for any other $\Sigma(\perp)$-algebra A, or show that F_Σ is isomorphic to (say) the usual algebra of $\Sigma(\perp)$-expressions (defined in Section 2) 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.*

PROOF. Assume $\bigsqcup_{i \in I} \langle t_{1,i}, \cdots, t_{n,i} \rangle$ exists in CT_Σ^n. Then by (4.7) it equals $\langle \bigsqcup_{i \in I} t_{1,i}, \cdots, \bigsqcup_{i \in I} t_{n,i} \rangle$, and we need

$$\sigma_{CT}(\bigsqcup_{i \in I} t_{1,i}, \cdots, \bigsqcup_{i \in I} t_{n,i}) = \bigsqcup_{i \in I} \sigma_{CT}(t_{1,i}, \cdots, t_{n,i}).$$

The left-hand side is

$$\begin{aligned} &\{\langle \lambda, \sigma \rangle\} \cup \bigcup_{j<n} \{\langle ju, \sigma' \rangle \mid \langle u, \sigma' \rangle \in \bigsqcup_{i \in I} t_{j+1,i}\} \\ &\quad = \{\langle \lambda, \sigma \rangle\} \cup \bigcup_{j<n} \bigcup_{i \in I} \{\langle ju, \sigma' \rangle \mid \langle u, \sigma' \rangle \in t_{j+1,i}\} \\ &\quad = \bigcup_{i \in I} (\{\langle \lambda, \sigma \rangle\} \cup \bigcup_{j<n} \{\langle ju, \sigma' \rangle \mid \langle u, \sigma' \rangle \in t_{j+1,i}\}), \end{aligned}$$

which is the right-hand side. □

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 strict monotonic Σ-homomorphism. Let $\boldsymbol{Palg}_\Sigma$ be a class of ordered Σ-algebras. Since the operations of F_Σ are monotonic (Proposition 4.6 and (4.6)), F_Σ is an ordered Σ-algebra. In fact,

PROPOSITION 4.7. *F_Σ is intial in $\boldsymbol{Palg}_\Sigma$.*

PROOF. Let A be an ordered Σ-algebra; make it into a $\Sigma(\perp)$-algebra by letting $\perp_A$ be the minimum element. Then by Proposition 4.4 there is a unique $\Sigma(\perp)$-homomorphism $h : F_\Sigma \to A$ which, of course, is also a Σ-homomorphism $F_\Sigma \to A$ preserving $\perp$. We show h is monotonic. If $t \sqsubseteq t'$ in F_Σ, then by Proposition 4.3 either $t = \perp$ (in which case $h(t) = \perp_A \sqsubseteq h(t')$), or $t = t' = \sigma_{CT} \in \Sigma_0$ (in which case $h(t) \sqsubseteq h(t')$), or there are $\sigma \in \Sigma_n$ and $t_1, \cdots, t_n$, $t'_1, \cdots, t'_n$ in CT_Σ such that $t = \sigma_{CT}(t_1, \cdots, t_n) \sqsubseteq \sigma_{CT}(t'_1, \cdots, t'_n) = t'$ and $t_i \sqsubseteq t'_i$. By induction on the cardinality of $def(t)$, we can assume $h(t_i) \sqsubseteq h(t'_i)$. But $h(t) = h(\sigma_{CT}(t_1, \cdots, t_n)) = \sigma_A(h(t_1), \cdots, h(t_n)) \sqsubseteq$ (monotonicity of σ_A) $\sigma_A(h(t'_1, \cdots, h(t'_n)) = h(\sigma_{CT}(t'_1, \cdots, t'_n)) = h(t')$. □

A Σ-algebra is *ω-continuous* iff its carrier is strict ω-complete 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 intial in the class of ω-continuous Σ-algebras with strict Δ-continuous Σ-homomorphisms.*

PROOF. The main idea is that every ω-continuous Σ-algebra A is in $\boldsymbol{Palg}_\Sigma$ so there is a unique homomorphism $h_A : F_\Sigma \to A$ which can be extended using Proposition 4.2 : $\bar{h}_A(t) = \bigsqcup_{n \in \omega} h_A(t^{(n)})$. We must prove $\bar{h}_A$ is a Σ-homomorphism, is Δ-continuous, and is unique.

Let A be an ω-continuous Σ-algebra. It is also in $\boldsymbol{Palg}_\Sigma$, so by Proposition 4.7 there is a unique (monotonic) Σ-homomorphism $h_A : F_\Sigma \to A$. We show it has a unique Δ-continuous extension $\bar{h}_A : CT_\Sigma \to A$ which is a Σ-homomorphism. In fact, define $\bar{h}_A(t) =$

$\bigsqcup_n h_A(t^{(n)})$ which exists because A is ω-complete.[25] Now uniqueness: If h' extends h_A and is Δ-continuous (even ω-continuous), then $h'(t) = h'(\bigsqcup_n t^{(n)} = \bigsqcup_n h'(t^{(n)}) = \bigsqcup_n h_A(t^{(n)}) = \bar{h}_A(t)$.

Next, $\bar{h}_A$ is monotonic Assume $t_0 \sqsubseteq t_1$ in CT_Σ with $t_0 = \bigsqcup_n t_0^{(n)}$ and $t_1 = \bigsqcup_n t_1^{(n)}$. For each k, $t_0^{(k)} \sqsubseteq t_1^{(k)}$; by monotonicity of h_A, $h_A(t_0^{(k)}) \sqsubseteq h_A(t_1^{(k)})$; and thus $\bar{h}_A(t_0) = \bigsqcup_k h_A(t_0^{(k)}) \sqsubseteq \bigsqcup_k h_A(t_1^{(k)}) = \bar{h}_A(t_1)$.

Now we show $\bar{h}_A$ is Δ-continuous. Assume $t = \bigsqcup_{\iota \in I} t_\iota$ for some directed set $\{t_\iota\}_{\iota \in I}$ in CT_Σ. We must show that $\bar{h}_A(t) = \bigsqcup_{\iota \in I} \bar{h}_A(t_\iota)$. First, $t_\iota \sqsubseteq t$ for all ι, so by monotonicity of $\bar{h}_A$, $\bar{h}_A(t_\iota) \sqsubseteq \bar{h}_A(t)$ for all ι; i.e. $\bar{h}_A(t)$ is an upper bound for $\{\bar{h}_A(t_\iota)\}_{\iota \in I}$. Let b be any other upper bound $\bar{h}_A(t_\iota) \sqsubseteq b$ for all $\iota \in I$. For each n, $t^{(n)} \sqsubseteq t = \bigsqcup_{\iota \in I} t_\iota$, and $\{t_\iota\}$ is directed; so by (4.10), $t^{(n)} \sqsubseteq t_j$ for some j. Thus for each n there is some j with $\bar{h}_A(t^{(n)}) \sqsubseteq \bar{h}_A(t_j) \sqsubseteq b$; i.e. $\bigsqcup_n \bar{h}_A(t^{(n)}) = \bar{h}_A(t) \sqsubseteq b$. Thus we have $\bar{h}_A(t) = \bigsqcup_{\iota \in I} \bar{h}_A(t_\iota)$ as required. (This may be surprising; since A is only ω-complete, there is no a priori reason for $\bigsqcup_{\iota \in I} \bar{h}_A(t_\iota)$ to exist.)

