MATHEMATICAL CENTRE TRACTS 109

FOUNDATIONS OF COMPUTER SCIENCE III

PART 2 : LANGUAGES, LOGIC, SEMANTICS

J.W. DE BAKKER (ed.)
J. VAN LEEUWEN (ed.)

SECOND PRINTING

MATHEMATISCH CENTRUM AMSTERDAM 1981

1980 MATHEMATICS SUBJECT CLASSIFICATION: 03B45, 68B10, 68C01, 68F05, 68F20

ACM-Computing Reviews-categories: 5.21, 5.23, 5.24

ISBN 90 6196 177 7
First printing 1979
Second printing 1981

CONTENTS

INFINITE WORDS, INFINITE TREES, INFINITE COMPUTATIONS

by

M. NIVAT

MATHEMATICAL CENTRE TRACTS 109(1979), 1-52

INFINITE WORDS, INFINITE TREES, INFINITE COMPUTATIONS

M.Nivat

University Paris VII and IRIA, France

In these notes we shall give an account of an attempt to define the semantics of recursive programs using the notion of successful infinite computation, which gave rise already to several papers [1,2,4,28].

The intuitive idea is very simple. In the now standard theory of computation in an ordered domain [10,11,17,18,29,34] there is the well-known equivalence between the definition of the computed function as the smallest fixed point of certain functional and the definition of the same function by means of terminating computation sequences of the program at a given point. This equivalence holds when the computation domain is a flat, i.e. discrete, domain in which different defined values are incomparable: the only converging sequences are the stationary sequences whose terms are all equal, for sufficiently large n to the limit of the sequence. In such a domain it is clear that any computed value is the result of some finite terminating computation sequence.

The situation is entirely different if, following D. Scott, one starts computing in a partially ordered domain which contains infinite ascending chains. A computed value may then be the lub of such a chain and as such it can fail to be the result of a finite computation. A typical example is the domain of real numbers: if basic functions are the four arithmetic operations and the initial values are rational numbers then after any finite amount of time one will have computed only a rational number, while the "final" result may be irrational.

We propose in this situation to give a meaning to succesful infinite computation sequences, which will be said to produce a result and to define the computed function by stating that its value at a given point is the set of results of both finite terminating and infinite succesful computation sequences at that point. In doing so one obviously has to accept that a computed function is many-valued, since there is absolutely no reason why all computation sequences would lead to the same result. But indeed many-valued functions were already considered as the normal output of a non-deterministic program. Thus our point of view amounts to considering deterministic programs as special cases of non-deterministic programs, with the advantage that our results will hold in the general case of non-deterministic programs (this was in fact the original motivation of this study).

In order to define successful computation sequences we found it extremely convenient to replace the order structure on the computation domain by a complete metric topology. (This does not mean at all that one cannot use the structure of a cpo to build a theory in many respects analogous to ours, like has been done in e.g. [30,35]).

The results we get to are mainly conditions for the equivalence of this definition of the computed function and a mathematical definition by means of fixed points: it happens that in a very natural way one is lead to consider greatest fixed points rather than smallest. Intuitively this corresponds to the idea that, at the beginning of the computation, we only know that the value of the computed function lies in a certain range, a priori the whole computation domain, and that in the course of the computation this range is narrowed (maybe to just one value but usually to a set of values). This is dual of the point of view expressed by Dana Scott that an a priori undefined initial value gets more and more defined in the course of the computation. We have borrowed this idea of decreasing range to a large extent from L. Nolin (in an uncountable number of discussions over twenty years!).

In the course of this study we will consider infinite words and infinite trees for the following reasons:

1. algebraic (or context-free) grammars are the only examples of non-deterministic recursive programs we really know (sufficiently at least to support intuition). If we extend the domain of finite words by adding those infinite words which are limits of sequences of finite words in the natural topology which measures the distance of two words by the inverse of the length of their common longest left factor, we have the typical case were infinite words cannot be the result of finite computations (or derivations). The idea of greatest fixed points and the major role played by closed subsets stem from our rather detailed study of infinite derivations of algebraic grammars which is presented below.
2. algebraic infinite trees which can be generated by a recursive program scheme are at the basis of the theory called "algebraic semantics" of recursive programs (see [10,11,17,18,24]). The algebraic tree thus

associated with a program scheme incorporates the whole semantics of the program in the sense that, when interpretation is defined as a morphism, the function computed by the program resulting from the interpretation of the scheme is the morphic image of this algebraic infinite tree. Whence many results concerning classes of interpretations and families of computation domains. Here infinite trees also play a role, in fact a crucial role, for the link between a semantics defined in an ordered structure and the semantics defined in a topological structure lies in the fact that the set of infinite trees $\Pi^{\infty}(F,V)$ has both an order structure and a topological structure which are closely related (in fact an increasing function is order-continuous iff it is continuous for the topology). The free complete F-image $\Pi^{\infty}(F,V)$ thus appears as the mother structure in which the phenomena of computation can be better described.

By no means we consider this theory as completed: many questions are raised which have to be studied further. The author is working on some of these in collaboration with André Arnold, without whom most of this work would never have been done. I gratefully acknowledge his constant help in producing these notes. Other people who were of great help at various stages of this writing are Luc Boasson and Bruno Courcelle. The author also thanks J.W. de Bakker and J. van Leeuwen for giving him the opportunity to teach this course and the Mathematical Centre for publishing these notes in such a pleasant typing and setting.

I. INFINITE WORDS

Let X be a finite alphabet. Let X^* denote the free monoïd generated by X, i.e. the set of finite words written over X, including the empty word ε. The length of a word $f \in X^*$ is denoted $|f|$.

Let Ω be a letter not in X. We define the mapping f from the set $\mathbb{P}$ of non-zero positive integers into $X \cup \{\Omega\}$ by

$$f(n) = \begin{cases} \text{the n-th letter of f if } n \leq |f| \\ \Omega \text{ otherwise.} \end{cases}$$

The relation $\leq$ on X^* is defined by

$$f \leq g \quad \text{iff} \quad \forall n \in \mathbb{P} \quad n \leq |f| \;\Rightarrow\; f(n) = g(n).$$

We say that f is a left factor of g iff $f \leq g$. We say that f is a proper left factor of g iff $f \leq g$ and $f \neq g$ and we write $f < g$ in this case. An infinite word on X is a mapping $u: \mathbb{P} \to X$. The letter $u(n)$ is called the n-th letter of u and we denote by $u[n]$ the finite word $u(1)u(2)\ldots u(n)$. The set of all infinite words is denoted X^{ω} and the set of all finite and infinite words $X^* \cup X^{\omega}$ is denoted X^{∞}. We extend the relation $\leq$ to X^{∞} by

$$\forall \alpha, \beta \in X^{\infty} \quad \alpha \leq \beta \iff \forall n \in \mathbb{P}\ (n \leq |\alpha| \Rightarrow \alpha(n) = \beta(n))$$

with the convention that $\alpha \in X^{\omega} \Rightarrow |\alpha| = \infty$ and

$$\forall n \in \mathbb{P} \quad n < \infty.$$

We thus have $\forall u \in X^{\omega}$, $\beta \in X^{\infty}$ $u \leq \beta \iff \beta = u$. We also have $\forall f \in X^*$, $u \in X^{\omega}$ $f \leq u \iff f = u[|f|]$ and clearly indeed $f < u$ in that case.
An ω-language is any subset of X^{ω}, an ∞-language is any subset of X^{∞} and a "language" will simply be any subset of X^*. A convenient notation is the following

$$\forall \alpha \in X^{\infty} \quad FG(\alpha) = \{f \in X^* \mid f \leq \alpha\}$$

$$\forall L \subset X^{\infty} \quad FG(L) = \cup\ \{FG(\alpha) \mid \alpha \in L\}.$$

To deal with infinite words we mainly use three lemmas.

<u>LEMMA 1</u>. *If* $u_1 \leq u_2 \leq \ldots \leq u_n \leq \ldots$ *is an increasing sequence of finite words in* X^* *ordered by* $\leq$ *then*

- *either the sequence is stationary, which means that*

$$\exists N \in \mathbb{P} : \forall n \geq N \quad u_n = u_N.$$

Clearly u_N *then is the least upper bound of the sequence* u_n.

- *or* $|u_n| \to \infty$ *when* $n \to \infty$.

Then there exists a unique infinite word $u \in X^{\omega}$ *such that*

$$\forall n \in \mathbb{P} \quad u_n \in FG(u).$$

This word u *is the least upper bound of the sequence* u_n.
In both cases the least upper bound is denoted as $\sup u_n$.

LEMMA 2. *For all* $u \in X^\omega$ *and* $L \subset X^\infty$

$$\mathrm{card}\ (FG(u) \cap FG(L)) = \infty \Rightarrow FG(u) \subset FG(L).$$

LEMMA 3 (Koenig's Lemma). *Let, for all* $n \in \mathbb{P}$, E_n *be a finite nonempty subset of a set* E *and* $R \subset E \times E$ *be a relation on* E *such that*

- $\mathrm{card}(\cup E_n \mid n \in \mathbb{P}) = \infty$
- $\forall n \in \mathbb{P} \quad \forall y \in E_{n+1} \quad \exists x \in E_n: (x,y) \in R$

Then there exists an infinite sequence of elements of E, $x_1, x_2, \dots, x_n, \dots$ *such that*

$$\forall n \in \mathbb{P} \quad x_n \in E_n \quad \textit{and} \quad (x_n, x_{n+1}) \in \mathbb{R}.$$

An equivalent way of stating lemma 3 is the following.

LEMMA 3'. *Every infinite tree of finite degree has an infinite branch.*

We shall mainly use lemma 3, more or less each time we shall have to prove the existence of an infinite object. Let us look at the monoid structure of X^∞. The standard multiplication in X^* can be defined by the formula

$$fg(n) = \begin{cases} f(n) & \text{if} \quad n \le |f| \\ g(n-|f|) & \text{if} \quad |f| < n \le |f| + |g| \\ \Omega \quad \text{if} \quad |f| + |g| < n. \end{cases}$$

The same formula may be applied to the definition of a multiplication in X^∞, with the result that for all $f \in X^*$ $u \in X^\omega$, fu is the infinite word

$$fu(n) = \begin{cases} f(n) & \text{if} \quad n \le |f| \\ \\ u(n-|f|) & \text{if} \quad |f| < n \end{cases}$$

(intuitively one writes f "in front of" u).
For all $\alpha \in X^\infty$, $u \in X^\omega$ one has $u\alpha = u$. (Obviously this is a convention and one could make other conventions with as a result a completely different theory, see COURCELLE [37]). We have the following lemma:

LEMMA 4. $\forall \alpha, \beta \in X^\infty \quad \alpha \leq \beta \Longleftrightarrow \exists \gamma \in X^\infty \; \alpha\gamma = \beta.$

We thus retrieve the standard definition of "is a left factor of". Extending the product to subsets of X^∞ by means of

$$LL' = \{\alpha\beta \mid \alpha \in L \; \beta \in L'\}$$

we can give simple rules to compute it. For $L \subset X^\infty$ define $L^{fin} = L \cap X^*$ and $L^{inf} = L \cap X^\omega$.

LEMMA 5. *For all* $L_1, L_2, L \subset X^\infty$

$$(L_1 \cup L_2)^{fin} = L_1^{fin} \cup L_2^{fin}$$

$$(L_1 \cup L_2)^{inf} = L_1^{inf} \cup L_2^{inf}$$

$$(L_1 L_2)^{fin} = L_1^{fin} L_2^{fin}$$

$$(L_1 L_2)^{inf} = L_1^{inf} \cup L_1 L_2^{inf}$$

$$(L^*)^{fin} = (L^{fin})^*$$

$$(L^*)^{inf} = (L^{fin})^* L^{inf}.$$

We shall also use another operation which produces an ω-language from a language. If $L \subset X^*$ we define

$$L^\omega = \{u \in X^\omega \mid \forall n \in \mathbb{P} \, \exists p_n \in \mathbb{P} : u[p_n] \in L^n \quad \text{and} \quad p_n \to \infty \text{ when } n \to \infty\}.$$

Another way of defining L^ω is to write

$$L^\omega = \{u \in X^\omega \mid u = f_1 f_2 \ldots f_n \ldots \quad \forall n \in \mathbb{P} \; f_n \in L \quad \text{and} \quad \text{card } \{n \in \mathbb{P} \mid f_n = \varepsilon\} < \infty\}.$$

The infinite power L^∞ of L is defined as

$$L^\infty = L^* \cup L^\omega.$$

In other words, L^{ω} is the set of infinite products of words in L which are equal to infinite words.

The set X^{∞} with the order $\leq$ is a good example of a complete partially ordered set. Completeness is defined by the fact that every directed subset has a least upper bound, where a directed subset Δ is a set satisfying

$$\forall\alpha,\beta \in \Delta \quad \exists\gamma \in \Delta \quad \alpha \leq \gamma \text{ and } \beta \leq \gamma.$$

If Δ is directed in X^{∞} then it has a least upper bound, because

- either $\Delta \cap X^{\omega} \neq \emptyset$ and then clearly $\Delta \cap X^{\omega}$ contains only one word, which is the least upper bound,
- or $\Delta \subset X^{*}$ and is then countable: if we write

$$\Delta = \{f_1, f_2, \ldots, f_n, \ldots\}$$

then we can build the increasing sequence $g_1, g_2, \ldots$ with $g_1 = f_1$ and for all $n \in \mathbb{P}$ g_{n+1} an element in Δ which is greater than g_n and f_{n+1} (such an element exists since Δ is directed). Clearly the least upper bound of g_n, which exists by lemma 1, is the least upper bound of Δ.

GENERATION OF INFINITE WORDS BY ALGEBRAIC GRAMMARS

Let $\Xi = \{\xi_1, \ldots, \xi_N\}$ be a set of symbols disjoint from X. The elements of Ξ are called non-terminals and the elements of X are called terminals. An algebraic (context-free) grammar G with terminal alphabet X and non-terminal alphabet Ξ will be written as a system of equations

$$G \left\{ \xi_i = P_i \right. \qquad (i = 1, \ldots, N)$$

where for all $i = 1, \ldots, N$ P_i is a finite subset of $(X \cup \Xi)^{*}$. For any such grammar G we define a relation $\xrightarrow[G]{}$ on $(X \cup \Xi)^{*}$ by

$$f \xrightarrow[G]{} f' \iff \exists g,h \in (X\cup\Xi)^{*}, i \in [N],\ m \in P_i :\ f = g\xi_i h \text{ and}$$
$$g' = gmh.$$

and from $\xrightarrow[G]{}$ we define $\xrightarrow[G]{*}$ as the reflexive and transitive closure of $\xrightarrow[G]{}$. We say that f' derives from f in G iff $f \xrightarrow[G]{*} f'$. In a standard way for all f and G we define the language generated by G from f (i.e. with f as an axiom) as

$$L(G,f) = \{f' \in X^* \mid f \xrightarrow[G]{*} f'\}.$$

A sequence of words $f_1,\ldots,f_{h+1}$ such that for all $j = 1,\ldots,h$ $f_j \xrightarrow[G]{} f_{j+1}$ is called a derivation from f_1 to f_{h+1} and, obviously, we can state that $f' \in X^*$ belongs to the language generated by G from f iff there exists a finite derivation going from f to f' in G.

It is quite natural to define an infinite derivation from f in G as an infinite sequence $f_1 = f, f_2,\ldots,f_n,\ldots$ such that for all $n \in \mathbb{P}$ $f_n \xrightarrow[G]{} f_{n+1}$.

It is immediate from the definition of $\xrightarrow[G]{}$ that, when a(f) is the largest left factor of f which belongs to X^* (we often call it the largest terminal left factor of f), we have

$$f \xrightarrow[G]{} f' \Rightarrow a(f) \le a(f').$$

Hence, if $f_1,f_2,\ldots,f_n,\ldots$ is an infinite derivation in G then the sequence $a(f_1) \le a(f_2) \le \ldots \le a(f_n) \le \ldots$ is an increasing sequence of words in X^*. We say that the derivation f_n is successful iff the sequence $a(f_n)$ has a least upper bound in X^ω, i.e. if $|a(f_n)| \longrightarrow \infty$ when $n \to \infty$. If f_n is successful we say that f_n produces the word $u = \sup a(f_n)$ and we write $f_1 \xrightarrow[G]{\omega} u$. Now the set of infinite words produced by G from f, also called the ω-language generated by G from f, is defined as

$$L^\omega(G,f) = \{u \in X^\omega \mid f \xrightarrow[G]{\omega} u\}.$$

It will be convenient to define

$$L^\infty(G,f) = L(G,f) \cup L^\omega(G,f).$$

We shall say that L is an algebraic ω-language (resp. ∞-language) iff there exists a grammar G and a non-terminal ξ_i such that

$$L = L^\omega(G,\xi_i) \text{ (resp. } L = L^\infty(G,\xi_i)).$$

The reader should notice that our definition of algebraic ω-languages is not the same as that of COHEN and GOLD [9] or LINNA [21], all authors who have recently written on algebraic languages of infinite words.

II. TOPOLOGICAL PROPERTIES OF X^∞

In this chapter we introduce the topology on X^∞ which will be used in the sequel. We shall see that it is a complete metric topology but in fact it did not present itself directly as a metric topology. Looking at the definition of $L^\omega(G,\xi_i)$ one is tempted to relate this set of words to the language $L(G,\xi_i)$ of finite words generated by the same grammar from the same non-terminal. This can be achieved in an elegant manner under very reasonable assumptions.

Say that a grammar G is in Greibach form iff

$$\forall i \in [N] \quad P_i \subset X(X\cup\Xi)^*$$

(in other words, every member of the right-hand side of a rule begins by a terminal) and that a grammar G is reduced iff

$$\forall i \in [N] \quad L(G,\xi_i) \neq \emptyset.$$

For any language $L \subset X^*$ define the adherence of L as

$$\mathrm{Adh}(L) = \{u \in X^\omega \mid FG(u) \subset FG(L)\}.$$

Then we obtain ([27])

__THEOREM 1__. *If* $G\colon \xi_i = P_i$, $i = 1,\dots,N$ *is in Greibach form and reduced, then for all* $i \in [N]$

$$L^\omega(G,\xi_i) = \mathrm{Adh}(L(G,\xi_i)).$$

__PROOF__. Consider $u \in L^\omega(G,\xi_i)$. There exists an infinite successful derivation $f_1 = \xi_1, f_2,\dots,f_n,\dots$ producing $u = \sup a(f_n)$. The grammar G being reduced, for all $g \in (X\cup\Xi)^*$ there exists $h \in X^*$ such that $g \xrightarrow[G]{*} h$. Thus for all $n \in \mathbb{P}$ there exists $\ell_n \in X^*$ such that $f_n \xrightarrow{*} \ell_n$. Obviously by transitivity $\xi_i \xrightarrow[G]{*} f_n$ and $f_n \xrightarrow[G]{*} \ell_n$ imply $\xi_i \xrightarrow[G]{*} \ell_n$ or, equivalently,

$\ell_n \in L(G,\xi_i)$. But now $u = \text{Sup } a(f_n)$ implies $n \in \mathbb{P}\ \exists p_n \in \mathbb{P}$ such that $u[n] \le a(f_{p_n})$ and, since $a(f_{p_n}) \le \ell_n$, we have $FG(u) \subset FG(L(G,\xi_i))$.

To prove the converse, one has to use Koenig's lemma.

Take $u \in Adh(L(G,\xi_i))$. Call ℓ_n a word in $L(G,\xi_i)$ such that $u[n] \le \ell_n$, by definition such a word ℓ_n exists for all n. As usual we define a left most derivation as a derivation

$$f_1,\dots,f_{h+1}$$

such that for all $j = 1,\dots,h$ $f_{j+1} = a(f_j)mh$ for some $m \in P_i$ if $f_i = a(f_j)\xi_i h$. (Thus, at each step of the derivation the rewriting is applied to the leftmost non-terminal)

Now define the following concepts:

- the derivation $f_1,\dots,f_{h+1}$ covers the word $g \in X^*$ iff $g \le a(f_{h+1})$
- the derivation $f_1,\dots,f_{h+1}$ strictly covers g iff

$$a(f_h) < g \le a(f_{h+1}).$$

All the following facts are well-known or easily proved.

- For every $g \in L(G,f)$ there exists a leftmost derivation going from f to g in G.
- If $f_1,\dots,f_{h+1}$ is a left most derivation which covers g, then there exists an initial segment $f_1,\dots,f_{\ell+1}$, $\ell \le h$ of $f_1,\dots,f_{h+1}$ that strictly covers g.
- If $f_1,\dots,f_{h+1}$ is a leftmost derivation in a grammar G in Greibach form then $a(f_{h+1}) \ge k$.

Now we can consider the sequence of sets E_n where, for all $n \in \mathbb{P}$, E_n is the set of leftmost derivations from ξ_i in G which strictly cover u[n]. We clearly have for all $n \in \mathbb{P}$:

- E_n is non-empty, since the leftmost derivation of ξ_i in ℓ_n covers u[n] (hence there exists an initial segment of it which strictly covers u[n]).
- E_n is finite, since if $f_1,\dots,f_{h+1}$ strictly covers u[n] one has $h < |a(f_h)| < n$ since $a(f_h) < u[n]$. We know that the set of derivations of length less than n is finite.
- for every derivation $f_1,\dots,f_{h+1} \in E_{n+1}$ there exists an initial segment of it which belong to E_n by the above remark ($f_1,\dots,f_{h+1}$ which covers u[n+1] a fortiori covers u[n]).

Hence, by Koenig's lemma there exist an infinite sequence $\delta_1,\dots,\delta_n,\dots$

of derivations such that for all n δ_n covers u[n] and δ_n is an initial segment of δ_{n+1}. This sequence has a least upper bound which is an infinite successful derivation producing u from ξ_i in G. □

PROPERTIES OF THE ADHERENCE

Obvious properties of adherences are

- $\mathrm{card}(L) < \infty \Rightarrow \mathrm{Adh}(L)$ is empty
- $L_1 \subset L_2 \Rightarrow \mathrm{Adh}(L_1) \subset \mathrm{Adh}(L_2)$
- $\mathrm{Adh}(L) = \mathrm{Adh}(FG(L))$

A subset A of X^ω is an adherence, in other words there exists $L \subset X$ such that $A = \mathrm{Adh}(L)$, if and only if

$$A = \mathrm{Adh}(FG(A))$$

[Clearly $A = \mathrm{Adh}(L) \Rightarrow FG(A) \subset FG(L) \Rightarrow \mathrm{Adh}(FG(A)) \subset A$ and for all $u \in A$ $FG(u) \subset FG(A) \Rightarrow u \in \mathrm{Adh}(FG(A))$].

We note that not all subsets of X^ω are adherences. For example

$$a^*b^\omega = \{a^nb^\omega \mid n \in \mathbb{N}\}$$

is not an adherence since

$$FG(a^*b^\omega) \supset a^*, \quad \textit{thus} \quad \mathrm{Adh}(FG(A)) \ni a^\omega$$

whereas $a^\omega \notin a^*b^\omega$.

<u>PROPERTY 1.</u> *If* $A = \mathrm{Adh}(L)$ *then*

$$FG(A) = \{f \in X^* \mid \mathrm{card}(fX^* \cap L) = \infty\}$$

FG(Adh(L)) is called the centre of L and is denoted by L^c.

The proof is a straightforward application of Koenig's lemma.

<u>COROLLARY 1.</u> L^c *and* Adh(L) *are empty if and only if* L *is finite* (*i.e.* $\mathrm{card}(L) < \infty$).

The following assertions are all easy to prove

$$L^c = FG(L^c)$$

$$(L^c)^c = L^c$$

$$Adh(L) = Adh(L^c)$$

$$L_1 \subset L_2 \Rightarrow L_1^c \subset L_2^c$$

$$Adh(L_1) = Adh(L_2) \iff L_1^c = L_2^c .$$

PROPERTY 2. *If* L_1 *and* L_2 *are two languages in* X^* *then*

(i) $\quad Adh(L_1 \cup L_2) = Adh(L_1) \cup Adh(L_2)$

(ii) $\quad Adh(L_1L_2) = Adh(L_1) \cup L_1 \, Adh(L_2)$

(iii) $\quad Adh(L_1^*) = L_1^* \, Adh(L_1) \cup L_1^\omega.$

The proof of (i) follows immediately from the observation that

$$\mathrm{card}(FG(u) \cap FG(L)) = \infty \Rightarrow FG(u) \subset FG(L).$$

We prove (ii). Suppose $FG(u) \subset FG(L_1L_2) = FG(L_1) \cup L_1 \, FG(L_2)$.
Then $FG(u) \subset FG(L_1)$ or $FG(u) \subset L_1 \, FG(L_2)$.
If $FG(u) \subset FG(L_1)$ then $u \in Adh(L_1)$. Suppose $FG(u) \not\subset FG(L_1)$. Then there exists a maximal n such that $u[n] \in FG(L_1)$. Thus for all p, $u[p] \in L_1 \, FG(L_2)$ can be factorized in $u[p] = f_pg_p$ with $f_p \in L_1$, $g_p \in FG(L_2)$ and $|f_p| \le m$. This implies that there exists $f \in L_1$ such that $f_p = f$ for an infinite number of p's. Whence $u = fu'$ with $u' \in Adh(L_2)$ and $u \in L_1 \, Adh(L_2)$.
The proof of (iii) is similar to the proof of (ii).

COROLLARY 2. *For* $L_1, L_2 \subset X^*$

$$(L_1 \cup L_2)^c = L_1^c \cup L_2^c$$

$$(L_1L_2)^c = \begin{cases} FG(L_1) \cup L_1L_2^c & \text{if } \mathrm{card}(L_2) = \infty \\ L_1^c & \text{if } \mathrm{card}(L_2) < \infty \end{cases}$$

$$L_1^c = \begin{cases} L_1^* \, FG(L_1) & \text{if } L_1 \neq \emptyset \text{ and } L_1 \neq \{\varepsilon\} \\ \emptyset \text{ if } L_1 = \emptyset \text{ or } L_1 = \{\varepsilon\} \end{cases}$$

Call a language central when it is equal to its centre $L = L^c$. A central language is either empty or infinite.

PROPERTY 3. *The family of non empty central languages is closed under union, product and star.*

Finally call an ∞-language $L \subset X^\infty$ closed when $L \supset \mathrm{Adh}(FG(L))$. We have

PROPERTY 4. *The family of closed ∞-language is closed under union, product and infinite power.*

We only prove the last assertion. Suppose $L \supset \mathrm{Adh}(FG(L))$.

Consider $L^\infty = L^* \cup L^\omega = (L^{fin})^\omega \cup (L^{fin})^* L^{inf}$. We have $FG(L^\infty) = (L^{fin})^* FG(L)$ whence $\mathrm{Adh}\ FG(L) = \mathrm{Adh}(L^{fin})^* \cup (L^{fin})^* \mathrm{Adh}(FG(L)) = (L^{fin})^* \mathrm{Adh}(L^{fin}) \cup$ $\cup (L^{fin})^\omega \cup (L^{fin})^* \mathrm{Adh}(FG(L))$ and this is clearly contained in L^∞. □

A NATURAL TOPOLOGY ON X^∞

Consider the mapping cl: $2^{X^\infty} \to 2^{X^\infty}$ defined by

$$cl(L) = L \cup \mathrm{Adh}(FG(L)).$$

We have already proved that this mapping satisfies the axioms

$$cl(\emptyset) = \emptyset$$

$$cl(cl(L)) = cl(L)$$

$$L \subset cl(L)$$

$$cl(L_1 \cup L_2) = cl(L_1) \cup cl(L_2).$$

These four axioms are sufficient for the family of complements of closed sets to be the open sets of a topology on X^∞. It happens that this topology is a metric topology. For all $\alpha, \beta \in X^\infty$ define the distance $d(\alpha,\beta) \in \mathbb{R}_+$ by

$$d(\alpha,\beta) = \begin{cases} 2^{-\min\{n \mid \alpha(n) \neq \beta(n)\}} & \text{if there exists an } n \text{ such that } \alpha(n) \neq \beta(n), \\ 0 & \text{if there does not exist such an } n \text{ (in which case } \alpha = \beta). \end{cases}$$

One easily verifies that d satisfies the axioms of an ultrametric-distance, namely

$$d(\alpha,\beta) = 0 \iff \alpha = \beta$$
$$d(\alpha,\beta) = d(\beta,\alpha)$$
$$d(\alpha,\beta) \leq \max(d(\alpha,\gamma), d(\beta,\gamma))$$

(this last axiom is stronger than the triangular inequality

$$d(\alpha,\beta) \le d(\alpha,\gamma) + d(\beta,\gamma)$$

and characterizes the ultrametric distances of BOURBAKI [8]).

The topology associated with this ultrametric is given by the following basis of neighbourhoods for each point $\alpha \in X^{\infty}$

$$\{B(\alpha,\frac{1}{2^n}) \mid n \in \mathbb{P}\}$$

where $B(\alpha,\frac{1}{2^n})$ is the open ball with center α and radius $\frac{1}{2^n}$, i.e.,

$$B(\alpha,\frac{1}{2^n}) = \{\beta \in X^{\infty} \mid d(\alpha,\beta) < \frac{1}{2^n}\}$$

or also,

$$B(\alpha,\frac{1}{2^n}) = \{\beta \in X^{\infty} \mid \alpha[n] = \beta[n]\}.$$

From this it is quite clear that for any $L \subset X^{\infty}$, Adh(L) is the set of cluster points of L. Indeed

$$\alpha \in \mathrm{Adh}(L) \iff \forall n \in \mathbb{P}\ \exists \ell_n \in L\colon \alpha[n] = \ell_n[n]$$

or

$$\alpha \in \mathrm{Adh}(L) \iff \forall n \in P\ \exists \ell_n \ne \alpha\ \ell_n \in B(\alpha,\frac{1}{2^n}) \cap L.$$

The topological closure of any set $L \subset X^{\infty}$ is the union of L and its set of cluster points, which is precisely

$$\mathrm{cl}(L) = L \cup \mathrm{Adh}(\mathrm{FG}(L))$$

Usually adherence and closure are synonymous, but we have consciously chosen to call adherence the set of cluster points in case $L \subset X^{*}$ to distinguish that part from the closure which is formed of infinite words.

We can say more about this topology:

- a sequence $\alpha_1,\alpha_2,\ldots,\alpha_n,\ldots$ is d-Cauchy iff

$$\forall n \in \mathbb{P}\ \exists N_n \in \mathbb{P}\ \forall p,q \in \mathbb{P}$$

$$p,q \ge N_n \Rightarrow \alpha_p[n] = \alpha_q[n]$$

- the sequence $\alpha_1,\alpha_2,\ldots,\alpha_n,\ldots$ converges to α, written as $\alpha_n \to \alpha$, iff

$$\forall n \in \mathbb{P}\ \exists N \in \mathbb{P}\ \forall p \in \mathbb{P}\quad p \geq N \;\Rightarrow\; \alpha_p[n] = \alpha[n].$$

Obviously, as in any metric space, each converging sequence is d-Cauchy. Conversely, suppose α_n is d-Cauchy. Clearly if we take for all $n \in \mathbb{P}$ the smallest N_n such that

$$p,q \geq N_n \;\Rightarrow\; \alpha_p[n],$$

the sequence $\alpha_{N_n}[n]$ is increasing for $\leq$. As such it has a least upper bound $\alpha = \sup \alpha_{N_n}[n]$ and $\alpha_n \to \alpha$. The space X^∞ equipped with the metric d is thus complete. The fact that X^∞ is compact for the d-topology derives from the observation that from every open covering

$$\{B(\alpha,\frac{1}{2^n}) \mid \alpha \in X^\infty\}$$

one can extract a finite covering:

$$\{B(f,\frac{1}{2^n}) \mid f \in X^{\leq p} = \{\varepsilon\} \cup X \cup X^2 \cup\ldots\cup X^p\}$$

is such a finite subcovering.
The metric d is called totally bounded and it is then known that X^∞ is compact (DUGUNDJI [12]).
A last but interesting remark is that every increasing function is d-continuous iff it is Sup-continuous.
The function $f\colon X^\infty \to X^\infty$ is d-continuous iff the inverse image $f^{-1}(L)$ of any closed set is closed or, equivalently iff for all L the image $f(\bar{L})$ of its closure is contained in the closure $\overline{f(L)}$. The function is Sup-continuous iff for every increasing sequence $\alpha_1 \leq \alpha_2 \leq \ldots \leq \alpha_n \leq \ldots$

$$f(\sup \alpha_n) = \sup(f(\alpha_n)).$$

The fact that d-continuity is equivalent to Sup-continuity derives from the easily proven fact that if $\alpha_n \to \alpha$ there exists a sequence β_n such that $\forall n \in \mathbb{P}\ \beta_n \leq \alpha_n$, the sequence β_n is increasing and $\sup \beta_n = \alpha$.
This equivalence constitutes the main link between the two structures of X^∞:

its structure as a cpo and its structure as a complete metric space.

III. MORE ON ALGEBRAIC ω-LANGUAGES AND FIXED POINT THEOREMS

We first give a substitution theorem which has interesting corollaries concerning the structure of algebraic ω-languages and the algebraicity of the center of an algebraic language.

We first have to extend the notion of substitution to X^∞, which is achieved by the following definition.

Let $\Xi = \{\xi_1,\dots,\xi_N\}$ be a non-terminal alphabet disjoint from the terminal alphabet X. Let $\vec{Q} = <Q_1,\dots,Q_N>$ be an N-vector of subsets of $(X \cup \Xi)^\infty$. The substitution $[\vec{Q}/\vec{\xi}]$ which maps the infinite word u onto the subset $u[\vec{a}/\vec{\xi}]$ is given by: if $u = \alpha_0\xi_{i_1}\alpha_1\xi_{i_2}\dots\alpha_{n-1}\ \xi_{i_n}\alpha_n\dots$ where $\alpha_0,\alpha_1,\dots,\alpha_n,\dots \in X^*$ and $\xi_{i_1},\xi_{i_2},\dots,\xi_{i_n},\dots \in \Xi$ $u[\vec{Q}/\vec{\xi}]$ is the set of infinite words of one of the two forms

$$- \alpha_0h_1\alpha_1h_2\dots\alpha_{n-1}h_n\alpha_n\dots \quad \text{where} \quad \forall n \in \mathbb{P}\ h_n \in Q_{i_n}^{fin}$$

$$- \alpha_0h_1\alpha_1h_2\dots\alpha_{n-1}\ w \quad \text{where} \quad m = 1,\dots,n-1\ h_m \in Q_{i_m}^{fin}$$

$$\text{and} \quad W \in Q_{i_n}^{inf}.$$

N.B. Note that in this definition we require that $u[\vec{a}/\vec{\xi}] \subset (X \cup \Xi)^\omega$, that is no infinite word can be mapped by substitution on a finite word. Without this requirement the following theorem would be false.

THEOREM 2. *Let* G *be the algebraic grammar* G: $\xi_i = P_i$, $i \in [N]$. *Let* $\Xi = \{\bar{\xi}_1,\dots,\bar{\xi}_N\}$ *be an alphabet in one to one correspondence with* Ξ *and disjoint from* $X \cup \Xi$. *Let* $\bar{G}$ *be the grammar* $\bar{\xi}_i = \bar{P}_i$ $i \in [N]$ *where* $\forall i \in [N]$ $\bar{P}_i = \{\alpha\bar{\xi}_j \mid \alpha \in (X \cup \Xi)^*,\ \xi_j \in \Xi$ *and* $\alpha\xi_j(X \cup \Xi)^* \cap P_i \neq \emptyset\}$. *Then for all* $i \in [N]$ *if* $\overrightarrow{L(G)}$ *is the vector* $<L(G,\xi_1),\dots,L(G,\xi_N)>$

$$L^\omega(G,\xi_i) = L\ (\bar{G},\bar{\xi}_i)\ [\overrightarrow{L(G)}/\vec{\xi}].$$

SKETCH OF A PROOF. The complete proof is long since it involves a number of definitions which the reader will find in [26].

The tool is an infinite derivation tree which contains an infinite successful derivation from ξ_i in u (such a tree is drawn below, see the example). If σ is such an infinite derivation tree, it contains a left-most infinite branch (by Koenig's lemma and obvious arguments). The infinite word u is the yield of that part of the tree which is on the left of the left-most infinite branch (the yield is the word formed by the leaves read from left to right).

Call σ' the smallest initial subtree of σ which contains the left-most infinite branch. If we bar the non-terminals which appear on this branch and delete all what is on the right of this branch then what remains of σ' is a derivation tree in $\bar{G}$, call it $\bar{\sigma}$. All what is on the left of the left-most infinite branch in σ is formed of $\bar{\sigma}$ and pending from each non-terminal which is a leaf in $\bar{\sigma}$ is a finite derivation tree in G. □

COROLLARIES 1. *For every algebraic grammar* G *and non-terminal* ξ, $L^{\omega}(G,\xi)$ *can be written as a finite union*

$$L^{\omega}(G,\xi) = \bigcup_{i=1}^{p} L_i(L_i')^{\omega}$$

where L_i, L_i' *are algebraic languages in* X^*.

2. *An adherence* $A = \mathrm{Adh}(L)$ *is the adherence of an algebraic language (we then call it an algebraic adherence) iff*

$$FG(A) = L^c$$

is algebraic.

3. *The* ∞*-algebraic language* L *is closed iff there exists an algebraic grammar in Greibach form such that* $L = L^{\omega}(G,\xi)$.