Now we have only to show that $\bar{h}_A$ is indeed a Σ-homomorphism. First check that $\sigma_{CT}(t_1, \cdots, t_n)^{(k)} = \sigma_{CT}(t_1^{(k-1)}, \cdots, t_n^{(k-1)})$ for $k > 1$ and that $t^{(0)} = \bot$. Then

$$
\begin{aligned}
\bar{h}_A(\sigma_{CT}(t_1, \cdots, t_n)) &= \bigsqcup_k h_A(\sigma_{CT}(t_1, \cdots, t_n)^{(k)}) && \text{(definition of } \bar{h}_A\text{)} \\
&= \bigsqcup_k h_A(\sigma_{CT}(t_1^{(k-1)}, \cdots, t_n^{(k-1)})) && \text{(definition of } t^{(k)}\text{)} \\
&= \bigsqcup_k \sigma_A(h_A(t_1^{(k-1)}), \cdots, h_A(t_n^{(k-1)})) && (h_A \text{ a homomorphism}) \\
&= \sigma_A(\bigsqcup_k h_A(t_1^{(k-1)}), \cdots, \bigsqcup_k h_A(t_n^{(k-1)})) && (\omega\text{-continuity of } \sigma_A) \\
&= \sigma_A(\bar{h}_A(t_1), \cdots, \bar{h}_A(t_n)) && \text{(definition of } \bar{h}_A\text{).} \quad \square
\end{aligned}
$$

The following result is a corollary of the proof of Theorem 4.8. We proved there was a unique strict ω-continuous Σ-homomorphism from CT_Σ to A, and then we were able to show (surprisingly) that in fact that homomorphism was Δ-continuous.

COROLLARY 4.9. *CT_Σ is initial in the class of ω-continuous Σ-algebras with strict ω-continuous homomorphisms.* $\square$

An algebra is *Δ-continuous* iff its carrier is Δ-complete and its operations are Δ-continuous. Let $\boldsymbol{\Delta alg}_\Sigma$ be the class of Δ-continuous Σ-algebras with strict 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 4.10. *CT_Σ is initial in class $\boldsymbol{\Delta alg}_\Sigma$ of Δ-continuous Σ-algebras with strict Δ-continuous homomorphisms.* $\square$

There are confusingly many initiality results for CT_Σ. For example, the following neither implies nor is implied by Theorem 4.8; details will be provided elsewhere when the relevance of the result is better understood.

THEOREM 4.11. *CT_Σ is initial in the class of Σ-algebras with ω-complete carriers, operations that are both ω-continuous and $\bigsqcup$-continuous and homomorphisms that are strict $\bigsqcup$-continuous Σ-homomorphisms.* $\square$

Since the following is a subclass of that in Theorem 4.11 and contains CT_Σ, we have

COROLLARY 4.12. *CT_Σ is initial in the class of Σ-algebras with strict Δ-complete carriers, $\bigsqcup$-continuous operations and strict $\bigsqcup$-continuous homomorphisms* $\square$

Before considering properties of derived operations on initial continuous algebras, we can now offer a bit more to the lattice versus poset "dispute" Since T_Σ is $\bigsqcup$-complete (Proposition 4.1), it follows that $CT_\Sigma^\top$ is a complete lattice (4.3) Make $CT_\Sigma^\top$ into a Σ-algebra by requiring that if any argument to σ_{CT} is $\top$, the value is $\top$ (the operations preserve $\top$). It follows immediately from Corollary 4.10 that $CT_\Sigma^\top$ is initial in the class of Σ-algebras with carriers which are complete lattices, operations which are Δ-continuous *and preserve* $\bot$, and homomorphisms which are doubly strict Δ-continuous (also pre-

[25] When the range of the index is obvious (as in this proof, $n \in \omega$), we delete that extra notation, e g $\bigsqcup_n = \bigsqcup_{n \in \omega}$

serve $\top$). It is fairly clear that the homomorphisms of lattice algebras should be doubly strict Δ-continuous, but it is equally clear that once the "overdefined" element $\top$ is introduced (we would say, artificially) to make the poset $\coprod$-complete, then we do not necessarily want the operations to preserve $\top$. In order to get the initial algebra in the class of lattice-Σ-algebras with Δ-continuous operations and doubly strict Δ-continuous homomorphisms, one has to put the obvious (though cumbersome) order on the algebra $CT_\Sigma(\top)$ (the algebra freely generated by $\{\top\}$). A common argument in support of lattices is that the only cost of using complete lattices is the occasional addition of an extra "overdefined" $\top$ to the domain. This initiality discussion shows that the argument is somewhat of an oversimplification, since many curious terms in $CT_\Sigma(\top)$ involving $\top$ have been added and are of dubious interpretation.

Having established the existence of initial continuous algebras, we can, as in Section 2, consider derived operators. Let A be any Z-continuous algebra (strict Z-complete carrier, Z-continuous operations) and let $a : X_n \to A$; then $\bar{a} : CT_\Sigma(X_n) \to A$ is the unique homomorphism determined by making A into a $\Sigma(X_n)$-algebra. As before $t_A(a) = \bar{a}(t)$. One can check that if $a \sqsubseteq a'$ (i.e. $a(x_j) \sqsubseteq a'(x_j)$, $1 \leq j \leq n$), then $\bar{a} \sqsubseteq \bar{a}'$ (i.e $\bar{a}(t) \sqsubseteq \bar{a}'(t)$, $t \in T_\Sigma(X_n)$).

PROPOSITION 4.13. *For each $n \in \omega$, $t \in CT_\Sigma(X_n)$, and Z-continuous Σ-algebra A, the derived operator $t_A : A^n \to A$ is Z-continuous for $Z \in \{\omega, \Delta, \ell'\}$.*

PROOF. Let $\{a_i\}_{i\in I}$ be a Z-set in A^n; then by the observation above, $\{\bar{a}_i(t)\}_{i\in I}$ is also a Z-set, and by Z-completeness of A we can define $\bar{a}(t) = \coprod_{i\in I}\bar{a}_i(t)$. We want to show that $\bar{a}$ is a homomorphism.

$$\begin{aligned}
\bar{a}(\sigma_{CT}(t_1, \cdots, t_n)) &= \coprod_{i\in I} \bar{a}_i(\sigma_{CT}(t_1, \cdots, t_n)) && \text{(definition of } \bar{a}\text{)},\\
&= \coprod_{i\in I} \sigma_A(\bar{a}_i(t_1), \cdots, \bar{a}_i(t_n)) && (\bar{a}_i \text{ a homomorphism}),\\
&= \sigma_A(\coprod_{i\in I} \bar{a}_i(t_1), \cdots, \coprod_{i\in I} \bar{a}_i(t_n)) && (Z\text{-continuity, (4.8)}),\\
&= \sigma_A(\bar{a}(t_1), \cdots, \bar{a}(t_n)) && (Z\text{-continuity, } \bar{a})
\end{aligned}$$

Clearly, $\bar{a}$ extends $\coprod_{i\in I}a_i$, and now we know $\bar{a}$ is a Σ-homomorphism, so $\bar{a} = \overline{\coprod_{i\in I}a_i}$. Thus $t_A(\coprod_{i\in I}a_i) = \bar{a}(t) = \coprod_{i\in I}a_i(t) = \coprod_{i\in I}t_A(a_i)$. □

Exactly 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 on initiality of CT_Σ.

PROPOSITION 4.14. *Substitution is Δ-continuous, i.e. $\leftarrow : CT_\Sigma^m(X_n) \times CT_\Sigma^n(X_k) \to CT_\Sigma^m(X_k)$ is a Δ-continuous function on the (Δ-complete) Cartesian product of Δ-complete sets.*

PROOF. For $t : X_m \to CT_\Sigma(X_n)$ and $t' : X_n \to CT_\Sigma(X_k)$, recall from Section 2 that $t \leftarrow t' : X_m \to CT_\Sigma(X_k)$ is defined by $(t \leftarrow t')(x_i) = \bar{t}'(t(x_i))$; i.e. $t \leftarrow t' = \bar{t}' \circ t$ where $\bar{t}'$ is the unique homomorphic extension of t'. For Δ-continuity we can consider each argument separately. Let $\{t_i\}_{i\in I}$ be a Δ-set Then $(\coprod_{i\in I}t_i) \leftarrow t' =$ (definition) $\bar{t}' \circ (\coprod_{i\in I}t_i) = (\bar{t}'$ is Δ-continuous) $\coprod_{i\in I}(\bar{t}' \circ t_i) =$ (definition) $\coprod_{i\in I}(t_i \leftarrow t')$. For the other argument, $t \leftarrow (\coprod_{i\in I}t_i') =$ (definition) $\overline{\coprod_{i\in I}t_i'} \circ t =$ (as in proof of Proposition 4.13) $(\coprod_{i\in I}\bar{t}_i') \circ t =$ (by 3.8) $\coprod_{i\in I}(\bar{t}_i' \circ t) =$ (definition) $\coprod_{i\in I}(t \leftarrow t_i')$. □

A simple example (arrived at with Steve Bloom) exemplifies substitution and shows that it is not u-continuous. Let $\Sigma_2 = \{a\}$ and take $I = \{1, 2\}$ with $t_1 = a(\bot x_1)$ and $t_2 = a(x_1\bot)$; then $t_1 \sqcup t_2 = a(x_1x_1)$, and $(t_1 \sqcup t_2) \leftarrow (t_1 \sqcup t_2) = a(a(x_1x_1)a(x_1x_1))$; but $(t_1 \leftarrow t_1) \sqcup (t_2 \leftarrow t_2) = a(\bot a(\bot x_1)) \sqcup a(a(x_1\bot)\bot) = a(a(x_1\bot)a(\bot x_1)) \neq a(a(x_1x_1)a(x_1x_1))$.

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 $t : \omega^* \dashrightarrow \Sigma$ 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) if $t(w)$ has arity $s_1 \cdots s_n$, then $i < n$ and $t(wi)$ has sort s_{i+1}.

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

THEOREM 4.15 *CT_Σ is initial in the class of ω-continuous Σ-algebras with strict Δ-continuous Σ-homomorphisms* □

5. *Applications of Initial Continuous Algebras*

Initiality of CT_Σ has important consequences in the theory of computation Just as initiality of T_Σ clarifies and simplifies "tree manipulation," CT_Σ clarifies and simplifies work with infinite trees which, for example, arise in the semantics of flow diagrams, recursive schemes, and other data structures.

CT_Σ has advantages in its abstract form (an initial Δ-continuous Σ-algebra) and in its concrete form (we can visualize elements of CT_Σ as (infinite) Σ-trees). The abstract formulation facilitates defining derived operations and solving equations in Δ-continuous Σ-algebras; in fact, this is easier than the conventional formulation employing the subset construction. The concrete representation permits clear understanding of the objects in CT_Σ; for example, our formulation of the lattice of flow diagrams is simpler, as well as more convincing than the inverse limit construction employed by Scott [61].

We know three papers[26] with general notions related to initial continuous algebras: Wand's [75] free μ-clones, Nivat's [48] algebra of schematic languages, and Bloom and Elgot's [9] free iterative algebraic theories. The diversity of these approaches, plus the somewhat sketchy character of the first two references and the novelty of initial continuous algebras, make it hard to describe all relationships between these approaches, but we do point out several connections

The discussion and examples below focus on the class $\boldsymbol{\Delta alg}_\Sigma$ of Δ-continuous Σ-algebras with strict Δ-continuous Σ-homomorphisms, thus we use Corollary 4.10, that CT_Σ is initial in $\boldsymbol{\Delta alg}_\Sigma$ Although what we say applies to other classes of algebras (e.g those mentioned in Theorems 4.8 and 4.11, and Corollaries 4.9 and 4.12), we don't give applications for these

5.1. SYSTEMS OF (REGULAR) EQUATIONS. Section 3 4 related initial algebra semantics to solving equations in the initial many-sorted Σ-algebra We used the phrase "make pA into a Σ-algebra by . . ." to define a derived operator $t_{pA} : (pA)^n \to pA$ for each $t \in T_\Sigma(X_n)$. An apparently ad hoc step then followed: extending derived operators to *sets* of terms by taking unions of values. For a Δ-continuous Σ-algebra, this isn't necessary since its order structure permits equations expressed in CT_Σ to be solved directly

A *system of n equations (expressed) in* CT_Σ is a function $E : X_n \to CT_\Sigma(X_n)$. This simplifies the informal definition " . a sequence or set of equations of the form $x_i = t_i$ where t_i is a term in n variables, $x_1, \cdots, x_n$." To read our system that way, write $x_i = E(x_i)$; $E(x_i)$ is the *right-hand side* of the ith equation.[27]

For any Δ-continuous algebra A, $E_A : A^n \to A^n$ is the derived operator of E over A; recall from Section 2 that $(E_A(a))_i = E(x_i)_I(a) = \bar{a}(E(x_i))$ for all $a \in A^n$ and $1 \le i \le n$. By Proposition 4.13, each component, $E(x_i)_A : A^n \to A$, of E_A is Δ-continuous, and by (3.7) so is their target-tuple $E_A = [E(x_1)_A, \cdots, E(x_n)_A] : A^n \to A^n$.

Thus E_A has a minimum fixed-point denoted $| E_A | \in A^n$ and called the *solution* of E over A We know that $| E_A | = \bigsqcup_{k\in\omega} E_A^k(\perp, \cdots, \perp)$; that $E_A(| E_A |) = | E_A |$; and that for all $a \in A^n$, if $E_A(a) = a$, then $| E_A | \sqsubseteq a$. For the special case $A = CT_\Sigma$, we write $| E |$ for the solution of E over CT_Σ.

Familiar results in the theory of equational sets follow nicely in this setting Recalling that $h_A : CT_\Sigma \to A$ is the unique Δ-continuous Σ-homomorphism from CT_Σ to the Δ-continuous Σ-algebra A, we write $| E |_A$ for $h_A^n(| E |)$.