PROOFS. The grammar G built in theorem 2 is right-linear, with the additional property that it generates only infinite words (for there are no terminal rules)

We can "solve" such a grammar: for example

$$G: \xi = A\xi \quad \text{gives} \quad \xi = A^{\omega}$$

and

$$G: \begin{cases} \xi_1 = A_{11}\xi_1 + A_{12}\xi_2 \\ \\ \xi_2 = A_{21}\xi_1 + A_{22}\xi_2 \end{cases}$$

gives $\xi_2 = A_{22}^{*}A_{21}\xi_1 + A_{22}^{\omega}$, which we can substitute for ξ_2 in the first equation to obtain

$$\xi_1 = (A_{11}+A_{12}A_{22}^{*}A_{21})\xi_1 + A_{12}A_{22}^{\omega}.$$

From this we derive

$$\xi_1 = (A_{11}+A_{12}A_{22}^{*}A_{21})^{*}A_{12}A_{22}^{\omega} + (A_{11}+A_{12}A_{22}^{*}A_{21})^{\omega}.$$

By induction on the number of equations one can easily show that for any right-linear grammar $\bar{G}$ with no terminal rules one has

$$L^{\omega}(\bar{G},\bar{\xi}) = \bigcup_{i=1}^{p} R_i(R_i')^{\omega}$$

where R_i, R_i' are rational subsets of X^*.
(This is indeed valid for all right-linear grammars and is a well-known theorem in the theory of infinite words recognized by finite automata, see EILENBERG [13]).
Now it suffices to substitute $\overrightarrow{L(G)}$ for $\vec{\xi}$ to get $L^{\omega}(G,\xi) = \bigcup_{i=1}^{p} L_i(L_i')^{\omega}$.
where $L_i = R_i[\overrightarrow{L(G)}/\vec{\xi}]$ and $L_i' = R_i'[\overrightarrow{L(G)}/\vec{\xi}]$ are algebraic subsets of X^*. □

2. It suffices to prove that the center of an algebraic language is algebraic. Take L algebraic and a reduced grammar G in Greibach form generating $L\backslash\{\varepsilon\}$. Then

$$L^{\omega}(G,\xi) = \mathrm{Adh}(L(G,\xi)) = \mathrm{Adh}(L\backslash\{\varepsilon\}) = \mathrm{Adh}(L)$$

and $L^{c} = \mathrm{FG}(\mathrm{Adh}(L)) = \mathrm{FG}(L^{\omega}(G,\xi))$. We are thus lead to look at the set of left factors of a finite union $\bigcup_{i=1}^{p} L_i(L_i')^{\omega}$. Obviously

$$\mathrm{FG}(L_i(L_i')^{\omega}) = \mathrm{FG}(L_i) \cup L_i(L_i')^{*}\mathrm{FG}(L_i')$$

and this is algebraic, since the set of left factors of an algebraic language is algebraic. (One can really build a grammar for L^{c} along these lines, see [26]). □

3. Corollary 3 follows from 2 and theorem 1.

EXAMPLE. A very simple reduced grammar in Greibach form is

$$G:\ \xi = a\xi\xi + b.$$

One knows that G generates the Lukasiewicz language

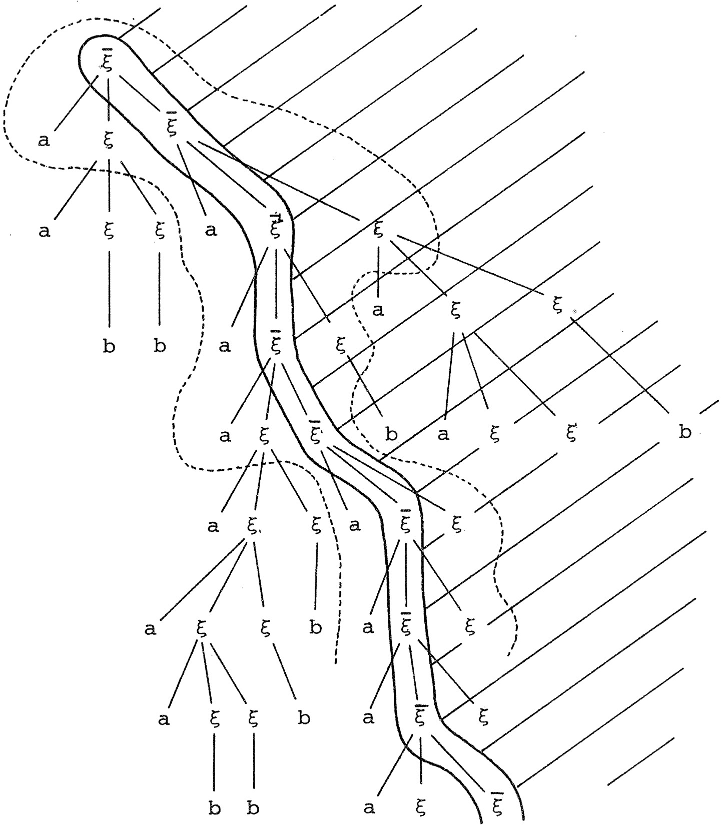
$$L = \{f \in \{a,b\}^* \mid |f|_a = |f|_b - 1 \text{ and } \forall f' < f\ |f'|_a \geq |f'|_b\}.$$

From theorem 1 we derive that

$$L^\omega(G,\xi) = \mathrm{Adh}(L) = \{u \in X^\omega \mid \forall f < u\ |f|_a \geq |f|_b\}.$$

(Indeed every f such that $|f|_a \geq |f|_b$ belongs to FG(L) and no word in L can be a left factor of a word in the adherence since L is prefix.) The tree we now exhibit should help in understanding theorem 2.

EXAMPLE.



The grammar $\bar{G}$ is $\bar{\xi} = a\bar{\xi} + a\xi\bar{\xi}$. We have $L^{\omega}(\bar{G},\bar{\xi}) = (a+a\xi)^{\omega}$, whence

$$L^{\omega}(G,\xi) = (a+a\xi)^{\omega}[L/\xi] = (a+aL)^{\omega}.$$

A grammar for the center is obtained in the following way: a grammar for $FG(L^{\omega}(\bar{G},\bar{\xi}))$ is

$$\bar{\xi} = a\bar{\xi} + a\xi\bar{\xi} + \varepsilon + a.$$

But we can obtain easily a grammar for $FG(L\ (\bar{G},\bar{\xi}))$ where the ξ's appearing at the end of a word are "primed":

$$\bar{\xi} = a\bar{\xi} + a\xi\bar{\xi} + \varepsilon + a\xi' + a\xi\xi'.$$

Now a grammar for the center is obtained by adding to this equation $\xi = a\xi\xi + b$, which will permit the substitution L/ξ, and $\bar{\xi}' = a\xi\xi' + a\xi' + \varepsilon$, which will permit the substitution $FG(L)/\xi$. Eventually we get the grammar

$$\begin{cases} \xi = a\xi\xi + b \\ \xi' = a\xi\xi' + a\xi' + \varepsilon \\ \bar{\xi} = a\xi\bar{\xi} + a\bar{\xi} + a\xi\xi' + a\xi' + \varepsilon. \end{cases}$$

IV. FIXED POINTS

Let G: $\xi_i = P_i$ $i \in [N]$ be an algebraic grammar. If we consider it as a system of equations, to be solved in $(2^{X^*})^N$, we call a solution of G any N-tuple $\vec{Q} = \langle Q_1,\ldots,Q_N \rangle$ of subsets of X^* which satisfies the equations in G i.e. such that

$$\forall i \in [N] \quad Q_i = P_i[\vec{Q}/\vec{\xi}].$$

We can also associate with G the mapping $\hat{G}$: $(2^{X^*})^N \to (2^{X^*})^N$ defined by $\hat{G}(\vec{Q}) = \langle P_1[\vec{a}/\vec{\xi}],\ldots,P_N[\vec{Q}/\vec{\xi}] \rangle$. The set of solutions is then the set of fixed points of $\hat{G}$. Now $(2^{X^*})^N$ ordered by component-wise inclusion is a lattice, and even a complete lattice. And the mapping $\hat{G}$ has the properties that

- $\hat{G}$ is increasing: $\forall \vec{Q},\vec{Q}'\ \vec{Q} \subset \vec{Q}' \Rightarrow \hat{G}(\vec{Q}) \subset \hat{G}(\vec{Q}')$
- $\hat{G}$ is weakly sup-continuous, this meaning that for all increasing sequences $\vec{Q}^{(1)} \subset \vec{Q}^{(2)} \subset \ldots \subset \vec{Q}^{(n)} \subset \ldots$ we have $\hat{G}(\cup_{n\geq 1}\vec{Q}^{(n)}) = \cup_{n\geq 1}\ \hat{G}(\vec{Q}^{(n)})$.

[The inclusion is obvious in one direction. Reversely take $t \in \hat{G}(\cup\vec{Q}^{(n)})_{\ell}$. There exists a word $t' \in P_{\ell}$, $t' = \alpha_0\xi_{i_1}\alpha_1\xi_{i_2}\ldots\alpha_{h-1}\xi_{i_h}\alpha_h$ and words $h_1,\ldots,h_k$ such that for all $j \in [k]$ $h_j \in (\cup Q^{(n)})i_j$ and

$$t = \alpha_0 h_1 \alpha_1 h_2 \ldots \alpha_{k-1} h_k \alpha_k .$$

Now $h_j \in (\cup Q^{(n)})_{i_j}$ implies that there exists n_j such that $h_j \in Q_{i_j}^{(n_j)}$. If we take $n_0 = \max\{n_1,\ldots,n_k\}$ then, for all $j, h_j \in Q_{i_j}^{(n_0)}$ since the sequence $Q^{(n)}$ is increasing. This also proves that $\hat{G}$ is not usually sup-continuous.]
It is well-known that every increasing weakly sup-continuous mapping σ of a complete lattice into itself has a smallest fixed point, $Y(\sigma) = \sup\{\sigma^{(n)}(0)\}$, where 0 is the smallest element in the lattice (Theorem of Knaster-Tarski.) Applying this result to $\hat{G}$ we get:

PROPERTY. For every algebraic grammar G the vector of languages $Y(\hat{G}) = \cup_{n\geq 1}\ \hat{G}^{(n)}(\vec{\emptyset})$ is the smallest solution of G.
An important, old result due to Schutzenberger is the following.

THEOREM 3. *For every algebraic grammar* G *the smallest solution of* G *is equal to the vector of languages*

$$\overrightarrow{L(G)} = \langle L(G,\xi_1),\ldots,L(G,\xi_N)\rangle .$$

Note that this is valid without any restriction on G.
We can now do the same work if we wish to get the greatest solution of an algebraic grammar G. Indeed the mapping $\hat{G}$ is also weakly inf-continuous i.e. satisfies that for all decreasing sequences $\vec{Q}^{(1)} \supset \vec{Q}^{(2)} \supset \ldots \supset \vec{Q}^{(n)} \supset \ldots$ of vectors of $(2^{X^*})^N$ one has

$$\hat{G}(\bigcap_{n\geq 1} \vec{Q}^{(n)}) = \bigcap_{n\geq 1} \hat{G}(\vec{Q}^{(n)}).$$

PROOF. $\hat{G}(\cap\vec{Q}^{(n)}) \subset \cap\hat{G}(\vec{Q}^{(n)})$ is clear. Reversely let $t \in \cap\hat{G}(\vec{Q}^{(n)})_{\ell}$. Then for all $n \in \mathbb{N}$ there exists $p_n \in P_{\ell}$ such that $t \in p_n[\vec{Q}^{(n)}/\vec{\xi}]$ and, since P_{ℓ} is finite, there exists $p \in P_{\ell}$ such that $t \in p[\vec{Q}^{(n)}/\vec{\xi}]$ for an infinity of n's.

Write $p = \alpha_0 \xi_{i_1} \alpha_1 \xi_{i_2} \ldots \alpha_{k-1} \xi_{i_k} \alpha_k$. There exists for every n in this infinity (call it N') words $p_1, \ldots, p_k$ such that

$$t = \alpha_0 p_1 \alpha_1 p_2 \ldots \alpha_{k-1} p_k \alpha_k \quad \text{and for} \quad j \in [h]\ p_j \in \vec{Q}^{(n)}_{i_j}.$$

But there exists only a finite number of factorizations of t into $\alpha_0 p_1 \alpha_1 p_2 \ldots \alpha_{k-1} p_k \alpha_k$. Thus there exists $p_1, \ldots, p_k$ such that $t = \alpha_0 p_1 \ldots \alpha_{k-1} p_k \alpha_k$ and for $j \in [k]$ $p_j \in \vec{Q}^{(n)}_{i_j}$ for an infinity of n's. But since $\vec{Q}^{(n)}$ is decreasing this implies $p_j \in \bigcap \vec{Q}^{n}_{i_j}$.

Now an increasing weakly inf-continuous mapping τ of a complete lattice into itself has a greatest fixed point

$$Z(\tau) = \text{Inf } \tau^{(n)}(1)$$

where 1 is the greatest element of the lattice. This is perfectly dual to the Knaster-Tarski theorem. The grammar G is said to be weakly-Greibach iff, $\forall i \in [N]$ $P_i \subset (X \cup \Xi)^* X (X \cup \Xi)^*$. One can prove

THEOREM 4. *If the grammar* G *is weakly-Greibach then the greatest fixed point of* $\hat{G}$ *exists and is equal to*

$$Z(\hat{G}) = \bigcap_{n \geq 0} \hat{G}^{(n)}(\vec{X^*}) = \overrightarrow{L(G)}.$$

An immediate consequence is that if G is weakly Greibach, G has a unique solution. Let us try now to extend these results to sets of infinite words generated by algebraic grammars. We would like to get the vector of ω-languages $\overrightarrow{L^\omega(G)}$ as a fixed point of some sort of $\hat{G}$. We have already defined the substitution in X^∞ and thus the mapping $\hat{G}$ in $(2^{X^\infty})^N$ still given by

$$\hat{G}(\vec{Q}) = \langle P_j[\vec{Q}/\vec{\xi}], \ldots, P_N[\vec{Q}/\vec{\xi}] \rangle.$$

The mapping $\hat{G}$ thus extended is obviously increasing and remains weakly sup-continuous [the above proof still works!].
Thus $\hat{G}$ has in $(2^{X^\infty})^N$ the same smallest fixed point as in $(2^{X^*})^N$, namely $Y(\hat{G}) = \bigcup_{n \geq 1} \hat{G}^{(n)}(\vec{\emptyset}) = \overrightarrow{L(G)}$.

The awkward phenomenon is that $\hat{G}$ is no longer weakly inf-continuous, as shown by the following example. Let G be the grammar

$$G: \begin{cases} \xi_1 = a\xi_1 b\xi_2 \\ \xi_2 = ab\xi_2 \end{cases}$$

We consider the following decreasing sequence

$$\vec{Q}^{(1)} \supset \vec{Q}^{(2)} \supset \ldots \supset \vec{Q}^{(n)} \supset \ldots$$

where $Q_1^{(n)} = (ba)^n(ba)^* \; Q_2^{(n)} = (ab)^\omega$. Then for all $n \in \mathbb{P}$

$$(ab)^\omega \in \overrightarrow{a\xi_1 b\xi_2[Q^{(n)}}/\vec{\xi}]$$

since

$$(ab)^\omega = a(ba)^n \; b(ab)^\omega.$$

Whence

$$(ab)^\omega \in \bigcap_n \hat{G}(\vec{Q}^{(n)})$$

but $(ab)^\omega \notin \hat{G}(\cap\vec{Q}^{(n)})$ since $\cap \; Q_1^{(n)} = \emptyset$.
Note that in this example the grammar is in Greibach form.
The fact that $\hat{G}$ is not weakly inf-continuous prevents us from using any form of the Knaster-Tarski theorem: one has to use specific combinatorial arguments to prove

THEOREM 5. *If the grammar* G *is in Greibach form then* $\hat{G}$ *has a greatest fixed point in* $(2^{X^\infty})^N$ *equal to*

$$Z_\infty(\hat{G}) = \bigcap_{n\geq 1} \hat{G}^n(\overrightarrow{X^\infty}) = \overrightarrow{L^\infty(G)}$$

where $\overrightarrow{L^\infty(G)}$ *is the vector* $\langle L^\infty(G,\xi_1),\ldots,L^\infty(G,\xi_N)\rangle$.

The proof can be found in [26]. However, there is a topological interpretation of these fixed point results: in order to give it we need to define the Hausdorff metric associated with d on the set of closed subsets of X^∞ which we denote Fer (X^∞). For all $\alpha \in X^\infty$, $L \subset X^\infty$ we define the distance

$$d(\alpha,L) = \min\{d(\alpha,\beta) \mid \beta \in L\}.$$

It is clear that for closed L: $d(\alpha,L) = 0 \iff \alpha \in L$. The distance of two sets $L,L' \subset X^\infty$ is then given by

$$d(L,L') = \max\{\max\{d(\alpha,L') \mid \alpha \in L\}, \max\{d(\alpha',L) \mid \alpha' \in L'\}\}.$$

This is a metric on $\mathrm{Fer}(X^\infty)$ since

$$d(L,L') = 0 \iff L = L'.$$

We have to remark that

$$d(L,L') < \frac{1}{2^n} \iff \pi_{n-1}(L) = \pi_{n-1}(L') \text{ and } FG_n(L) = FG_n(L')$$

where

$$\pi_n(L) = L \cap X^{\leq n} = \{f \in L \mid |f| \leq n\}$$

and

$$FG_n(L) = FG(L) \cap X^n.$$

This stems from the fact that $|f| < n$ and $d(f,\alpha) < \frac{1}{2^n}$ imply $f = \alpha$. We shall now prove that $\mathrm{Fer}(X^\infty)$ with this Hausdorff metric is a complete metric space. The result is obtained from a few lemmas.

<u>LEMMA 6</u>. *If α_n is a sequence of words in X^∞ then it contains an infinite converging subsequence, i.e. there exists a sequence of integers $n_1,n_2,\ldots,n_p,\ldots$ such that α_{n_p} is convergent and $n_p \to \infty$ when $p \to \infty$.*

<u>PROOF</u>. This is an easy application of Koenig's lemma. Two cases arise:

1- $\{\alpha_n \mid n \in \mathbb{P}\}$ is finite. Certainly then there exists $\alpha \in \{\alpha_n\}$ such that $\alpha_n = \alpha$ for an infinite number of n's and the corresponding stationary sequence converges to α.

2- $\{\alpha_n \mid n \in \mathbb{P}\}$ is infinite. Then for all $p \in \mathbb{P}$ $E_p = \{f \in X^p \mid \mathrm{card}\{n \mid \alpha_n \in fX^\infty\} = \infty\}$ is finite and non-empty. Moreover for every $g \in E_{p+1}$ there exists $f \in E_p$ such that $g \in fX$. Whence an infinite sequence $f_p \in E_p$ such that $f_{p+1} \in f_pX$ which has a limit u. Now take n_1 minimal such that $\alpha_{n_1} \in f_1X^\infty$, then $n_2 > n_1$ minimal such that $\alpha_{n_2} \in f_2X^\infty$ and so on. □

We now consider d-Cauchy sequences of sets. The sequence $L_1,L_2,\dots,L_p,\dots$ is d-cauchy iff

$$\forall n \in \mathbb{P} \quad \exists N_n \in \mathbb{P} : \quad p,q \geq N_n \Rightarrow d(L_p,L_q) < \frac{1}{2^n} .$$

<u>LEMMA 7</u>. *Suppose the sequence of sets* L_p *is* d-*Cauchy. Suppose there exists a sequence of integers* p_ℓ, *with* $p_\ell \to \infty$ *when* $\ell \to \infty$, *and a sequence* α_ℓ *where* $\alpha_\ell \in L_{p_\ell}$ *for all* $\ell \in \mathbb{P}$ *such that* $\alpha_\ell \to \alpha$. *Then there exists a sequence* β_p, *where for all* $p \in \mathbb{P}$ $\beta_p \in L_p$ *and such that* $\beta_p \to \alpha$.

<u>PROOF</u>. Suppose $\alpha_\ell \to \alpha$, $\alpha \in X^*$. Then there exists N such that for all $\beta > N$ $\alpha_\ell = \alpha$. If $|\alpha| = n-1$ take N_n given by the d-Cauchy condition such that $p,q \geq N_n \Rightarrow d(L_p,L_q) < 1/2^n$. One has $\alpha \in L_{p_N}$ and thus for all $q \geq$ $\geq \max(p_N,N_n) = M_n$ $\alpha \in L_q$. We can take β_p = any word in L_p for $p < M_n$ and $\beta_p = \alpha$ for $p \geq M_n$ to get the desired sequence.

Suppose $\alpha_\ell \to \alpha$, $\alpha \in X^\omega$, then for all $n \in \mathbb{P}$ there exists $\ell_n \in \mathbb{P}$ such that

$$q \geq \ell_n \Rightarrow \alpha[n] \leq \alpha_q .$$

Consider N_n given by the d-Cauchy condition and $M_n = \min\{p_\ell \mid p_\ell \geq p_{\ell_n}$ and $p_\ell \geq N_n\}$ then for all $p \geq M_n$ one has $\alpha[n] \in FG_n(L_p)$ since

$$\alpha[n] \in FG_n(L_{p_\ell}) \quad \text{and} \quad FG_n(L_p) = FG_n(L_{p_\ell}) .$$

The sequence β is built by taking for all p, $M_n \leq p < M_{n+1}$ β_p = any element in L_p such that $\alpha[n] \leq \beta_p$. □

We now come to

THEOREM 6. *Let* L_p *be a* d-*Cauchy sequence of closed sets. Then* L_p *converges to the closed set*

$$L = \{\alpha \in X^\infty \mid \exists \alpha_n \in L_n : \alpha_n \to \alpha\}.$$

PROOF. We verify that L is closed, i.e. contains the adherence of FG(L). We have $u \in \mathrm{Adh}\ FG(L)$ iff $FG(u) \subset FG(L)$. Or else $\forall n \in \mathbb{P}\ \exists \alpha^{(n)} \in L$: $u[n] \le \alpha^{(n)}$. Since $\alpha^{(n)} \in L \Rightarrow \exists \alpha_m^{(n)} \in L_m\ \alpha_m^{(n)} \to \alpha^{(n)}$ we can write $\forall n \in \mathbb{P}\ \exists p_n \in \mathbb{P} : u[n] \le \alpha_{p_n}^{(n)}$ which means that the sequence $\alpha_{p_n}^{(n)} \to u$. And this implies $u \in L$ by lemma 6.

We need to prove now that $d(L_p,L) \to 0$ when $p \to \infty$. It is clear that if $\alpha \in L$, $\alpha_p \to \alpha$ then $d(\alpha_p,\alpha) \to 0$ when $n \to \infty$ and, since $d(L_p,\alpha) \le d(\alpha_p,\alpha)$, the distance $d(L_p,\alpha) \to 0$ when $u \to \infty$. It is also true that $d(\alpha_p,L)$ can be taken arbitrarily small for all $\alpha_p \in L_p$, p sufficiently large. To be precise we have $\forall n \in \mathbb{P}\ \exists N_n \in \mathbb{P}$ such that

$$p \ge N_n \quad \Rightarrow \quad d(\alpha_p,L) < \frac{1}{2^n}\ .$$

Consider N_n as given by the d-Cauchy condition. Then either $\alpha_p \in L_p$, $p \ge N_n$, is in $\pi_{n-1}(L_p)$ and then it is in $\pi_{n-1}(L_q)$ for all $q \ge N_n$ which implies $\alpha_p \in L$ whence $d(\alpha_p,L) = 0$, or $\alpha_p[n] \in FG_n(L_p)$ and also by the choice of N_n $\alpha_p[n] \in FG_n(L_q)$ for all $q \ge N_n$. But then there exists a sequence β_q such that $\beta_q \in L_q$ and $\alpha_p[n] \le \beta_q$. By lemma 6 there is a converging subsequence with limit α such that $\alpha_p[n] \le \alpha$ and by lemma 7 we get that $\alpha \in L$ whence $d(\alpha_p,L) < \frac{1}{2^n}$. □

The following lemmas are now useful.

LEMMA 8. *For every algebraic grammar* G, $\hat{G}$ *is continuous as a mapping of* $(2^{X^\infty})^N$ *with its metric topology into itself.*

PROOF. First define the metric on $(2^{X^\infty})^N$ in the ordinary way, that is

$$d(\overrightarrow{Q^{(1)}},\overrightarrow{Q^{(2)}}) = \max_i d(Q_i^{(1)},Q_i^{(2)}).$$

Then one has

$$d(\hat{G}(\vec{Q}^{(1)}),\ \hat{G}(\overrightarrow{Q^{(2)}})) \le d(\overrightarrow{Q^{(1)}},\overrightarrow{Q^{(2)}}).$$

Coming back to the definition it suffices to show for all $f \in (X \cup \Xi)^*$

$$d(f[\overrightarrow{Q^{(1)}}/ \vec{\xi}], f[\overrightarrow{Q^{(2)}}/\vec{\xi}]) \leq d(\overrightarrow{Q^{(1)}},\overrightarrow{Q^{(2)}}).$$

Write $f = \alpha_0\xi_{i_1}\cdots\alpha_{k-1}\xi_{i_k}\alpha_k$ and consider

$$h = \alpha_0h_1\cdots\alpha_{k-1}h_k\alpha_k \in f[\overrightarrow{Q^{(1)}}/\vec{\xi}].$$

Call ℓ the smallest integer such that $h_\ell \notin Q^{(2)}_{i_\ell}$: if such an ℓ does not exist then obviously

$$h \in f[\overrightarrow{Q^{(2)}}/\vec{\xi}] \quad \text{and} \quad d(h,f[\overrightarrow{Q^{(2)}}/\vec{\xi}]) = 0.$$

If ℓ exists then $d(h,f[\overrightarrow{Q^{(2)}}/\vec{\xi}])$ is certainly less than

$$d(\alpha_0h_1\cdots\alpha_{\ell-1}h_\ell\alpha_k\cdots\alpha_k,\ \alpha_0h_1\cdots\alpha_{\ell-1}h'_\ell\cdots\alpha_k)$$

where $h'_\ell \in Q^{(2)}_{i_\ell}$. And this distance is less than

$$\frac{1}{2^{|\alpha_0h_1\cdots\alpha_{\ell-1}|}} \times d(h_\ell,Q^{(2)}_{i_\ell})$$

which is less than $d(\overrightarrow{Q^{(1)}},\overrightarrow{Q^{(2)}})$. □

LEMMA 9. *For every algebraic grammar G, and every fixed point $\vec{Q}$ of $\hat{G}$ the closure $\overrightarrow{cl(Q)} = \langle cl(Q_1),\ldots,cl(Q_N)\rangle$ is a fixed point of $\hat{G}$.*

PROOF. The mapping $\hat{G}$ is closed since X^∞ is compact (see above remark) and $(2^{X^\infty})^N$ is Hausdorff (DUGUNDJI [12]). Whence $\hat{G}(\overrightarrow{cl(Q)})$ is closed. Since $\hat{G}$ is continuous we have

$$\hat{G}(\overrightarrow{cl(Q)}) \subset cl(\hat{G}(\vec{Q})).$$

From $\vec{Q} = \hat{G}(\vec{Q})$ we thus obtain $\overrightarrow{cl(Q)} \subset \hat{G}(\overrightarrow{cl(Q)})$ since $\hat{G}(\vec{Q}) \subset G(\overrightarrow{cl(Q)})$ (by the fact that $\hat{G}$ is increasing), whence $\vec{Q} \subset \hat{G}(cl(\vec{Q}))$ and $\overrightarrow{cl(Q)} \subset \hat{G}(cl(\vec{Q}))$ since $\hat{G}(\overrightarrow{cl(Q)})$ is closed. Finally we have $\overrightarrow{cl(Q)} \subset \hat{G}(\overrightarrow{cl(Q)}) \subset \overrightarrow{cl(Q)}$. □

LEMMA 10. *If* G *is in Greibach form then* $\hat{G}$ *is contracting, in particular*

$$d(\hat{G}(\overrightarrow{Q^{(1)}}),\ \hat{G}(\overrightarrow{Q^{(2)}})) \le \frac{1}{2}\, d(\overrightarrow{Q^{(1)}}, \overrightarrow{Q^{(2)}}).$$

PROOF. Like the proof of lemma 8.

Then we can state, using the Banach fixed point theorem,

THEOREM 7. *For every grammar* G *in Greibach form, the mapping* $\hat{G}$ *of* $\mathrm{Fer}(X^{\infty})^N$ *into itself has a unique fixed point*

$$Z_{cl}(\hat{G}) = \lim_{n\to\infty} \hat{G}^n(\vec{Q}\ )$$

whichever is the initial vector $\vec{Q} \subset \mathrm{Fer}(X^{\infty})^N$. *For every fixed point* $\vec{Q}$ *of* $\hat{G}$ *in* $(2^{X^\infty})^N$ *one has*

$$\overrightarrow{cl(Q)} = Z_{cl}(\hat{G}).$$

REMARK. This last theorem provides a topological proof of several statements made above. We have indeed

$$\overrightarrow{cl(L(\hat{G}))} = Z_{cl}(\hat{G}) = \bigcap_{n\ge 1} \hat{G}^n(\overrightarrow{X^{\infty}})$$

for the limit of a decreasing sequence of closed sets is just their intersection. Take

$$L_1 \supset L_2 \supset \ldots \supset L_n \supset \ldots \qquad \forall n\ L_n \in \mathrm{Fer}(X^{\infty}).$$

If $\alpha \in \bigcap_n L_n$ then the sequence $\alpha_n = \alpha$ for all n converges to α, whence $\alpha \in \lim L_n$. If the sequence α_n, $\alpha_n \in L_n$, converges to α then for all n, α is the limit of a sequence of elements in L whence, since L_n is closed, $\alpha \in L_n$. The same argument shows that $\bigcap_{n\ge 1} \hat{G}^n(\overrightarrow{X^{\infty}})$ is always the greatest fixed point of $\hat{G}$, because it is a fixed point since

$$\hat{G}(\lim \hat{G}^n(\overrightarrow{X^{\infty}})) = \lim \hat{G}^{n+1}(\overrightarrow{X^{\infty}})$$

by the continuity of $\hat{G}$ and if $\vec{Q}$ is any other fixed point, then

$$\vec{Q} \subset \overrightarrow{X^{\infty}} \Rightarrow \vec{Q} = \hat{G}(\vec{Q}) \subset \hat{G}(\overrightarrow{X^{\infty}})$$

and by induction $\hat{G}^n(\vec{Q}) = \vec{Q} \subset \hat{G}^n(\overrightarrow{X^\infty})$ whence $\vec{Q} \subset \bigcap_{n\ 1} \hat{G}^n(\overrightarrow{X^\infty})$. This last result is not in contradiction with the remark that $\hat{G}$ is not weakly inf-continuous: indeed we have

$$\hat{G}(\bigcap \overrightarrow{Q^{(n)}}) = \bigcap G(\overrightarrow{Q^{(n)}})$$

if the sequence is a decreasing sequence of *closed* sets. Similar results on closed subsets of cpo's are to be found in PLOTKIN [30].

V. INFINITE TREES

Most of the results we have mentioned up to now still hold for infinite trees: the set $M^\infty(F,V)$ of finite and infinite trees has a very similar structure to X^∞. In this chapter we give the necessary definitions, stress the differences with X^∞ and state the results, leaving to the reader the easy exercise of adapting the proofs.

Let F be a finite set of function symbols, each one given with an arity $a(f) \in \mathbb{N}$. F_0 is the subset of symbols of arity 0. Let V be a set of variables disjoint from F and Ω be a symbol not in $F \cup V$. We denote by M the maximum of the arities, $M = \max\{a(f) \mid f \in F\}$, and by [M] the set $[M] = \{1,2,\dots,M\}$. A partial F-tree on V is a partial mapping $\sigma\colon [M]^* \to$ $\to F \cup V$ satisfying the following conditions

$$\forall u \in \mathrm{dom}(\sigma) \qquad v < u \;\Rightarrow\; v \in \mathrm{dom}(\sigma)$$

$$\forall u \in \mathrm{dom}(\sigma) \qquad \sigma(u) \in F_0 \cup V \;\Rightarrow\; u[M] \cap \mathrm{dom}(\sigma) = \emptyset$$

$$\forall u \in \mathrm{dom}(\sigma) \qquad \sigma(u) = f \in F \setminus F_0 \;\Rightarrow\; u[M] \cap \mathrm{dom}(\sigma) = \{1,\dots,a(f)\} \text{ or } \emptyset.$$

Elements of the domain are called nodes. It is very convenient to consider a tree as a total mapping of $[M]^*$ into $F \cup V \cup \{\Omega\}$ by letting $\sigma(u) = \Omega$ for $u \notin \mathrm{dom}(\sigma)$. The set of partial trees is then ordered in the same way as X^∞. By definition $\sigma \le \sigma' \iff \forall u \in \mathrm{dom}(\sigma)\ \sigma'(u) = \sigma(u)$. Special attention has to be given to the maximal elements, i.e. to all partial trees σ satisfying $\forall \tau\ \sigma \le \tau \Rightarrow \sigma = \tau$. It is easily seen that σ is maximal iff it satisfies

$$\forall u \in \mathrm{dom}(\sigma) \qquad \sigma(u) = f \in F \setminus F_0 \Rightarrow u[M] \cap \mathrm{dom}(\sigma) = \{1,\dots,a(f)\}$$

(In fact if $\sigma \leq \tau$, τ is obtained by adding nodes pending from the nodes in σ such that $\sigma(u) = f \in F\backslash F_0$ and $u[M] \cap \mathrm{dom}(\sigma) = \emptyset$.)
The set of partial F-trees on V is denoted $M^\infty_\Omega(F,V)$. The set of trees $M^\infty(F,V)$ is exactly the set of maximal partial F-trees on V (which is exactly the free complete F-magma generated by V which appears in [3,5,24]).
One can check that the set of partial trees ordered by $\leq$ is a complete partially ordered set (cpo):

- the empty tree Λ such that $\mathrm{dom}(\Lambda) = \emptyset$ is the smallest element
- suppose the set of partial trees Δ is directed i.e. satisfies

$$\forall \sigma_1, \sigma_2 \in \Delta \quad \exists \sigma_3 \in \Delta \quad \sigma_1 \leq \sigma_3 \quad \text{and} \quad \sigma_2 \leq \sigma_3.$$

Then if $u \in \mathrm{dom}(\sigma_1) \cap \mathrm{dom}(\sigma_2)$ we have

$$\sigma_1(u) = \sigma_3(u) \quad \text{and } \sigma_2(u) = \sigma_3(u) \quad \text{whence} \quad \sigma_1(u) = \sigma_2(u).$$

In other words, all trees in Δ defined at a given node u have the same value at that node. We can thus define the tree Sup Δ given by

$$\mathrm{dom}(\mathrm{Sup}\ \Delta) = \cup \{\mathrm{dom}(\sigma) \mid \sigma \in \Delta\}$$

$$\forall u \in \mathrm{dom}(\mathrm{Sup}(\Delta))$$

$\mathrm{Sup}(\Delta)(u)$ is the common value of all $\sigma(u)$ such that $u \in \mathrm{dom}(\sigma)$. One easily checks that $\mathrm{Sup}(\Delta)$ is the least upper bound of Δ.

We shall be mainly interested in conditions for $\mathrm{Sup}(\Delta)$ to be maximal,

LEMMA 11. $\mathrm{Sup}(\Delta)$ *is maximal (i.e. belongs to* $M^\infty(F,V)$) *iff the greatest lower bound* $\mathrm{Inf}\{2^{-h(\sigma)} \mid \sigma \in \Delta\} = 0$ *where*

$$h(\sigma) = \min\{|u| \mid u \in \mathrm{dom}(\sigma),\ \sigma(u) \in F\backslash F_0 \ \textit{and} \ u[M] \cap \mathrm{dom}(\sigma) = \emptyset\}.$$

PROOF. $\mathrm{Inf}\{2^{-h(\sigma)} \mid \sigma \in \Delta\} > 0$ implies that $\forall \sigma \in \Delta$ there exists $u \in \mathrm{dom}(\sigma)$, $|u| = n$ such that

$$\sigma(u) \in F \backslash F_0 \quad \text{and} \quad u[M] \cap \mathrm{dom}(\sigma) = \emptyset$$

where n is the smallest integer such that $2^{-n} \leq \varepsilon$. There are only finitely many nodes $u \in [M]^n$ whence for infinitely many $\sigma \in \Delta$ the same u: this implies that in $\mathrm{Sup}(\Delta)$ we have

$$u \in \mathrm{Sup}(\Delta), \quad \mathrm{Sup}(\Delta)(u) \in F \backslash F_0 \quad \text{and} \quad u[M] \cap \mathrm{dom}(\mathrm{Sup}(\Delta)) = \emptyset.$$

Clearly $\mathrm{Sup}(\Delta)$ is not maximal since one can add nodes pending from that node u. The if part is straightforward. □

The set of partial F-trees on V, $M_\Omega^\infty(F,V)$, is turned into a complete metric space by defining an ultrametric distance d by

$$d(\sigma,\sigma') = \begin{cases} 2^{-\min\{|u| \mid \sigma(u) \neq \sigma'(u)\}}, \text{ if there exists } u \in [M]^* \text{ such} \\ \text{that } \sigma(u) \neq \sigma'(u) \\ 0, \text{ if for all } u \in [M]^* \ \sigma(u) = \sigma'(u). \end{cases}$$

One checks immediately that d is an ultrametric distance. In order to describe this topology further it is extremely convenient to use converging filter bases rather than converging sequences.