[26] Reynolds's work [55] employing some methods of this paper will probably also interest the reader, and should appear soon after this paper does

[27] The notation $E(x_i)$ is correct because E is a function with source $X_n = \{x_1, \quad, x_n\}$, but note that $E(x_i)$ is an element of $CT_\Sigma(X_n)$ possibly involving all the variables $x_1, \quad, x_n$

The following analogue of Mezei and Wright [45, Th. 5.5] says that solving equations in the initial algebra, then interpreting the solution; and interpreting the equations, and then solving in the algebra, give the same answer. (Engelfriet and Schmidt [18] appropriately refer to these as "Mezei-Wright-like results"; cf. the following proposition.)

PROPOSITION 5.1. *For any system of equations* $E : X_n \to CT_\Sigma(X_n)$ *and any* Δ*-continuous* Σ*-algebra* A, $| E |_A = | E_A |$.

PROOF. By induction on k and Proposition 2.5, $h_A^n(E^k(\perp, \cdots, \perp)) = E_A^k(\perp_A, \cdots, \perp_A)$ where, for clarity, $\perp$ is minimum in CT_Σ and $\perp_A$ is minimum in A (In general, $h_A^n(E(t_1, \cdots, t_n)) = E_A(h_A(t_1), \cdots, h_A(t_n))$.) Then $h_A(| E |) = h_A(\bigsqcup_k E^k(\perp, \cdots, \perp)) =$ (by continuity of h_A) $\bigsqcup_k h_A(E^k(\perp, \cdots, \perp)) = \bigsqcup_k E_A^k(\perp_A, \cdots, \perp_A) = | E_A |$. □

Call a system $E : X_n \to CT_\Sigma(X_n)$ *ideal* (cf. Elgot [17]) iff each $E(x_i)$ is nontrivial, i.e. for some $m > 0$, $\sigma \in \Sigma_m$, and $t_1, \cdots, t_m \in CT_\Sigma(X_n)$, $E(x_i) = \sigma_{CT}(t_1, \cdots, t_m)$.

THEOREM 5.2. *Solutions of ideal systems of equations over* CT_Σ *are* unique

PROOF. We indicate the method of proof; full detail will appear elsewhere. If $t = \langle t_1, \cdots, t_n \rangle$ is a solution for a system $E : X_n \to CT_\Sigma(X_n)$, then for all k, $t = E_{CT}^k(t) = \underbrace{E \leftarrow E \leftarrow \cdots \leftarrow E \leftarrow E}_{k} \leftarrow t$. By induction on length of strings in ω^*, one shows for ideal E, if $w \in def(t_i)$ has length k, then $w \in def(E^k(\perp, \cdots, \perp)_i)$; i.e. $def(t_i) \subseteq def(| E |_i)$; but $| E | \sqsubseteq t$, so $| E | = t$. □

This rather surprising result seems to say the order relation on CT_Σ is superfluous once solutions are obtained. This suggests a strong connection with Elgot's [17] iterative algebraic theories where, in effect, ideal equations must have unique solutions and no ordering is involved.

A system E of equations is *finite* iff each right-hand side is finite; i.e. $E : X_n \to F_\Sigma(X_n)$. Now E_A can be defined over any ordered algebra A (Proposition 4 7) and $a \in A^n$ is the *solution* of E over A iff it is the minimum fixed point of E_A.

An ordered algebra A is *equationally complete* iff every *finite* 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 having an ω-complete carrier. The following important case illustrates this

Let R_Σ denote the set of equational elements of CT_Σ; i e $R_\Sigma = \{| E |_i \mid E : X_n \to F_\Sigma(X_n), n > 0 \text{ and } 1 \le i \le n\}$. The "$R$" in R_Σ stands for "recognizable" or "regular" as applied to sets of finite trees in Doner [15] and in Thatcher and Wright [70]. (For appropriate Σ discussed below, Scott [61] calls F_Σ rational, R_Σ algebraic, and $CT_\Sigma - R_\Sigma$ transcendental) The following two propositions indicate the connection.

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

PROOF Using Proposition 4.4, we transform a system $E : X_n \to F_\Sigma(X_n)$ of equations over CT_Σ to the system $\hat{E} : X_n \to \boldsymbol{p}(F_\Sigma(X_n))$ in the Mezei-Wright sense [45] by $\hat{E}(x_i) = \{E(x_i), \perp\}$ (cf. Section 3.4) Let $| \hat{E} |$ be the solution of $\hat{E}$ over F_Σ as in Mezei-Wright, each $| \hat{E} |(x_i)$ is a recognizable subset of F_Σ These recognizable sets are called "schematic languages" by Nivat [48], who shows that every schematic language is directed (in fact a lattice) with respect to the ordering on F_Σ. So $\bigsqcup | \hat{E} |$ exists in CT_Σ (Proposition 4.1) and we want to show $| E | = \bigsqcup | \hat{E} |$. We know $| E | = \bigsqcup_k E^k(\perp, \cdots, \perp)$ and, by induction on k, $\{E^k(\perp, \cdots, \perp)\} \subseteq | \hat{E} |$, so that $| E | \sqsubseteq \bigsqcup | \hat{E} |$. On the other hand, for each $t \in | \hat{E} |$ there exists k such that $t \sqsubseteq E^k(\perp, \cdots, \perp)$ (by induction on the definition of $| \hat{E} | = \bigcup_m \hat{E}^m_{\boldsymbol{p}F_\Sigma}(\varnothing, \cdots, \varnothing)$), so $| E | \supseteq \bigsqcup | \hat{E} |$, i.e. $| E | = \bigsqcup | \hat{E} |$. □

The second result relating the equational elements to recognizability uses the concrete construction of CT given in Section 4; we leave its proof to the reader.

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. □

As indicated in the proof of Proposition 5 3, a *schematic language* (Nivat [48]) is a recognizable subset of F_Σ (viewed as the initial $\Sigma(\perp)$ algebra) defined by equations (in

the Mezei-Wright sense) whose right-hand sides are of the form $\{t, \perp\} \subseteq F_\Sigma$ By reversing the translation in the proof of Proposition 5 3, it is easy to see that the least upper bound of any schematic language is an equational element of CT_Σ. Nivat [48] shows that the schematic languages form a Σ-algebra, and his proof also shows that R_Σ is a subalgebra of CT_Σ. Because R_Σ is equationally complete by definition, we have

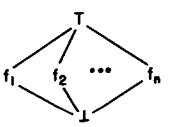
PROPOSITION 5.5. *R_Σ is the smallest equationally complete subalgebra of CT_Σ.* □

5.2 THE LATTICE OF FLOW DIAGRAMS. Our construction of initial continuous algebras permits simple alternative formulations of Scott's [61] lattice of flow diagrams. Since those alternatives are specific initial algebras, the lattice of flow diagrams appears as a special case of a very general situation. Our comparison is slightly encumbered by the fact that Scott uses complete lattices ($\bigsqcup$-complete posets), while we favor strict Δ-complete posets.