Let us recall a few definitions (see DUGUNDJI [12]). A filter base $\mathcal{B}$ on a set E is a collection of subsets of E satisfying

$$\forall B \in \mathcal{B} \quad B \neq \emptyset$$

$$\forall B_1, B_2 \in \mathcal{B} \quad \exists B_3 \in \mathcal{B}: \quad B_3 \subset B_1 \cap B_2.$$

If E is equipped with a metric d, the filter base $\mathcal{B}$ is said

- to accumulate at point e iff every open ball

$$B(e,\varepsilon) = \{e' \in E \mid d(e,e') < \varepsilon\}$$

  intersects all the elements of the filter base. The point e is called a cluster point of $\mathcal{B}$
- to converge to e (we write $\mathcal{B} \rightarrow e$) iff every open ball $B(e,\varepsilon)$ contains an element of the filter base
- to be d-Cauchy iff for every $\varepsilon > 0$ $\mathcal{B}$ contains an element of diameter less

than ε. The diameter $d(B)$ is given by

$$d(B) = \mathrm{Sup}\{d(e_1,e_2) \mid e_1,e_2 \in B\}.$$

It is clear that a converging filter base is d-Cauchy.

THEOREM 8. *Every* d*-Cauchy filter base in* $M^\infty(F,V)$ *is convergent, that is* $M^\infty(F,V)$ *with the metric* d *is a complete metric space.*

PROOF. We use a correspondence between filter bases and directed sets which relies on the existence of greatest lower bounds in $M^\infty_\Omega(F,V)$. Suppose B is a subset of $M^\infty(F,V)$ and define

$$\mathrm{dom}(\mathrm{Inf}(B)) = \{u \in [M]^* \mid \forall\sigma,\sigma' \in B\ \sigma(u) = \sigma'(u)\}.$$

$\mathrm{Inf}(B)(u)$ is the set of common values of all $\sigma(u)$, $\sigma \in B$. The tree $\mathrm{Inf}(B)$ exists and is clearly the greatest lower bound of B in $M^\infty_\Omega(F,V)$. The correspondence goes both ways: with the filter base $\mathcal{B}$ one associates the directed set $\Delta_{\mathcal{B}} = \{\mathrm{Inf}(B) \mid B \in \mathcal{B}\}$; $\Delta_{\mathcal{B}}$ is directed, for $B_3 \subset B_1 \cap B_2 \Rightarrow$ $\Rightarrow \mathrm{Inf}(B_1) \le \mathrm{Inf}(B_3)$ and $\mathrm{Inf}(B_2) \le \mathrm{Inf}(B_3)$; with the directed subset Δ one associates the filter base $\mathcal{B}_\Delta = \{\cup B(\sigma) \mid \sigma \in \Delta\}$ where $\cap B(\sigma) = \{\sigma' \mid \sigma \le \sigma'\}$. We can now show that every d-Cauchy filter base converges to $\mathrm{Sup}(\Delta_{\mathcal{B}})$. We shall rely on the lemmas:

LEMMA 12. *For all directed subset* $\Delta \subset M^\infty_\Omega(F,V)$, *and all* $n \in \mathbb{P}$ *there exists* $\sigma \in \Delta$ *such that* $\mathrm{dom}(\sigma) \cap [M]^{\le n} = \mathrm{dom}(\mathrm{Sup}(\Delta)) \cap [M]^{\le n}$.

PROOF. $\mathrm{Dom}(\mathrm{Sup}(\Delta)) \cap [M]^{\le n}$ is certainly finite and we can order its elements $u_1,u_2,\ldots,u_p$. For all $i = 1,\ldots,p$ there exists $\sigma_i \in \Delta$ such that $u_i \in \mathrm{dom}(\sigma_i)$. Then we can form a sequence of elements of Δ

$$\tau_1 = \sigma_1, \quad \tau_2,\ldots$$

where for all $i = 1,\ldots,p-1$ τ_{i+1} is an element of Δ such that

$$\tau_i \le \tau_{i+1} \text{ and } \sigma_{i+1} \le \tau_{i+1}.$$

It is clear that $\tau_p \in \Delta$ has the desired property. □

LEMMA 13. *The diameter* d(∩B(σ) *is equal to* $2^{-h(\sigma)-1}$ *(this follows directly from a remark already made)*.

Suppose now $\mathcal{B}$ is d-Cauchy. First we note that $\mathrm{Sup}(\Delta_{\mathcal{B}})$ is maximal, for

$$d(B) \le \frac{1}{2^{n+1}}\ ,\ B \subset \cup\, B(\mathrm{Inf}(B)) \ \Rightarrow\ h(\mathrm{Inf}(B)) \ge n$$

and the condition of lemma is satisfied. Secondly we verify that $\mathcal{B}$ converges to $\mathrm{Sup}(\Delta_{\mathcal{B}})$, i.e. $\forall n \in \mathbb{P}$ $B(\mathrm{Sup}(\Delta_{\mathcal{B}}), \frac{1}{2^n})$ contains an element in B.
We apply the preceding lemma: since Sup(Δ) is maximal, the element Inf(B) which satisfies

$$\mathrm{dom}(\mathrm{Inf}(B)) \cap [M]^{\le n} = \mathrm{dom}(\mathrm{Sup}(\Delta)) \cap [M]^{\le n}$$

is such that h(Inf(B)) ≥ n. Thus $d(\cup B(\mathrm{Inf}(B)) \le \frac{1}{2^{n+1}}$ and a fortiori d(B). But $B(\mathrm{Sup}(\Delta_{\mathcal{B}}), \frac{1}{2^n}\ )$ contains Inf(B), whence it contains $B(\mathrm{Inf}(B), \frac{1}{2^n})$ and this contains B. □

N.B. A strange phenomenon is that, whenever a filter base converges, it converges to maximal element: this is why we can restrict ourselves to filter bases in $M^{\infty}(F,V)$ whose elements contain only maximal trees.

We shall now retrieve the notion of an adherence. First we define M(F,V) (resp. $M_{\Omega}(F,V)$, $M^{\omega}(F,V)$ and $M_{\Omega}^{\omega}(F,V)$) as the set of all finite (resp. partial finite, infinite and partial infinite) trees.
Let $L \subset M^{\infty}(F,V)$ and $\bar{L}$ be its topological closure. We know that $\sigma \in \tau$ iff there exists a filter base $\mathcal{B}$ such that

$$\forall B \in \mathcal{B} \qquad B \cap L \neq \emptyset \qquad \text{and} \qquad \mathcal{B} \to \sigma.$$

Let FG(τ) denote the set of finite left factors of the tree τ, i.e.

$$FG(\tau) = \{\sigma \mid \sigma \le \tau\}$$

and let $FG(L) = \cup\, \{FG(\tau) \mid \tau \in L\}$.
Then $B \cap L \neq \emptyset$ implies $\mathrm{Inf}(B) \in FG(L)$. Whence $\sigma \in \bar{L}$ iff = Sup(Δ) for some

directed subset $\Delta \subset FG(L)$. We can now define the adherence of $L \subset M(F,V)$ to be

$$\mathrm{Adh}(L) = \{\sigma \in M^{\omega}(F,V) \mid FG(\sigma) \subset FG(L)\}$$

and state

THEOREM 9. *The topological closure* $\bar{L}$ *of* $L \subset M^{\infty}(F,V)$ *is equal to* $L \cup \mathrm{Adh}(L)$. *The set* $L \subset M^{\infty}(F,V)$ *is closed iff* $L \supset \mathrm{Adh}(FG(L))$.

ALGEBRAIC TREE GRAMMARS

Before going further we change our notations. We define the set of expressions written with F as function symbols and V as variables as the smallest set of words, Term (F,V), on the alphabet composed of F,V, the left and right parenthesis "(" and ")" and the comma "," which satisfies

$$F_0 \cup V \subset \mathrm{Term}(F,V)$$

$$\forall f \in F \quad \forall t_1,\dots,t_{a(f)} \in \mathrm{Term}(F,V) \qquad f(t_1,\dots,t_{a(f)}) \in \mathrm{Term}(F,V).$$

One can identify M(F,V) with Term(F,V) via the one to one mapping

- $\forall f \in F_0$ $\quad \hat{f}$ is the tree such that $\mathrm{dom}(\hat{f}) = \{\varepsilon\}$, $\quad \hat{f}(\varepsilon) = f$
  $\forall v \in V$ $\quad \hat{v}$ is the tree such that $\mathrm{dom}(\hat{v}) = \{\varepsilon\}$ $\quad \hat{v}(\varepsilon) = v$
- if $\hat{t}_1,\dots,\hat{t}_{a(f)}$ are the trees corresponding to $t_1,\dots,t_{a(f)}$

then the tree corresponding to $f(t_1,\dots,t_{a(f)})$ has the domain

$$1\ \mathrm{dom}(\hat{t}_1) \cup 2\ \mathrm{dom}(\hat{t}_2) \cup \dots \cup a(f)\ \mathrm{dom}(\hat{t}_{a(f)})$$

and for $u = pu'$ in this domain, $1 \le p \le a(f)$, the value $f(t_1,\dots,t_{a(f)})(pu') = \hat{t}_p(u')$. From now on we shall write finite trees as expressions (or terms) and use the factorisation in the free monoid $F \cup V \cup \{(\ , \ )\}$ to write equalities of the form

$$t = \alpha t' \beta$$

to indicate that t' is a subtree of t (where a subtree of σ is any tree τ such that for some $u \in \mathrm{dom}(\sigma)$, $\mathrm{dom}(\tau) = \{v \mid uv \in \mathrm{dom}(\sigma)\}$ and $\tau(v) = \sigma(uv)$ for all $v \in \mathrm{dom}(\tau)$). We shall also use the notion of substitution: if

$t \in M(F,\{v_1,\ldots,v_k\})$, $t[t_1/v_1,\ldots,t_h/v_k]$ is obtained by substituting for every occurrence of v_1 in t the tree $t_1,\ldots,$ for every occurrence of v_k in t the tree t_k. We believe all this is well-known. Next we define an algebraic tree grammar Σ as a system of equations of the form

$$\Sigma \begin{cases} \phi_1(v_1,\ldots,v_{n_i}) = P_i \\ \\ i = 1,\ldots,N \end{cases}$$

where $\Phi = \{\phi_1,\ldots,\phi_N\}$ is a finite set of unknown variables, ϕ_i with arity n_i and, for all i, P_i is a finite subset of $M(F\cup\Phi,\{v_1,\ldots,v_{n_i}\})$. This is a special case of non-deterministic recursive program scheme (ndrps, see below). With each grammar Σ there corresponds a relation $\xrightarrow[\Sigma]{}$ on $M(F\cup\Phi,V)$ defined by

$$t \xrightarrow[\Sigma]{} t' \iff t = \alpha\phi_i(t_1,\ldots,t_{n_i})\,\beta \text{ and } t' = \alpha s[t_1/v_1,\ldots,t_{n_i}/v_{n_i}]\beta$$

$$\text{for some } \alpha,\beta,\ i \in [N],\ s \in P_i.$$

The relation $\xrightarrow[\Sigma]{*}$ is the reflexive and transitive closure of $\xrightarrow[\Sigma]{}$. A derivation (or computation) of Σ from t is a sequence of trees $t_1,t_2,\ldots,t_n,\ldots$ which may be finite or infinite such that $t_1 = t$ and for all $n \in \mathbb{P}$ $t_n \xrightarrow[\Sigma]{} t_{n+1}$.

A finite sequence is said to terminate (or to be terminal) iff $t_n \in M(F,V)$. The set of trees which are results of finite terminating derivations have been considered by several authors (for example FISCHER [16], ENGELFRIET-SCHMIDT [14], ROUNDS [31]).

We write $T(\Sigma,t) = \{t' \in M(F,V) \mid t \xrightarrow[\Sigma]{*} t'\}$ and call it the tree language generated by Σ from t. Our problem is now to define successful infinite derivations. This is where the notion of filter base we have introduced is becoming useful. For $t \in M(F\cup\Phi,V)$ define $\pi_M(t)$ in the following way

$$t \in F_0 \cup V \Rightarrow \pi_M(t) = \{t\}$$

$$t = f(t_1,\ldots,t_{a(f)}) \Rightarrow \pi_M(t) = f(\pi_M(t_1),\ldots,\pi_M(t_{a(f)}))$$

$$= \{f(t'_1,\ldots,t'_{a(f)}) \mid t'_p \in \pi_M(t_p)\}$$

$$t = \phi_i(t_1,\dots,t_{n_i}) \Rightarrow \pi_M(t) = M(F,V).$$

Then clearly

$$t \xrightarrow[\Sigma]{} t' \Rightarrow \pi_M(t') \subset \pi_M(t)$$

and if $t_1,\dots,t_n,\dots$ is an infinite derivation, the sequence $\pi_M(t_1),\dots,\pi_M(t_n),\dots$ is a decreasing sequence of subsets of $M(F,V)$ and hence a filter base.

We say that the derivation is successful iff $\pi_M(t_n)$ is a d-Cauchy filter base which satisfies $\forall\varepsilon > 0\ \exists n \in \mathbb{P}$ such that $d(\pi_M(t_n)) < \varepsilon$. Clearly if $\pi_M(t_n)$ is d-Cauchy it converges to a limit $t' \in M^\infty(F,V)$. This limit is taken as the result of the successful derivation.

NB. One can see where the awkwardness of the use of left factors comes from to define successful derivations. Indeed one can easily define the greatest terminal left factor of t_n: $a(t_n)$ is the terminal partial tree s_n whose domain is maximal (for the inclusion order) such that $s_n \leq t_n$. But even if the trees s_n grow to infinity, i.e. $\text{card}(\text{dom}(\sigma_n)) \to \infty$ when $n \to \infty$, the least upper bound $\text{Sup}(s_n)$ has no reason to belong to $M^\infty(F,V)$. It will usually be only a partial tree. In order that the limit belongs to $M^\infty(F,V)$ one must require that $h(s_n) \to \infty$ when $n \to \infty$.

Let us now define $T^\omega(\Sigma,t) = \lim\{\pi_M(t_n) \mid t_n$ is a successful derivation of Σ from $t\}$ and state results about fixed points which generalize our previous results. To Σ we attach a mapping $\hat{\Sigma}$ of the set $\hat{Q}$ of N-vectors

$$\vec{Q} = \langle Q_1,\dots,Q_N\rangle,\ Q_i \in M^\infty(F,\{v_1,\dots,v_{n_i}\}$$

into itself. We define the substitution $[\vec{Q}/\vec{\Phi}]$, whose result when applied to t in $M(F,V)$ is $t[\vec{Q}/\vec{\Phi}]$, recursively as follows:

if t is terminal $t \in M(F,V)$ then $t[\vec{Q}/\vec{\Phi}] = \{t\}$
if $t = f(t_1,\dots,t_{a(f)})$ then $t[\vec{Q}/\vec{\Phi}] = f(t_1[\vec{Q}/\vec{\Phi}],\dots,t_{a(f)}[\vec{Q}/\vec{\Phi}])$
if $t = \phi_i(t_1,\dots,t_{n_i})$ then $t[\vec{Q}/\vec{\Phi}] = \cup\ \{s[\vec{Q}/\Phi] \mid s \in Q_i\}$.

The mapping $\hat{\Sigma}$ maps $\vec{Q}$ onto

$$\hat{\Sigma}(\vec{Q}) = \vec{P}[\vec{Q}/\vec{\Phi}].$$

We have the two major results.

<u>THEOREM 10</u>. *The infinite intersection* $\bigcap_{n\geq 1} \hat{\Sigma}^n(\overrightarrow{M^\infty(F,V)})$ *is the greatest fixed point of* $\hat{\Sigma}$ *in* Q. *In case* Σ *is Greibach, that is* $P_i \in F((M(F \cup \Phi_1\{v_1,\ldots,v_{n_i}\})^N)$, *then* $\bigcap_{n\geq 1}$ $(\overrightarrow{M^\infty(F,V)})$ *is the unique fixed point of* $\hat{\Sigma}$ *in* Q *which is closed and it is equal to*

$$\overrightarrow{T^\infty(\Sigma)} = \overrightarrow{T(\hat{\Sigma})} \cup \overrightarrow{T^\omega(\Sigma)}$$

where

$$\overrightarrow{T(\hat{\Sigma})} = <T(\Sigma,\phi_1 v_1,\ldots,v_{n_1})),\ldots,T(\Sigma,\phi_N(v_1,\ldots,v_{n_N}))>$$

$$\overrightarrow{T^\omega(\Sigma)} = <T^\omega(\Sigma,\phi_1(v_1,\ldots,v_{n_1}),\ldots,T^\omega(\Sigma,\phi_N(v_1,\ldots,v_{n_N}))>.$$

<u>PROOF</u>. One can give two proofs. One is combinatorial by nature (see ARNOLD-NIVAT [2]). One follows the same lines as the proof we gave above of similar results for algebraic grammars on X^∞: the set $\hat{Q}$ is equipped with an ultra metric distance and the same results of completeness of $\hat{Q}$, continuity of $\hat{\Sigma}$ in the general case, contractivity of $\hat{\Sigma}$ in the Greibach case and compactness of $\hat{Q}$ are established, leading easily to the result. Unfortunately the continuity of $\hat{\Sigma}$ is not very easily proved: one should refer to ARNOLD-NIVAT [3]. □

NB. A difficulty should be underlined here which is that it is not true that every algebraic tree language can be generated by a Greibach grammar. This is a well known fact (BOUDOL [7]). We cannot characterize closed algebraic ∞-tree languages by a condition analogous to Corollary 3 of theorem 2.

VI. METRIC INTERPRETATIONS AND RECURSIVE PROGRAMS

In this last chapter we come to the real motivations of the whole study: we build a fixed point semantics of recursive programs whose computation domains are complete metric spaces rather than cpo's.

We first need to consider the algebraic structure of $M^\infty(F,V)$ which has played no role until now but which is essential in what follows. If F is a set of function symbols, we call an F-magma a structure

$<E_I, \{f_I \mid f \in F\}>$ formed of a non empty domain E_I and a collection of mappings f_I, $f \in F$ where f_I maps $E_I^{a(f)}$ into E_I. A morphism between F-magma is a mapping ϕ of E_I into E_J such that for all $f \in F$, $e_1,\ldots,e_{a(f)} \in E_I$

$$\phi(f_I(e_1,\ldots,e_{a(f)})) = f_J(\phi(e_1),\ldots,(e_{a(f)})).$$

The F-magma structure exists naturally on $M^\infty(F,V)$: on $M(F,V)$ it is given by the very definition of Term(F,V), the mapping f_M: $\mathrm{Term}(F,V)^{a(f)} \to \to \mathrm{Term}(F,V)$ simply maps

$$t_1,\ldots,t_{a(f)} \quad \text{onto} \quad f(t_1,\ldots,t_{a(f)}).$$

The magma constituted by $M(F,V)$ and $\{f_M \mid f \in F\}$ has the important property that every mapping ϕ of V into E_I can be extended in a unique way to an F-magma morphism of $<M(F,V) \mid f_M>$ into the F-magma $<E_I,f_I>$. In other words $<M(F,V) \mid f_M>$ is the free F-magma generated by V (see NIVAT [24], COURCELLE-NIVAT [11]). The mappings f_M can be extended by continuity to $M^\infty(F,V)$: suppose the elements $T_1,\ldots,T_{a(f)}$ in $M^\infty(F,V)$ are defined as limits of filter bases $\mathcal{B}_1,\ldots,\mathcal{B}_{a(f)}$. The family of subsets $\{f_M(B_1,\ldots,B_{a(f)}) \mid B_i \in \mathcal{B}_i\}$ is a filter base since

$$\forall i B_i' \subset B_i \quad \text{implies} \quad f_M(B_1',\ldots,B_{a(f)}') \subset f_M(B_1,\ldots,B_{a(f)}).$$

If for all i, $\mathcal{B}_i$ is d-Cauchy then the filter base just defined is also d-Cauchy, for the diameter satisfies

$$d(f_M(B_1,\ldots,B_{a(f)})) < \max\{d(B_i) \mid i = 1,\ldots,a(f)\}$$

whence it converges to a limit which will be taken as $f_M(t_1,\ldots,T_{a(f)})$. It is a straightforward matter to verify that this limit does not depend on a particular choice of the $\mathcal{B}_1,\ldots,\mathcal{B}_{a(f)}$.

Let us now suppose that the F-magma $<E_I,f_I>$ is also a complete metric space for a metric d_I defined on E_I such that the f_I are continuous mappings. We call the structure $<E_I,d_I,f_I>$ with these properties a complete metric F-magma: the continuity of the f_I means that if $\mathcal{B}_1,\ldots,\mathcal{B}_{a(f)}$ are converging filter bases on E_I with $\mathcal{B}_i \to e_i$ then $f_I(e_1,\ldots,e_{a(f)}) = \lim \mathcal{B}$, where $\mathcal{B}$ is the filter base $\{f_I(B_1,\ldots,B_{a(f)}) \mid B_i \in \mathcal{B}_i\}$.

Now let ϕ be a mapping of V into E_I: we know how to define a unique F-magma morphism of M(F,V) into E_I. And we can from there define for all $t \in M(F,V)$ a mapping t_I of $E_I^{\text{card } V}$ into E_I by letting

$$t_I(e_1,e_2,\ldots) = \phi(t)$$

where $\phi\colon V \to E_I$ is given by $\phi(v_n) = e_n$ for all n.

We adopt vector notation. Denoting the vector $<e_1,e_2,\ldots>$ as $\vec{e}$, we write $t_I(\vec{e}) = \phi_{\vec{e}}(t)$ and also $t.\vec{e}$ for the element $t[e_1/v_1,\ldots,e_n/v_n,\ldots]$ of $M(F,E_I)$ whose value $(t.\vec{e})_I$ is precisely $\phi_{\vec{e}}(t)$. If we try to extend the morphism $\phi_{\vec{e}}$ to $M^\infty(F,V)$, it is natural to do it by continuity: if $T \in M^\infty(F,V) = \lim \mathcal{B}$, for some filter base $\mathcal{B}$, we would like

$$\phi_{\vec{e}}(\mathcal{B}) = \{\phi_{\vec{e}}(B) \mid B \in \mathcal{B}\}$$

to be a convergent filter base and to write $T_I(\vec{e}) = \phi_{\vec{e}}(T) = \lim \phi_{\vec{e}}(\mathcal{B})$. Clearly there is no reason that $\phi_{\vec{e}}(\mathcal{B})$ be d-Cauchy and this is generally not the case. We are thus lead to define the following notion of convergence: the mapping T_I is convergent at point $\vec{e}$ iff the image $\phi_{\vec{e}}(\mathcal{B})$ of a convergent filter base with limit T is d-Cauchy. We then define $T_I(\vec{e}) = \lim \phi_{\vec{e}}(\mathcal{B})$. In case T_I is not convergent at $\vec{e}$, it is said to be divergent. The extension $\phi_{\vec{e}}$ is thus only a partial morphism.

<u>EXAMPLES</u> 1) $F = \{f\}, a(f) = 1, V = \{x\}$

$$M(F,V) = \{f^n(x) \mid n \in \mathbb{N}\}$$

$$M^\infty(F,V) = M(F,V) \cup \{f^\infty\} \text{ where}$$

$$f^\infty = \lim \mathcal{B}, \text{ with } \mathcal{B} = \{\{f^p(n) \mid p > n\}\ n \in \mathbb{N}\}.$$

Consider the F-magma $E_I = [0,1]$, the unit interval of $\mathbb{R}$.

$$d_I\colon E_I \times E_I \to \mathbb{R}_+$$

is given by $d_I(x,y) = |x-y|$ and

$$f_I\colon E_I \to E_I \text{ is the function } \lambda x.\ \frac{1}{1+x}.$$

We may consider f_I^∞ as the continued fraction $\cfrac{1}{1+\cfrac{1}{1+\ddots}}$

One can see that the diameter of

$$\phi_e(B_n) = \phi_e\{f^p(x) \mid p > n\}$$

tends to 0 when $n \to \infty$ for all e. Indeed this diameter is bounded by

$$s_n = \max\left\{ \left| \underbrace{\cfrac{1}{1+\cfrac{1}{1+\ddots\cfrac{1}{1+x}}}}_{n \text{ times}} - \cfrac{1}{1+\cfrac{1}{1+\ddots\cfrac{1}{1+y}}} \right| \; x,y \in [0,1] \right\}$$

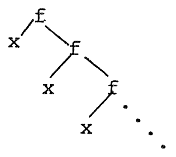
and we can easily establish that $s_n = \frac{1}{n}$. Thus f_I^∞ converges in our sense, which is the same as the classical sense of convergence for continued fractions: the n-th approximant

$$\underbrace{\cfrac{1}{1+\cfrac{1}{1+\ddots\cfrac{1}{1+x}}}}_{n \text{ times}}$$

tends to a limit independent of x for all $x \in [0,1]$.

2. Take $F = \{f\}, a(f) = 2, V = \{x,y\}$.

We cannot describe $M^\infty(F,V)$ as easily as in the first example but surely the infinite tree

T = 

belongs to $M^\infty(F,V)$

A filter base which converges towards T is $\mathcal{B} = \{B_n \mid n \in \mathbb{N}\}$

$$B_n = \{\overbrace{f(x,f(x\ldots f(x}^{n \text{ times}},t) \mid t \in M(F,V)\}.$$

Consider the same $E_I = [0,1]$ with metric $d_I(x,y) = |x-y|$. Take $f_I = \lambda xy.$ $1 \dot{-} x \times y$ which maps $[0,1]^2$ into $[0,1]$. We can show that T_I converges for every $x \in [0,1]$: the diameter of $\phi_x(B_n)$ is indeed

$$\max\{(1-x+x^2 \ldots +(-1)^n xy(-(1-x+x^2+\ldots+(-1)^n xz) \mid z,y \in [0,1]\}$$

and this can be proved to be less than $x^n \times |y-z|$. Clearly $\phi_2(B_n) \to 0$ when $u \to \infty$ for all $0 \le n < 1$. The limit of $\phi_x(B_n)$ thus exists and is equal to

$$\lim(1-x+x^2+\ldots+(-1)^n x^n) = \lim \frac{1+(-1)^n x^{n+1}}{1+x} = \frac{1}{1+x} .$$

Thus T_I, which can be viewed as the power series $1-x+x^2+\ldots+(-1)^n x^n+\ldots$ converges to $\frac{1}{1+x}$ for all $0 \le x < 1$, which is what the theory of power series tells us.

COMPUTATIONS OF RECURSIVE PROGRAMS

A recursive program is a pair formed by
- a non deterministic recursive program scheme Σ
- a metric interpretation.

The ndrps Σ is as above but for the <u>or</u> operator which we introduce to describe a possibility of choice at each step of the computation

$$\Sigma \begin{cases} \phi_i(v_1,\ldots,v_{n_i}) = P_i \\ \\ i = 1,\ldots,N \end{cases}$$

P_i is a finite subset of $M(F \cup \phi \cup \{\underline{or}\}, \{v_1,\ldots,v_{n_i}\})$, the operator <u>or</u> is of arity 2. The meaning of <u>or</u> is operational: it is a choice operator which enters in the definition of computations below.

The metric interpretation I is just a complete metric F-magma

$$M_I = \langle E_I, d_I, \{f_I \mid f \in F\}\rangle.$$

A computation of Σ under I at $t \in M(F\cup\phi\cup\{\underline{or}\}, E_I)$ is a finite or infinite sequence $t = t_1, t_2, \ldots, t_n, \ldots$ where for all n

$$t_n \xrightarrow[\Sigma]{} t_{n+1} \text{ or } t_n \xrightarrow[I]{} t_{n+1} \text{ or } t_n \xrightarrow[or]{} t_{n+1}.$$

We define

$$t_n \xrightarrow[\Sigma]{} t_{n+1} \iff t_n = \alpha\phi_i(s_1,\dots,s_{n_i})\beta \text{ and}$$

$$t_{n+1} = \alpha s[s_1/v_1,\dots,s_{n_i}/v_{n_i}]\beta \text{ for some } s \in P_i$$

$$t_n \xrightarrow[\underline{or}]{} t_{n+1} \quad \text{or} \quad t_n = \alpha(s_1 \text{ \underline{or} } s_2)\beta \text{ and } t_{n+1} = \alpha s_1\beta \text{ or } \alpha s_2\beta$$

$$t_n \xrightarrow[I]{} t_{n+1} \iff t_n = \alpha f(e_1,\dots,e_{a(f)})\beta \text{ and } t_{n+1} = \alpha e\beta$$

$$\text{for some } e_1,\dots,e_{a(f)} \in E_I \quad e = f_I(e_1,\dots,e_{a(f)})$$

When $t_n \xrightarrow[\Sigma]{} t_{n+1}$ we say that t_n is rewritten to t_{n+1}, when $t_n \xrightarrow[I]{} t_{n+1}$ we say that t_n is simplified to t_{n+1}, when $t_n \xrightarrow[\underline{or}]{} t_{n+1}$ we say that t_n is divided (We can introduce many more simplicifation rules and retain the results we state below for this larger set of rules. It may seem awkward indeed not to have rules of the type $0 \times \phi_i(v_1,\dots,v_{n_i}) = 0$ or <u>if</u> $0 = 0$ <u>then</u> 2 <u>else</u> $\phi_i(v_1,\dots,v_{n_i}) = 2$. The definition of a good set of rules, large enough to represent the simplifications actually performed in computers and small enough to keep the Church-Rosser property and commutation with rewritings however, is a completely different problem from the one we are treating here, see HUET [20]).

A computation terminates iff it is finite and the last element obtained is in E_I: obviously this last element is the result of the computation. An infinite computation $t_1, t_2, \dots, t_n, \dots$ is successful iff the filter base $\pi_I(t_n)$ converges to a limit, which is then taken as the result. The mapping π_I: $M(F \cup \Phi \cup \{\underline{or}\}, E_I) \to 2^{E_I}$ is given by

$$\pi_I(e) = \{e\} \quad \text{for all } e \in E_I$$

$$\pi_I(f) = \{f_I\} \quad \text{for all } f \in F_0$$

$$\pi_I(f(s_1,\dots,s_{a(f)})) = f_I(\pi_I(s_1),\dots,\pi_I(s_{a(f)}))$$

$$\pi_I(e_i(s_1,\dots,s_{n_i})) = E_I$$

$$\pi_I(t \text{ \underline{or} } t') = \pi_I(t) \cup \pi_I(t').$$

One immediately checks that

$$t \xrightarrow[\Sigma]{} t' \Rightarrow \pi_I(t') \subset \pi_I(t)$$

$$t \xrightarrow[I]{} t' \Rightarrow \pi_I(t) = \pi_I(t')$$

$$t \xrightarrow[\underline{or}]{} t' \Rightarrow \pi_I(t') \subset \pi_I(t)$$

and it follows that $\pi_I(t_n)$ is a decreasing sequence of subsets of E_I. Thus the condition for $t_1,\dots,t_n,\dots$ to be successful is that $d(\pi_I(t_n))$ tends to zero: intuitively, since $\pi_I(t_n)$ certainly contains the possible values of the term t_n when the unknown function symbols are replaced by anything, the condition means that in the course of the computation one restricts the possible set of values of the computed term, with the effect that in the limit just one value is retained.

EXAMPLES. 1. Σ is the recursive program scheme

$$\phi(x) = f(x,\phi(x))$$

and the interpretation I is the metric F-magma

$$E_I = [0,1]$$

with its ordinary metric $d_I(x,y) = |x-y|$ and

$$f_I(x,y) = \frac{1}{1+xy} .$$

There is a unique infinite computation at point (1) which is

$$\phi(1) \xrightarrow[\Sigma]{} \frac{1}{1+\phi(1)} \xrightarrow[\Sigma]{} \frac{1}{1+\frac{1}{1+\phi(1)}} \longrightarrow \text{-----}.$$

We can compute $\pi_I(t_n)$ in this case.

$$\pi_I(t_1) = \pi_I(\phi(1)) = 1$$

$$\pi_I(t_2) = \pi_I(\frac{1}{1+\phi(1)}) = \{\frac{1}{1+z} \mid z \in [0,1]\} = [\tfrac{1}{2},1].$$

If we define the Fibonaci sequence by

$$\text{fib}(n+1) = \text{fib}(n) + \text{fib}(n-1),\ \text{fib}(1) = 1,\ \text{fib}(0) = 0$$

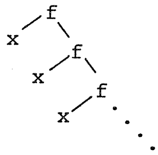
then we have

$$\pi_I(t_{2n}) = \left[\frac{\text{fib}(2n-1)}{\text{fib}(2n)}, \frac{\text{fib}(2n)}{\text{fib}(2n+1)}\right]$$

$$\pi_I(t_{2n+1}) = \left[\frac{\text{fib}(2n+1)}{\text{fib}(2n+2)}, \frac{\text{fib}(2n)}{\text{fib}(2n+1)}\right].$$

The result of this infinite computation is the limit of $\pi_I(t_n)$, which is also the limit of the sequence $\frac{\mathrm{fib}(n)}{\mathrm{fib}(n+1)}$ which is known to be $\frac{-1+\sqrt{5}}{2}$. This example is treated in VUILLEMIN [36].

2. The same Σ computing in the metric F-magma $M(F,\{x\})$ at point $\phi(x)$ computes the infinite tree



for there is only one infinite computation

$$\phi(x) \xrightarrow[\Sigma]{} f(x,\phi(x)) \xrightarrow[\Sigma]{} f(x,f(x,\phi(x))) \longrightarrow ----.$$

This computation is successful: the set $\pi(t_n)$ is

$$\{f(x,f(x,\ldots f(x,s)\ldots)) \mid s \in M^{\infty}(F,V)\}$$

and its diameter is clearly equal to $1/2^n$. □

We now state results. (ARNOLD-NIVAT [4,28]). If $c = t_1,\ldots,t_n,\ldots$ is a finite terminating or infinite successful computation, let $\mathrm{Res}(c)$ denote its result. If Σ is a ndrps, I a complete metric interpretation and

$$t \in M(F\cup\Phi,\{v_1,\ldots,v_k\})$$

define the function

$$\mathrm{Val}_I\Sigma(t): E_I^k \longrightarrow 2^{E_I}$$

by

$$\mathrm{Val}_I\Sigma(t)(e) = \{\mathrm{Res}(c) \mid c \text{ is a finite terminating or an infinite successful computation of } \Sigma \text{ under } I \text{ at point } t.\vec{e}\}.$$

(Usually one is interested in $\text{Val}_I\Sigma(\phi_1(v_1,\ldots,v_{n_1}))$, called the function computed by Σ under I). Denote by M the free interpretation, which is

$$<M^{\infty}(F,V),d,\{f_M \mid f \in F\}>.$$

Every ndrps has such a free interpretation, usually called the Herbrand interpretation.

LEMMA 14. *If* $c = t_1,\ldots,t_n,\ldots$ *is an infinite successful computation of* Σ *under* I *at point* $t.\vec{e}$, *then there exists an infinite successful computation* c' *of* Σ *under* M *at point* t *such that*

$$\text{Res}(c')_I \quad \textit{converges at } \vec{e} \textit{ and} \quad \text{Res}(c')_I(\vec{e}) = \text{Res}(c).$$

A very similar result holds for finite terminating computations (see a proof in NIVAT [24]). The two results make use of the property of pseudo computation, the importance of which was stressed by ROSEN [32].

LEMMA 15. *If* $t_1,t_2,t_3 \in M(F\cup\Phi,E_I)$ *are such that* $t_1 \xrightarrow[I]{} t_2$ and $t_2 \xrightarrow[\Sigma]{} t_3$ *then there exists* t_4 *such that* $t_1 \xrightarrow[\Sigma]{} t_4$ *and* $t_4 \xrightarrow[I]{*} t_3$.

LEMMA 16. *If* c *is a successful infinite computation of* Σ *under* M *at point* t *and* $\text{Res}(c)_I$ *converges at* $\vec{e}$, *then there exists a successful infinite computation* c' *of* Σ *under* I *at* $t.\vec{e}$ *such that* $\text{Res}(c') = \text{Res}(c)_I(\vec{e})$.

From lemmas 14 and 16 one easily obtains

THEOREM 11. $\text{Val}_I\Sigma(t) = \text{Val}_M\Sigma(t)_I$.

This theorem is an extension of theorem 6 of [24]: intuitively it amounts to say that it is equivalent to interpret first and compute afterwards or to compute first and interpret afterwards.

FIXED POINTS OF RECURSIVE PROGRAMS

Consider the ndrps Σ and the complete metric interpretation I. Call $\mathcal{G}$ the set of N-tuples of mappings $\vec{g} = <g_1,\ldots,g_N>$ where g_i maps $E_I^{n_i}$ into 2^{E_I} for all $i = 1,\ldots,N$. The set $\mathcal{G}$ is naturally ordered by inclusion in the following sense

$$\forall g_1 g' \in [E_I^n \to 2^{E_I}]:\ g \subset g' \iff \forall \vec{e} \in E_I^n:\ g(\vec{e}) \subset g'(\vec{e})$$

$$\forall \vec{g}_1 \vec{g}' \in G:\ \vec{g} \subset g' \iff \forall i \in [N]:\ g_i \subset g_i'.$$

With this ordering G is a complete lattice. To Σ,I one associates the mapping $\hat{\Sigma}_I$ of G into itself defined by

$$\hat{\Sigma}_I(\vec{g}) = \vec{p}[\vec{g}/\vec{\Phi}]_I$$

(if $t \in M(F\cup\Phi,V)$,$t[\vec{g}/\vec{\Phi}]$ is obtained by litteraly substituting the symbol g_i for the symbol ϕ_i for all $i \in [N]$, and $t[\vec{g}/\vec{\Phi}]_I$ is obtained as the interpretation under I extended by the rule $\vec{g}_I = \vec{g}$). Now define a contracting interpretation (it could be called Lipschitz) as an interpretation I satisfying the following conditions

$$C_1:\ \forall f \in F_n\ \exists 0 \le \alpha < 1\ \forall e_1,\ldots,e_n,\ e_1',\ldots,e_n' \in E_I$$

$$d_I(f_I(e_1,\ldots,e_n),f_I(e_1',\ldots,e_n')) \le \alpha \max_j d_I(e_j,e')$$

C_2: the diameter $d_I(E_I)$ is finite.