Most simply put, Scott's lattice of flow diagrams is the solution to the *domain equation*

$$E \cong \{I\} + F + E \times E + B \times E \times E, \qquad (*)$$

where $+$ is the coalesced lattice sum, $\times$ is the usual lattice product, and $\{I\}$, F, and B are the ("flat") lattices which result from adjoining $\perp$ and $\top$ to, respectively, the set consisting of a single function symbol I denoting the identity function, a set F of function symbols (operations occurring in boxes of a flow diagram), a set B of predicate symbols (occurring as tests in flow diagram). So if the set $F = \{f_1, \cdots, f_n\}$, then the lattice for F is:



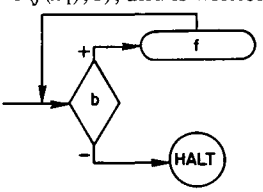
Scott's inverse limit construction provides a solution for E as a complete lattice. However, Gordon [26] argues the advantages of Δ-complete posets (calling them semidomains) over complete lattices (domains) and shows the inverse limit construction (using separated sum) works just as well in that context. Passing over pragmatic and aesthetic points discussed earlier (Section 4), we make our comparison with Gordon's solution to (*), thinking of it as the *semidomain of flow diagrams*.

We describe three semidomains 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 the algebraic approach (as opposed to the pure ordered structure approach) leads directly to *algebras* of flow diagrams rather than ordered sets of flow diagrams. (Nivat [48] stresses this point, and it seems that Scott also saw it [61, Sec. 4].) We hope this creates a stronger position from which to consider the questions about equivalence of flow diagrams raised by Scott [61, Secs. 8 and 9].

(I) Let $\Sigma = \langle\{I\},F,B\rangle$ (that is, $\Sigma_0 = \{I\}$, $\Sigma_1 = F$, $\Sigma_2 = B$, and $\Sigma_k = \varnothing$ for $k > 2$). $CT_\Sigma^\top$ (see (4.3)) is *a* lattice of flow diagrams, with every flow diagram represented by its unfoldment. But (quite aside from questions about $\top$) $CT_\Sigma^\top$ doesn't have quite the algebraic, or the order theoretic, structure of Scott's lattice. The equational elements (in R_Σ) represent the flow diagrams. The idea, to be developed formally 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 is written $x_1 = (b \to (f; x_1), I)$ by Scott.



The element of R_Σ which is the component for the start node is the unfoldment of the given flow diagram (cf. Goguen [20]). Conversely, any finite system of equations over CT_Σ can be reduced to "primitive" equations of the above form (much as in Mezei and Wright [45] or in Wand [75, Th. 2]), from which a standard flow diagram can be retrieved.

Wand [75] takes Scott's construction "trivially modified" as a starting point, calling $E(\Sigma)$ the lattice of flow diagrams with operator symbols Σ. He says this follows Elgot's [17] idea of handling tests (rank 2 or greater) and operations (rank 1) uniformly. His $E(\Sigma)$ seems to be our $CT_\Sigma^\top$; his "finite diagrams" to be our $F_\Sigma(X_n)$, with "return codes" $\{x_1, \cdots, x_n\}$ (also paralleling Elgot [17] because of the n "exits"); and his "loop-representable" flow diagrams to be our $R_\Sigma(X_n)$ (that is, equational elements of $CT_{\Sigma(X_n)}$). Wand [75] is rather informal, but seems to miss the distinction brought out by our second formulation for the lattice of flow diagrams below

One can define context-free sets over initial algebras (Rounds [57]), and this applies to initial continuous algebras as well. Alternatively (Maibaum [41, 42], Turner [71], and Wand [76]), one can make the function symbols zero-ary and introduce a symbol for composition; then the context-free sets are equational in this modified algebra

(II) In that spirit, let $\Sigma' = \langle F \cup \{I\}, \varnothing, B \cup \{;\}\rangle$. Again, every flow diagram (over F and B) is represented in $CT_{\Sigma'}$ by an unfoldment. But now the algebraic structure corresponds to Scott's idea: Every element of $CT_{\Sigma'}$ is either I, an element of F, $(d; d')$, or $b(d, d')$ (written $b \to d, d'$) for diagrams d and d'.

However, the equational elements of this algebra correspond to unfoldings of *recursive* flow diagrams. For example, the solution to the equation $x_1 = b(f;(x_1;f'),I)$, problematic in Scott [61, Fig. 21], is the unfolding of the recursive flow diagram,

$$x_1 : \textbf{if } b \textbf{ then } (f; \textbf{call } x_1; f') \textbf{ else halt.}$$

Nivat's [48] *program schemes over* $\langle F \cup \{I\}, B\rangle$ are reformulated in this setting as primitive[28] systems of equations $E : X_n \to F_{\Sigma'}(X_n)$, which are connected (meaning $x_1 < x_i$ for $1 < i \leq n$, where $<$ is the transitive closure of x_i "calls" x_j, i.e. if x_j occurs in $E(x_i)$).

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

(III) For our last formulation, let Σ'' be a $\{b,d\}$-sorted operator domain with $\Sigma''_{\langle\lambda,b\rangle} = B$, $\Sigma''_{\langle\lambda,d\rangle} = \{I\} \cup F$, $\Sigma''_{\langle dd,d\rangle} = \{;\}$, and $\Sigma''_{\langle bdd,d\rangle} = \{\to\}$. Then $CT_{\Sigma'',b}$ contains only $B \cup \{\perp\}$, but $CT^\top_{\Sigma'',d}$ is isomorphic[29] (in both order and algebraic structure) to Scott's lattice E of flow diagrams. (This corresponds, as in [41, 42, 71, 76], to making *all* the function symbols of $\langle\{I\}, F, B\rangle$ zero-ary; the "composition symbols" are ";" and "$\to$".) But all that has been added are diagrams such as $(\perp \to d, d')$, 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 in the previous formulation

5.3. Semantics of Flow Diagrams Theorem 4.8 says that *any* Δ-continuous Σ-algebra is a possible semantic algebra for CT_Σ. Looking at the second formulation for the lattice of flow diagrams $\Sigma' = \langle F \cup \{I\}, \varnothing, B \cup \{;\}\rangle$, take, as Scott does, the Δ-complete (4.9) poset $[S \overset{\omega}{\to} S]$ of "partial state transformations" as carrier of a semantic algebra ambiguously denoted S Given (Scott's [61]) interpretations of the function symbols $\mathscr{F} : F \to [S \overset{\omega}{\to} S]$ and predicate symbols $\mathscr{B} : B \to [S \overset{\omega}{\to} \{0, 1\}]$, make $[S \overset{\omega}{\to} S]$ 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.(cf. 3.2, 3.7, 3.8), so initiality of $CT_{\Sigma'}$ gives a unique homomorphism $\mathscr{V}_S : CT_{\Sigma'} \to S$, Scott's [61] "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

[28] Much as used above, each $E(x_i)$ is either $f \in F$, I, (α_1, α_2), or $(b \to \alpha_1, \alpha_2)$, where $\alpha_i \in F \cup \{I\} \cup X_n$