We have the analogue of theorem 7 of [24].

<u>THEOREM 12</u>. *If* Σ *is a Greibach* ndrps *and* I *a contracting interpretation of* Σ *then the* N-*tuple of functions*

$$\mathrm{Val}_I\Sigma(\vec{\Phi}) = \langle \mathrm{val}_I\Sigma(\phi_i(v_1\ldots v_{n_1})),\ldots,\mathrm{val}_I\Sigma(\phi_N(v_1,\ldots,v_{n_N})\rangle$$

is the greatest fixed point of $\hat{\Sigma}$ *in* G, *and the following equality holds*

$$\mathrm{Val}_I\Sigma(\vec{\Phi}) = \bigcap_{n\ge 1} \hat{\Sigma}_I^n(\overrightarrow{E_I})$$

where $\overrightarrow{E_I} \in G$ *is the* N-*tuple of functions whose* N *components map the set of their arguments onto* E_I.

<u>EXAMPLE</u>. Σ_I is the recursive program

$$(x) = \frac{\phi(x)}{4} + \frac{1}{2} \underline{\text{or}} \frac{\phi(x)}{4} + \frac{1}{4} .$$

We compute in [0,1], equipped with its usual metric d defined by

$$d(x,y) = |x-y| .$$

It is clear that C_1 and C_2 are satisfied and thus we can compute the infinite intersection $\bigcap_{n\geq 1} \hat{\Sigma}_I^n(E_I)$, which has to be equal to $\mathrm{Val}_I\ \Sigma(\vec{\Phi})$

$$\hat{\Sigma}_I[0,1] = \frac{[0,1]}{4} + \frac{1}{2} \cup \frac{[0,1]}{4} + \frac{1}{4} = [\frac{1}{4}, \frac{3}{4}]$$

$$\hat{\Sigma}_I^2[0,1] = \frac{[\frac{1}{4}, \frac{3}{4}]}{4} + \frac{1}{2} \cup \frac{[\frac{1}{4}, \frac{3}{4}]}{4} + \frac{1}{4} = [\frac{9}{16}, \frac{11}{16}] \cup [\frac{5}{16}, \frac{7}{16}] .$$

We can see a "Cantor process" appearing: we first divide [0,1] into 4 equal subintervals and delete the left most and right most. Then on each subinterval left we perform the same process: eventually we get to the totally disconnected and measure-0 compact subset of reals whose diadic expansion is of the form 0.u, u $\in$ $(01+10)^{\omega}$. Otherwise an infinite computation of the program is of the form

$$\phi(n) \rightarrow \frac{1}{2^{1+\varepsilon_0}} + \frac{\phi(n)}{2^2} \rightarrow \frac{1}{2^{1+\varepsilon_0}} + \frac{1}{2^{3+\varepsilon_1}} + \frac{\phi(x)}{2^4} \rightarrow ----$$

which eventually leads to a result of the form

$$\sum_{n\geq 0} \frac{1}{2^{2n+1+\varepsilon_n}}$$

with $\varepsilon_n = 0$ or 1. The set of such results is easily seen to be equal to the preceding set. □

REFERENCES

[1] ARNOLD, A. & M. NIVAT, *Non-deterministic recursive program schemes in Fundamentals of Computation Theory*, Lecture notes in Computer Science no 56, Springer Verlag (1977) pp.

[2] ARNOLD, A. & M. NIVAT, *Algebraic semantics of non-deterministic recursive program schemes,* Rapport LITP 78-4, Université Paris VII, (1978), to appear in Math. Syst. Theory.

[3] ARNOLD, A. & M. NIVAT, *The metric space of infinite trees,* Rapport IRIA Laboria no. 323, Rocquencourt (1978).

[4] ARNOLD, A. & M. NIVAT, *Metric interpretations of infinite trees and semantics of non deterministic recursive programs,* Report no. IT.3-78 Université de Lille.

[5] BAKKER DE, J.W., *The fixed point approach in semantics: theory and application,* in Foundations of Computer Science, Mathematical Centre Tract no 63, Amsterdam (1975) pp. 3-53.

[6] BOASSON, L. & M. NIVAT, *Adherences of languages,* Report no. 79-13, Laboratoire d'Informatique Théorique et Programmation, Paris.

[7] BOUDOL, G., *Languages polyadiques algébriques,* Thèse 3$^{\text{ème}}$ cycle, Université Paris VII (1975).

[8] BOURBAKI, N., *Topologic Générale,* Hermann, Paris (1977).

[9] COHEN, R. & A. GOLD, *Theory of ω-languages,* Journ. Comp. Syst. Sci., Vol. 15 (1977) pp. 169-208.

[10] COURCELLE, B. & M. NIVAT, *Algebraic families of interpretations,* in 17th Symposium on Foundations of Computer Science, Houston (1976) pp. 137-146.

[11] COURCELLE, B. & M. NIVAT, *The algebraic semantics of recursive program schemes,* in 7th Symposium on Mathematical Foundations of Computer Science, Lecture Notes in Computer Science no 64, Springer Verlag (1978) pp. 16-30.

[12] DUGUNDJI, T., *Topology,* Allyn and Bacon, Boston (1966).

[13] EILENBERG, S., *Automata, languages and machines,* Vol. A, Academic Press, New York (1974).

[14] ENGELFRIET, J. & E. SCHMIDT, *IO and OI,* Journ. Comp. Syst. Sci. Vol. 15 (1977) pp. 328-353 and Vol. 16 (1978) pp. 67-99.

[15] ELGOT, C., S. BLOOM & R. TINDELL, *On the algebraic structure of rooted trees,* Journ. Comp. Syst. Sci. Vol. 16 (1978) pp. 362-399.

[16] FISCHER, M., *Grammars with macro-like productions,* Doctoral dissertation, Harvard University (1968).

[17] GOGUEN, J.A. & J. THATCHER, *Initial algebra semantics,* in 15th Symposium on Switching and Automata Theory, New Orleans (1974) pp. 63-77.

[18] GOGUEN, J.A., J. THATCHER, E. WAGNER & J. WRIGHT, *Initial algebra semantics and continuous algebras,* Journ. Assoc. Comp. Mach. Vol. 24 (1977) pp. 68-95.

[19] GUESSARIAN, I., *Schemas de programmes recursifs polyadiques,* Thèse de doctorat d'état, Université Paris VII (1975).

[20] HUET, G., *Confluent reductions,* in 18th Symposium on Foundations of Computer Science, Providence (1977) pp. 30-47.

[21] LINNA, M., *On ω-words and ω-computations,* Annales Universitatis Turkueusis (1975).

[22] MANNA, Z. & J. VUILLEMIN, *Fixpoint approach to the theory of computation,* Comm. Assoc. Comp. Mach. Vol. 15 (1972) pp. 528-536.

[23] MYCIELSKI, J. & W. TAYLOR, *A compactification of the algebra of terms,* Algebra Universalis Vol. 6 (1976) pp. 159-163.

[24] NIVAT, M., *On the interpretation of recursive polyadic program schemes,* Symposia Mathematica Vol. 15 (1975) pp. 255-281.

[25] NIVAT, M., *Interpretation universelle d'un schema de programme recursif,* Rivista di Informatica, Vol. 7 (1977), pp. 9-16.

[26] NIVAT, M., *Mots infinis engendreś par une grammaire algebrique,* RAIRO Informatique Théorique Vol. 11 (1977) pp. 311-327.

[27] NIVAT, M., *Sur les ensembles de mots infinis engendrés par une grammaire algébrique,* RAIRO Informatique Théorique Vol. 12 (1978) pp. 259-278.

[28] NIVAT, M. & A. ARNOLD, *Calculs infinis, interpretations métriques et plus grands points fixes,* in Colloque de Math. Appli., Palaiseau (1978) pp. 191-208.

[29] PARK, D., *Fixpoint induction and proof of program properties,* in Machine Intelligence no 5, Edinburgh University Press (1969) pp. 59-78.

[30] PLOTKIN, G., *A power domain construction,* Soc. Ind. Appl. Math. Journ. Comp. Vol. 5 (1976) pp. 452-487.

[31] ROUNDS, W., *Mappings and grammars on trees,* Math. Syst. Theory Vol. 4 (1970) pp. 257-287.

[32] ROSEN, B., *Tree manipulating systems and Church-Rosser theorems,* Journ. Assoc. Comp. Mach. Vol. 20 (1973) pp. 160-188.

[33] SCOTT, J., *The lattice of flow diagrams,* in Symposium on Semantics of Algorithmic Languages, Lecture notes in Mathematics no 188, Springer-Verlag (1971) pp. 311-366.

[34] SCOTT, D., *Outline of a theory of computation,* Oxford Programming Research Group Memo no PRG 2 (1972).

[35] SMYTH, M., *Power domains,* Journ. Comp. Syst. Sci. Vol. 16 (1978) pp. 23-36.

[36] VUILLEMIN, J., *Syntaxe, semantique et axiomatique d'un langage de programmation simple,* Thèse de doctorat d'état, Université de Paris VII (1974).

Additional bibliography (added in proof)

[37] COURCELLE, B., *Frontiers of Infinite trees,* RAIRO Informatique Théorique, Vol. 12 (1978) pp. 319-337.

DYNAMIC LOGIC

V.R. Pratt

MIT, Cambridge, USA

The distinction made here between static and dynamic logic has a very simple character, yet can play a central and unifying role in logic as a vantage point from which one can compare propositional calculus, predicate calculus, intensional logics such as modal logic and temporal logic, various algorithmic logics (logics of programs), and Quine's notions of transparency and opacity.

Background

Logic is metamathematics, that is, its objects of study are the unremarked-on and unnamed expressions that are used to make remarks and name objects in ordinary mathematics. Typical expressions are "0", "3x+2", "x=y+2", "$(x+y)(x-y) = x^2-y^2$", "$\forall x\exists y[p(x,y)\vee q(y,x)]$". The logician collects a set L of such expressions, calls this set a *language*, and proceeds to study its meaning *(semantics)* and its manipulation *(axiomatics)*.

Meaning is specified with the help of a *semantic domain* D; D might contain natural numbers, truth values, functions on the reals, predicates on polynomials, and so on. The connection between L and D is made with a meaning function or *interpretation* $\mu:L\to D$, assigning to each expression an object in the semantic domain.

If every expression in L had an agreed-on meaning in the above sense, a single (fixed) interpretation would suffice. Such a language would consist only of constants (that is to say, constant-value expressions), and would contain nothing worthy of the attention of a logician per se. If on the other hand every expression in L could have an arbitrary meaning, any element of $L\to D$ could serve as an interpretation, whence L would in effect consist only of variables and again would not be worth studying. What makes logic interesting is that the range of possible interpretations lies between these two extremes.

For a function from L to D to pass muster as an interpretation it must satisfy a collection of constraints. For example we might have the following constraints on μ.

$$\begin{aligned} \mu(0) &= \text{the zero (unique additive identity) of D} \\ \mu(x+y) &= \mu(x) \text{ plus } \mu(y) \\ \mu(p\wedge q) &= \textit{true} \text{ if } \mu(p) = \textit{true} \text{ and } \mu(q) = \textit{true} \\ &\quad \textit{false} \text{ otherwise.} \end{aligned}$$

These constraints have a special form. We shall assume throughout this paper that every expression is of the form <s,t>, consisting of a symbol s (the *operator*) together with an n-tuple $t = (t_1,\ldots,t_n)$ of expressions (the *arguments*). (Ordinary constants have a zero-tuple of arguments). The most general form of the above constraints is:

$$\mu(<s,t>) = F_s(\mu(t_1),\ldots,\mu(t_n)).$$

That is, the meaning of the expression <s,t> is defined recursively as some function F_s of the meanings of the arguments, the function depending on s but not on μ. (If we want to have f(x) in L, meaning that f denotes some function (which one depending on μ) to be applied to x, we will write the application of f to x explicitly as $\gamma(f,x)$. The more usual convention of always applying μ to s as well as to the t_i's is less appropriate for our account of dynamic logic.) Constraints of this form we shall call *semantic constraints*.

The domain (in the functional sense) of F_s in the above is significant insofar as it acts as an additional constraint, namely on the possible values of the t_i's. For example, if there is a constraint for the expression $p \wedge q$ with $F_\wedge$ being conjunction, a function with domain {*true, false*}, then even if the expression p (or q) has no explicit constraint of its own, its possible values *in the expression* $p \wedge q$ are confined to the domain of $F_\wedge$, *true* and *false*.

The expression x is said to be *interpreted* when some semantic constraint $\mu(x) = \ldots$ applies to it, and otherwise is *uninterpreted*. We write L_0 for the uninterpreted subset of L. It should be apparent that if all expressions are finite (actually, well-founded is sufficient) then for *every* element of $L_0 \to D$ there is a *unique* extension of that element to an interpretation (i.e. to an element of $L \to D$ satisfying the semantic constraints). Since every interpretation uniquely determines an element of $L_0 \to D$, it follows that there is a one-to-one correspondence between the elements of $L_0 \to D$ and the interpretations. For this reason it is common in logic texts to call instead the elements of $L_0 \to D$ the interpretations. Our reluctance to follow this convention is due to the fickle nature of L_0, discussed below.

Clearly, what conventionally pass for variables will have to be uninterpreted expressions in this scheme of things. A little loosely perhaps, we will henceforth refer to *any* uninterpreted expression as a variable. (A little later we will discuss some consequences of this somewhat non-standard view of variables.)

Notice that, as defined above, a semantic constraint mentions only one interpretation, the one it is constraining. This is the defining characteristic of a *static* logic. If one uses the constraints to evaluate an expression recursively, the interpretation remains unchanged (static) as the evaluation proceeds; there are no side effects, so to speak. Quine refers to operators defined in this way as being *referentially transparent* [44]; what the operator's arguments refer to can be seen from "outside" the expression, i.e. the operator does not "block the view".

The Limits of Static Logic

Consider the following expressions.

(1) $\forall x \exists y(x=y)$

(2) Necessarily $x+y = y+x$

(3) After setting x to 1, $x = 1$.

Each of these, we argue, involves concepts beyond the scope of static logic. The reason is that there is no function F such that the meaning of $\forall xp$ can be specified with a constraint of the form $\mu(\forall xp) = F(\mu(p))$, and similarly for the other constructs.

States

To give an account of these expressions we introduce the notion of *state*. One quite workable arrangement is to define a state to be an interpretation. However it will be slightly more convenient for us (and consistent with ordinary practice in modal logic) to postulate *a priori*, as part of D, a set of states $W = \{u,v,w,\ldots\}$, along with a function $\pi{:}W \to (L \to D)$ which assigns to each state an interpretation. We shall frequently abbreviate $\pi(u)(e)$ to $u \models e$ (think of $\models$ as π on its side). When e is a formula $u \models e$ will be a truth value; this special case coincides with the conventional usage of $\models$. We define $\hat{\pi}$: $L \to (W \to D)$ as $\hat{\pi}(e)(u) = \pi(u)(e)$. Notice that π need not be an injection (1-1), that is, it is possible to have $u \models e = v \models e$ for all expressions $e \in L$ and still not have $u = v$; such pairs of states are *equivalent* but not *equal*.

Now the meaning of expression (1) above is that no matter what x is changed to y can then be changed so that $x = y$. Putting it more precisely, we take $u \models \forall x \exists y(x=y)$ to be *true* just when $v \models \exists y(x=y)$ is true for *every* v such that $u \models z = v \models z$ for all variables z other than x. In turn $v \models \exists y (x=y)$ is true just when $w \models x=y$ is true for *some* w such that $v \models z = w \models z$ for all variables z other than y.

We can put this a little more succinctly if we let R_x denote the binary relation on states such that uR_xv whenever $u \models z = v \models z$ for all variables z other than x. (Thus R_x is an equivalence relation on W.) Then

$$u \models \forall xp = \bigwedge_{uR_xv} v \models p \qquad \text{(the conjunction of } v \models p \text{ for all } v \text{ s.t. } uR_xv)$$

Similarly we may take $\exists xp$ to be defined thus.

$$u \models \exists xp = \bigvee_{uR_xv} v \models p.$$

The only difference is the use of $\vee$ in place of $\wedge$. Notice that $u \models \exists xp$ and $u \models \neg\forall x\neg p$ must be the same for all u in W, that is, these are *equivalent* expressions. Just as $\vee$ is the dual of $\wedge$, so is $\exists x$ the dual of $\forall x$.

The advantage of this way of looking at $\forall x$ and $\exists x$ is that it will also be how we shall look at concepts like "necessarily" and "after setting x to 1". This treatment of "necessarily" was first defined carefully by KRIPKE [25, 26]. While Kripke (deliberately) did not propose any particular binary relation, let us consider the relation R such that uRv for all states u and v, the so-called complete binary relation on states. Then define "necessarily" as follows, writing "[]" for "necessarily".

$$u \models []p = \bigwedge_{uRv} v \dashv p.$$

This says that p is necessarily true just when it is true in *all* states v, since R is complete. It follows that u $\models$ []p is the same for all u's. This particular interpretation of "necessarily" is not implausible.

As with $\forall$x, "necessarily" has a dual, "possibly", written "<>". We have as before

$$u \models \langle\rangle p = \bigvee_{uRv} v \models p$$

We also have the equivalence of <>p with $\neg[\,]\neg$p.

Now consider our third example, "after setting x to 1, x = 1". Even this formula can be fitted into the above framework. Let uRv hold just when uR_xv (as defined for example (1)) holds and v $\models$ x = u $\models$ 1. That is, x is set to 1, and there are no other effects (on L_0). Then

$$u \models (\text{after setting } x \text{ to } 1,\ x = 1) = \bigwedge_{uRv} v \models x = 1$$

Of course the value of this is *true*, as with the preceding two examples. In this case it doesn't make any difference whether we write $\wedge$ or $\vee$, so that "after setting x to 1" is its own dual.

For notational convenience we abbreviate "set x to 1" as "x := 1", and, in imitation of the notation for "necessarily", we abbreviate "after x := 1" as "[x:=1]", which as we remarked is the same as its dual "<x:=1>".

Form of constraint in DL

We would like to think of the above equations for quantifiers etc. as semantic constraints. To do so, however, we must abandon the requirement that a semantic constraint mention only a single state. In so doing we make the transition from static to dynamic logic.

The general form of a semantic constraint in dynamic logic is

$$\hat{\pi}(\langle s,t\rangle) = F_s(\hat{\pi}(t_1),\ldots,\hat{\pi}(t_n)). \qquad (\text{Recall } \hat{\pi}(e)(u) = \pi(u)(e)).$$

This is actually the same as the general form for static logic, with $\hat{\pi}$ in place of μ. The difference is that $\hat{\pi}$ is a dynamic meaning function; it yields the meaning of an expression *as a function of state*. In this framework, our original concept of a semantic constraint in static logic takes the form

$$u \models \langle s,t\rangle = F_s(u \models t_1,\ldots,u \models t_n) \qquad \text{for all u in W.}$$

We shall henceforth refer to this special form as a *static constraint*, and the more general one above as a *dynamic constraint*. From now on, as a notational matter, we use $u \models$ in place of the (shortlived!) μ.

The general form permits $u \models \langle s,t\rangle$ to depend on values of the t_i's in other states than u. It does not however permit it to depend arbitrarily on the t_i's themselves, which *are* evaluated, even if not in u. In this way, although we cannot substitute equals as we could in static logic (e.g. $u \models (x=y \supset \phi(x) \equiv \phi(y))$ is no longer always true where ϕ is an arbitrary formula - consider $x=y \supset [\,](x=y) \equiv [\,](y=y)$), we *can* still substitute equivalents. That is, if $\hat{\pi}(a) = \hat{\pi}(b)$ then we do know that $u \models (\phi(a) = \phi(b))$ is *true* for all u in W.

(For LISP afficionados: note that the extent to which dynamic logic is a step up from static logic is less than the extent to which FEXPR's in LISP are a step up from EXPR's. The additional power of a FEXPR over an EXPR is that the FEXPR can inspect the form of the arguments, for example being able to distinguish $p \wedge q$ from $q \wedge p$, which is beyond the power of dynamic logic.)

Loose ends

We are now in a position to point out a peculiarity of our view of variables. When extra constraints are taken notice of (as might happen in the course of following an argument, when it becomes apparent that say propositional reasoning alone does not support the argument), certain expressions that hitherto were treated as variables now become constrained. A simple example would be the expression 0. As long as 0 remains uninterpreted it acts as a variable, and peculiar expressions such as $\exists 0(x=0)$ then have the same meaning as $\exists y(x=y)$. As soon as 0 is assigned a fixed interpretation, $\exists 0(x=0)$ means something else. According to the definition of $\exists y$ above, $\exists 0(x=0)$ would be equivalent to $x = 0$ when 0 is interpreted.

It is a simple enough matter to include a syntactic condition on $\exists$, :=, and other variable-manipulating operators, so that only never-to-be-interpreted variables can be so manipulated. However such a condition would play no significant role in dynamic logic, and would require us to draw a distinction between the unconstrained expressions and variables. Thus we omit it from the theory in the interests of minimizing baggage.

For the remainder of this paper we adopt the convention of writing all expressions we want to be considered uninterpreted using single letters. Thus any occurrence of "x+y" is understood to be interpreted.

An operator definable in dynamic logic but not in static logic is said to be *referentially opaque,* again following Quine. Actually this is a somewhat more mathematically precise definition than Quine had to offer.

The examples of dynamic logic we have seen thus far, without getting into any depth, have already given some idea of the range of domains that can be served by dynamic logic: quantificational calculus, modal logic, and algorithmic logic (i.e. logic of programs, $x := 1$ being a program).

In this connection it may be worth remarking that the semantics of LUCID [1] are presented with emphasis on $\hat{\pi}$; in fact there is no global set of states in Lucid semantics, and instead each variable takes on values in a series of states defined essentially by the lifetime, or *extent*, of that variable.

The Kripke Operator

All the examples we have seen so far of dynamic logic constraints fit a much narrower description than the above, namely Kripke's semantics for modal logic. The reason for our rather general characterization of a dynamic constraint is that later we will want to deal with certain adverbial constructs that transcend the Kripke formula. For the time being however, we will stick with Kripke semantics.

It is natural to introduce the names of binary relations into the language L. We reserve a,b,c,... as variables for this purpose. We call such expressions *actions*.

It is also natural to take $u \models a$ to be $\{v \mid uR_a v\}$, the set of states accessible from u via R_a. Then $u \models [a]p$ can be defined as $(u \models a) \models p$, if we adopt the usual convention in logic that for a set $U \subseteq W$, $U \models p$ means that $u \models p$ for every u in U.

The notation $[a]p$, though almost universal in the dynamic logic literature, starting with [39, 40], nevertheless obscures what we would like to call the *Kripke operator*. And later we will want to combine a's and p's in other ways, giving rise to other operators similar to the Kripke operator, each needing their own syntax. For these reasons we introduce the Kripke operator $\rfloor$, and replace the notation $[a]p$ with $a\rfloor p$. The semantics remains unchanged:

$$u \models (a{\lrcorner}p) \equiv (u \models a) \models p$$

For the syntax of the language $\{\neg \supset \lrcorner\}$ we adopt the convention that parentheses may be omitted from any of the following without changing the parsing. (We make use of this convention later, as new operators are introduced, to give a succinct account of the syntax of each new operator.)

$p{\supset}(q{\supset}r)$	(so $\supset$ is right associative)
$(\neg p){\supset}q$	(so $\neg$ binds tighter than $\supset$)
$a{\lrcorner}(b{\lrcorner}p)$	(so $\lrcorner$ is right associative)
$(a{\lrcorner}p){\supset}q$	(so $a{\lrcorner}$ behaves like $\neg$)
$(p{\supset}q)\ \ \supset\ r$	(so spaces count)

That part of dynamic logic confined to Kripke semantics can draw on a wealth of knowledge about modal logic. Of particular interest is the *theory* of this logic, the set of *valid* formulae, that is, formulae p such that $u \models p$ is *true* for all states u, where π satisfies

$$
\begin{array}{lcl}
u \models \neg p & \equiv & u \not\models p \\
u \models (p{\supset}q) & \equiv & u \models p \text{ implies } u \models q \\
u \models (a{\lrcorner}p) & \equiv & (u \models a) \models p
\end{array}
$$

Two questions we shall ask about the set of valid formulae are: how hard is it to decide validity, and what useful axiomatizations do they have?

In the case when L contains only one action, that action being in L_0, the theory coincides with the theory of system K of modal logic (that is, just the modal operator [], with no restrictions on the binary relation corresponding to []). The complexity of the theory of K (along with several related systems) was shown by R. LADNER [28] to be log-space complete in polynomial space. That is, first there is a computer program that will determine of a given formula of length n whether it is valid using $O(n^d)$ units of storage for some d. (This is to say that the theory of K is in polynomial space. Most reasonable notions of "units of storage" will suffice here.) This by itself is not very exciting, and so second (which is what makes the result much more interesting), one cannot do any better. That is, for *any* set in polynomial space there is a computer program which could determine whether an element of length n was in that set in only $O(\log n)$ units of space if only it had access to an "oracle" for the theory of K, a program that gives us at no charge answers to questions of membership in that theory. This is a very strong sense in which validity in K is as hard as any problem solvable in polynomial space.

Ladner's results extend without change to the case when L contains any number of actions, so long as they all belong to L_0. Shortly we shall see what happens when one introduces semantic constraints on actions.

The following will serve as a complete axiomatization of $\{\rceil \supset \rfloor\}$.

K	$p{\supset}q{\supset}p$
S	$p{\supset}q{\supset}r \supset p{\supset}q \supset p \supset r$
N	$\rceil p{\supset}\rceil q \supset \rceil p{\supset}q \supset p$
$\rfloor$	$a\rfloor(p{\supset}q) \supset a\rfloor p \supset a\rfloor q$
MP	From p, $p{\supset}q$ derive q (Modus Ponens)
$\rfloor$Nec	From p derive $a\rfloor p$ (Necessitation)

(The names K and S come from combinatory logic. N is for Negation. Note the similarity between S and $\rfloor$; when a is p? (a test, see below), S and $\rfloor$ coincide.)

Regular Dynamic Logic

It is not very interesting to consider just a set of unrelated actions with no visible internal structure. Hence we are inspired to introduce operations on actions. In so doing we shall be combining Tarski's calculus of binary relations [48] with Kripke's semantics for modal logic. This combination is without doubt the most interesting facet of that part of dynamic logic constructed around Kripke semantics. Even more interestingly, when we encounter analogues of the Kripke operator, the combination will become even more fruitful.

Propositional Dynamic Logic, PDL

We begin with four operators, ? ; $\cup$ *, that together with $\rceil \supset \rfloor$ give rise to the language Propositional Dynamic Logic (PDL).

Tests. Conditionals in a programming language are usually introduced with "if-then-else". However the rules of reasoning can be simplified by using a "smaller" notion of conditional, the test, which can be used in conjunction with the next two constructs to synthesize if-then-else. x>0? is an instance of a test, as is $j=0 \vee p(j)=t(k)?$.

A test p? is constructed from a formula p of the logical language. The idea of a test is that a computation may proceed past a test just when that test evaluates to *true* in the current environment, otherwise the computation must block (which for our purposes is equivalent to going into an infinite loop). Formally:

$$\begin{aligned} u \models p? \quad = \quad & \{u\} \text{ if } u \models p \\ & \emptyset \;\text{ otherwise.} \end{aligned}$$

Most of what we say holds even for tests containing $\rfloor$, permitting for example the side-effect-free programming construct "if p *would be* the result of running a then ..."

The axiom for tests is

T $\qquad p?\rfloor q \equiv p \supset q$

Composition. A familiar concept to programmers is that of executing one program after another; we may execute first a and then b. In terms of binary relations this means applying the first relation to a state to nondeterministically yield another state, and then applying the second relation to the resulting state. The *composition* of a and b, written a;b, is the relation describing the net effect of executing first a and then b. Formally:

$$u \models (a;b) \quad = \quad (u \models a) \models b$$

where $U \models b$, for $U \subseteq W$, is the union of the $u \models b$'s for each u in U.

The following axiom completely captures composition in dynamic logic.

C $\qquad a;b\rfloor p \equiv a\rfloor b\rfloor p. \qquad$ (Syntax: $(a;b)\rfloor p$)

Union. Another concept, slightly less familiar to programmers, is that of having a choice of which action to carry out. The action $a \cup b$ offers the choice of actions a or b. Formally:

$$u \models (a \cup b) \quad = \quad u \models a \cup u \models b.$$

Though $\cup$ may be less familiar than ; it has a static definition, unlike;. It is a nondeterministic concept; the closest deterministic programming concept is that of the conditional "if p then a else b" where a choice is given between a and b but in the same breath the criterion for making the choice, the formula p, is also given. In dynamic logic these two concepts of *choice* and *testing* are factored out, to simplify the domain of discourse and its attendant rules of reasoning. We can define "if p then a else b" in regular dynamic logic as $(p?;a) \cup (\neg p?;b)$.

The following axiom completely captures union in dynamic logic.

U $\quad a \cup b \lrcorner p \equiv a \lrcorner p \wedge b \lrcorner p.$ $\quad$ (Syntax: $(a \cup b) \lrcorner p$).

From these axioms we may infer that the validity problem for the language $\{\neg \supset \lrcorner \; ; \; \cup\}$ is decidable - in fact in exponential space - simply by using the axioms for ; and $\cup$ to eliminate all occurrences of ; and $\cup$ from the input to yield a formula at most exponentially larger.

Iteration. In order to get a program to run for a substantial time some way of executing programs repeatedly is called for. The most elementary form of repetition is *iteration,* which in dynamic logic means execution of an action an arbitrary number of times. We write a^* (a-star) for the iteration of a. Formally

I $\quad u \models I \;=\; \{u\}$ (I is the identity action, needed for the next line)

R $\quad u \models a^* \;=\; u \models (I \cup a \cup a;a \cup a;a;a \cup \ldots)$

The closest deterministic programming construct to this is "while p do a", which executes the program a a number of times determined by the test p. Again we have reduced things to more fundamental concepts just as we did with if-then-else, this time separating while-do into iteration and testing. We can define "while p do a" as $(p?;a)^*;\neg p?$.

Axioms for iteration are not as easy to come by as for union and composition. In fact when we introduce assignment later we will not be able to get a complete axiomatization of iteration. Without assignment however, we can achieve an axiomatization of iteration as follows.

Refl: $\quad [a^*]p \supset p$

Step: $\quad [a^*]p \supset [a][a^*]p$

Ind: $\quad p \wedge [a^*](p \supset [a]p) \supset [a^*]p.$

It is not at all apparent that these axioms generate all the valid formulae of PDL. The fact that they do was first announced (minus tests) in the Notices of the AMS by K. SEGERBERG [47]. Later (Jan. 1978) Segerberg found a lacuna in his proof, which he repaired some months after. Meanwhile R. PARIKH [36] and the present author [42], working semi-independently, found completeness proofs. Also D. GABBAY [11] gave a sketch of a completeness proof, though much detail would appear to be necessary to convert this sketch into a convincing proof.

It is also not at all apparent that the theory of PDL is decidable; this was shown by M. FISCHER and R. LADNER [8]. The proof uses a modal logic technique called filtration to show that a satisfiable formula of PDL has a finite model, whence satisfiability and validity can be determined by a finite search for a model. Normally filtration proofs are straightforward, but in the case of PDL a minor difficulty arises with ; . Fischer and Ladner were able to show that the theory of PDL is in NTIME(2^n) (nondeterministic Turing machine exponential time), but not in DTIME(c^n) for some $c > 1$. The upper bound has recently been improved to DTIME(8^n), as described in a revised version of [42].

First Order Regular Dynamic Logic.

The transition to any first order logic is made when terms are introduced into the language L. A term denotes an arbitrary domain element, not merely a truth value as in the case of a formula, or a set of states as in the case of an action.

Random Assignment. A random assignment is an action x:=? where x is any expression and ":=?" is the random assignment operator. It is defined by

$$u \models x{:=}? = \{v \mid uR_x v\} = \{u \models z = v \models z \text{ for all variables } z \text{ other than } x\}$$

Note that since random assignments involve the notion of variable, changing the semantic constraints may affect the meaning of x:=?.

The main role for x:=? is for defining quantifiers. ∀x is just x:=?⌋. The following two axioms for random assignment are, in the absence of other actions, just enough to completely axiomatize first order predicate calculus.

A1 $\quad p \supset \forall x p$ when x does not occur free in p

A2 $\quad \forall x p(x) \supset p(e)$ for any expression e

The definition of "occurs free in" is as follows. For e in L_0, x occurs free in e just when e is x. x occurs free in $a;b$ ($a \rfloor p$) just when x occurs free in a or <x is not bound in a and x occurs free in b (p)>. x occurs free in any other expression when x occurs free in one or more of its arguments. (Intuitively, x occurs free in $u \models e$ when there is a chance that the value of e might "depend on" the value of x in u.) x is *bound* in a when a is $x{:=}?$ or $x{:=}e;$, when a is $b;c$ and x is bound in b or c; or when a is $b \cup c$ and x is bound in b and c. (Intuitively, x is bound in a when it is guaranteed that a assigns some value to x.)

In this paper we shall forbid random assignment to action expressions.

Assignment. An assignment is a pair of expressions $x{:=}e$. The idea is that an assignment is the action of changing the state so as to make the value of x in the new state that of e in the old. Thus the corresponding binary relation consists of those pairs u,v such that $uR_x v$ and $v \models x \; = \; u \models e$. (Recall that R_x was the binary relation corresponding to $\forall x$ and consisting of all pairs u,v such that $u \models z \; = \; v \models z$ for all variables z other than x.) So we have

$$u \models x{:=}e = \{v \mid uR_x v \text{ and } v \models x = u \models e\}$$

There is no axiom for assignments as satisfactory as the axioms we have been encountering for other constructs. If $p(x)$ is a formula involving some "x-e-visible" occurrences of x (an x-e-visible occurrence has only operators "above" it in the expression that are referentially transparent to x and e, i.e. clearly don't change x or e), then the following axiom is adequate.

Ass: $\quad x{:=}e \rfloor p(x) \equiv p(e) \qquad$ (HOARE [20])

As we have thus far only constrained formulae and programs, the only assignments whose effects can be felt thus far are assignments to formulae and programs. We shall forbid the latter kind entirely. GRABOWSKI [12] has shown that algorithmic logic with this construct has a decidable validity problem. Extending this result to dynamic logic poses no insurmountable obstacles.

If we include = in L, with its standard interpretation on whatever domain takes our fancy, matters become more complex. It is now possible for information about the values of non-formulae to propagate up to the formula level; for example, we may now deduce the validity of $x{:=}y \rfloor x{=}y$ for variables x and y.

Problem: Determine whether validity is decidable for $\{\neg \supset \rfloor \cup ; {}^{*} =\}$; for $\{\neg \supset \rfloor \cup ; {}^{*} = {:=}\}$.

Including application, γ, (with the conditions that γ's first argument be a free variable, i.e. not one occurring inside an expression which contains assignments to that variable, and excluding variable actions) gives us first order predicate calculus with uninterpreted function symbols. We define γ_n (application for n-ary functions) thus.

$$u \models \gamma_n(f,x_1,\ldots,x_n) = (u \models f)(u \models x_1,\ldots,u \models x_n)$$

It was shown in [40] that the theory of what may take to be $\{\neg \supset \rfloor {}^{*} \forall := = \gamma\}$ was not recursively enumerable (r.e.), even if attention was restricted to formulae of the form $p \supset (x{:=}\gamma(f,x))^{*} \rfloor q$ where p and q were $\rfloor$-free. In [13] this result was strengthened to show that that fragment of the theory was Π_2^0-complete (cf. [45]). A. MEYER has shown that the whole theory is Π_1^1-complete (again cf. [45]). All these results indicate very definitely that a complete axiomatization of dynamic logic at this level of richness is out of the question.

Applications to Program Verification

Program verification is the art of showing that a program meets its specifications using formal logic. There is no doubt that HOARE's p{a}q construct [20] is of considerable interest to program verification. Since we can embed p{a}q in dynamic logic, as $p \supset a \rfloor q$, it follows that at least that fragment of dynamic logic is relevant to program verification. However, if *total* correctness is to be established, program verification also needs to deal with the problem of termination, which is not expressible using the p{a}q construct. Because of the ability to negate *all* formulae in dynamic logic, termination can be represented with no language extensions or informal arguments.

To test the extent to which dynamic logic could help in program verification, the author, with S. LITVINTCHOUK, implemented a proof checker for dynamic logic proofs [31]. Thus far the largest program we have demonstrated the total correctness of is the KNUTH-MORRIS-PRATT pattern-matching algorithm [22].

One interesting aspect of our perspective on DL is the decomposition of quantifiers such as ∀x into random assignment and the Kripke operator. A result of this is that less code is needed to cope explicitly with quantification, since half of what is known about quantification is actually general knowledge about arbitrary programs. This general knowledge is subsumed under axiom ⌟ and rule Nec. The axioms specific to quantification itself are then A1 and A2, which are so like the axioms for reasoning about assignment that only a small amount of additional code is needed to deal explicitly with quantifiers.

This situation should be contrasted with the usual approach to program verification, which is a two-stage affair in which verification conditions are generated and then sent to a theorem prover. Knowledge about programs in general and assignments in particular is kept in the verification condition generator, completely separate from knowledge about quantifiers, which is kept in the theorem prover.

Another point is that one does not always want to generate all verification conditions before starting to work on the sort of logical manipulation done by the theorem prover. Consider for example the two programs

```
a:          while p do b
aa:         while p do (b;b)
```

where p might be "x≤eps+y" and b might be (x:=y-z×x; y:=x+z×y). Now it turns out that regardless of what p and b are, the termination of aa implies the termination of a, a fact expressible in DL as "⌝aa⌟*false* ⊃ ⌝a⌟*false*". Yet if this statement, with p and b spelled out in full, is given to a two-stage system (even assuming it could handle things like having two ⌟'s in the problem), it will think hard about the assignments in b before getting to the logic. In a system that works top down (i.e. starting at the "top" of the formula to be proved, a characteristic of natural deduction systems for one), the validity of the above claim can become apparent even before any assignments are contemplated.

Applications to Natural Language

Consider the following sentences.

(1) Whether you strike a match or operate a cigarette lighter you get a flame.

This may be formalized as $M\cup C\rfloor F$, where M stands for the proposition that you have struck a match, C that you have operated a cigarette lighter, and F that you have a flame.

(2) If you strike a match you get a flame, and if you operate a cigarette lighter you get a flame.

Similarly this amounts to $M\rfloor F\wedge C\rfloor F$. The intuitive equivalence between (1) and (2) is formalized (and therefore subject to automatic verification) by the assertion $M\cup C\rfloor F \equiv M\rfloor F\wedge C\rfloor F$.

(3) When you open the door and walk through it you enter the room.

(4) When you open the door then when you walk through it you enter the room.

The equivalence of (3) and (4) is summarized in $O;W\rfloor E \equiv O\rfloor W\rfloor E$. Notice that we do not get as equivalent

(5) When you walk through the door and open it you enter the room.

or

(6) When you walk through the door, then when you open it you enter the room.

even though (5) and (6) are equivalent to each other, $W;O\rfloor E \equiv W\rfloor O\rfloor E$. If we were to try to capture the meaning of (3) or (4) using the propositional calculus alone, we might end up with $O\wedge W\supset E \equiv O\supset E\supset W$, which is certainly valid. Unfortunately $O\wedge W\supset E$ is equivalent to $W\wedge O\supset E$, which reveals the limitation of propositional calculus for reasoning about action in this way.

(7) If your TV won't work and you kick it it still won't work.

(8) If your TV won't work then no matter how many times you kick it it still won't work.

If (7) is true in all circumstances then (8) ought also to be true in all circumstances. This amounts to the soundness of the rule, from $W\supset K\rfloor W$ infer $W\supset K^{*}W$. This rule can be derived by starting with $W\supset K\rfloor W$, applying Necessitation to get $K^{*}\rfloor(W\supset K\rfloor W)$, then applying Modus Ponens to it and the induction axiom (reformulated slightly using propositional reasoning to read

$K^{*}\lrcorner(W\supset K\lrcorner W) \supset W\supset K^{*}\lrcorner W)$ to get $W\supset K^{*}\lrcorner W$ as desired.

Reasoning About Processes

So far the Kripke operator $\lrcorner$ has been our only operator relating actions to formulae. We now introduce some other operators that, like $\lrcorner$, find application to both algorithmic logic and to natural language reasoning. A price we must pay for these operators is the redefinition of the meaning of actions, which as defined so far do not contain enough information.

So far we have taken $u \models a$ to be the set of states that a might halt in when started in state u. We now take it to be the set of *sequences* of states that a goes through, starting from u. We let s,t,... range over sequences. Sequences can be viewed as functions from an initial segment of ordinals to states. In the event that a runs forever, the sequence will be infinite. If a is blocked by a test that evaluates to *false*, the limbo state $\Lambda\in W$ is entered. Sequences always have a final state, called s_f, whence infinite sequences need a limit element, indexed by the ordinal ω, which will always by Λ. Λ may not appear as a non-final state of a sequence.

The distinguished state Λ has a special behaviour as regards formulae; $\Lambda \models p$ is *true* for all formulae. For actions, $\Lambda \models a$ is $\{(\Lambda)\}$ for all actions. Semantic constraints of the form $u\models e = \ldots$ do not include the case $u = \Lambda$.

We also insist that $u \models a$ never be the empty set, for any action a, interpreted or not, even if this means taking $u \models a$ to be $\{(u,\Lambda)\}$.

With this notion of an action it becomes possible to define the new operators. But first we should adjust the definition of $\lrcorner$ so that it retains its meaning.

$$u \models a\lrcorner p = \bigwedge_{s\in u\models a} s_f \models p$$

The next operator is $\#$, as in $a\#p$, pronounced "a maintains p". The idea is that $u \models a\#p$ is *true* just when $v \models p$ is *true* in every state v of every sequence $u \models a$. Formally:

$$u \models a\#p = \bigwedge_{s\in u\models a} \; \bigwedge_{v\in s} v \models p$$

(We loosely write $v\in s$ to mean $v=s_i$ for some element s_i of s.)

The following completely axiomatizes $\{\neg \supset \#\}$, if we take S, K, N and MP.

$\#_1$ $a\#(p\supset q) \supset a\#p \supset a\#q$

$\#_2$ $a\#p \supset p$

$\#_3$ From p derive $a\#p$

The proof of completeness may be found in either of [42] or [43].

The third operator is $\perp$, as in $a\perp p$, pronounced "a promises p". Here $u \models a\perp p$ is *true* just when $v \models p$ is *true* in some state v of every sequence of $u \models a$. Formally:

$$u \models a\perp p = \bigwedge_{s\in u\models a} \; \bigvee_{v\in s} \; v\models p$$

Notice that in any sequence ending in Λ, everything is "promised". The idea here is that if a sequence ends in limbo you aren't supposed to care, just as for $\dashv$. This is important for programs where iteration is implemented using $*$ and tests. Careful inspection of the possible sequences reveals many ending in Λ that we would not want to compromise the intuitive notion of "promises" in conjunction with while loops.

The following completely axiomatizes $\{\neg \supset \perp\}$.

$\perp_1$ $p \supset a\perp p$

$\perp_2$ From $p\supset q$ infer $a\perp p \supset a\perp q$

The proof of completeness may be found in [43].

Finally we have $\int$, as in $a\int p$, pronounced "a preserves p". Here $u \models a\int p$ is *true* just when if $v \models p$ is *true* for any state v in $s\in u \models a$ then $w \models p$ is *true* for all states w in s after v. Formally:

$$u\models a|p = \bigwedge_{s\in u\models a} \; \bigwedge_{v\leq w\in s} \; v\models p\supset w\models p$$

where "$v\leq w\in s$" means that $v = s_i$, $w = s_j$, with $i \leq j$. The axiomatization is somewhat more complex; again see [43]:

$\int_1$ $p \supset a\int p \supset a\int(p\supset q) \supset a\int q$

$\int_2$ $p\equiv q \supset a\int(p\equiv q) \supset a\int p \supset a\int q$

$\int_3$ $p \supset a\int p \supset a\int\neg p$

$\int_4$ $a\int p \supset a\int q \supset a\int(p\wedge q)$

$\int_5$ $a\int p \supset a\int q \supset a\int(p\vee q)$

$\int_6$ From p infer $a\int p$

So far we have considered just the languages $\{\neg \supset x\}$ for various operators x. When these are combined to form $\{\neg \supset \rfloor \# \perp \int\}$ we need some additional axioms.

$a\#p \equiv p \wedge a\int p$ (suggests taking $\#$ as abbreviation only)
$a\int p \supset a\perp p \supset a\rfloor p$
$\neg(a\#p \wedge a\perp\neg p)$ (depends on fact that $u \models a$ is never empty)

None of this deals with operations on actions. The definitions of $\cup$ and $*$ need not change. We do however require definitions for ? ; and := . First let us define the operation ; on individual sequences, as

$(s;t)_i = s_i$ where s_i is defined
$(s;t)_{S+i} = t_i$ where S is the length of s and t_i is defined.

Then the definition of the action operators are:

$u \models I = \{(u)\}$
$u \models p? = \{(u)\}$ if $u \models p$
$\{(u,\Lambda)\}$ otherwise
$u \models a;b = \{s;t \mid s \in u \models a,\ t \in u \models b \text{ and } s;t \text{ is defined}\}$
$u \models x{:=}e = \{(u,v) \mid uR_x v \text{ and } v \models x = u \models e\}$

The following axiomatize $\int$ in conjunction with these. (With the axiom $a\#p \equiv p \wedge a\int p$ it becomes unnecessary to give further axioms for $\#$).

$p?\int q$
$a \cup b\int p \equiv a\int p \wedge b\int p$
$a;b\int p \equiv a\int p \wedge a\rfloor b\int p$
$a^*\int p \equiv a^*\rfloor a\int p$

This leaves open the problem of axiomatizing $\perp$ with the action operators. A little reflection shows that while $\cup$ and * can be axiomatized ($a^*\perp p \equiv p$), ; cannot be axiomatized with a single equivalence in any obvious way. To get around this we introduce a new operator, $\rfloor\!\|$, as in $a\rfloor\!\|p,q$, which takes two arguments p and q. It is defined thus.

$$u \models a\rfloor\!\|p,q = \forall s \in u \models a(\exists i(s_i \models p) \vee s_f \models q).$$

This says that for every sequence s in $u\models a$, either p holds in some state of s or q holds in the final state of s (including the case when $s_f = \Lambda$, which satisfies both p and q). This rather odd construct has the properties that ; can be axiomatized with it, and both $\lrcorner$ and $\perp$ can be treated as abbreviations, thus:

$$a\lrcorner p \equiv a\lrcorner\!\!| \mathit{false},p$$
$$a\perp p \equiv a\lrcorner\!\!| p,\mathit{false}.$$

The following axiomatize $\lrcorner\!\!|$ in conjunction with $\int$, treating $\lrcorner$, $\#$ as mere abbreviations. This axiomatization is probably not complete.

$\lrcorner\!\!|$ 1 $\quad a\lrcorner\!\!| p,(q\supset r) \supset (a\lrcorner\!\!| p,q \supset a\lrcorner\!\!| p,r)$

$\lrcorner\!\!|$ 2 $\quad a\lrcorner\!\!| p,\neg p$

$\lrcorner\!\!|$ 3 $\quad p \supset a\lrcorner\!\!| p,q$

$\lrcorner\!\!|$ 4 $\quad$ from $p\supset q$ derive $a\lrcorner\!\!| p,r \supset a\lrcorner\!\!| q,r$

$\lrcorner\!\!|$ 5 $\quad a\int p \supset (a\perp p \supset a\lrcorner p)$

$\lrcorner\!\!|$ 6 $\quad \neg(a\#p \wedge a\perp\neg p)$

$\lrcorner\!\!|$ 7 $\quad (a\cup b)\lrcorner\!\!| p,q \equiv a\lrcorner\!\!| p,q \wedge b\lrcorner\!\!| p,q$

$\lrcorner\!\!|$ 8 $\quad (a;b)\lrcorner\!\!| p,q \equiv a\lrcorner\!\!| p,(b\lrcorner\!\!| p,q)$

$\lrcorner\!\!|$ 9 $\quad a^*\lrcorner\!\!| p,q \supset p\vee q$

$\lrcorner\!\!|$ 10 $\quad a^*\lrcorner\!\!| p,q \supset a\lrcorner\!\!| p,(a^*\lrcorner\!\!| p,q)$

$\lrcorner\!\!|$ 11 $\quad a^*\lrcorner\!\!| p,(q \supset a\lrcorner\!\!| p,q) \supset (q \supset a^*\lrcorner\!\!| p,q)$ (Harel induction)

$\lrcorner\!\!|$ 12 $\quad p?\lrcorner\!\!| q,r \equiv q\vee(p\supset r)$

Applications of process logic to algorithmic logics

The $\#$ operator is perhaps the simplest operator one might wish to apply to a program that was designed to run forever (e.g. an operating system, or an interactive editor), for which the $\lrcorner$ operator is worthless.

The $\perp$ operator is relevant to the issues of fairness and starvation, concepts that arise occasionally in the literature on verification of operating systems. If we view the scheduler as a nondeterministic program (and even if we assume that the operating system is a deterministic mechanism we cannot really work that determinacy into our proofs in practice), then we would like to be able to say of the system as a whole, nondeterminism and all, that there will come a time when a certain state of affairs (e.g. such-and-such a process getting service) will hold.

The $\int$ operator arises naturally in talking about a system that only manages to keep p true throughout its execution by assuming it is true to begin with and depending throughout on its staying true. This idea is embodied in the axiom $p \wedge a\int p \supset a^{\#}p$. The $\int$ operator is used implicitly by OWICKI in her thesis [34].

Applications of process logic to natural language

We give a further series of examples of natural language formulae embodying arguments formalizable within dynamic logic and using other operators besides the Kripke operator.

(1) While stacking up blocks, if the box becomes empty it will remain empty for the duration of the stacking process.

(2) Sometime during the stacking of blocks the box is guaranteed to be empty.

(3) When you stack up blocks you end up with the box being empty.

It is apparent that if both (1) and (2) are the case then so is (3), as can be seen from the valid formula $S\int E \wedge S\perp E \supset S\rfloor E$.

(4) If a defect appears in the wall while laying bricks the defect will stay there for the rest of the bricklaying.

(5) After laying any number of bricks, if a defect is found in the wall while laying the next brick the defect will stay there for the rest of the laying of that brick.

With a little thought it can be seen that (4) and (5) are equivalent, as formalized by $L^{*}\int D \equiv L^{*}\rfloor L\int P$.

Pointers to the dynamic logic literature

The earliest formal dynamic logic was FREGE's quantificational calculus [10]. The idea of viewing quantifiers in terms of a relation R_x, thereby making the connection between modal logic and the quantificational calculus and so permitting the latter to viewed as a dynamic logic, is relatively recent. Modal logic as discussed here is due to LEWIS [29]. The semantics we are using is due principally to KRIPKE [25], who also contributed to issues of decidability in [26]. An excellent reference work on modal logic is [21].

Following ENGELER [7] and the Polish school of algorithmic logic [4, 46], we shall call a dynamic logic whose actions are deterministic programs, described either by flowcharts or if's and while's, an *algorithmic logic*. The earliest work on proving programs correct [49, 50] amounted to informal algorithmic logics for flowchart programs. In the early sixties J. McCARTHY [32] proposed the use of a static approach to program correctness by programming with recursively defined functions, thereby avoiding the problem of reasoning about states.

In 1967 FLOYD [9] described in detail for the first time an algorithmic logic, built around flowcharts as with [49, 50]. In 1969 HOARE [20] described a more conventional algorithmic logic oriented towards textual programs using if's and while's. Hoare's logic introduced the notion of a partial correctness assertion $p\{a\}q$ as an expression having a status different from that of an ordinary formula, in particular not being subject to Boolean operations. Though Hoare gave only an informal semantics for $p\{a\}q$, it seems beyond debate that he meant it to have the semantics of $\models p\supset a\lrcorner q$. In 1970 Salwicki developed a similar algorithmic logic (and applied ENGELER's term algorithmic logic [7] to it). The most striking difference from Hoare's logic was that all of Salwicki's formulae were subject to Boolean operations; as such, Salwicki's logic is the first true algorithmic logic. It may be characterized as dynamic logic using $\lrcorner$, if, while, and having functional rather than relational actions, as behoves a deterministic programming language. (Engeler's algorithmic logic [7] is rather weaker, permitting in effect only *false* as the second argument to $\lrcorner$.) Salwicki's work prompted a veritable flood of papers from Warsaw on algorithmic logic, mostly by members of H. Rasiowa's group; a comprehensive survey of work up to 1974, including a bibliography of some 40 papers on algorithmic logic, may be found in [4].

The idea of modelling programs with binary relations, taking advantage of TARSKI's calculus of $\cup,;,^*$ [48], goes at least as far back as EILENBERG and ELGOT [6]. DE BAKKER [2], and with DE ROEVER [3], developed the idea considerably further, adding a fixed point operator to Tarski's calculus to model recursion. Independently of de Bakker, but motivated by EILENBERG and ELGOT, D. PARK [19], with P. Hitchcock, also used the fixed point operator in a relational treatment of flowchart programs.

The combination of modal logic and Tarski's operators was first developed by the author [39] in response to a suggestion of R. Moore, a student in the author's program semantics course. It was brought to the attention of a wider audience in [40] some two and a half years later, in a paper that prompted several people, including M. Fischer, D. Harel, R. Ladner and A. Meyer, to work on dynamic logic. This gave rise to a paper by HAREL, MEYER and the author [13] on the complexity of the theory of first-order dynamic logic, along with a relative-completeness proof of the axiomatization given in [40], and another paper by FISCHER and LADNER [8] on the validity problem for PDL, including not only the result that it was decidable but giving good bounds on the complexity of the problem. A little later, Harel and the author reported on work on Dijkstra's notion of total correctness *("weakest precondition")*, proposing definitions for the concepts Dijkstra was attempting to define via axioms, and giving a relatively complete axiom system for DIJKSTRA's language [14].

At about the same time several people began asking questions about definability in dynamic logic. A. Meyer addressed the question of whether DL^+, the language defined in [14] in part to formalize Dijkstra's language, was expressible in regular first-order dynamic logic. This problem turned out to have a very elusive answer. Meyer was able to show that DL^+ was no more expressive than DL provided the programming language permitted array assignments [33]. Later WINKLMANN [51] was able to obtain the same result without requiring array assignments, but using ┘ within tests. Eventually he was able to eliminate ┘ from tests [52]. Meanwhile the author showed that PDL^+ is strictly stronger than PDL, complementing [52].

F. BERMAN and M. PATERSON, in a remarkably delicate argument, showed that PDL was strictly strengthened by the inclusion of tests [5]. MEYER and PARIKH showed that regular first-order DL with ┘-free tests was strictly weaker than constructive $L\omega_1\omega$ [35].

D. HAREL developed the relative completeness ideas of [13] further, drawing a distinction between relative completeness and arithmetic completeness. Using a result of LIPTON [30], Harel showed in essence that arithmetic completeness is all that one wants.

The question of finding a complete axiomatization for PDL was raised in [8]. There is an account earlier in this paper of the origins of [47, 36, 42, 11] as answers to this question.

A very thorough and detailed treatment of Harel's many contributions to DL may be found in his thesis [18]. In addition Harel has authored a close-to-exhaustive survey of logics of deterministic programs, using the [a]p/<a>p notation as a lingua franca in order to make it easier to see the similarities and differences between the various logics.

Motivated by the concept of a Boolean algebra underlying propositional calculus, Harel asks the question what the appropriate algebra for PDL is. A partial answer to this may be found in [17].

The author, with S. Litvintchouk, explored the question of how to implement a proof checker for DL. A proposal for such a system is described in [31]. Some of the techniques used in the first implementation of the system are alluded to briefly in [41]. The system has been operational since August 1977, when it was able to check a 20-theorem proof of the total correctness of the Knuth-Morris-Pratt pattern-matching algorithm, taking 45 seconds to do so. In the process of making the system more automatic, some theoretical questions about deciding validity in PDL arose, giving rise to an algorithm described in [42] that is more "practical" than the algorithm of [8]. A revision of [42] improved the upper bound on PDL validity from two exponentials to one.

In [42] the issue of discussing programs intended not to halt is raised. Several constructs are proposed, namely those discussed above in the section on logics of processes. The author has recently been able to show completeness of various axiom systems for some of those constructs [43]. The semantics for processes is intimately related to PNUELI's semantics for temporal logic [38]. Parikh has shown, using Rabin's remarkable decidability result for the weak second order theory of n successors, that a very much stronger language has a decidable theory, although unlike the theory of [42], it is not elementary recursive.

The Interest in Nondeterministic Programs

The following is taken from [14], and may be of interest to those wondering why someone interested in deterministic algorithmic logic would want to get involved in the greater generality of dynamic logic and Tarski's relational calculus.

First, nondeterministic programs have been proposed as a model of parallel processes. Such parallelism arises in timeshared computers, where nondeterminism expresses the apparent capriciousness of the scheduler. It also arises in the management of external physical devices, where the nondeterminism captures the unpredictable behaviour of physical devices.

Second, nondeterminism is gaining credence as a component of a programming style that imposes the fewest constraints on the processor executing the program. For example a certain program may run correctly provided that initially x is even. If the programmer requires the processor to set x to an even number of the programmer's choosing, the processor may be unduly constrained. On a byte-oriented machine where integers are represented as four-byte quantities, setting x to a particular number requires four operations, but if the programmer has merely requested setting it to an arbitrary even number the processor can satisfy the request with one operation, by setting the low-order byte to, say, zero.

Third, nondeterminism supplies one methodology for interfacing two procedures that, though written independently, are intended to cooperate on solving a single problem. The approach is to make one procedure an "intelligent" interpreter for the other. Wood's Augmented Transition Networks supply an instance of the style. The user of this system writes a grammar for a specific natural language which amounts to a nondeterministic program to be run on Wood's interpreter, which though ignorant of the details of specific languages nevertheless contributes much domain-independent parsing knowledge to the problem of making choices left unspecified by the user's program. This technique is in wide use in other areas of Artificial Intelligence, and supplies a way of viewing such AI programming languages as QA-3, PLANNER, and a number of more recent languages.

Fourth, from a strictly mathematical viewpoint, there is something dissatisfying about taking such constructs as if-then-else and while-do as primitive constructs. If-then-else involves the two concepts of *testing* and *choosing*, and while-do involves the two concepts *testing* and *iterating*. A more basic approach is to develop these concepts separately. However, in isolating the concept of testing from the concepts of choosing and iterating, we have removed the parts of the if-then-else and while-do constructs responsible for their determinism.

Fifth, from a practical point of view, when reasoning about deterministic programs it can sometimes be convenient to make what amounts to claims about nondeterministic programs. When we argue that "if $x > 0$ then $x := x-1$ else $x := x+1$" cannot affect y, a part of our argument might be that, whether we execute $x := x-1$ or $x := x+1$, y will not change. The fact that the whole program is deterministic played no role in this argument, which amounts to the observation that the nondeterministic program $x := x-1 \cup x := x+1$ cannot change y. By the same token the observation that "while $x < 0$ do $x := x+2$" leaves the parity of x unchanged depends principally on the fact that executing $x := x+2$ arbitrarily often, i.e. executing $(x:=x+2)^*$, leaves the parity of x unchanged. This illustrates the appropriateness of applying nondeterministic reasoning to deterministic programs.

REFERENCES

[1] ASHCROFT, E.A. & W.W. WADGE, *Lucid, a Nonprocedural Language with Iteration,* Comm. ACM, 20, 7, 519-526, July 1977.

[2] DE BAKKER, J.W. & D. SCOTT, *An outline of a theory of programs,* Unpublished manuscript, 1969.

[3] DE BAKKER, J.W. & W.P. DE ROEVER, *A calculus for recursive program schemes,* In M. Nivat (ed.), Automata, Languages and Programming, North Holland, pp. 167-196, 1972.

[4] BANACHOWSKI, L., A. KRECZMAR, G. MIRKOWSKA, H. RASIOWA, A. SALWICKI, *An Introduction to Algorithmic Logic; Metamathematical Investigations in the Theory of Programs,* In A. Mazurkiewicz & Z. Pawlak (eds.), Math. Found. of Computer Science, Banach Center Publications, Warsaw, 1977.

[5] BERMAN, F. & M. PATERSON, *Test-Free Propositional Dynamic Logic is Strictly Weaker than PDL,* T.R. 77-10-02, Dept. of Computer Science, Univ. of Washington, Seattle, November 1977.

[6] EILENBERG, S. & C.C. ELGOT, *Recursiveness,* Academic Press, N.Y. 1970.

[7] ENGELER, E., *Algorithmic properties of structures,* Math. Syst. Th. 1, 183-195, 1967.

[8] FISCHER, M.J. & R.E. LADNER, *Propositional Modal Logic of Programs,* Proc. 9th Ann. ACM Symp. on Theory of Computing, 286-294, Boulder, Col., May 1977.

[9] FLOYD, R.W., *Assigning Meanings to Programs,* In J.T. Schwartz (ed.), Mathematical Aspects of Computer Science, AMS Proc. Symp. Applied Math. 19, pp. 19-32, 1967.

[10] FREGE, G., *Begriffsschrift,* Halle, 1879.

[11] GABBAY, D., *Axiomatizations of Logics of Programs,* Manuscript, under cover dated November 1977.

[12] GRABOWSKI, M., *The Set of Tautologies of Zero-order Algorithmic Logic is Decidable,* Bull. Acad. Pol. Sci., Ser. Math. Astr. Phys., 20, 575-582, 1972.

[13] HAREL, D., A.R. MEYER & V.R. PRATT, *Computability and Completeness in Logics of Programs,* Proc. 9th Ann. ACM Symp. on Theory of Computing, 261-268, Boulder, Col., May 1977.

[14] HAREL, D. & V.R. PRATT, *Nondeterminism in Logics of Programs,* Proc. 5th Ann. ACM Symp. on Principles of Programming Languages, 203-213, Tucson, Arizona, January 1978.

[15] HAREL, D., *Arithmetical Completeness in Logics of Programs,* In G. Ausiello & C. Böhm (eds.), Proc. 5th Int. Colloq. on Automata, Languages and Programming (Udine), Springer Lecture Notes in Comp. Sci 62, pp. 268-288, 1978.

[16] HAREL, D., *On the Correctness of Regular Deterministic Programs; A Unified Survey,* Submitted for publication.

[17] HAREL, D., *Relational Logic,* Internal report, MIT, 1977.

[18] HAREL, D., *Logics of Programs: Axiomatics and Descriptive Power,* Ph.D. thesis, Dept. of EECS, MIT, June 1978.

[19] HITCHCOCK, P. & D. PARK, *Induction Rules and Termination Proofs,* In M. Nivat (ed.), Automata, Languages and Programming, North-Holland, 1972.

[20] HOARE, C.A.R., *An Axiomatic Basis for Computer Programming,* CACM 12, 576-580, 1969.

[21] HUGHES, G.E. & M.J. CRESSWELL, *An Introduction to Modal Logic,* London: Methuen and Co. Ltd., 1972.

[22] KNUTH, D.E., J.H. MORRIS & V.R. PRATT, *Fast Pattern Matching in Strings,* SIAM J. on Computing, 6, 2, 323-250, June 1977.

[23] KRECZMAR, A., *The set of all tautologies of algorithmic logic in hyperarithmetical,* Bull. Acad. Pol. Sci., Ser. Sci. Math. Astr. Phys., Vol. 19, 781-783, 1971.

[24] KRECZMAR, A., *Degree of recursive unsolvability of algorithmic logic,* Bull. Acad. Pol. Sci., Ser. Sci. Math. Astr. Phys., Vol. 20, 615-617, 1972.

[25] KRIPKE, S.A., *Semantical considerations on Modal Logic,* Acta Philosophica Fennica, 83-94, 1963.

[26] KRIPKE, S.A., *Semantical analysis of modal logic I: normal modal propositional calculi,* Zeitschr. f. Math. Logik und Grundlagen d. Math., Math., 9, 67-96, 1963.

[27] KROEGER, F., *Logical Rules of Natural Reasoning about Programs,* In Automata, Languages and Programming, S. Michaelson & R. Milner (eds.), Edinburgh University Press, pp. 87-98, 1976.

[28] LADNER, R.E., *The Computational Complexity of Provability in Systems of Modal Propositional Logic,* SIAM J. on Computing, 6, 3, 467-480, 1977.

[29] LEWIS, C.I., *A Survey of Symbolic Logic,* Berkeley, 1918.

[30] LIPTON, R.J., *A Necessary and Sufficient Condition for the Existence of Hoare Logics,* Conf. Rec. 18th Ann. IEEE Symp. on Found. of Computer Science, Providence, R.I., pp. 1-6, October 1977.

[31] LITVINTCHOUK, S.D. & V.R. PRATT, *A Proof-checker for Dynamic Logic,* Proc. 5th Int. Joint Conf. on AI, Boston, 552-558, August 1977.

[32] McCARTHY, J., *A Basis for a Mathematical Theory of Computation,* In P. Braffort & D. Hirschberg (eds.), Computer Programming and Formal Systems, North-Holland Publ. Comp., pp. 33-70, 1963.

[33] MEYER, A.R., *Equivalence of DL, DL+ and ADL for Regular Programs with Array assignments,* Unpublished report, MIT, August 1977.

[34] MEYER, A.R. & R. PARIKH, *Definability in Dynamic Logic,* Talk presented at NSF-CBMS Research Conference on the Logic of Computer Programming, Troy, N.Y., May 1978.

[35] OWICKI, S., *A consistent and complete deductive system for the verification of parallel programs,* Proc. 8th Ann. ACM Symp. on Theory of Computing, Hershey, pp. 73-86, May 1976.

[36] PARIKH, R., *A Completeness Result for PDL,* In M. Karpinski (ed.), Mathematical Foundations of Computer Science, Zakopane, pp. September 1978.

[37] PARIKH, R., *Second Order Process Logic,* Conf. Rec. 19th Ann. IEEE Symposium on Foundations of Computer Science, Ann Arbor, pp. 177-183, October 1978.

[38] PNUELI, A., *The Temporal Logic of Programs,* Conf. Rec. 18th Ann. IEEE Symposium on Foundations of Computer Science, Providence, 46-57, October 1977.

[39] PRATT, V.R., *Semantics of Programming Languages,* Lecture notes for CS 6.892, Fall 1974, M.I.T.

[40] PRATT, V.R., *Semantical Considerations on Floyd-Hoare Logic,* Conf. Rec. 17th Ann. IEEE Symp. on Found. of Computer Science, Houston, 109-121, 1976.

[41] PRATT, V.R., *Two Easy Theories Whose Combination is Hard,* Internal Report, Lab. for Computer Science, MIT, September 1977.

[42] PRATT, V.R., *A Practical Decision Method for Propositional Dynamic Logic,* Proc. 10th Ann. ACM Symp. on Theory of Computing, San Diego, pp. 326-337, May 1978. (Revised as *A Near Optimal Method for Reasoning About Action,* LCS TM-113, MIT, Sept. 1978.)

[43] PRATT, V.R., *Process Logic,* Proc. 6th Ann. ACM Symp. on Principles of Programming Languages, San Antonio, Texas, pp. , Jan. 1979.

[44] QUINE, W.V.O., *Word and Object,* MIT Press, MA., 1960.

[45] ROGERS, H., *Theory of Recursive Functions and Effective Computability,* McGraw-Hill, 1967.

[46] SALWICKI, A., *Formalized Algorithmic Languages,* Bull. Acad. Pol. Sci., Ser. Sci. Math. Astr. Phys., Vol. 18, No. 5, pp. , 1970.

[47] SEGERBERG, K., *A completeness Theorem in the Modal Logic of Programs,* Preliminary report, Notices of the AMS, 24, 6, A-552, October 1977.

[48] TARSKI, A., *On the Calculus of Relations,* J. Symbolic Logic, 6, 73-89, 1941.

[49] TURING, A., *Checking a Large Routine,* In Rep. Conf. High Speed Automatic Calculating Machines, Inst. of Comp. Sci. Univ. of Toronto, Ontario, Can., January 1950.

[50] VON NEUMANN, J., *Collected Works.,* Vol. 5, pp. 91-99, MacMillan, New York, 1963.

[51] WINKLMANN, K., *Equivalence of DL and DL+ for regular programs without array assignments but with DL-formulas in tests,* Manuscript, Lab. for Comp. Sci., MIT, 1978.

[52] WINKLMANN, K., *Equivalence of DL and DL+ for regular programs*, Manuscript, Lab. for Comp. Sci., MIT, March 1978.

NOTES ON ALGEBRAIC FUNDAMENTALS
FOR
THEORETICAL COMPUTER SCIENCE

by

J.W. Thatcher, E.G. Wagner, J.B. Wright

MATHEMATICAL CENTRE TRACTS 109(1979), 83-163

NOTES ON ALGEBRAIC FUNDAMENTALS
FOR
THEORETICAL COMPUTER SCIENCE

James W. Thatcher, Eric G. Wagner and Jesse B. Wright
IBM Thomas J. Watson Research Center
Yorktown Heights, New York 10598
USA

1. INTRODUCTION

From categories to many-sorted algebras to rational and iterative algebraic theories, these notes cover basic definitions and results that we feel provide the groundwork for comprehensive and mathematically sound developments in theoretical computer science. The belief that the ideas presented here are key, comes from our experience over the last eight years in developing and applying these concepts. That experience, in which Joe Goguen's participation and insights were essential, has shown us that basic (or "first order") ideas from category theory and universal algebra (generalized to the many-sorted case following Birkhoff and Lipson (1970)) have wide applicability as indicated by many entries in the Bibliography.

These Notes were originally prepared for the Summer School on Foundations of Artificial Intelligence and Computer Science, Pisa, Italy, 19-30 June 1978. (Sections 1, 2, 3, 6, 8, 10, 11, 12, and 13 covering the bare algebraic fundamentals.) For The 3rd Advanced Course on Foundations of Computer Science, Amsterdam, The Netherlands, 21 August - 1 September, 1978, we still left that coverage quite sparse while beginning to add material on the way the algebra may be applied. Section 4 begins an algebraic treatment of flowcharts (continued briefly in Section 13); Section 7 describes the basic ideas of algebraic semantics; and, Section 9 presents a few of the fundamentals of the ADJ approach to abstract data types. There are many ways in which we plan on expanding the coverage of these Notes. The algebraic treatment of flowchart computation (as presented here) can be extended to flowcharts with recursive procedures. The semantics of flowchart computation through rational and continuous algebraic theories needs to be made clear and complete. We would like to extend the methods of data type specification to ordered and continuous algebras, as well as to algebraic theories of many flavors (ala Burstall and Goguen (1977)). We want to combine the methods of algebraic semantics (Section 7) with those of flowchart computation (Sections 4 and 13) to provide an algebraic methodology along the lines suggested by Morris (1973,1973a) for proving correctness of compilers. (Our paper, ADJ (1979), makes a significant step in this direction and is a direct outgrowth of the preparation of these Notes and the lectures for the Amsterdam Summer School.)

Before moving to technical details, we want to discuss some general considerations about applications of algebra and category theory in particular.

As Graetzer (1968) points out in his introduction, A.N. Whitehead (1898) foresaw the future of and the value of universal algebra:

> "Such algebras are worthy of comparative study, for the sake of the light thereby thrown on the general theory of symbolic reasoning and algebraic symbolism in particular. The comparative study necessarily presupposes some previous special study, comparison being impossible without knowledge."

Such prophesy is quite remarkable, especially considering the fact that it was over a third of a century later when the first results on universal algebra (Birkhoff (1933, 1935)) were published.

Much past work on semantics of computation has the flavor of the specialized study for algebras that Whitehead referred to. Universal algebraic methods, including category theory, provide a framework for comparative study, thereby, we hope, throwing light on the general theory of computation.

Although certainly not always, it is sometimes the case that, as algebraicists, we find ourselves involved in reworking, in algebraic language, what others have studied or talked about with less discipline and less structured tools. An example of this is the study of "monadic computation" or "flowchart computation" by Elgot (1973) using iterative algebraic theories, or by ADJ (1976d) using rational algebraic theories. You might ask, what is the point in restating the familiar in algebraic terms? Our answer is that the "restatement" is more than a matter of changing words; it involves a very significant reexamination of the underlying concepts. The result, when successful, is a formulation that is both broader and deeper than the original. Thus, the study of "flowchart computation" in terms of algebraic theories provides a formulation which, as shown in ADJ (1978a), also applies to other forms of programming languages. This case illustrates the rather amazing phenomenon that, repeatedly, algebraic concepts developed for special purposes turn out to have broad (and deep) applications elsewhere. It is difficult to imagine that the discoverer of algebraic theories (Lawvere (1963)) could have foreseen the applications to the theory of computation; but even more, it is a pleasant surprise indeed, that the applicability of algebraic theories in computer science also reaches far beyond "flowchart computation."

There seems to be a tendency for some people to talk about "categorical computer science" in much the same way that one talks of "automata theory" or "complexity theory," as subfields of computer science. This is a mistake; "categorical computer science" is no more a subfield of computer science than is "set theoretic computer science" or "graph theoretic computer science." The practice probably comes from the fact that theoretical computer scientists using categorical algebra form a small subgroup, and that their work seems relatively inaccessible to someone with a typical computer science background in mathematics. We hope that this will change through the improvement of mathematical education in computer science curricula and through continued efforts to make the algebraic work more accessible to the computer scientist.

We should confront another aspect of the isolation of the subgroup using algebraic (categorical) methods; it is a kind of arrogance displayed by some, first, in not acknowledging the fact when they are involved in redoing what others have done in one form or another, and second, in giving the impression that category theory solves all the problems and that they have solutions at hand. We have to dismiss this by saying that it too is an unfortunate mistake.