[29] Ignoring the extra overdefined elements discussed in Section 4

To get the intended semantics of flow diagrams when expressed in CT_Σ ($\Sigma = \langle\{I\}, F, B\rangle$), appeal to *continuations* of C. Wadsworth and L. Morris (see Reynolds [53] and Strachey and Wadsworth [66]) seems inevitable. Taking the same carrier for the semantic algebra, operations ($f \in F$) are of type $[S \nrightarrow S] \to [S \nrightarrow S]$, or equivalently $[S \nrightarrow S] \times S \to S$ taking a "continuation" in $[S \nrightarrow S]$ and a state in S giving a new state. The extent to which continuations can be 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 for further investigation.[30]

5.4. SYSTEMS OF (REGULAR) EQUATIONS WITH PARAMETERS. Equational functions can be defined much as equational elements of an algebra were. In the spirit of Wagner [73, 74] and Wand [75], a function[31] $E : Y_n \to CT_{\Sigma(X_p)}(Y_n)$ is a *system of n-equations with p parameters* $\{x_1, \cdots, x_p\}$. For any algebra A in $\boldsymbol{\Delta alg}_\Sigma$ and for each $a \in A^p$, 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 $E_{A(a)} : A^n \to A^n$, which has a minimum fixed point $| E_{A(a)} |$. The function $| E_A | : A^p \to A^n$ determined by system E is defined by $| E_A | (a) = | E_{A(a)} |$.

The equation of Proposition 5.1 holds for equational functions as well. Given E as above, $| E | : Y_n \to CT_{\Sigma(X_p)}$ (solving in $CT_{\Sigma(X_p)}$), so $| E |_A : A^p \to A^n$, and

PROPOSITION 5.1′. *For a system of equations* $E : Y_n \to CT_{\Sigma(X_p)}(Y_n)$ *with p parameters and any* Δ*-continuous* Σ*-algebra* A, $| E_A | = | E |_A : A^p \to A^n$.

PROOF. By definition of solution, $| E_A |(a) = | E_{A(a)} |$. But Proposition 5.1 gives $| E_{A(a)} | = | E |_{A(a)}$, and from the definition of derived operator, $| E |_{A(a)} = | E |_A(a)$. □

Solving a finite system of equations with parameters $E : Y_n \to F_{\Sigma(X_p)}(Y_n)$ over $CT_{\Sigma(X_p)}$ yields $| E | \in R^n_{\Sigma(X_p)}$, by definition. But as sets $R_{\Sigma(X_p)} = R_\Sigma(X_p)$, so we can talk about solving a system of equations whose right-hand sides are in $R_{\Sigma(X_p)}$; this gives nothing new. (As proved by Nivat [48] within his schematic language formulation, it is a consequence of the substitution theorem for context-free languages [19].)

PROPOSITION 5.6 (Nivat). *Systems of equations with right-hand sides in* $R_\Sigma(X_p)$ *have solutions in* R_Σ. □

We believe the principal connections with Wand [75] and with Bloom and Elgot [9] are to be found here. First, it seems clear that $\langle R_\Sigma(X_p)\rangle_{p\in\omega}$ is the free μ-clone generated by Σ, the construction of which is Wand's principal result. Next, let $\bar{R}_\Sigma(X_p)$ be the set of maximal elements of $R_\Sigma(X_p)$. (In the concrete construction of $CT_{\Sigma(X_p)}$, t is maximal *iff* $t(u) \in \Sigma_k$ implies $ui \in def(t)$, $0 \le i < k$.) The algebraic structure of $\langle \bar{R}_\Sigma(X_p)\rangle_{p\in\omega}$ under substitution ($t : X_n \to R_\Sigma(X_p)$, $t' : X_p \to R_\Sigma(X_q)$; $t \leftarrow t' . X_n \to R_\Sigma(X_q)$; see Section 2) gives rise to an *algebraic theory* (cf. Elgot [17]). Theorem 5.2 and Proposition 5.6 say this algebraic theory is *iterative*, which means that every ideal system of equations $E : X_n \to \bar{R}_{\Sigma(X_p)}(X_n)$ has a unique solution $| E | : X_n \to \bar{R}_\Sigma(X_p)$ (called $E\dagger$ in [17]) 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 believe[32] that this algebraic theory is the "free iterative theory generated by Σ," the construction of which is the principal result of Bloom and Elgot [9].

6 *Conclusion*

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

(A) How is the conventional theory of equational sets (e.g. Mezei and Wright [45]) expressed in the (simpler) continuous algebra setting?

(B) If every finite system of equations has a solution in an ordered (not necessarily ω-

[30] Reynolds [55] considers this in careful and interesting detail

[31] X_p and $Y_n = \{y_1, \cdots, y_n\}$ are, technically, assumed to be disjoint copies of the respective sets of variables

[32] S. Ginali proves this in her forthcoming University of Chicago dissertation

complete) algebra, does it follow that $| E_A | = \bigcup_k E_A^k(\perp)$ anyway?

(C) How do the works of Maibaum [42], Turner [71], and Wand [76] (which identify context-free sets over one algebra with recognizable sets in another) relate to the two ranked alphabets Σ and Σ' in the discussion of the lattice of flow diagrams, or more generally to the solution of regular equations versus the solution of context-free equations? Engelfriet and Schmidt [18] tackle this area (and (A) above) in their typically precise way.

(D) Gordon [26] proves the equivalence of a denotational (Scott-like) semantics and an interpretative semantics for pure LISP. The proof is very long. Why? Could methods here contribute to further simplification?

(E) Our initial algebra formulation of denotational semantics (Section 3) differs significantly from previous formulations. The "structural induction" and "finite approximation" proof techniques used by Gordon are implicit in initial continuous algebras Could these yield a more workable semantics for "real" programming languages?

(F) We know that T_Σ is initial in the class of Σ-algebras, that CT_Σ is initial in the class of Δ-continuous Σ-algebras, and that these results bear fruit On either side of CT, we ask the question: In what class of algebras is R_Σ (and $\bar{R}_\Sigma$) initial, and in what class of algebras in Nivat's $PS_{\Sigma'}$ initial? The latter is essentially the question posed in Burstall and Thatcher [11].

(G) Chandra [12] uses infinite tree flow diagrams which are outside R_Σ but 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 relevant, how are his "meanings" defined from an initial algebra point of view?

(H) Can the questions raised by Scott in [61] be answered in the initial continuous algebra setting? Should they?

(I) The practical aspects of the work reported in this paper should be further explored. Pottenger [51] has programmed a system which acts as a compiler generator: Given Σ and σ_A for each σ in Σ, the function h_A becomes available. The implementation is somewhat inefficient, but is useful as it is, and can presumably be improved in various ways Also, there seem to be a number of implications for programming language design which should be followed up.

(J) What is the connection of the present formulation (that is, CT_Σ, etc.) with the V-category formulation suggested in Goguen [20], where syntax is an "algebraic theory" whose hom-sets are sufficiently nice posets, and semantics is a sufficiently nicely ordered algebra of this theory?