Even though there seems to be some argument on the matter (c.f. Wegner (1972) and Goguen's answer, (1972)), we take as the starting point that mathematics underlies theoretical computer science. Unfortunately much modern mathematics does not appear to be directly relevant to theoretical computer science. (Exceptions certainly exist: topology in Scott (1972); matrix theories in Elgot (1974); extension bases in Markowski and Rosen (1976), etc.) What one consistently encounters is a certain well understood collection of concepts and methods from elementary set theory (217-tuples and the like), combinatorics and graph theory. These tools do quite well in certain problem areas but in many cases they serve only to build models which, while precise, are not manipulatable; they are not internally "coherent;" they are heavily notationally dependent; and, they fail to reflect the essential structure of the problem under consideration. The question is not asked, "what's wrong with this formulation?" It is not asked because these disciplines offer practically no guides to the answer. We feel that the algebraic (or categorical) approach meets these objections in many cases, by providing guidance as well an increased coherence and ease of manipulation. Some of the reasons why this is so for the categorical approach are spelled out in the "doctrines" given in the introduction to ADJ (1973) which express the naturalness and ubiquity of the categorical concepts. (E.g., "every construction is representable (extendible to) a functor"; "every 'canonical' construction is (part of) an adjunction.") The same phenomenon is true of the universal algebra approach although it may not lend itself as readily to such "doctrines". Those familiar with our applications of universal algebra to computer science have probably noticed the recurrent emphasis on free or initial algebras. These basic algebraic concepts can be used to define, explain, and unify a great number of computer science concepts. When one becomes familiar with such concepts (and the results concerning them) they provide a guide as to what one should look for, and as to how to formulate one's definitions and results. The reason why the approach is so successful is that, in truth, these algebraic and categorical concepts are the precise formulations of the "recurrent phenomena" of mathematics (and thus of theoretical computer science). Indeed, if, in your research, you find yourself repeatedly having to work through cases of "almost the same thing" then you may be practically assured that by going to an algebraic (and/or categorical) approach you can capture the "thing" and replace the many cases by one.

Finally, do not succumb to a feeling that you must understand *all* of universal algebra and category theory before you can put it to use. When one talks of a "set theoretic" model for some computing phenomenon, s/he is not thinking of a formulation in terms of measurable cardinals! Similarly, a category theoretic model does not *necessarily* involve the Kan extension theorem or double categories. It is our hope that the material in these notes will provide sufficient background to enable you to enjoy the benefits of the algebraic approach.

2. SETS, RELATIONS, AND FUNCTIONS.

We shall use standard notation for sets.

(2.1.1)	$\emptyset$	The empty set.
(2.1.2)	$a \in A$	Membership.
(2.1.3)	$A \subseteq B$	Set inclusion.
(2.1.4)	$\wp A$	Power set (set of subsets) of A.
(2.1.5)	$\wp_\omega A$	Finite subsets of A.
(2.1.6)	#A	Cardinality of A.
(2.1.7)	$A \cup B$	Set union.
(2.1.8)	$A \cap B$	Set intersection.
(2.1.9)	$\bigcup S$	Set intersection for $S \subseteq \wp A$.
(2.1.10)	$\bigcap S$	Set intersection for $S \subseteq \wp A$.

The set of nonnegative integers (natural numbers) is denoted ω and it is sometimes convenient to use the ordinal notation where $n = \{0,1,...,n\text{-}1\}$ so that $n \leq m$ iff $n \in m$ as sets. For "one-origin" indexing, the set $[n] = \{1,...,n\}$ is very handy. Note that $[0] = \emptyset$ and also in ordinal notation, $0 = \emptyset$. We will use $[\omega]$, consistent with the other square bracket notation, to denote the natural numbers starting with 1, $[\omega] = \{1,2,...\}$.

For ordered pairs we write $<a,b>$ with angle brackets; we could define them using sets, $<a,b> = \{\{a\},\{a,b\}\}$, or take the notion as primitive, by saying that the pair $<a,b>$ is uniquely determined by its first and second components. $A \times B$ is the (Cartesian) product of A and B; $A \times B = \{<a,b> \mid a \in A \text{ and } b \in B\}$. Similarly, $<a_1,\ldots,a_n>$ is an n-tuple uniquely determined by its components.

A *relation* R from A to B is usually viewed as a subset of $A \times B$. However, this is really inadequate because there is no way to retrieve A and B from R. So we define a *relation* to be an ordered triple $<A,R,B>$ where A and B are sets and $R \subseteq A \times B$ is called the *graph* of the relation, A is its *source* and B is its *target*. We write $R{:}A \rightarrow B$ to mean R is the graph a relation from A to B and often speak of just R when A and B are understood from context. As is the standard convention, we sometimes write aRb to mean $<a,b> \in R$.

Given $R{:}A \rightarrow B$ and $S{:}B \rightarrow C$, the graph of the *composite* relation, $R \circ S{:}A \rightarrow C$ is defined by:

(2.2) $\quad R \circ S = \{<a,c> \mid \text{there exists } b \text{ with } <a,b> \in R \text{ and } <b,c> \in S\}$.

This is the Pierce product of (the graphs of) the relations. The operation of forming the composite of relations is called *composition*. Note that we are writing composition of relations in the *natural* diagrammatic order as indicated in the following diagram.

$$\begin{tikzcd} & B \arrow[dr, "S"] & \\ A \arrow[ur, "R"] \arrow[rr, "R \circ S"'] & & C \end{tikzcd}$$

When we want to talk about the *image* of a subset A' of A under a relation $R{:}A \rightarrow B$, we write

$$(A')R = \{b \mid <a,b> \in R \text{ and } a \in A'\}.$$

(a)R is the special case, abbreviating $(\{a\})R$, giving the set of $b \in B$ such that $<a,b> \in R$. R is *defined on* a if (a)R is nonempty. The *domain of definition* of R is the set of $a \in A$ such that R is defined on a. The *range* of R is $(A)R = \{b \mid <a,b> \in R\}$.

The following is a selection of important properties of relations. Let $R{:}A \rightarrow B$ be a relation from A to B.

(2.3.1) R is *functional* iff $<a,b> \in R$ and $<a,b'> \in R$ implies $b = b'$.

(2.3.2) R is *total* iff for all $a \in A$ there exists $b \in B$ with $<a,b> \in R$.

(2.3.3) R is *surjective* iff for all $b \in B$ there exists $a \in A$ with $<a,b> \in R$.

(2.3.4) R is *injective* iff $<a,b> \in R$ and $<a',b> \in R$ implies $a = a'$.

(2.3.5) R is *bijective* iff R is injective and surjective.

Given $R{:}A \rightarrow B$, the *opposite, converse,* or *inverse of* R is $R^{-1} = \{<b,a> \mid <a,b> \in R\}$. The following equivalences hold for any $R{:}A \rightarrow B$.

(2.4.1) R is functional iff R^{-1} is injective.

(2.4.2) R is total iff R^{-1} is surjective.

(2.4.3) R is a *function* iff R^{-1} is bijective.

For any set A there exists a unique relation from $\emptyset$ to A; it is $<\emptyset,\emptyset,A>$ and is denoted λ_A or simply λ. λ is functional and total. There is also a unique relation from A to $\emptyset$, $<A,\emptyset,\emptyset>$, which is bijective.

Assume $R{:}A \rightarrow A$ is a relation *on* A.

(2.5.1) R is *reflexive* iff $<a,a> \in R$ for all $a \in A$.

(2.5.2) R is *symmetric* iff $<a,a'> \in R$ iff $<a',a> \in R$.

(2.5.3) R is *antisymmetric* iff $<a,a'> \in R$ and $<a',a> \in R$ implies $a=a'$.

(2.5.4) R is *transitive* iff $<a,a'> \in R$ and $<a',a''> \in R$ implies $<a,a''> \in R$.

A *preorder* is a transitive and reflexive relation. A *partial order* is a preorder which is also antisymmetric. An *equivalence relation* is a transitive, reflexive and symmetric relation, i.e., a preorder which is symmetric.

If $\equiv$ is an equivalence relation on a set A ($\equiv{:}A \rightarrow A$), then $A/\equiv$ is the set of equivalence classes determined by $\equiv$. Let $[a]=\{b \mid a \equiv b\}$, then $A/\equiv \; = \{[a] \mid a \in A\}$ which is a partition of A because $a \equiv b$ implies $[a]=[b]$ (by transitivity and symmetry), which means that the equivalence classes are disjoint and each $a \in A$ is in some class, namely $[a]$ (by reflexivity).

An important fact in later sections is that any preorder, $\underset{\sim}{\sqsubset}$, determines an equivalence relation, $\sim$, by:

$$a \sim a' \quad \text{iff} \quad a \underset{\sim}{\sqsubset} a' \quad \text{and} \quad a' \underset{\sim}{\sqsubset} a.$$

Then $A/\sim$ is partially ordered by

$$[a] \sqsubseteq [a'] \quad \text{iff} \quad a \underset{\sim}{\sqsubset} a'.$$

A *partial function from* A *to* B is a functional relation from A to B. Given $f{:}A \rightarrow B$ we write $(a)f$ to denote the unique $b \in B$ (if it exists) such that $<a,b> \in R$. (Uniqueness is guaranteed by functionality.) A *(total) function* from A to B is a relation that is both functional and total. We will sometimes want to use the notation $[A \rightarrow B]$ for the set of total functions from A to B (later continuous functions when that is clear from context) and $[A \multimap B]$ for the set of partial functions from A to B.

Composition for partial functions and for (total) functions is given by relational composition (2.2). For all we write the composite in diagrammatic order as discussed above. Thus given $f:A\to B$ and $g:B\to C$, the composite is $fg:A\to C$. This is why we make the (at times bothersome) decision to usually (!) write the argument of a function for function evaluation on the left of the function. So $(a)fg = ((a)f)g$. Since this is generally unfamiliar, we will try always to put parentheses around the argument of a function reminding the reader that this (natural) left-to-right order is being assumed.

Let S be any set. Then $S^n = \{w:[n]\to S\}$ is the set of *strings* over S of *length* n. (Note we might well write $S^{[n]}$ instead of S^n, but the latter is more conventional and convenient.) We write $|w|$ for the length of the string w, in this case, n. S^0 has one element, λ_S, usually denoted λ, and called the *empty string* or the *null string*; $|\lambda| = 0$. Strings $w\in S^n$ are often denoted $w_1\ldots w_n$ or $<w_1,\ldots,w_n>$ where $(i)w=w_i\in S$. Note that there is an isomorphism between the set of n-tuples less formally defined above and the set S^n of strings over S of length n. We at least needed to begin with pairs because our definition of strings depends on the definition of function. The important thing is that S^n, equipped with its *projections*, $\pi_i:S^n\to S$ $((w)\pi_i = w_i)$ has the *universal property* that for any indexed family $<f_i:T\to S \mid i\in[n]>$ there exists a unique function $[f_1,\ldots,f_n]:T\to S^n$ such that $[f_1,\ldots,f_n]\pi_i = f_i$ for all $i\in[n]$. And indeed, defining $(t)[f_1,\ldots,f_n] = <(t)f_1,\ldots,(t)f_n>$ gives that required function.

$S^*=\bigcup_{n\geq 0}S^n$ is the set of all *strings* over S. $S^\infty = S^* \cup S^{[\omega]}$ is the set of S-*sequences*.

While on the subject of strings, we will have occasion in Section 4 to use strings in a quite general way. Given $f:[n]\to S$ and $g:[m]\to S$, the function (string) $(f,g):[n+m]\to S$ is defined by $(i)(f,g)=$ (*if* $i\in[n]$ *then* $(i)f$ *else* $(i-n)g$). It is the unique function with source $[n+m]$ satisfying the property that $\iota_1(f,g)=f$ and $\iota_2(f,g) = g$ where $\iota_1:[n]\to[n+m]$ is the inclusion and $\iota_2:[m]\to[n+m]$ sends j to n+j. (We usually think of this operation, (f,g), as the *concatenation* of strings f and g.) Given $f_i:[n_i]\to[m_i]$, then $f_1+f_2:[n_1+n_2]\to[m_1+m_2]$ is $(f_1\iota_1,f_2\iota_2)$ where $\iota_i:[m_i]\to[m_1+m_2]$ as above. Finally, for any set S, 0_S is the unique function from [0] to S and with notation already introduced, $1_{[n]}$ is the identity function on [n]. (Since $[0]=\emptyset$, 0_S was denoted λ_S several paragraphs above.)

We will have many occasions to deal with indexed families of sets, functions and relations. This seems to be characteristic of applications to computer science. Thus it is important to realize that the the notation for sets and functions carries over to indexed families.

Let S be a set (an indexing set), then an S-indexed family of sets $A = <A_s \mid s\in S>$ can be viewed as a function from S to sets, preferably to the set of subsets of some larger set U. We use set-theoretic operations and function notation on indexed families with the same ease as in the unindexed case. If A and B are S-indexed families then $A \subseteq B$ iff $A_s \subseteq B_s$ for all $s \in S$. $A \cup B$ is the indexed family with $(A \cup B)_s = A_s \cup B_s$. A function $f:A \rightarrow B$ on S-indexed families is itself an S-indexed family of functions, $<f_s:A_s \rightarrow B_s \mid s \in S>$. For S-indexed families A and B we will write A^B to mean the set of all S-indexed families of functions f from A to B. With context permitting, we can write $\emptyset$ for the S-indexed family $<\emptyset_s = \emptyset \mid s \in S>$ of empty sets. For $a \in A_s$, $\{a\}$, in this extended notation stands for the S-indexed family which has every component empty except the one indexed by s and that has singleton a.

Cartesian products of members of an S-indexed family A are conveniently described with a notation generalizing the power notation, A^n, for the unindexed case. Let $w = s_1 \ldots s_n$ be a string of elements of S. Then

$$A^w = A_{s_1} \times A_{s_2} \times \ldots A_{s_n}.$$

For the special case when $w = \lambda$, $A^w = A^0 = \{\lambda\}$. Corresponding to $h^n:A^n \rightarrow B^n$ we have $h^w:A^w \rightarrow B^w$ where A and B are S-indexed families and $w = s_1 \ldots s_n$, defined by

$$(a_1,\ldots,a_n)h^w = <(a_1)h_{s_1},\ldots,(a_n)h_{s_n}>.$$

3. GRAPHS AND CATEGORIES.

Definition 3.1. A *directed graph* $G = <E, V, s, t>$ consists of a set E of *edges*, a set V of *vertices* and two functions s,t from the set of edges to the set of vertices. (e)s is the *source vertex* of e and (e)t is the *target vertex* of e. □

Notation. For and edge e, we write $e:v \rightarrow v'$ when $(e)s = v$ and $(v)t = v'$. Further, $G(v,v')$ denotes the set of all edges of G with source v and target v'

Definition 3.2. A *morphism of directed graphs* $h:G \rightarrow G'$ is a pair of functions, $h_E:E \rightarrow E'$ and $h_V:V \rightarrow V'$ which preserve the source and target functions, i.e., $sh_V = h_E s'$ and $th_V = h_E t'$. □

The situation is summarized in Figure 3.1.

$$\begin{tikzcd}
G \arrow[d, "h"] & E \arrow[r, "s", "t"'] \arrow[d, "h_E"] & V \arrow[d, "h_V"] \\
G' & E' \arrow[r, "s'", "t'"'] & V'
\end{tikzcd}$$

Figure 3.1

Definition 3.3. Let G be a directed graph. Then a *path* in G is a triple <u,p,v> where u and v are vertices and $p \in E^*$ is a string of zero or more edges, $p = e_1...e_n$, such that there exist (unique) vertices $v_0,...,v_n$, such that $u=v_0$, $v=v_n$ and for each $i \in [n]$, $(e_i)s=v_{i-1}$ and $(e_i)t=v_i$. We say that <u,p,v> is a path *from* u *to* v. Note that if n=0 then $p=\lambda$ and u=v. Let Pa(G) be the set of paths of G. □

A category is very much like a directed graph with edges (they are called morphisms) and vertices (they are called objects) except that in a category there is a composition operation defined on the edges (morphisms) and there are distinguished morphisms which act as identities for that composition operation. (Also V and E may be proper classes.)

Definition 3.4. A *category* G = <V, E, s, t, ∘,1> consists of a class V of *objects*, a class E of *morphisms*, and two functions s,t from E to V called *source* and *target* respectively. ∘ is an operation on E called *composition* and it is defined on a pair of morphisms e,e′ if and only if (e)t = (e′)s. When this condition is satisfied, then (e∘e′)s = (e)s and (e∘e′)t = (e′)t. When defined, composition is associative:

(3.4.1) $$(e \circ e') \circ e'' = e \circ (e' \circ e'').$$

1 is a function from objects to morphisms; (v)1, written 1_v, is the *identity morphism* for v satisfying:

(3.4.2) $$1_{(e)s} \circ e = e = e \circ 1_{(e)t}.$$

G is said to be a *small category* if both E and V are sets. □

Notation. Just as with directed graphs, for a morphism e, we write $e:v \to v'$ when (e)s = v and (e)t = v′. Similarly $G(v,v') = \{e \mid e:v \to v'\}$ Composition is defined for $e:v \to v'$and $e':v' \to v''$ giving $e \circ e':v \to v''$.

Rather than a partial function, composition should be viewed as a family of functions,

(3.4.3) $$\circ_{v,v',v''}:G(v,v') \times G(v',v'') \to G(v,v'')$$

indexed by all triples <v,v′,v″> of objects of G.

We will try to always use bold-face type for categories. For a category **C**, **C** will ambiguously stand for the class of morphisms of **C** and |**C**| denotes its class of objects. When referring to different categories, say **B** and **C**, we will use the same symbols for source, target , composition and identities (s,t,∘,1). This is a familiar practice in elementary algebra; in studying additive groups, for example, one uses the same symbols +,−,0, for addition, additive inverse, and identity in different groups.

Definition 3.5. Let **A** and **B** be categories. A is a *subcategory* of B iff |**A**| ⊆ |**B**|, the morphisms of **A** are contained in the morphims of **B** (**A**⊆**B**) and all the operations of A are the appropriate restrictions of those for $\mathbb{B}$. A is a *strict* subcategory of $\mathbb{B}$ iff it is a subcategory and $|A| = |\mathbb{B}|$. A is a *full subcategory* of $\mathbb{B}$ iff for all objects A,A′ in $|A|$, $\mathbb{A}(A,A') = \mathbb{B}(A,A')$.

When there is only one object, a category is nothing more than a monoid (semigroup with identity). An arbitrary partially ordered set P (Section & posec.) can be viewed as a (small) category $\mathbb{P}$ with objects, the set P, and morphisms all pairs <u,v> such that $u \sqsubseteq v$. The source of <u,v> is u and the target is v. Composition of <u,v> with <v,w> is <u,w> and $u \sqsubseteq w$ by transitivity. The identity for v is <v,v> (and indeed, $v \sqsubseteq v$ by reflexivity). In $\mathbb{P}$ there is a unique morphism from u to v if there is any morphism from u to v. Associated with any (small) category is a pre-order (reflexive, transitive relation) on the objects given by $u \sqsubseteq v$ iff $\mathbb{G}(u,v)$ is not empty.

Our first example of a non-trivial (and large) category is the category **Gph** of directed graphs. Its objects consist of all directed graphs and its morphisms are graph morphisms. It is an easy exercise to check that the composite of graph morphisms is a graph morphism and that $1_G = <1_E, 1_V>$, the function which is the identity on edges and vertices, is the identity graph morphism.

Lets mention some other useful (large) categories. **Rel** is the category whose objects are sets and whose morphisms are relations. Recall from Section 2 that technically a relation is a triple <A,R,B> where R is a subset of $A \times B$. The source of <A,R,B> is A and the target is B. Composition of relations is the composition operation in **Rel**. The identity morphism for a set A is actually the identity function, $<A,1_A,A>$, where $1_A = \{<a,a> \mid a \in A\}$

The category **Pfn** is the strict subcategory of **Rel** in which the morphisms are partial functions. The category **Set** is in turn the subcategory of **Pfn** in which the morphisms are total functions. In the notation of Section 2, **Pfn**$(A,B)=[A \rightharpoonup B]$ and **Set**$(A,B)=[A \to B]$.

As we've already mentioned in Section 2, we will make considerable use of S-indexed families of sets and functions. When the set S is indexing the carriers of many-sorted algebras (Section 6) it is called a set of *sorts*. The corresponding (large) category is denoted $\mathbf{Set}^S$. It has S-indexed families of sets as objects and S-indexed families of functions as morphisms. So $f{:}A \to B$ in $\mathbf{Set}^S$ is actually a family $<f_s{:}A_s \to B_s \mid s \in S>$. Composition in $\mathbf{Set}^S$ is composition by components, $(f \circ g)_s = f_s g_s$.

Definition 3.5. A *category morphism* $F{:}\mathbb{G} \to \mathbb{G}'$ is called a *functor*. It consists, as does a graph morphism, of a pair of functions, $F_E{:}E \to E'$ and $F_V{:}V \to V'$ which preserve source and target functions and also composition and identities.

(3.5.1) $\quad F_E s = s F_V$

(3.5.2) $F_E t = tF_V$

(3.5.3) $(e \circ e')F_E = (e)F_E \circ (e')F_E$

(3.5.4) $(1_v)F_E = 1_{(v)F_V}$. □

Notation. It is sometimes convenient to use a notation for functors that is consistent with the object-morphism notation for categories, especially in cases where context does not easily distinguish between the object and morphism parts of a functor. So in such cases we let F ambiguously stand for the morphism part of F (i.e., F_E) and $|F|$ stands for the object part of F (for F_V). Often we will use F for both parts.

Let Σ be a set; the category $\mathbf{Gph}_\Sigma$ of Σ-(vertex) labeled graphs is one of importance in computer science. The objects of $\mathbf{Gph}_\Sigma$ are pairs $<G,\ell:V \to \Sigma>$ consisting of a directed graph G and a labeling function ℓ from the vertices of G to the labeling set, Σ. If G and G′ are labeled graphs, then a morphism $F:G \to G'$ is a directed graph morphism that preserves labeling, i.e., $(v)F_V\ell' = (v)\ell$.

Such "labeling" constructions are special cases of an important construction in categorical algebra, called a *comma category* (c.f. Mac Lane(1971) p.46). In this special case we have a functor $U:\mathbf{Gph} \to \mathbf{Set}$ which picks out the set of vertices of the graph and a designated object Σ in **Set**. The comma category is denoted $(U\downarrow\Sigma)$ and called the category of *U-objects over* Σ. In pictures:

objects
$$\begin{tikzcd} GU \arrow[d, "\ell"] \\ \Sigma \end{tikzcd}$$
morphisms
$$\begin{tikzcd} GU \arrow[rr, "(h)U"] \arrow[dr, "\ell"'] & & G'U \arrow[dl, "\ell'"] \\ & \Sigma & \end{tikzcd}$$

Two small categories are N and $\mathbf{Str}_S$. They are the "base categories" for one-sorted and S-sorted algebraic theories respectively (see Section 12). The category N has natural numbers for objects and $N(n,p) = \mathbf{Set}([n],[p])$, i.e., all functions from [n] to [p]; composition is function composition (composition in Set). Let S be a set (called a set of *sorts*). $\mathbf{Str}_S$ has strings on S (S^*) as objects and $\mathbf{Str}_S(u,v) = \{f:[\,|u|\,] \to [\,|v|\,] \mid fv=u\}$.

$$\begin{tikzcd} {[n]} & u & \\ f & & S \\ {[m]} & v & \end{tikzcd}$$

Composition in $\mathbf{Str}_S$ is function composition and it is easy to see that the commuting condition is preserved. A function in $\mathbf{Str}_S(u,v)$ is a morphism of strings in that it can be viewed as sending the i^{th} symbol of u to the $(i)f^{th}$ symbol of v; the commuting condition says $u_i = v_{(i)f}$. Hopefully one sees that $\mathbf{Str}_S$ is also a comma category, namely the comma category (In:$\mathbf{N} \rightarrow \mathbf{Set}$, S) of "In-objects over S." As with the labeled graphs above, $\mathbf{Str}_S$ is labeling of [n] for all $n \in |\mathbf{N}| = \omega$.

The "category of categories" may at first seem overly abstract, and perhaps even contradictory, but it embodies several basic facts about functors and is important in a wide range of applications. We state the necessary properties in the following

Proposition 3.6. Given functors $F:\mathbf{A} \rightarrow \mathbf{B}$ and $H:\mathbf{B} \rightarrow \mathbf{C}$, define their *composite*, $FH:\mathbf{A} \rightarrow \mathbf{C}$ to have object part, $(A)|FH| = (A)|F|\,|H|$ for objects $A \in |\mathbf{A}|$. and morphism part, $(f)FH = ((f)F)H$ for all morphisms $f \in \mathbf{A}$ Then:

(3.6.1) The composite FH is a functor.

(3.6.2) Composition of functors is associative.

(3.6.3) Given a category B, then defining $1_{\mathbf{B}}:\mathbf{B} \rightarrow \mathbf{B}$ by $(B)|1_{\mathbf{B}}| = B$ for B an object of $\mathbf{B}$, and $(f)1_{\mathbf{B}} = f$ for morphisms f of $\mathbf{B}$, gives a functor which is the identity for composition of functors in the sense that the equations

$$F1_{\mathbf{B}} = F \quad \text{and} \quad 1_{\mathbf{B}}G = G$$

hold. $1_{\mathbf{B}}$ is called the *identity functor* on $\mathbf{B}$. □

Let **Cat** denote the category with categories as objects and functors as morphisms. Then Proposition 3.6 gives the properties required by Definition 3.4 that indeed make **Cat** into a category. Its "size" is clearly stupendous in some sense, and one might wonder if it is an object in itself. The reader worried about this and related foundational matters is referred to Mac Lane (1971) or Feferman (1971). One easy way out of this puzzle is to consider only the category $\mathbf{Cat}_S$ of small categories. It is a *subcategory* of **Cat** but is not itself small.

Associated with any (small) category $\mathbf{G}$ is a graph $(\mathbf{G})U$ which is obtained by just "forgetting" the composition operation and the identities. In more detail, given a small category $\mathbf{G} = \langle E,V,s,t,\circ,1\rangle$, we get a graph $(\mathbf{G})U = \langle E,V,s,t\rangle$. It should be obvious that this object mapping (from small categories to directed graphs) immediately extends to a functor from $\mathbf{Cat}_S$ to $\mathbf{Gph}$ because a functor $F:\mathbf{G} \rightarrow \mathbf{G}'$ (in $\mathbf{Cat}_S$) gives a unique graph morphism $(F)U:(\mathbf{G})U \rightarrow (\mathbf{G}')U$; it is the same pair of functions, $\langle F_E,F_V\rangle$, which preserve the source and target functions; we are just forgetting that they also preserve composition and identities. Although "forgetful" functors like this $U:\mathbf{Cat}_S \rightarrow \mathbf{Gph}$ may not be very interesting in themselves (they are kind of "unconstructing" constructions), the exciting thing is that there is usually a "free" construction paired with such a forgetful functor. In this case it is a familiar and useful construction that yields the category of paths of a directed graph.

Before getting into that construction, let's rehearse the process with a special case that may be even more familiar. A monoid is an algebra $<M,\circ,e>$ where M is a set and $\circ$ is a binary associative operation with identity element e. If we forget the operation and the identity we just get a set. So in this case, there is a forgetful functor from the category **Mon** of monoids with monoid homomorphisms, to the category **Set** of sets and functions.

Now the free construction $F:\mathbf{Set}\rightarrow\mathbf{Mon}$ takes an arbitrary set X and gives the monoid freely generated by X, $<X^*,\circ,\lambda>$, where X^* is the set of strings over X, $\circ$ is concatenation, and λ is the empty string. An important and often overlooked ingredient of this construction is the injection (a set map) $I_X:X\rightarrow X^*$ which tells us how to find each generator x as a string $(x)I_X$ of length 1, consisting exactly of x. In Section 2 we saw that a string was a function $w:[n]\rightarrow X$ so $(x)I_X:[1]\rightarrow X$ has the value x on the one element in its source: $(1)((x)I_X) = x$.

The *universal property* associated with the free monoid construction says that for any function $f:X\rightarrow M$ (where $<M,\circ,e>$ is a monoid), there exists a *unique* monoid homomorphism $f^{\#}$ from $<X^*,\circ,\lambda>$ to $<M,\circ,e>$ which extends f in the sense that $I_Xf^{\#} = f$ as set maps. $f^{\#}$ just takes a string $x_1...x_n$ and multiplies out the images in M: $(x_1...x_n)f^{\#} = (x_1)f\circ...\circ(x_n)f$ and, of course, $(\lambda)f^{\#} = e$.

The above is an example of an adjunction, one of the most important concepts of categorical algebra. We give it a precise formulation in the following

Definition 3.7. Let $\mathbb{A}$ and $\mathbb{X}$ be categories, U a functor from $\mathbb{A}$ to $\mathbb{X}$, and $|F|$ an object map, $|F|:|\mathbb{X}|\rightarrow|\mathbb{A}|$, and $I = <I_X:X\rightarrow(X)|F|U>$ a family of morphisms indexed by objects in $|\mathbb{X}|$. Then we say U is a *right adjoint functor with respect to* $<|F|,I>$ when the following universal condition is satisfied.

(*) For every object A of $|\mathbb{A}|$ and morphism $h:X\rightarrow(A)U$ of $\mathbb{X}$ there exists a unique morphism $h^{\#}:(X)|F|\rightarrow A$ such that

$$\begin{tikzcd}
X \arrow[r, "I_X"] \arrow[dr, "h"'] & (X)|F|U \arrow[d, "(h^{\#})U"] & (X)|F| \arrow[d, "h^{\#}"] \\
 & (A)U & A
\end{tikzcd}$$

commutes in $\mathbb{X}$.

When context clearly determines what (*forgetful*) functor $U:\mathbb{A}\rightarrow\mathbb{X}$ is involved in the adjunction, we will refer to the pair, $<X|F|,I_X>$ as the *free* $\mathbb{A}$-object *generated by* X, and usually "$\mathbb{A}$-object" is linguistically given as in "the free monoid generated by X." □

It is common practice in algebra to give the object part of a "free construction" as we have seen for the free monoid. There is good reason for this; the morphism part of that construction is uniquely determined as given in the following proposition.

Proposition 3.8. Given that U is a right adjoint functor with respect to $<|F|,I>$, there is a uniquely determined functor $F:\mathbf{X}\rightarrow\mathbf{A}$ (which is *left-adjoint* to U) and has object part $|F|$. That functor is defined on morphisms as follows: let $g:X\rightarrow X'$ be a morphism in $\mathbf{X}$. Then $gI_{X'}:X\rightarrow(X')|F|U$ and $(g)F$ is defined to be $(gI_{X'})^{\#}:(X)|F|\rightarrow(X')|F|$. The family of morphisms I is called the *unit* of the adjunction. □

In the monoid example, the forgetful functor is $U:\mathbf{Mon}\rightarrow\mathbf{Set}$, a right adjoint functor with respect to $<\text{—}^{*},I>$, i.e., with respect to the object map taking X to $<X^{*},\bullet,\lambda>$ and injections $I_X:X\rightarrow X^{*}$. For a set map $g:X\rightarrow Y$, Proposition 3.8 says g^{*} is $(gI_Y)^{\#}:X^{*}\rightarrow Y^{*}$, i.e., the monoid extension of the mapping of x to the string of length one consisting of (x)g in Y^{*} That extension takes $x_1...x_n$ to $(x_1)g\bullet...\bullet(x_n)g$ (making the usual identification of y with $(y)I_Y$), a familiar and important process.

We have gone from an easy rehearsal to an abstract statement of what an adjunction is; now let us return to the directed graphs. From Definition 3.3, (G)PA is the set of paths of G; the source and target of a path are obvious: $(<u,p,v>)s = u$ and $(<u,p,v>)t = v$. The composite $<u,p,v>\circ<v,p',w>$ is $<u,pp',w>$ which can be easily checked to be a path from u to w. The identity for each vertex is the empty path, $<v,\lambda,v>$, which is indeed the identity for composition. So $(G)\mathbf{PA} = <(G)PA,V,s,t,\circ,1>$, defined in this way, is a category. It is called the *path category of* G. In just the same way as we injected the generators of the free monoid into X^{*}, so we need a graph morphism $I_G:G\rightarrow(G)\mathbf{PA}U$ and it is the obvious one that takes an edge e to the path $<(e)s,e,(e)t>$ of length one. (Note that we might have pedantically written $<(e)s,(e)I_E,(e)t>$, where I_E is the injection of E into E^{*}.)

$(G)\mathbf{PA}$ is the category freely generated by G and that is what the following proposition says, but in the language of Definition 3.7.

Proposition 3.9. $U:\mathbf{Cat}_S\rightarrow\mathbf{Gph}$ is a right adjoint functor with respect to $<\mathbf{PA},I>$ given above. □

We indicate the proof. Given any graph morphism $h:G\rightarrow(\mathbf{C})U$ one extends h to $h^{\#}:(G)\mathbf{PA}\rightarrow\mathbf{C}$ in the same way that the extension to a monoid homomorphism was obtained. A path $<u,e_1...e_n,v>$ goes to $(e_1)h\circ(e_2)h\circ...\circ(e_n)h$, where that composition is composition in $\mathbf{C}$. One just composes in $\mathbf{C}$ the morphisms which are the images of edges of G.

We can say things about a graph in terms of its path category. For example, G is *acyclic* iff for all vertices v, if $p:v\to v$ in $(G)\mathbb{PA}$ then $p = 1_v$. v is a *root* of G iff $p:v'\to v$ implies $v'=v$ and $p = 1_v$. G is a *forest* iff there is at most one path p from any v to any v′. G is a *tree with root* v iff v is *initial* in $(G)\mathbb{PA}$ This is an important concept of which we make considerable use, so we give

Definition 3.10. An object I is *initial* in a category $\mathbb{C}$ iff for every object C there exists a unique morphism $h:I\to C$. Dually, T is a *terminal object* in $\mathbb{C}$ iff for every object C there is a unique morphism $h:C\to T$. A *zero object* in $\mathbb{C}$ is an object which is both initial and terminal. □

The root of a tree is an example of an initial object (in a small category), namely, in the path category of the tree. Looking on a larger scale, whenever we have a free construction of the kind described here, if there is an initial object in the source of that free construction, then there is initial object in the target. Look at the universal property of Definition 3.7. Let J be initial in $\mathbb{X}$; for every object A of $\mathbb{A}$, there exists a unique morphism $h:J\to(A)U$ (because J is initial) and thus there is a unique $h^{\#}:(J)F\to A$ that extends h $((I_J \circ h^{\#}U) = h)$. If there were another $g:(J)F\to A$, then $I_J(gU):J\to(A)U$, and by J's initiality that composite has to be h so g $= h^{\#}$.

The initial object in **Set** is $\emptyset$; the unique morphism $\lambda_A:\emptyset\to A$ was mentioned in Section 2. The monoid freely generated by $\emptyset$ just has the identity element which is not very interesting, but it is initial in the category of monoids, nonetheless. The free category gives us an even less interesting example because the initial graph is the empty graph ($V = E = \emptyset$), and the path category (the initial category) is also empty. All this is far from typical. The initial algebras of Sections 6, 10, and 11 play an important role in algebraic semantics and the specification of data types. Looking to those applications, the following proposition is important.

Proposition 3.11. If I and J are initial objects in a category $\mathbb{C}$ then I and J are (uniquely) isomorphic. Also if I is initial and I is isomorphic to J then J is initial. The same holds for terminal and zero objects. □

There are common and important "conventions" in mathematics, and in algebra in particular, that can be understood and justified with the material of this section. We have already seen one: the morphism part of a free construction is uniquely determined by its definition on objects so it is unnecessary to say how that construction works on (homo)morphisms (this is Proposition 3.8). Another convention is that we speak of "*the* initial object" or, in particular "*the* initial algebra." That language is justified by Proposition 3.11: and two initial objects are isomorphic and anything isomorphic to an initial object is initial.

Initiality determines the objects up to isomorphism which is usually all we care about, especially viewing isomorphism as inessential notational variation.

The reader should be wondering about "*the* free object," "*the* free algebra" or very particularly, "*the* free monoid." Indeed Proposition 3.11 is a special case of a much more general result which we now state.

Proposition 3.12. If $U:A \rightarrow X$ is a right adjoint functor with relative to $<|F|, I>$ and also relative to $<|G|, J>$, then $|F|$ and $|G|$ are "naturally isomorphic." □

4. GRAPHS AND FLOWCHARTS

There are many ways that graphs enter into the formulation of models for computation. We wish to explore a selection of those here. All grow from the fundamental definition of directed graph (Definition 3.1) and morphisms of directed graphs (Definition 3.2). At least three distinct attitudes are discernible. In the first, the edges of the graph are labeled with partial functions (employing the "Karp device" (Karp (1959)): here the emphasis is on simulation, homomorphisms, and proofs of correctness of programs. (Milner (1971), Burstall (1972a), Goguen (1974), and ADJ (1976c).) In the second, the edges are still labeled, but emphasis is on the definition of and properties of recursive flowchart schemes (Burstall and Thatcher (1974), Goguen and Meseguer (1977) and Gallier (1977,1978)). Finally there is the detailed study of directed graphs *qua* flowcharts where the nodes are labeled as in the flowcharts with which we are all familiar. The emphasis here is on the operations such as parallel and serial composition used to build up the flowcharts. (See especially Elgot (1971,1973,1976,1977) and Elgot and Shepherdson (1977).) We begin with the last direction of study because it contains ideas that are important for the other areas. In particular it points to the fact that there are many ways to treat flow diagrams and that one has to be particularly careful with the mathematical definitions.