(K) In fact, it would be interesting to give a thorough development of semantics in terms of algebraic theories [32]; we have restricted ourselves to initial algebras primarily to avoid imposing odious mathematical prerequisites. The question is, could more than convenience and conciseness be achieved? We are proposing some suggestions in this direction in [25].

These are just a few questions; if the reader has faced infinite structures from semantics of programs or data structures, he may well have many others. We hope this paper will stimulate such questions and that the initial algebra viewpoint will be fruitful in answering them.[33]

ACKNOWLEDGMENTS. 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 to thank M. Gordon, S. MacLane, B. Rosen, and M. Wand for their suggestions and criticisms. R.M. Burstall

[33] This has actually happened to the authors of this paper, who have, subsequent to the work reported here, proposed an initial algebra approach to abstract data types [24]

and J.C. Reynolds deserve special gratitude for their extensive efforts to improve both the content and presentation of the material.

REFERENCES

(Note References [14, 21, 22, 37, 56, 58, 72] are not cited in the text)

1 AHO, A V , AND ULLMAN, J D Properties of syntax directed translations *J Computer and Syst. Scis. 3* (1969), 319-334

2 AHO, A B , AND ULLMAN, J D Translations of a context-free grammar *Inform and Contr 19* (1971), 439-475

3 BAKER, B S Tree transductions and families of tree languages Tech Rep TR9-73, Center for Research in Computing Technology, Harvard U , Cambridge, Mass , 1973

4 BEKIĆ, H Definable operations in general algebra and the theory of automata and flowcharts Research Rep , IBM Laboratory, Vienna, Austria, 1969

5 BÉNABOU, J Structures algébriques dans les catégories Thèse, fac sci , Université de Paris, March 1966 Also, *Cahiers de Topologie et Géométrie Différentielle 10* (1968), 1-126

6 BIRKHOFF, G Structure of abstract algebras *Proc Cambridge Phil Soc 31* (1938), 433-454

7 BIRKHOFF, G , AND LIPSON, J D Heterogeneous algebras *J Combinatorial Theory 8* (1970), 115-133

8 BLIKLE, A Equational languages *Inform and Contr 21* (1972), 134-147

9 BLOOM, S L , AND ELGOT, C C The existence and construction of free iterative theories. Research Rep RC-4937, IBM Thomas J Watson Research Center, Yorktown Heights, N Y , 1974

10 BURSTALL, R M , AND LANDIN, P J Programs and their proofs An algebraic approach *Machine Intelligence, Vol 4,* B Meltzer and D Michie, Eds , Edinburgh U Press, Edinburgh, Scotland, 1969, pp 17-43

11 BURSTALL, R M , AND THATCHER, J W The algebraic theory of recursive program schemes. *Lecture Notes in Computer Science, Vol. 25 Category Theory Applied to Computation and Control,* Springer, Berlin, 1974, pp 126-131.

12 CHANDRA, A K Degrees of translatability and canonical forms of program schemes Pt I Proc Sixth Ann ACM Symp on Theory of Computing, Seattle, 1974, pp 1-12

13 COHN, P M *Universal Algebra* Harper and Row, New York, 1965

14 COURCELLE, B , AND VUILLEMIN, J. Semantics and axiomatics of a simple recursive language, Proc Sixth Ann ACM Symp on Theory of Computing, Seattle, 1974, pp 13-26

15 DONER, J E Tree acceptors and some of their applications *J Computer and Syst Scis 4* (1970), 406-451

16 EILENBERG, S , AND WRIGHT, J B Automata in general algebras *Inform and Control 11* (1967), 452-470

17 ELGOT, C C Monadic computation and iterative algebraic theories Research Rep. RC-4564, IBM Thomas J Watson Research Center, Yorktown Heights, N Y , 1973, also Proc Logic Colloquium '73, Bristol, England, North-Holland Pub Co , Amsterdam, 1975, pp 175-230

18 ENGELFRIET, J , AND SCHMIDT, E M IO and OI Datalogisk Afdeling Rep , DAIMI PB-47, Aarhus U , Aarhus, Denmark, July 1975

19 GINSBURG, S *The Mathematical Theory of Context-Free Languages* McGraw-Hill, New York, 1962

20 GOGUEN, J A On homomorphisms, correctness, termination, unfoldments and equivalence of flow diagram programs Proc 13th Ann IEEE Symp on Switching and Automata Theory, 1972, pp 52-60 A portion of this paper appears in expanded form in *J Computer and Syst Scis 8* (1974), 333-365

21 GOGUEN, J A Semantics of computation *Lecture Notes in Computer Science, Vol 25 Category Theory Applied to Computation and Control,* Springer, Berlin, 1974, pp 151-163

22 GOGUEN, J A , AND THATCHER, J W Initial algebra semantics Extended Abstract, Research Rep RC-4865, IBM Thomas J Watson Research Center, Yorktown Heights, N Y , May 1974, also, Proc 15th Ann IEEE Symp on Switching and Automata Theory, 1974, pp 63-77

23 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 Pt 1, Research Rep RC-4526, Pt 2, Research Rep RC-5908, IBM Thomas J Watson Research Center, Yorktown Heights, N Y , 1973, 1976

24 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. Conference on Computer Graphics, Pattern Recognition and Data Structure. May 1975, pp 89-93