Definition 4.1. Let $G = <V,E,s,t>$ be a directed graph. G is *finite* if both E and V are finite. G is *locally finite* iff for each $v \in V$, $\{e \mid (e)s = v\}$ is finite, i.e., the set of edges leaving a vertex must be finite. When G is locally finite there is a natural ranking function on the set of vertices:

$$(v)r_G = \#\{e \mid (e)s = v\},$$

and we say that v has *rank* $(v)r_G$. □

For node-labeled flowcharts, the edges leaving a node must be ordered to distinguish, say, between the *true* and *false* branches of a test. One especially simple way to achieve this is given in Elgot (1977):

Definition 4.2. An *outedge directed graph* $G = <V,E,s,t>$ has for its edges,

(4.2.1) $\quad E = \{<v,i> \mid v\in V \text{ and } i\in[(v)r_G]\}$

and the source function is given by,

(4.2.2) $\quad <v,i>s = v.$ □

Thus in an outedge directed graph, the edges leaving a vertex are naturally ordered by

$$<v,1>,<v,2>,\ldots,<v,(v)r_G>.$$

Note that because of (4.2.2) all the information of the outedge directed graph is given by the triple $<V,E,t>$ and actually the relevant data consists of the set V of vertices, the rank function $r_G:V\rightarrow\omega$ and the target function,

$$t:\{<v,i> \mid v\in V \text{ and } i\in[(v)r_G]\}\rightarrow V.$$

The same effect is obtained with the notion of ordered directed graph given by Arbib and Giveon (1968):

Definition 4.3.. A (locally finite) *ordered directed graph*[†] G consists of a set V of vertices and a function $\tau:V\rightarrow V^*$. The *rank* of a vertex v is the length of $(v)\tau$. □

The sense in which the two definitions have the same effect is that for any ordered directed graph $<V,\tau>$ there is an outedge directed graph $<V,E,t>$ where

$$E = \{<v,i> \mid i\in[(v)\tau]\}$$

and,

$$<v,i>t = (i)(v)\tau.$$

Similarly one can construct an ordered directed graph from an outedge directed graph and that construction is inverse to what we just described. Take $<V,E,t>$ and define $\tau:V\rightarrow V^*$ by:

$$(i)(v)\tau = <v,i>t, \text{ for } i\in[(v)r_G].$$

We will return to a very natural use of ordered directed graphs in a while.

Let $\Sigma = <\Sigma_0,\Sigma_1,\ldots,\Sigma_n,\ldots>$ be a *ranked alphabet* of disjoint sets. We sometimes use the notation Σ for $\cup\Sigma_i$ which has an associated *ranking function*, $r_\Sigma:\Sigma\rightarrow\omega$,

$$(\sigma)r_\Sigma = i \text{ iff } \sigma\in\Sigma_i.$$

Ranked alphabets are objects in $\mathbf{Set}^\omega$ and *ranked alphabet maps* are morphisms there, i.e., families of functions $f=<f_i:\Sigma_i\rightarrow\Sigma_i'>$. The alternative is to view ranked alphabets as objects in the comma category of **Set**-objects over ω (see Section 3).

[†] Of course Arbib and Giveon called them directed ordered graphs with the acronym DOG.

For any p, let X_p denote a set (of *variables*) $\{x_1,...,x_p\}$ disjoint from Σ. Then we write $\Sigma(X_p)$ for the ranked alphabet with X_p adjoined as symbols of rank zero, $\Sigma(X_p)_0=\Sigma_0 \cup X_p$ and $\Sigma(X_p)_i=\Sigma_i$ for $i\neq 0$. We also view the set of edges of an outedge directed graph as a ranked alphabet with ranking function r_G. The following is a variation of the corresponding definition in Elgot (1977).

Definition 4.4. Let Σ be a ranked alphabet. For natural numbers n and p, a *Σ-chart* F *from* n *to* p consists of the following data.

(4.4.1) An outedge directed graph $G=<V,E,t>$.

(4.4.2) A *pointing* (or *begin*) *function* $b:[n]\rightarrow V$.

(4.4.4) A ranked-alphabet map $\ell:V\rightarrow\Sigma(X_p)$ called the *labeling function.*

If $(v)\ell\in X_p$ then v is called a *terminus* and the set of termini is denoted V^{term}. If ℓ is bijective restricted to V^{term}, then the termini are called *exits*. $S=V-V^{term}$ is called the set of *internal vertices* of F and F is said to be of *weight* S. The pair $<G,\ell>$ is called the *underlying labeled graph* of the F. The class of Σ-charts from n to p is denoted $Ch_\Sigma(n,p)$. □

When $\Sigma_i=\emptyset$ for all i except i = 1,2, then we have the usual class of flowcharts where Σ_1 is the set of operation symbols and Σ_2 is the set of predicate symbols (c.f., Elgot (1976)). A crucially important difference from many treatments, however, is the fact that the charts have n "begins" and p "exits" for arbitrary n and p in ω. It is this additional structure (giving the structure of a category eventually) that makes things work as well as they do.

There are a couple of things that are troublesome about Definition 4.4. First, we know of no computational significance for what could be vast parts of a chart that are "inaccessible" from the begin vertices. And second, $Ch_\Sigma(n,p)$ is *large* since any outedge directed graph could provide the underlying graph structure for a Σ-chart. We take care of the first problem with the definition of "accessibility" to follow. But when we consider operations of flowcharts, we are interested in finite flowcharts and we can take a subclass of $Ch_\Sigma(n,p)$ which is a set by choosing "canonical" sets of vertices.

Definition 4.5. Let F be a Σ-chart from n to p. F is *accessible* iff every vertex v lies on a path from some begin vertex, i.e., $(G)PA((i)b,v)\neq\emptyset$ for some $i\in[n]$. □

Note that by this definition, every vertex in an accessible Σ-chart either lies on a path from an begin vertex to a terminus, is on a loop out of which one can *not* reach a terminus, or is in a path terminating in a vertex labeled by some element of Σ_0. The significance of the last case is a curious matter!

Definition 4.6. A **morphism** $H:F\rightarrow F'$ of Σ-charts from n to p is a morphism of their underlying labeled graphs which preserves the begin function, (4.6.1) $(i)bH_V = (i)b'$,

and the ordering:

(4.6.2) $<v,i>H_E = <(v)H_V,i>$.

We denote the category of Σ-charts from n to p with bold letters, $\mathbf{Ch}_\Sigma(n,p)$. $\mathbf{ACh}_\Sigma(n,p)$ denotes the full subcategory of $\mathbf{Ch}_\Sigma(n,p)$ whose objects are the accessible Σ-charts from n to p. □

Note that a Σ-chart morphism is completely determined by its vertex part, H_V, and so much structure has been put on the charts that we have the following rather surprising fact (Elgot (1977),(3.2)).

Fact 4.7. There exists at most one morphism $H:F \to F'$ in $\mathbf{ACh}_\Sigma(n,p)$. □

Let us write $F \lesssim F'$ iff there is a morphism from F to F′ in $\mathbf{ACh}_\Sigma(n,p)$. The relation $\lesssim$ is a preorder (see Section 2 or the *definition* of preorder in Mac Lane (1971), p.11). The induced equivalence relation ~ from Section 2 is, by Fact 4.7, the isomorphism relation. Furthermore, $\mathbf{ACh}_\Sigma(n,p)/\sim$ is partially ordered by $[F] \sqsubseteq [F']$ iff $F \lesssim F'$. In fact this is a partially ordered *set* of isomorphism classes of Σ-charts from n to p. It seems to be quite different from other posets of flowcharts in the literature (e.g. Scott's lattice of flow diagrams (Scott (1971) or alternatives in ADJ (1975), Nivat (1973), or Wand (1972)).

Elgot (1977) views $\mathbf{ACh}_\Sigma(n,p)$ in a different way:

Proposition 4.8. Let ≈ be the transitive closure of the union of $\lesssim$ and its converse (see Section 2.) This relation is an equivalence relation on the accessible Σ-charts from n to p. Let $[F]_\approx$ be the full subcategory of $\mathbf{ACh}_\Sigma(n,p)$ determined by F's ≈-equivalence class. Then $[F]_\approx$ has an initial object I and a terminal object M. The initial object is an n-tuple of (labeled) trees which is the "complete unfolding" of F and the terminal object M is the minimal chart with the same "strong behavior" as F. □

We are really getting ahead of ourselves in that we didn't plan to discuss behaviors of flowcharts until we have introduced algebraic theories. In the terminology of Burstall and Thatcher (1974) we have been looking at the "vertical structure" of the category of Σ-charts (there n=p=1 and edges were labeled instead of vertices but the deterministic case corresponds to $\mathbf{ACh}_\Sigma(1,1)$ for an appropriate Σ). Now we are interested in the horizontal structure (composition of Σ-charts) and without actually saying so we are coming close to a double category; something we said in the Introduction would not be necessary.

As promised above we are now going to pick out a subclass of the Σ-charts which is a set and which includes representatives of all the finite charts. On this set we will define the basic operations that are used to build flowcharts: composition, pairing and iteration. That these are the essential operations on charts is a key contribution of Elgot (1973).

The way we pick the subclass is to use the set [s+p] for the set of vertices of a Σ-chart with s internal nodes and p exits. Since the p exits must have outdegree zero and since the i^{th} exit is always labeled with x_i we make this information implicit by defining the underlying ordered directed graph to be a function from [s] to $[s+p]^*$ and the labeling is only defined on [s]. There is an additional wrinkle. We want to allow, in any flowchart, a box labeled "undefined." To be consistent with Definition 4.4, we define the ranked alphabet $\Sigma_\perp$ to be exactly like Σ except that $(\Sigma_\perp)_1=\Sigma_1\cup\{\perp\}$.

Recall from Section 2: $0_A:[0]\to A$ is the unique function from $[0]=\emptyset$ to A; and, $I_A:A\to A^*$ is the set injection of A into the underlying set of the free monoid A^*.

Definition 4.9. A (*normalized*) *$\Sigma_\perp$-flowchart from* n *to* p *of weight* s consists of a triple $<b,\tau,\ell>$ where:

begin function	$b:[n]\to[s+p]$
underlying graph	$\tau:[s]\to[s+p]^*$
labeling function	$\ell:[s]\to\Sigma_\perp$,

such that $|(i)\tau|=((i)\ell)r_{\Sigma_\perp}$. (i)b is called a *begin vertex*, $i\in[s]$ is an *internal vertex*, $i\in s+[p]$ is an *exit* and in particular, s+j is the j^{th} *exit vertex*. $(i)\ell$ is the operation symbol labeling the i^{th} internal vertex; it must have rank $|(i)\tau|$. Let $Fl_{\Sigma_\perp}(n,p)$ be the set of $\Sigma_\perp$-flowcharts from n to p. □

Definition 4.10. The *identity* $\Sigma_\perp$-flowchart from n to n, denoted 1_n, has weight 0 and:

begin function	$1_{[n]}:[n]\to[n]$
underlying graph	$0_{[n]^*}:[0]\to[n]^*$
labeling function	$0_{\Sigma_\perp}:[0]\to\Sigma_\perp$

□

Definition 4.11. The *composite* of $\Sigma_\perp$-flowcharts, $F=<b,\tau,\ell>$ from n to p of weight s and $F'=<b',\tau',\ell'>$ from p to q of weight s' is $F\circ F'$ from n to q of weight s+s' with:

begin function	$bf:[n]\to[s+s'+q]$
underlying graph	$(\tau f^*,\tau' g^*):[s+s']\to[s+s'+q]^*$
labeling function	$(\ell,\ell'):[s+s']\to\Sigma_\perp$

where f and g are the following functions,

$$f=1_{[s]}+b':[s+p]\to[s+s'+q]$$

$$g=0_{[s]}+1_{[s'+q]}:[s'+q]\to[s+s'+q].$$

□

Informally $F \circ F'$ is obtained by identifying the p exits of F with the q begin vertices of F'. At the same time the vertices of F' are "translated" by s, i.e., a vertex j of F' becomes s+j in $F \circ F'$.

Theorem 4.12. For each $n,p \in \omega$, let $\mathbf{Fl}_{\Sigma_\perp}(n,p)$ be the set of $\Sigma_\perp$-flowcharts from n to p (i.e., $\mathrm{Fl}_{\Sigma_\perp}(n,p)$). Then $\mathbf{Fl}_{\Sigma_\perp}$ is a category with the nonnegative integers as objects, with composition given by Definition 4.11, and with identities given by Definition 4.10. That is, composition is associative and the identity charts satisfy 3.4.2. □

Definition 4.13. The *pairing* or *coalesced sum* of two $\Sigma_\perp$-flowcharts $F=<b,\tau,\ell>$ from n to p of weight s and $F'=<b',\tau',\ell'>$ from n' to p of weight s' is (F,F') from $n+n'$ to p of weight $s+s'$ where

begin function $(bf,b'g):[n+n'] \to [s+s'+p]$

underlying graph $(\tau f^*,\tau' g^*):[s+s'] \to [s+s'+p]^*$

labeling function $(\ell,\ell'):[s+s'] \to \Sigma_\perp$

where

$$f=1_{[s]}+0_{[s']}+1_{[p]}:[s+p] \to [s+s'+p]$$

$$g=0_{[s]}+1_{[s'+p]}:[s'+p] \to [s+s'+p].$$

□

Informally, the effect of pairing is to put the two charts F and F' next to each other identifying the p exits of F with those of F'.

Proposition 4.14. Pairing of $\Sigma_\perp$-flowcharts is associative, i.e.,

$$(F_1,(F_2,F_3))=((F_1,F_2),F_3)$$

for F_1, F_2, F_3 where the pairing is defined. □

The last is perhaps the most important operation; it is the only one that employs '$\perp$'. Thus all the definitions above apply to Ω-flowcharts with arbitrary Ω replacing our special $\Sigma_\perp$. The idea is that for an $\Sigma_\perp$-flow chart from n to n+p of weight s, the "iterate" of F, denoted $F^\dagger$, identifies the ith exit with the ith begin node, thus introducing 'loops,' the result has p exits and weight s. The construction is more complicated than that, however, because the ith exit might be the ith begin (for example) and this iteration has to yield an nonterminating loop ($\perp$).

Definition 4.15. Let $F=<b,\tau,\ell>$ be a $\Sigma_\perp$-flow chart from n to n+p of weight s. Further, let $f=(x^{s+n+p}_{(1)},b,x^{s+n+p}_{(3)}):[s+n+p] \to [s+n+p]$ and factor f^n to

$$f^n = h \circ (1_s+g+1_p):[s+n+p] \to [s+n+p],$$

where $h:[s+n+p] \to [s+u+p]$ and $g:[u] \to [n]$ and u is the smallest natural number yielding such a factorization. The *iterate* of F is the flow chart $F^\dagger$ from n to p of weight s+u with:

begin functionb$\circ$h:$[n] \to [s+u+p]$ underlying graph
$(\tau \circ h^*, \lambda^u))$:$[s+u] \to [s+u+p]^*$ labeling function$(\ell, \perp^u)$:$[s+u] \to \Sigma_\perp$,

where $\lambda^u:[u] \to [s+u+p]^*$ sends each $i \in [u]$ to $\lambda \in [s+u+p]^*$ and $\perp^u$ sends each $i \in [u]$ to $\perp \in \Sigma_\perp$.

Definition 4.16. For any function $f:[n] \to [p]$ we define an associated $\Sigma_\perp$-flowchart $f^\wedge$ from n to p of weight 0; $f^\wedge = <f, 0_{[p]^*}, 0_{\Sigma_\perp}>$. □

Using the charts corresponding to maps (Definition 4.16) and coalesced pairing (Definition 4.13) we define *the separated sum* of F_i from n_i to m_i $(i \in [2])$ to be the chart

$$F_1 \oplus F_2 = (F_1 \circ f_1^\wedge, F_2 \circ f_2^\wedge)$$

where $f_i:[s_i+m_i] \to [s_1+s_2+m_1+m_2]$ are the obvious injections for $i = 1,2$.

Flowcharts will be interpreted much later. The idea (of Elgot (1973) and Wagner (1974)) is to construct an equivalent category of "Σ-normal descriptions," which consists of pairs of morphisms from the algebraic theory freely generated by Σ. Once the operations and tests (Σ) have been interpreted in a rational theory, the interpretation of the normal descriptions, and thus flowcharts, is uniquely determined by the requirement that the maps, composition, pairing, and iteration be preserved.

5. POSETS

As we saw in Section 2, a *poset* is a set P equipped with a partial order $\sqsubseteq$ on P. Let $S \subseteq P$; then $u \in P$ is an *upper bound* for S is $p \sqsubseteq u$ for every $p \in S$; u is a *least upper bound* for S if u is an upper bound for S and $u \sqsubseteq v$ for every upper bound v of S. We write $\bigsqcup S$ for the least upper bound of S if it exists.

Let P and P′ be posets: then a mapping $f:P \to P'$ is said to be *monotonic* if it preserves the ordering, i.e., for all $p, p' \in P$, $p \sqsubseteq p'$ implies $(p)f \sqsubseteq (p')f$. The collection of all posets together with the monotonic mappings forms a category **Po**, called the *category of posets.*

We say that an element $\perp$ of a poset P is the *bottom* element of P if it is minimum in P, i.e., if $\perp \sqsubseteq p$ for all $p \in P$. We say a poset is *strict* if it has a bottom element; a monotonic mapping $f:P \to P'$ between strict posets is *strict* if $\perp f = \perp$. The strict posets together with the strict monotonic mappings between them form a category $\mathbf{Po}_\perp$ called the *category of strict posets,* which is a subcategory of **Po** (though not a strict one!).

A very simple and familiar construction takes a set X (from **Set**) to the *flat* poset $X_\perp$, where $X_\perp$ is X with a new element $\perp$ adjoined, ordered by $x \sqsubseteq y$ in $X_\perp$ iff $x=y$ or $x=\perp$. This gives us another example of an adjunction. In particular if $I_X:X \to X_\perp$ is the inclusion (in **Set**) then $\langle I_X, X_\perp \rangle$ is the "free strict poset generated by X." This means that for any strict poset P and any set map $f{:}X \to P$ there exists a unique strict monotonic map $f^{\#}{:}X_\perp \to P$ extending f ($f^{\#}$just sends $\perp$ to the minimum element of P).

Definition 5.1. Let **Po** be the category of posets with monotonic functions as morphisms. A *subset system* on **Po** is a function Z which assigns to each poset P, a set Z[P] of subsets of P such that:

(5.1.1) For each poset P and $p \in P$, $\{p\} \in Z[P]$.

(5.1.2) If $f{:}P \to P'$ in **Po** and $S \in Z[P]$, then $Sf = \{ sf \mid s \in S \} \in Z[P']$.

We call the elements of Z[P], the Z-sets of P, and say "S is a Z-set in P", when $S \in Z[P]$. □

Definition 5.2. Given a subset system Z on **Po**, a poset P is Z-*complete* iff every Z-set of P has a least upper bound in P. A morphism $f{:}P \to P'$ is Z-*continuous* iff for every Z-set S in P such that $\bigsqcup S$ exists,

$$(\bigsqcup S)f \;=\; \bigsqcup\{ sf \mid s \in S \} = \bigsqcup(S)f.$$

So Z-continuous functions preserve least upper bounds of Z-sets that exist in their source. □

We first used the term "Z-set" in ADJ (1975a) as a notational device, without the definition (5.1) of a "subset system." We were interested in a few instantiations of Z which we enumerated and which are included in the list below. In effect, we left it to the reader to check that the results worked for the various Z's and didn't realize that that checking essentially depended on the main part of the definition, i.e., closure of Z-sets under monotonic maps.

The following are examples of subset systems; most have appeared in one form or another in the literature.[†] The examples are presented by saying, in each case, what it means for a subset S of a poset P to be a Z-set in P.

S is a $\bigsqcup$-set iff S is nonempty.

S is an n-set iff S is nonempty and $\equiv S \leq n$.

S is a PC-set iff S is nonempty and every pair from S has an upper bound in P (a *pairwise compatible* subset of P).

S is a C-set iff S is nonempty and every finite subset of S has an upper bound in P (a *compatible* subset of P.

[†] The term "pairwise compatible" is from Markowski and Rosen (1976) although the concept is to be found in Egli and Constable (1976). wo-completeness is used by Tiuryn (1976).

S is a $\dot{\bigsqcup}$-set iff S is nonempty and has an upper bound in P (a *bounded* subset of P).

S is a Δ-set iff S is nonempty and every pair from S has an upper bound in S (iff every finite subset of S has an upper bound in S) (a *directed* subset of P).

S is an ℓ-set iff S is nonempty and linearly ordered (a *chain*).

S is a wo-set iff S is a nonempty well-ordered chain.

S is an α-set iff S is a chain of order type α.

S is a $\sqcup$-set iff S is a finite nonempty set (a finite $\bigsqcup$-set).

S is a pc-set iff S is a finite PC-set.

S is a $\dot{\sqcup}$-set iff S is a finite bounded set (a finite $\dot{\bigsqcup}$ set).

The many notions of completeness and continuity are instances of Definition 5.2 for Z's taken from the list above. Thus "chain-complete" is ℓ-complete, "continuous with respect to directed sets" is Δ-continuous. In general, we can pick out the subcategory of **Po** whose objects are the Z-complete posets and whose morphisms are the Z$'$-continuous functions; the result will be denoted **Po**[Z,Z$'$]. We already have one trivial example. Taking Z to be sets of cardinality less than or equal to one (this is *not* Z=1 from the above list) Z-completeness requires $\bigsqcup\emptyset$ exist (since $\bigsqcup\{p\}=p$) and this is $\perp$; Z-continuity of f means that f is strict (preserves $\perp$). Thus, in this case, **Po**[Z,Z] is $\mathbf{Po}_\perp$.

For any Z, if there is a P with $\emptyset \in Z[P]$, then preservation of Z-sets under monotonic maps ensures $\emptyset \in Z[P]$ for all P, so we will say Z is *strict* if $\emptyset \in Z[P]$ for any (all) P; in this case **Po**[Z,Z] is always a subcategory of $\mathbf{Po}_\perp$. All of the Z's listed above are non-strict and it is sometimes convenient to have a notation for their strict counterparts. So let $Z_\perp[P] = Z[P] \cup \{\emptyset\}$. Generally we can just use the qualifier "strict" and also, by example, write $\mathbf{Po}_\perp[\Delta]$ for $\mathbf{Po}[\Delta_\perp,\Delta_\perp]$. (In this notation $\mathbf{Po}_\perp = \mathbf{Po}[1_\perp,1_\perp]$, as seen from the previous paragraph.)

Definition 5.3. A subset X of a poset P is an *ideal* iff it is downward closed: p $\in$ X and p$'$ $\sqsubseteq$ p implies p$'$ $\in$ X. Let S be an arbitrary subset of P; then the *ideal generated* by S is

$$\hat{S} = \{p \mid p \sqsubseteq p' \text{ for some } p' \in S\}.$$

X is a Z-*ideal* if X is generated by a Z-set.□

Definition 5.4. Let Z be a subset system on **Po** and P a poset. An element p $\in$ P is Z-*isolated* in P if for each Z-set D in P such that $\bigsqcup$D exists, if p $\sqsubseteq$ $\bigsqcup$D then p $\sqsubseteq$ d for some d $\in$ D. We call the set of Z-isolated elements of P the Z-*core* of P and denote it Core[P]. □

In ADJ (1977) we used the term "compact," following Birkhoff (1967) and Markowski and Rosen (1976), instead of "isolated." We now believe that "isolated" (following Scott

(1972a)) is better. However, for the next concept, we seem to be fighting the tide. The term "algebraic" (as in Graetzer (1968), Scott (1972a) and Courcelle and Nivat (1976)) is gaining wide acceptance. We find such constructions as "algebraic algebras" and, worse, "algebraic algebraic theories" to be so unpleasant that we stick to the terminology of ADJ (1977) and use "inductive."

Definition 5.5. A poset P is *Z-inductive* iff every $p \in P$ is the least upper bound of some Z-set in Core[P]. □

The poset of partial functions on a set is one of the handiest examples to illustrate these concepts. Looking at a fixed source and target, say a set A, we consider the graphs of the partial functions ordered by set inclusion. Call this poset **Pfn**(A,A) (consistent with the notation in Section 3). The Δ-isolated elements of **Pfn**(A,A) are the finite functions while the C-isolated (also $\mathring{\sqcup}$- and PC-) elements are the singleton or one-point functions. The poset **Pfn**(A,A) is Δ-inductive and C-inductive because every partial function $f{:}A \to A$ is the least upper bound of (a) the directed set of finite functions contained in f and (b) the C-set of one-point functions making up f. **Pfn**(A,A) is also Δ-complete and C-complete; it is not $\sqcup$-complete (compare with **Rel**(A,A)).

The crucial property of Z-inductive posets is that monotonic maps defined on their core extend uniquely to continuous maps on all of the poset.

Theorem 5.6. (The Extension Theorem for Z-inductive posets.) Let P and Q be posets with P, Z-inductive and Q, Z-complete. Let $f{:}\mathrm{Core}[P] \to Q$ be monotonic. Then there exists a unique monotonic Z-continuous $f^{\#}{:}P \to Q$ which extends f. In particular, if $p \in P$, and $S \in Z[P]$ such that $p = \sqcup S$, then $(p)f^{\#} = \sqcup(Sf)$. □

Our statement of the Extension Theorem includes the method of proof; the extension of $f{:}\mathrm{Core}[P] \to Q$ is defined for $p \in P$ by $(p)f^{\#} = \sqcup(S)f$ where S is a Z-set in Core[P] with $\sqcup S = p$; such an S exists because P is Z-inductive; the least upper bound exists in Q because Q is Z-complete.

Inductive posets are intimately related to the "extension basis" for posets discussed in Markowsky and Rosen (1976). We give their definition relativized to Z.

Definition 5.7. A subset B of a Z-complete P is a *Z-extension basis* iff, for every Z-complete poset Q and monotonic map $f{:}B \to Q$, there exists a unique monotonic Z-continuous $f^{\#}{:}P \to Q$ that extends f. □

Corollary 5.8. If a Z-complete poset P is Z-inductive then Core[P] is a Z-extension basis for P. □

Definition 5.9.[†] Let I[P] denote the poset of Z-ideals ordered by set inclusion. An ideal is *principal* if it is generated by a singleton; let C[P] be the poset of principal ideals. Since every singleton set is a Z-set we know $C[P] \subseteq I[P]$. Let $\iota_P : P \to I[P]$ be the monotonic map sending each $p \in P$ to the principal ideal generated by $\{p\}$. □

Fact 5.10. $\iota_P : P \to I[P]$ is an injection and its target restriction to C[P], $\iota_P : P \to C[P]$, is an isomorphism. □

The partially ordered set I[P] (ordered by set inclusion) is the "Z-inductive completion" of P. We need, however, an additional property of Z for the construction to work properly.

Definition 5.11. Z is *union-complete* iff I[P] is Z-complete and for any Z-set $S \in Z[I[P]]$, $\bigsqcup S = \bigcup S$. Equivalently, Z is union-complete iff the set union of Z-ideals is a Z-ideal. □

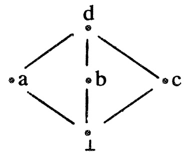


Figure 5.1.

Most of the Z's listed above are union-complete; exceptions include the finite cardinals and chains.[‡] A simple example illustrates this for finite cardinals. Let P be the poset in Figure 5.1 and take Z=2. The doubleton set of Z-ideals, $\{\{a,b,\bot\},\{c,\bot\}\}$ (a 2-set in I[P]), has a least upper bound in I[P], namely P, which is not the union of $\{a,b,\bot\}$ and $\{c,\bot\}$.

For contrast, consider any directed set D of ideals generated by directed sets, and let S be the union of the generators. Then S is directed: for any x_1, $x_2 \in S$, x_i comes from a directed set s_i in D. But D is directed so s_1 and s_2 are contained in a directed set s; thus x_1 and x_2 are members of s and have an upper bound x in s and thus in S. And it is easy to see that $\hat{S} = \bigcup D$. We have actually shown that Δ possesses a stronger property than union-completeness; we call it "strong-union complete."

Definition 5.12. A subset system Z is *strong union-complete* iff Z[P] is Z-complete and for every Z-set D in Z[P], $\bigsqcup D = \bigcup D$.□

[†] We really should write $I_Z[P]$ since the construction depends on Z; hopefully the Z will always be clear from context.

[‡] In ADJ (1977) we stated that only the finite cardinals were *not* union complete. Banaschewski and Nelson (1979) point out that Z = chains is also not union complete. Meseguer (1979) has shown that every Z naturally determines one that is union complete and for which continuity and completeness are the same.

Under the condition that Z is union-complete (or strong union-complete), the completion, I[P], has the desired properties.

Theorem 5.13. If Z is union-complete then for each poset P, I[P] is Z-inductive and Z-complete. □

Corollary 5.14. If Z is union-complete and P is Z-inductive and Z-core complete then P ≅ I[Core[P]]. □

Also under the condition of union-completeness, we can tie up the connection with extension bases of Markowski and Rosen (1976).

Proposition 5.15. Assume Z is union-complete. If B is an Z-extension basis for a Z-complete poset P then P is Z-inductive and B = Core[P]. □

We will write **Po**[Z] instead of **Po**[Z,Z] for the category of Z-complete posets with Z-continuous monotonic morphisms. Let $I: |\mathbf{Po}| \to |\mathbf{Po}[Z]|$ be the mapping which takes each each poset P to I[P], the poset of Z-ideals of P. The completion of Theorem 5.13 gives an adjunction which we state in the language of Definition 3.7 from Section 3.

Theorem 5.16. When Z is union-complete, the inclusion functor from **Po**[Z] to **Po** is a right adjoint functor with respect to the object map $I: |\mathbf{Po}| \to |\mathbf{Po}[Z]|$, and the family of morphisms $\iota_P{:}P \to I[P]$. Equivalently, given $P \in |\mathbf{Po}|$, a Z-complete poset Q, and a monotonic map $f{:}P \to Q$, then there exists a unique mapping $f^{\#}{:}I[P] \to Q$ extending f in the sense that $\iota_P \circ f^{\#} = f$. In particular, if $X \in I[P]$ then $Xf^{\#} = \bigsqcup\{\, xf \mid x \in X \,\}$.

$$\begin{tikzcd}
P \arrow[r, "\iota_P"] \arrow[dr, "h"'] & I[P] \arrow[d, "h^{\#}"] \\
& Q
\end{tikzcd}$$

□

It is consistent with the language of Section 3 to say that $<I[P],\iota_P>$ is the "free Z-complete, poset generated by P," which is just saying that I[P] enjoys the same universal property in relation to posets and monotonic maps as the free monoid with respect to sets and functions; it is all just saying that I is the object part of a functor which is left adjoint to a forgetful functor, in this case an inclusion (forgetting that a poset is Z-complete).

In Section 11 we want to talk about completing algebras rather than just posets. The idea is that we complete the poset P to I[P] to get a Z-complete and Z-inductive carrier for the algebra. The functions or operations of the algebra are not just defined on P but on powers, P^n, of P. Thus we need to consider what happens to completions of powers (or products) of P in relation to completion of P. In particular, it is necessary to extend monotonic functions defined on P^n to Z-continuous functions defined on $I[P]^n$. For all of this we need some additional definitions and facts.

Definition 5.17. Given posets $P_1,...,P_n$ where P_i has partial order $\sqsubseteq_i$, we define the product of the P_i (in the usual way) to be the poset with underlying set $P_1 \times ... \times P_n$ and ordering $\sqsubseteq$ where for $<p_1,...,p_n>$, $<p'_1,...,p'_n> \in P_1 \times ... \times P_n$, $<p_1,...,p_n> \sqsubseteq <p'_1,...,p'_n>$ iff $p_i \sqsubseteq_i p_i'$ for each $i \in [n]$. Let π_i^n denote the ith *projection function*, i.e., $\pi_i^n : P_1 \times ... \times P_n \rightarrow P_i$, taking $<p_1,...,p_n>$ to p_i. □

Fact 5.18. Let Z be a subset system, let $P_1,...,P_n$ be posets and let $S \in Z[P_1 \times ... \times P_n]$. Then, for each $i \in [n]$, $S\pi_i^n = \{ s\pi_i^n \mid s \in S \} \in Z[P_i]$. □

Fact 5.19. Least upper bounds of Z-sets work by components on cartesian products, i.e., for any Z-set $D \subseteq P_1 \times ... \times P_n$

$$\bigsqcup D = <\bigsqcup(D)\pi_1^n,...,\bigsqcup(D)\pi_n^n>. \qquad \square$$

Corollary 5.20. If S is a Z-set in $P_1 \times ... \times P_n$ then $\bigsqcup S$ exists iff $\bigsqcup(S\pi_i^n)$ exists for each $i \in [n]$. □

The well-known weaker concept of continuity by components has the following Z-statement.

Definition 5.21. A monotonic mapping $f:P_1 \times ... \times P_n \rightarrow P$ is *Z-continuous by components* iff for each $i \in [k]$, $a_1 \in P_1, ... , a_{i-1} \in P_{i-1}, a_{i+1} \in P_{i+1}, ..., a_k \in P_k$, and Z-set $S \subseteq P_i$,

$$f(a_1,..., a_{i-1}, \bigsqcup S, a_{a+1}, ..., a_k) = \bigsqcup\{f(a_1,..., a_{i-1}, s, a_{a+1}, ..., a_k) \mid s \in S\}. \qquad \square$$

Fact 5.22. For each n, if $f:P^n \rightarrow P$ is Z-continuous then f is Z-continuous by components. □

Proposition 5.23. Let P and $P_1,...,P_n$ be posets and let $f:P_1 \times ... \times P_n \rightarrow P$ be monotonic. Then f is Z-continuous by components iff for all $i \in [n]$, and every Z-set $S_i \in Z[P_i]$ such that $\bigsqcup S_i$ exists,

$$f(\bigsqcup S_1,...,\bigsqcup S_n) = \bigsqcup< f(s_1,...,s_n) \mid s_i \in S_i, i \in [n]>. \qquad \square$$

Definition 5.24. Let S and T be subsets of a poset P. Then we say S is *cofinal* in T, and write $T \underset{\sim}{\sqsubset} S$, iff for every $t \in T$ there exists $s \in S$ such that $t \sqsubseteq s$. If S is cofinal in T, and T is cofinal in S then we say S and T are *mutually cofinal*, and write $S \sim T$. □

Definition 5.25. Let Z be a subset system. We say Z is *crossed-down* if for all posets P_1 and P_2 and each Z-set $S \subseteq P_1 \times P_2$, S and $S\pi_1 \times S\pi_2$ are mutually cofinal. □

Theorem 5.26. Let Z be a crossed-down subset system. Then $f:P_1 \times P_2 \rightarrow P$ is Z-continuous by components implies f is Z-continuous. □

Theorem 5.27. Let Z be subset system which is crossed down and union complete. Then for all posets P_1 and P_2, $I[P_1 \times P_2] \cong I[P_1] \times I[P_2]$. □

6. ALGEBRAS.

The basic concepts of universal algebra are to be found in Birkhoff (1935, 1967), Cohn (1965), Graetzer (1968) and, perhaps more leisurely in ADJ (1973, 1975). An algebra in the sense of Birkhoff is simply a set, called the carrier of the algebra, together with an indexed family of operations on Cartesian powers of that carrier. The indexing system for the operations is what is called an operator domain or ranked alphabet (as in Section 4). A ranked alphabet is itself an indexed family of sets, $\Sigma = <\Sigma_i \mid i \in \omega>$, where Σ_n is the set of names of operations of rank or arity n.

But computer science applications seem to need "heterogeneous" or as we shall call them, "many-sorted" algebras. A many-sorted algebra is just like an algebra in the sense of Birkhoff described above except that you have several carriers instead of one and they are indexed by a set S, called the set of "sorts." Then the operations of a many-sorted algebra go from Cartesian products of those carriers into one of them. The indexing on the operation symbols must reflect this additional complexity. (See the definition of "signature" below.) For example, the sort set might be *{real,int,bool}*. An algebra of this kind would have three carriers, A_{real}, A_{int} and A_{bool} together with some operations such as $EXP:A_{real} \times A_{int} \rightarrow A_{real}$ and $COND:A_{bool} \times A_{real} \times A_{real} \rightarrow A_{real}$ corresponding to exponentiation and conditional respectively.

A finite automaton, in the sense of Rabin and Scott (1959), 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" $<S,S\Sigma>$ which yields a state for each state and input pair. The initial state is a constant of sort S.

The definition of many-sorted algebra that we give below is equivalent to that of "heterogeneous" algebra given by Birkhoff and Lipson (1970) (except carriers must be allowed to be empty) and also equivalent to Higgin's (1963) "algebra with a scheme of operators". The main point is that the theory of universal algebra carries over with "undiminished force" to the many-sorted case. Birkhoff and Lipson (1970) give computer science examples but the first explicit uses for new results in computer science seem to be in Maibaum (1972,1973) and in F.L. Morris (1972,1973a).

Definition 6.1. An S-*sorted signature* Σ consists of a set S, called the set of *sorts* and an indexed family $<\Sigma_{s,w} \mid s\epsilon S$ and $w\epsilon S^*>$ of disjoint sets, called the *operator symbols*. Call $\sigma\epsilon\Sigma_{s,w}$ an operator symbol of *type* $<s,w>$, *arity* w, *sort* s, and *rank* $|w|$. A *constant symbol of sort* s is an element σ in $\Sigma_{s,\lambda}$ □

When S is a singleton, say {a}, the indexing of the signature consists of pairs $<a,a^n>$ and this is the "one-sorted" or "homogeneous" case mentioned at the beginning of this section. Since the sort is always the same it can be ignored in the indexing, and all that remains is the length of the arity, i.e., the *rank*, in this case, n. Thus the {a}-sorted signature reduces to the ranked alphabet, $\Sigma = <\Sigma_i \mid i\epsilon\omega>$ with Σ_n naming operations of rank n.