25 GOGUEN, J A , THATCHER, J W , WAGNER, E G , AND WRIGHT, J B. Programs in categories (summary), in preparation

26 GORDON, M Models of pure LISP Ph D Th , Edinburgh U , Edinburgh, Scotland, 1973

27 GRAETZER, G *Universal Algebra* Van Nostrand, Princeton, N J , 1968

28 HIGGINS, P J Algebras with a schema of operators *Math Nachr 27* (1963), 115-132

29. IRONS, E T A syntax directed compiler for ALGOL 60 *Comm ACM 4* (1961), 51-55

30 KNUTH, D E Semantics of context-free languages *Math Syst Theory 2* (1968), 127-145

31 LANDIN, P J A program machine symmetric automata theory *Machine Intelligence 5*, B Meltzer and D Michie, Eds , Edinburgh U. Press, Edinburgh, Scotland, 1970, pp 99-120.
32. LAWVERE, F W Functorial semantics of algebraic theories. *Proc. Nat Acad Sci. 50* (1963), 869-872.
33 LEWIS, C H , AND ROSEN, B K Recursively defined data types, Pt. 1 Proc ACM Symp on Principles of Programming Languages, 1973, pp 125-138, Pt 2, Research Rep. RC-4713, IBM Thomas J Watson Research Center, Yorktown Heights, N Y , 1974
34 LEWIS, P.M II, AND STEARNS, R E Syntax-directed transduction *J ACM 15*, 3 (July 1968), 465-488
35 LUCAS, P , LAUER, P , AND STIGLEITNER, H Method and notation for the formal definition of programming languages Tech Rep TR 25 087, IBM Laboratory, Vienna, Austria, 1968
36 MCCARTHY, J Towards a mathematical science of computation Proc IFIP Cong , North-Holland Pub Co , Amsterdam, 1962, pp 21-28
37 MCCARTHY, J A formal description of a subset of ALGOL In "Formal Language Description Languages for Computer Programming," Proc IFIP Working Conf 1964, T B Steel, Jr , Ed , North-Holland Pub Co , Amsterdam, 1966, pp 1-12
38 MCCARTHY, J , AND PAINTER, J Correctness of a compiler for arithmetic expressions In "Mathematical Aspects of Computer Science," Proc of Symposia in Applied Mathematics, Vol 19, J T Schwartz, Ed , Amer Math Soc , Providence, R I , 1967, pp 33-41
39 MACLANE, S *Category Theory for the Working Mathematician* Springer, New York, 1971
40 MAGIDOR, M , AND MORGAN, G Finite automata over finite trees Tech Rep 30, Hebrew U , Jerusalem, Israel, 1969
41 MAIBAUM, T S E The characterization of the derivation trees of context-free sets of terms as regular sets Proc 13th Ann IEEE Symp on Switching and Automata Theory, 1972, pp. 224-230
42 MAIBAUM, T S E Generalized grammars and homomorphic images of regular sets Research Rep CS-73-30, U of Waterloo, Waterloo, Ontario, Canada, 1973
43 MANNA, Z Properties of programs and the first-order predicate calculus *J ACM 16*, 2 (April 1969), 244-255
44 MARKOWSKY, G Chain-complete posets and directed sets with applications Research Rep RC-5024, IBM Thomas J Watson Research Center, Yorktown Heights, N Y , Aug 1974
45 MEZEI, J , AND WRIGHT, J B Algebraic automata and context-free sets *Inform and Contr 11* (1967), 3-29
46 MORRIS, F L Correctness of translations of programming languages Ph D Th , Computer Science Memo CS72-303, Stanford U , Stanford, Calif , 1972
47 MORRIS, F.L Advice on structuring compilers and proving them correct Proc Symp on Principles of Programming Lanugages, Boston, 1973, pp 144-152
48 NIVAT, M Languages algébraic sur le magna libre et sémantique des schémas de programme In *Automata, Languages and Programming*, M Nivat (Ed), North-Holland Pub Co , Amsterdam, 1972, pp 293-308
49 PARK, D Fixpoint induction and proofs of program properties *Machine Intelligence, Vol 5*, B Meltzer and D Michie, Eds , Edinburgh U Press, Edinburgh, Scotland, 1970, pp 59-78
50 PETRONE, L Syntactic mappings of context-free languages Proc IFIP Cong 1965, Vol 2, North-Holland Pub Co , Amsterdam, pp 590-591
51 POTTENGER, R ARBOL, a system for defining functions on trees A I Memo 3, UCLA, 1976
52 RABIN, M P , AND SCOTT, D Finite automata and their decision problems *IBM J Res Develop 3* (1959), 114-125
53 REYNOLDS, J C Definitional interpreters for higher-order programming languages Proc 25th National ACM Conference, Boston, Mass , Aug 1972, pp 717-740
54 REYNOLDS, J C On the relation between direct and continuation semantics *Lecture Notes in Computer Science, Vol 14 Automata, Languages and Programming*, Springer, Berlin, 1974, pp 141-156
55 REYNOLDS, J C. Semantics of the lattice of flow diagrams. Manuscript, Syracuse U , Syracuse, N Y , submitted for publication, July 1975
56 ROSEN, B K Program equivalence and context-free grammars Proc 13th Ann IEEE Symp on Switching and Automata Theory, 1972, pp 7-18, revised as Research Rep RC-4822, IBM Thomas J Watson Research Center, Yorktown Heights, N Y , 1974
57 ROUNDS, W C Mappings and grammars on trees *Math Syst Theory 4* (1970), 256-287
58 SCHUTZENBERGER, M P Context-free languages and push down automata *Inform and Contr 6* (1963), 246-264
59 SCHWARTZ, J T Semantic definition methods In *Formal Semantics of Programming Languages*, R Rustin, Ed , Prentice-Hall, Englewood Cliffs, N J , 1972, pp 1-23
60 SCOTT, D Outline of a mathematical theory of computation Proc 4th Ann Princeton Conf on Information Sciences and Systems, 1970, pp 169-176
61 SCOTT, D The lattice of flow diagrams Tech Monograph PRG 3, Oxford U Computing Lab , Oxford U , Oxford, England, also, *Lecture Notes in Mathematics, Vol 182 Semantics of Algorithmic Languages*, E Engeler, Ed , Springer, Berlin, 1971, pp 311-366
62 SCOTT, D Continuous lattices Tech. Monograph PRG 7, Oxford U Computing Lab , Oxford U ,

Oxford, England, 1971, also, *Lecture Notes in Mathematics, Vol 274,* Springer, Berlin, 1971, pp 97–136

63 SCOTT, D Data types as lattices Unpublished notes, Amsterdam, 1972

64 SCOTT, D Data types as lattices Unpublished notes, Oxford, 1974

65 SCOTT, D , AND STRACHEY, C Towards a mathematical semantics for computer languages Tech Monograph PRG 6, Oxford U. Computing Lab., Oxford U., Oxford, England, 1971, also, *Computers and Automata,* J Fox, Ed , Wiley, New York, 1971, pp 19–46

66 STRACHEY, C , AND WADSWORTH, C P Continuations – A mathematical semantics for handling full jumps Tech Monograph PRG-11, Programming Research Group, Oxford U Computing Lab , Oxford, England, 1974

67. THATCHER, J.W. Characterizing derivation trees of context free grammars through a generalization of finite automata theory *J Computer and Syst Scis 1* (1967), 317–322

68 THATCHER, J W Generalized² sequential machines *J Computer and Syst. Scis 4* (1970), 339–367

69 THATCHER, J W Tree automata· An informal survey In *Currents in Computing,* A V. Aho, Ed , Prentice-Hall, Englewood Cliffs, N J , 1973, 143–172

70. THATCHER, J W., AND WRIGHT, J B. Generalized finite automata theory with an application to a decision problem of second-order logic *Math Systems Theory 2* (1968), 57–81

71 TURNER, R Doctoral Diss , U of London, London, England, 1973

72. VUILLEMIN, J Syntaxe, Sémantique et Axiomatique d'un Langage de Programmation Simple These d'Etat, Université Paris 6, France, 1974

73 WAGNER, E.G An algebraic theory of recursive definitions and recursive languages. Proc. Third Annual ACM Symposium on Theory of Computing, 1971, pp 12–23

74 WAGNER, E G Languages for defining sets in arbitrary algebras. Proc 11th Ann IEEE Symp on Switching and Automata Theory, 1971, pp 191–201

75 WAND, M A concrete approach to abstract recursive definitions In *Automata, Languages and Programming,* M Nivat, Ed , North-Holland Pub Co., Amsterdam, 1972, pp 331–341.

76 WAND, M An algebraic formulation of the Chomsky hierarchy *Lecture Notes in Computer Science, Vol 25 Category Theory Applied to Computation and Control,* Springer, Berlin, 1974, pp 209–213

77 VAN WIJNGAARDEN, A., Ed Report on the algorithmic language ALGOL 68 *Numer Math 14* (1969), 79–218

RECEIVED JANUARY 1975, REVISED MARCH 1976