Definition 6.2. Let Σ be an S-sorted signature. An *algebra* A *with signature* Σ or briefly, a Σ-*algebra* consists of an indexed family of sets, $<A_s \mid s\epsilon S>$ and an indexed family of operations,

$$<\sigma_A : A^w \rightarrow A_s \mid \sigma \epsilon \Sigma_{s,w}>.$$

The set A_s is called the *carrier of* A *of sort* s. The operation σ_A is the *operation of* A *named by* σ. For $\sigma\epsilon\Sigma_{s,\lambda}$, $\sigma_A\epsilon A_s$ is a *constant of* A. □

Definition 6.3. Let A and B be Σ-algebras. Then A is a *subalgebra of* B, written $A\subseteq B$, iff for each $s\epsilon S$, $A_s\subseteq B_s$ and for each operator symbol $\sigma\epsilon\Sigma_{s,w}$ and $<a_1,...,a_n>\epsilon A^w$,

$$(a_1,...,a_n)\sigma_A = (a_1,...,a_n)\sigma_B. \qquad □$$

Definition 6.7. If A and B are both Σ-algebras, A Σ-*homomorphism*. $h:A\rightarrow B$ is a function $h:A\rightarrow B$, i.e., a family of functions $<h_s:A_s\rightarrow B_s \mid s\epsilon S>$, that preserve the operations:

(6.7.1) if $\sigma\epsilon\Sigma_{s,\lambda}$ then $(\sigma_A)h = \sigma_B$;

(6.7.2) if $\sigma\epsilon\Sigma_{s,s_1...s_n}$ and $<a_1,...,a_n>\epsilon A_{s_1}\times...\times A_{s_n}$ then

$$((a_1,...,a_n)\sigma_A)h_s = ((a_1)h_{s_1},...,(a_n)h_{s_n})\sigma_B. \qquad □$$

The composite of homomorphisms is again a homomorphism and composition is an associative operation. The identity function, 1_A, on the carrier of A (so actually the S-indexed family of identity functions) is a Σ-homomorphism which is the identity for composition. Thus we have a category $\mathbf{Alg}_\Sigma$ whose objects are Σ-algebras and whose morphisms are Σ-homomorphisms.

We are now going to define the S-indexed family of Σ-*expressions* with *generators* or *variables* from an indexed family of sets, $X = <X_s \mid s\epsilon S>$. This will give us the carrier of the Σ-algebra freely generated by X.

Let Σ (ambiguously) denote the *set* of all operator symbols in the S-sorted signature Σ, i.e., $\bigcup\{\Sigma_{s,w} \mid s\in S \text{ and } w\in S^*\}$. Also let X be an S-indexed family of disjoint sets which is also disjoint from the set of operator symbols (i.e., from Σ). (We will also write X for $\bigcup X_s$.) Then define $T_\Sigma(X) = \langle T_\Sigma(X)_s \mid s\in S\rangle$ to be the smallest S-indexed family of sets of strings over the set $\Sigma\cup X\cup\{(,)\}$ satisfying the following two conditions. (Here $\{(,)\}$ is a two element set disjoint from both Σ and X, but later we will be writing expressions over $\Sigma\cup X$ without using the special brackets, (and).)

(6.8.1) $\quad \Sigma_{s,\lambda} \subseteq T_\Sigma(X)_s$;

(6.8.2) $\quad$ if $\sigma\in\Sigma_{s,s_1\ldots s_n}$ and $t_i\in T_\Sigma(X)_{s_i}$ then, $\sigma(t_1\ldots t_n) \in T_\Sigma(X)_s$.

Example 6.1. Take $S = \{d,s\}$, $\Sigma_{d,\lambda} = \{*\}$, $\Sigma_{s,\lambda} = \{\star,\Lambda\}$, $\Sigma_{s,s} = \{POP\}$, $\Sigma_{s,ds} = \{PUSH\}$, $\Sigma_{d,s} = \{TOP\}$, $X_s = \emptyset$ and finally $X_d = \{x_1,x_2,\ldots\}$. Then $T_\Sigma(X)_s$ contains the likes of $\star$, Λ, PUSH($x_i\Lambda$), PUSH($x_i\star$), PUSH(x_iPUSH($x_j\Lambda$)), TOP(PUSH($x_i\Lambda$)), etc. Whereas $T_\Sigma(X)_d$ has $*$, TOP($\star$), TOP(Λ), TOP(PUSH($x_i\star$)), TOP(PUSH($x_i\Lambda$)), etc. □

Then we make $T_\Sigma(X)$ into a Σ-algebra by defining the operations σ_T for each $\sigma\in\Sigma$ (Note we are using σ_T instead of the typographically bothersome $\sigma_{T_\Sigma(X)}$, for the operation named by σ in $T_\Sigma(X)$.)

(6.9.1) $\quad$ For $\sigma\in\Sigma_{s,\lambda}$, $\sigma_T = \sigma \in T_\Sigma(X)_s$.

(6.9.2) $\quad$ For $\sigma\in\Sigma_{s,s_1\ldots s_n}$ and $t_i\in T_\Sigma(X)_{s_i}$,

$$(t_1,\ldots,t_n)\sigma_T = \sigma(t_1\ldots t_n) \in T_\Sigma(X)_s.$$

Theorem 6.10. Let $I_X: X\to T_\Sigma(X)$ be the (S-indexed family of set) injection(s) of the generators X into the carrier of $T_\Sigma(X)$ Then $\langle I_X, T_\Sigma(X)\rangle$ is the algebra freely generated by X in $\mathbf{Alg}_\Sigma$. That is, for any Σ-algebra A and any map $h: X\to A$ (again, S-indexed family) there exists a unique Σ-homomorphism $h^\#: T_\Sigma(X)\to A$ such that $I_X h = h^\#$ as set maps.

$$\begin{tikzcd}
X \arrow[r, "I_X"] \arrow[dr, "h"'] & T_\Sigma(X) \arrow[d, "h^\#"] \\
 & A
\end{tikzcd}$$

□

Corollary 6.11. $T_\Sigma(\emptyset) = T_\Sigma$ is the initial algebra in the category of Σ-algebras, i.e., for any Σ-algebra A, there is a unique homomorphism $h_A: T_\Sigma \to A$. □

7. ALGEBRAIC SEMANTICS.

A fundamental tenet of algebraic semantics is that syntactic constructs reside in free (or initial) objects and that semantics is completely determined by specifying an algebra with the same signature as the syntactic algebra and by specifying the values of the generators (if any); then the semantic function is the unique homomorphism from the syntactic algebra to the semantic one guaranteed by Theorem 6.10 or Corollary 6.11.

In the discussion here we focus on the free and initial Σ-algebras, but the fundamental process that is going on applies equally well to the other free and initial algebras and algebraic theories to be introudced in the sequel. Algebraic semantics is applicable in all those cases as well as it is here.

Example 7.1. Consider the following $\{int,Bool\}$-sorted signature for integer and Boolean expressions (borrowed from Kamin (1979)).

$$\Sigma_{int,\lambda}=\{0,1,2,\ldots\} \qquad \Sigma_{int,int}=\{\bar{\ },\mathrm{Pr},\mathrm{Su}\} \qquad \Sigma_{int,int\ int}=\{+,-,*\}$$

$$\Sigma_{Bool,\lambda}=\{\mathrm{tt},\mathrm{ff}\} \qquad \Sigma_{Bool,Bool}=\{\neg\} \qquad \Sigma_{Bool,Bool\ Bool}=\{\wedge\ \vee\}$$

$$\Sigma_{Bool,int}=\{\mathrm{even}\} \qquad \Sigma_{Bool,int\ int}=\{<,>,=\} \qquad \Sigma_{int,Bool\ int\ int}=\{\mathrm{cond}\}.$$

All other $\Sigma_{s,w}$ are empty. $T_{\Sigma,int}$ is the set (or algebra) of integer valued expressions and $T_{\Sigma,Bool}$ is the set of Boolean valued expressions. The expected semantic algebra is the algebra S with $S_{int}=\mathbb{Z}$ (the integers) and $S_{Bool}=[2]$ (the Boolean values, which we take to be $\{1,2\}$ for technical convenience). We know how to define all the operations, like $+_S$ and $\wedge_S$, on these carriers and so we don't need to spell them out. It would be consistent with the notation of Scott and Strachey (1971) to write $[\![e]\!]$ for the value of the expression e under the unique homomorphism from T_Σ to S. If we introduce variables, $X_{Bool}=\{x_{Bool,1}, x_{Bool,2},\ldots\}$ and $X_{int}=\{x_{int,1},x_{int,2},\ldots\}$, then for any assignment $\rho:X\to S$ of values (of appropriate sort) to the variables (or identifiers) there is a unique homomorphism $\rho^{\#}:T_\Sigma(X)\to S$ and we might well write $[\![e]\!]\rho$ instead of $(e)\rho^{\#}$.□

"Structural induction" is a consequence of initial algebra semantics. Let Σ be an S-sorted signature and let $<P_s \mid s\in S>$ be a family of predicates indexed by the sorts. Then "structural induction" says the following.

IF (0) $P_s(\sigma)$ for all $\sigma\in\Sigma_{s,\lambda}$;

AND (1) For all $\sigma\in\Sigma_{s,s_1\ldots s_n}$, and $t_i\in T_{\Sigma,s_i}$

$$P_{s_1}(t_1)\wedge\ldots\wedge P_{s_n}(t_n) \Rightarrow P_s(\sigma(t_1\ldots t_n))$$

THEN For all $s\in S$ and $t\in T_{\Sigma,s}$, $P_s(t)$.

Let $\overline{P}_s$ be the set of $t \in T_{\Sigma,s}$ such that $P_s(t)$ is true. Make $\overline{P}$ into a Σ-algebra by;

$$\sigma_{\overline{P}} = \sigma$$

$$(t_1,\ldots,t_n)\sigma_{\overline{P}} = \sigma(t_1\ldots t_n)$$

Then $\overline{P}$ is a subalgebra of T_Σ by cases (0) and (1) of the hypotheses of structural induction. We have a unique homomorphsim $h_{\overline{P}}:T_\Sigma \rightarrow \overline{P}$. And the inclusion from $\overline{P}$ to T_Σ is also a homomorphism. The composite, by uniqueness, must be the identity on T_Σ; i.e., $\overline{P}$ is all of T_Σ. This is the desired conclusion of "structural induction."

A very general instance of algebraic semantics is found in the semantics of context-free languages as described in ADJ (1975a).[†] Let $G=<N,T,P>$ be a context-free grammar. N is the set of *non-terminals*, T is the set of *terminals*, and $P \subseteq N \times (N \cup T)^*$ is a set of *productions*.[†] Let $V = N \cup T$, and for $w \in V^*$, define (w)var to be the string of nonterminals occurring in w. (More precisely, $\text{var}:V^* \rightarrow N^*$ is the unique extension to a monoid homomorphism of the map $V \rightarrow N^*$ which is the identity on N and sends each $t \in T$ to $\lambda \in N^*$.)

Now make G into an N-sorted signature where for each $<A,w> \in N \times N^*$,

$$G_{A,w} = \{p \in P \mid p=<A,w'> \text{ and } (w')\text{var}=w \}.$$

Thus a production $A \rightarrow u_0A_1u_1A_2\ldots u_{n-1}A_nu_n$ $(A_i \in N,\ u_i \in T^*)$ is, in this setting, an operator symbol of type $<A,A_1\ldots A_n>$. The initial G-algebra T_G has carriers $T_{G,A}$ which are parse trees for derivations in G from the non-terminal A.

The impact of algebraic semantics here is that any G-algebra whatsoever (a set S_A for each non-terminal A and a function

$$p_S:S_{A_1} \times \ldots \times S_{A_n} \rightarrow S_A$$

for each production p of type $<A,A_1\ldots A_n>$) provides a semantics for the context-free language generated by G. T_G, being initial, gives a unique homomorphism $h_S:T_G \rightarrow S$ 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 a simple example, make T^* into a G-algebra (call it T^*, with every carrier T^*) by letting

$$(v_1,\ldots,v_n)p_{T^*} = u_0v_1u_1\ldots u_{n-1}v_nu_n$$

[†] The importance of the algebraic structure associated with a context free grammar seems to have been independently discovered (and precisely formulated) by Morris (1973a) and Rus (1974).

[‡] We do not want to exclude the possibility of any of these sets being infinite. Effective presentation of such systems must be given careful consideration. (See van Wijngaarden (1969).)

for every production $p = <A, u_0A_1u_1...u_{n-1}A_nu_n>$ in G. Then the unique homomorphism $h_S:T_G \to T^*$ assigns to each derivation in $t \in T_{G,A}$ the string in T^* that is derived. Thus, even "string generated " is a semantics for G.

Example 7.2. We illustrate the general process described above with a specific example of a programming language intimately related to the one employed by Morris (1973a) in his "Advice on structuring compilers and proving them correct." It is a slight enrichment of the language used as an example by Milner (1976) and it is the one we use in ADJ (1979a). Our grammar will have non-terminals {<st>,<ae>,<be>} for "statements", "arithmetic expressions" and "Boolean expressions." The terminals include the symbols in the signature Σ of Example 7.1 plus those other letters in boldface occurring in the productions below. Further, we assume given a set X of variables or identifiers.

We list the productions of G giving each a name which we can use in defining the semantic algebra. Thus, for example, when G is viewed as an operator domain, ifthenelse is an operator symbol to denote a function that takes three arguments of sorts <be>,<st>,<st>, respectively, and yields a result of sort <st>. Similarly result takes two arguments of sort <st> and <ae> and yields a result of sort <ae>.

(L1)	continue	<st> ::= **continue**	
(L2)	x:=	<st> ::= **x:=**<ae>	For $x \in X$
(L3)	ifthenelse	<st> ::= **if**<be>**then**<st>**else**<st>	
(L4)	;	<st> ::= <st>;<st>	
(L5)	whildo	<st> ::= **while**<be>**do**<st>	
(L6)	c	<ae> ::= **c**	For $c \in \Sigma_{int,\lambda}$
(L7)	x	<ae> ::= **x**	For $x \in X$
(L8)	aop1	<ae> ::= **aop1**<ae>	For $aop1 \in \Sigma_{int,int}$
(L9)	aop2	<ae> ::= <ae>**aop2**<ae>	For $aop2 \in \Sigma_{int,int\ int}$
(L10)	cond	<ae> ::= **if**<be>**then**<ae>**else**<ae>	
(L11)	result	<ae> ::= <st>**result**<ae>	
(L12)	letx	<ae> ::= **letxbe**<ae>**in**<ae>	For $x \in X$
(L13)	bc	<be> ::= **bc**	For $bc \in \Sigma_{Bool,\lambda}$
(L14)	prop	<be> ::= **prop**<ae>	For $prop \in \Sigma_{Bool,int}$
(L15)	rel	<be> ::= <ae>**rel**<ae>	For $rel \in \Sigma_{Bool,int\ int}$
(L16)	bop1	<be> ::= **bop1**<be>	For $bop1 \in \Sigma_{Bool,Bool}$
(L17)	bop2	<be> ::= <be>**bop2**<be>	For $bop2 \in \Sigma_{Bool,Bool\ bool}$

Now we define the semantic algebra M. For this we need the set Env of "environments," Env = $[X \to \mathbb{Z}]$. Then the three carriers are:

$$M_{<st>} = [\mathrm{Env} \mathrel{-\!\!\circ\!\!\rightarrow} \mathrm{Env}] \quad M_{<ae>} = [\mathrm{Env} \mathrel{-\!\!\circ\!\!\rightarrow} \mathrm{Env} \times \mathbb{Z}] \quad M_{<be>} = [\mathrm{Env} \mathrel{-\!\!\circ\!\!\rightarrow} \mathrm{Env} \times [2]].$$

Here $[A \to B]$ is the set of (total) functions from A to B and $[A \mathrel{-\!\!\circ\!\!\rightarrow} B]$ is the (po)set of partial functions from A to B (see Section 2).

The definitions of the seventeen operations on M (corresponding to the grammar's seventeen productions) involve certain primitive operations on M's carriers along with standard (and some not so standard) combining forms.

We first list the primitive operations:

$$\mathrm{assign}_x : \mathrm{Env} \times \mathbb{Z} \to \mathrm{Env} \qquad (z)\langle e,v\rangle \mathrm{assign}_x = \begin{cases} v \text{ if } z=x \\ (z)e \text{ if } z \neq x \end{cases}$$

$$\mathrm{fetch}_x : \mathrm{Env} \to \mathrm{Env} \times V \qquad (e)\mathrm{fetch}_x = \langle e,(x)e\rangle$$

We also have available all the operations σ_S, for $\sigma \in \Sigma$, from Section 2; e.g., $+_S$ is addition on the integers.

Given two (partial) functions, $f_i : A \to B$, define the *source tuple*, $(f_1,f_2) : A \times [2] \to B$, by

$$\langle a,i\rangle(f_1,f_2) = (a)f_i.$$

Define the *sum*, $f_1+f_2 : A \times [2] \to B \times [2]$, of functions $f_i : A \to B$ for $i \in [2]$ by:

$$\langle a,i\rangle(f_1+f_2) = \langle (a)f_i,i\rangle.$$

If $\iota_i : B \to B \times [2]$ is the injection sending $b \in B$ to $\langle b,i\rangle$, for $i \in [2]$, then $f_1+f_2=(f_1 \circ \iota_1, f_2 \circ \iota_2)$. $B \times [2]$ is the disjoint union, sum or coproduct of B with itself, and more generally $B \times [n]$ is the coproduct of B with itself n times (n disjoint "copies" of B); $\iota_i : B \to B \times [n]$ sends b to $\langle b,i\rangle$, for $i \in [n]$. Context will usually distinguish the source of an injection and for this paper, the target will always be clear. When necessary to distinguish sources, we will write $\iota_j^B : B \to B \times [n]$.

Given a partial function $f : A \to A \times [2]$, define the *iterate*, $f^\dagger : A \to A$, to be the least upper bound (i.e. union) of the sequence $f^{(k)}$ defined by:

$$f^{(0)} = \emptyset$$
$$f^{(k+1)} = f \circ (f^{(k)}, 1_A),$$

where $\emptyset$ is the empty partial function from A to A.

Given (partial) functions $f_i : A \to B_i$, define the *target tuple*, $[f_1,f_2] : A \to B_1 \times B_2$, by:

$$(a)[f_1,f_2] = \langle (a)f_1,(a)f_2\rangle.$$

Note that if either f_1 or f_2 is undefined at a, then $[f_1,f_2]$ is undefined at a. The projection function $\pi_i:A_1\times\ldots\times A_n\rightarrow A_i$ takes $<a_1,\ldots,a_n>$ to a_i. Given functions $f_i:A_i\rightarrow B_i$, define their *product*, $f_1\times f_2:A_1\times A_2\rightarrow B_1\times B_2$, by:

$$<a_1,a_2>(f_1\times f_2) = <(a_1)f_1,(a_2)f_2>.$$

Paralleling the sum case above, the product of functions is defined in terms of target tupling and projections: $f_1\times f_2=[\pi_1\circ f_1,\pi_2\circ f_2]$.

Now for the definitions of M's operations; τ,τ_1,τ_2, range over $M_{<st>}$; α,α_1,α_2 range over $M_{<ae>}$; and, β,β_1,β_2 range over $M_{<be>}$.

(M1) $\mathsf{continue}_M = 1_{Env}$

(M2) $(\alpha)\mathsf{x{:=}}_M = \alpha\circ \mathrm{assign}_x$

(M3) $(\beta,\tau_1,\tau_2)\mathsf{ifthenelse}_M = \beta\circ(\tau_1,\tau_2)$

(M4) $(\tau_1,\tau_2)\mathsf{;}_M = \tau_1\circ\tau_2$

(M5) $(\beta,\tau)\mathsf{whiledo}_M = (\beta\circ(\tau+1_{Env}))^{\dagger}$

(M6) $\mathsf{c}_M = 1_{Env}\times \mathsf{c}_S$

(M7) $\mathsf{x}_M = \mathrm{fetch}_x$

(M8) $(\alpha)\mathsf{aop1}_M = \alpha\circ(1_{Env}\times \mathsf{aop1}_S)$

(M9) $(\alpha_1,\alpha_2)\mathsf{aop2}_M = \alpha_1\circ(\alpha_2\times 1_{\mathbb{Z}})\circ[\pi_1,\pi_3,\pi_2]\circ(1_{Env}\times \mathsf{aop2}_S)$

(M10) $(\beta,\alpha_1,\alpha_2)\mathsf{cond}_M = \beta\circ(\alpha_1,\alpha_2)$

(M11) $(\tau,\alpha)\mathsf{result}_M = \tau\circ\alpha$

(M12) $(\alpha_1,\alpha_2)\mathsf{letx}_M = \mathrm{fetch}_x\circ[(\alpha_1\circ \mathrm{assign}_x\circ\alpha_2)\times 1_{\mathbb{Z}}]\circ[\pi_1,\pi_3,\pi_2]\circ(\mathrm{assign}_x\times 1_{\mathbb{Z}})$

(M13) $\mathsf{bc}_M = 1_{Env}\times \mathsf{bc}_S$

(M14) $(\alpha)\mathsf{prop}_M = \alpha\circ(1_{Env}\times \mathsf{prop}_S)$

(M15) $(\alpha_1,\alpha_2)\mathsf{rel}_M = \alpha_1\circ(\alpha_2\times 1_{\mathbb{Z}})\circ(1_{Env}\times \mathsf{rel}_S)$

(M16) $(\beta)\neg_M = \beta\circ(\iota_2,\iota_1)$

(M17a) $(\beta_1,\beta_2)\wedge_M = \beta_1\circ(\iota_1,\beta_2)$

(M17b) $(\beta_1,\beta_2)\vee_M = \beta_1\circ(\beta_2,\iota_2)$

The Boolean expressions are treated differently from the arithmetic expressions. In the definition of $\wedge_M$, for example, β_1 can give the value false (1) and β_2 will not be evaluated, i.e., could be non-terminating: if $(e)\beta_1=<e',1>$ (false with new environment e′), then $(e)\beta_1\circ(\iota_1,\beta_2)=<e',1>$ independent of β_2.

Calling our grammar above, G, we have made $M = <M_{<st>}, M_{<ae>}, M_{<be>}>$ into a G-algebra with the seventeen definitions, (M1-M17). The algebraic semantics for G is the unique homomorphism $\theta:T_G\rightarrow M$. □

8. EQUATIONAL CLASSES.

$T_\Sigma(X)$ is sometimes called the "anarchic" (no laws) or "totally free" Σ-algebra. We use $T_\Sigma(X)$ for syntax for presenting classes of algebras satisfying certain properties, first "equational properties." A *Σ-equation of sort* s is just a pair of expressions $e = <e_1,e_2>$ from $T_\Sigma(X)_s$. And an *equational system* (*over* $T_\Sigma(X)$) is a set (pedantically, family) E of Σ-equations. We will write $e_1=e_2$ for $<e_1,e_2>\in E$, but what makes these pairs "equations" is how we interpret them or use them; as they are, they are just pairs of expressions with variables X. When treating the generators X as variables, we speak of $\theta:X\to A$ as an *assignment* of values in the algebra A to the variables X, or an *interpretation*. Any assignment θ, extends by Theorem 6.10 to a unique homomorphism $\theta^\#:T_\Sigma(X)\to A$; θ gives the values in A for the variables and A, being a Σ-algebra, gives values for the operation symbols σ -- $\theta^\#$ is *evaluation* and nothing more.

For an equation $e=<e_1,e_2>$, let var(e) be the variables[†] occurring in e; this is a (finite) subfamily of X. A Σ-algebra A *satisfies* e iff for every assignment[‡] θ:var(e)$\to$A,

$$(e_1)\theta^\# = (e_2)\theta^\#.$$

We can also say e is *valid* in A or E *holds* in A; our use of "satisfies" is consistent with past writing but may be confusing since satisfaction suggests existential quantifiers whereas from a logical point of view, when an an algebra A satisfies e it means that the universally quantified equation is valid in A. Another alternative would be to say A is a *model for* e instead of A "satisfies" e

For an equational system E, a Σ-algebra A *satisfies* E iff it satisfies every $e\in E$. We then say that A is a (Σ,E)-algebra. $\mathbf{Alg}_{\Sigma,E}$ is the category of all Σ-algebras satisfying E with the Σ-homomorphisms. It is a full subcategory of $\mathbf{Alg}_\Sigma$, i.e., $\mathbf{Alg}_{\Sigma,E}(A,B) = \mathbf{Alg}_\Sigma(A,B)$ because if $h:A\to B$ is a Σ-homomorphism then it is still one if A and B both satisfy E

[†] One usually doesn't bother with things like this, but technically var(e) = var(e_1)$\cup$var(e_2). But what about var(e_i)? Define a Σ-algebra VAR with $VAR_s = p_\omega(X)$, i.e., all carriers of VAR consist of finite subfamilies of X. All the operations are union; $(Y_1,\ldots,Y_n)\sigma_{VAR}=Y_1\cup\ldots\cup Y_n$. Now define $v:X\to VAR$ for $x\in X_s$ by $(x)v_s=\{x\}$ and $(x)v_{s'}=\emptyset$ for $s'\neq s$, i.e., (x)v is the singleton family for x. Finally apply Theorem 6.10 to get $v^\#:T_\Sigma(X)\to VAR$. Then var(e_i) = $(e_i)v^\#$!

[‡] As pointed out in ADJ (1976) one has to take care here. Birkhoff and Lipson (1970) made the error of defining many-sorted (there *heterogeneous*) algebras to have nonempty carriers, thus rendering false results such as the existence of free algebras and the lattice of subalgebras; you must permit empty carriers. But with empty carriers care must be taken with the definition of satisfaction. If a variable of sort s does not occur in an equation and the carrier A_s is empty then there are *no* assignments for the variable x, so that if we universally quantify over assignments including x, then the equation is vacuously valid. Thus we must restrict assignments to the variables occurring in the equation.

$\mathrm{Alg}_{\Sigma,E}$ also has free algebras and the way one is constructed is to take the "quotient" of $T_\Sigma(X)$ modulo the "congruence relation" determined by the equational system E. This results in identifying all expressions in $T_\Sigma(X)$ forced to be so identified by the equations in E. For this purpose we need the following definitions.

Definition 8.1. A Σ-*Congruence*, $\equiv$ on a Σ-algebra A is a family $<\equiv_s \mid s\in S>$ of equivalence relations, $\equiv_s$ on A_s, with the *substitution property*: for all $\sigma\in\Sigma_{s,s_1\ldots s_n}$, if $a_i,b_i\in A_{s_i}$ and if $a_i\equiv_{s_i}b_i$ for $i=1,\ldots,n$ then

$$(a_1,\ldots,a_n)\sigma_A \equiv_s (b_1,\ldots,b_n)\sigma_A. \qquad \Box$$

If A is a Σ-algebra and $\equiv$ is a Σ-congruence on A, let $A/\equiv$ be the S-indexed family of sets of equivalence classes, $A/\equiv\ = <A_s/\equiv_s \mid s\in S>$. As in Section 2, let $[a]_s$ (or just $[a]$) be the equivalence class of $s\in A_s$. We now make $A/\equiv$ into a Σ-algebra by defining the operations $\sigma_{A/\equiv}$.

(8.1.1) If $\sigma\in\Sigma_{s,\lambda}$, then $\sigma_{A/\equiv} = [\sigma_A]$.

(8.1.2) If $\sigma\in\Sigma_{s,s_1\ldots s_n}$ and $[a_i]\in(A/\equiv)_{s_i}$ then
$$([a_1],\ldots,[a_n])\sigma_{A/\equiv} = [(a_1,\ldots,a_n)\sigma_A].$$

Proposition 8.2. If A is a Σ-algebra and $\equiv$ is a Σ-congruence on A, then $A/\equiv$, as defined above, is a Σ-algebra, called the *quotient* of A by $\equiv$. $\Box$

Proposition 8.3. The canonical map $h:A\to A/\equiv$ which sends each a to [a] (for $a\in A_s$, $(a)h_s=[a]_s$) is a Σ-homomorphism. Furthermore, if $h:A\to B$ is a Σ-homomorphism and $\equiv_h$ is defined by:

$$a \equiv_h a' \text{ iff } (a)h = (a')h,$$

then $\equiv_h$ is a Σ-congruence and if h is surjective then $A/\equiv_h\ \cong B$. $\Box$

In order to define the free (Σ,E)-algebra we want the smallest congruence relation on $T_\Sigma(X)$ containing all substitution instances of E; for this the following result is key.

Proposition 8.4. Let A be any Σ-algebra. Then the congruence relations on A form a complete lattice. $\Box$

let $\Theta(A)$ be the lattice of congruence relations on A. The minimum element of $\Theta(A)$ is the identity relation (family of identity relations) on A and the maximum element is the universal relation, $<A,A\times A,A>$. Because the properties that make a relation a congruence relation are of a closure type (see 8.1), it follows that an arbitrary intersection of congruence relations is again a congruence relation. This makes $\Theta(A)$ a lattice. If R is any family of relations on A, then the *congruence generated by* R is the intersection of all congruences containing R. For example, the join of two congruences, $\equiv_1$, and $\equiv_2$ is the congruence generated by $\equiv_1 \cup \equiv_2$ because in general, the union of congruence relations will not be a congruence relation.

If $e = <e_1,e_2>$ is a Σ-equation, then a *substitution instance* of e (in $T_\Sigma(X)$) is the result of substituting expressions in $T_\Sigma(X)$ for the variables in e. This substitution is given by Theorem 6.10. For any assignment θ:var(e)$\rightarrow T_\Sigma(X)$, $<(e_1)\theta^{\#},(e_2)\theta^{\#}>$ is a substitution instance of e. We get all substitution instances of e as we let θ vary over all possible assignments to $T_\Sigma(X)$. Let $E(T_\Sigma(X))$ be the family of relations obtained in this way, i.e., $E(T_\Sigma(X))_s=\{(e)\theta^{\#} \mid e \in E_s$ and θ:var(e)$\rightarrow T_\Sigma(X)\}$. Define $\equiv_E$ to be the smallest congruence containing $E(T_\Sigma(X))$ which exists by Proposition 8.4. We define $T_{\Sigma,E}(X)$ to be $T_\Sigma(X)/\equiv_E$, the quotient of the free Σ-algebra by the congruence relation generated by substitution instances of E. Finally, let $I_X:X \rightarrow T_{\Sigma,E}(X)$ be the canonical map, $I_X:x \mapsto [x]$, taking each x to the congruence class of x relative to $\equiv_E$.

Theorem 8.5. $<I_X,T_{\Sigma,E}(X)>$ is the algebra freely generated by X in the category $\mathbf{Alg}_{\Sigma,E}$ of all Σ-algebras satisfying E: for any Σ-algebra A which satisfies E and for any set map $h:X \rightarrow A$, there exists a unique homomorphism $h^{\#}:T_{\Sigma,E}(X) \rightarrow A$ such that

$$\begin{tikzcd} X \arrow[r, "I_X"] \arrow[dr, "h"'] & T_{\Sigma,E}(X) \arrow[d, "h^{\#}"] \\ & A \end{tikzcd}$$

commutes in **Set**. □

Corollary 8.6. $T_{\Sigma,E} = T_{\Sigma,E}(\emptyset)$ is the initial algebra in $\mathbf{Alg}_{\Sigma,E}$. □

There are two ways to treat equational systems from a deductive point of view. The first is "reduction," which is the basis of most automatic algebraic simplifiers or theorem provers; the second is more like a "logistic system" where a "proof" consists of a sequence of pairs (equations) satisfying certain proof rules.

Let us assume that we have a fixed equational system E, so that we don't have to subscript everything we write with E. For reduction, we define a relation $\Rightarrow$ on $T_\Sigma(X)$, called "directly reduces to." Then $\Rightarrow$* is the reflexive transitive closure of $\Rightarrow$ and is called "reduces to." This is most simply described by going back to the fact that the carriers of the free algebra, $T_\Sigma(X)$, in Theorem 6.10 are actually sets of strings on $\Sigma \cup X \cup \{(,)\}$.

Definition 8.7. Let E be an equational system over $T_\Sigma(X)$. Let $t,t' \in T_\Sigma(X)_s$; then $t \Rightarrow_s t'$ iff there exists $u \in T_\Sigma(X)_s$ and $x \in X_{s'}$ with $u = u_1xu_2$ and there also exists a pair $<v,v'>$ which is a substitution instance of $(E \cup E^{-1})_{s'}$, and finally, $t=u_1vu_2$ and $u_1v'u_2 = t'$. If $t \Rightarrow_s t'$, we say t *directly reduces to* t' and where $\Rightarrow^*$ is the reflexive transitive closure of $\Rightarrow$, we say t *reduces to* t' when $t \Rightarrow^* t'$. □

The important thing about this deductive system is that t reduces to t' (by E) iff $t \equiv_E t'$.

Proposition 8.8. Let E be an equational system over $T_\Sigma(X)$, and let $\Rightarrow^*$ be as in Definition 8.7. Then for all $t,t' \in T_\Sigma(X)$, t reduces to t' by E iff t is congruent to t' modulo E, i.e., $\Rightarrow^* = \equiv_E$. □

Definition 8.9. Let E be an equational system over $T_\Sigma(X)$. An *equational proof* is a sequence of equations (pairs) from $T_\Sigma(X)$ with the property that if $e=<e_1,e_2>$ is the i^{th} member in the sequence then one of the following holds.

(8.9.1) e is a substitution instance of E.

(8.9.2) $e_1 = t = e_2$ for some $t \in T_\Sigma(X)$.

(8.9.3) $<e_2,e_1>$ appears earlier in the sequence.

(8.9.4) Both $<e_1,e_1'>$ and $<e_1',e_2>$ appear earlier in the sequence for some $e_1' \in T_\Sigma(X)$.

(8.9.5) There are pairs $<e_i',e_i''>$, $i = 1,\dots,n$ earlier in the sequence with e_i', e_i'' of sort s_i, and there is some $\sigma \in \Sigma_{s,s_1\dots s_n}$ with

$$e_1=\sigma(e_1'\dots e_n') \text{ and } \sigma(e_1''\dots e_n'')=e_2.$$

e is *equationally deducible* from E iff there exists an equational proof with e as it last line. □

Proposition 8.10. An equation $e = <e_1,e_2>$ is equationally deducible from E iff $e_1 \equiv_E e_2$. □

9. ABSTRACT DATA TYPES.

There has been a continuing active interest in both the practice and the theory of the specification of abstract data types. In fact, a number of recent papers in the bibliography begin with a variant of that same statement. The divergence of attitudes, approaches, and perceptions of the problem can well be illustrated by a perusal of the Proceedings of the SIGPLAN/SIGMOD Conference on Data: Abstraction, Definition and Structure, Salt Lake City, Utah, March, 1976.

The fundamental idea in abstract data type specification is to precisely present (describe, specify) a data type independent of any representation of the data objects and independent of any implementation of the operations of the type. The idea has, at least in part, grown from the suggestion of Hoare (1972):

> In the development of programs by stepwise refinement, the programmer is encouraged to postpone the decision on representation of his data until after he has designed his algorithm and has expressed it as an 'abstract' program operating on 'abstract' data.

The literature is replete with what can be viewed as responses to this suggestion.

Our work has followed the same lines as Guttag, Horowitz, Liskov, Musser and Zilles in that we view data types as many-sorted algebras. At least since F. L. Morris (1973), it seems to be generally accepted that types are not just sets, but sets equipped with operations, and that is *exactly* what many-sorted algebras are. Many other authors have the same view but are not exploiting the mathematics of universal algebra.

As we approach it, what is "abstract" about an abstract data type is that it consists of an isomorphism class of algebras rather than any concrete representative of the class. When it comes to specifying an abstract data type one can display a particular algebra and define the abstract data type as the isomorphism class of that algebra. The proposed alternative is to characterize the isomorphism class using axioms written in terms of the operations of the types.

Spitzen and Wegbreit (1975) list three properties inherent in describing the type by axioms.

> First, they are declarative and hence avoid programming details and language dependencies. Second, they are intuitively reasonable descriptions of the behavior of various structures. Finally, they are sufficiently rigorous to permit a proof that a particular realization of the data structures is faithful to the specifications.

And Guttag (1977) adds:

> They are easy to read and comprehend, thus facilitating informal verification of the fact that they do indeed conform to the intent of their creator.

There is also the advantage of potential automatic (though perhaps inefficient) implementation of the type directly from the specification as in Goguen and Tardo (1977). Perhaps summarizing all these points, the language for writing algebraic specifications is simple, fixed, and algebraic as opposed to the open-ended ways one can go about describing a particular algebra. (Brand (1978) raises serious objections to some of these points.)

Our approach is distinguished from others' by the fact that we actually want to live in both worlds; we believe there has to be a standard of *correctness* of a specification, i.e., a criterion for *formally* verifying that we have specified what we want. When we begin to write down a specification we have in mind some particular algebra and that specification is correct iff the specified algebra is isomorphic to the intended algebra.

We use the concept of initial algebras as the key to providing both the syntax and semantics for specifications. We were originally attracted to this approach because it offered a comparatively simple way to provide absolutely rigorous semantics for data type specifications, and it worked very nicely for simple examples; we were able to specify such data types as ***int***, ***bool***, ***string***, etc. in a very natural way (see ADJ (1976a)).

As we continued working in this area it became evident that things were not as simple as they had looked at first.

The need to handle "exceptional states" and/or "errors" (e.g., the result of "reading" an empty stack) led to very complex specifications (see ADJ (1976a) for examples.) An approach using "error algebras" is proposed in ADJ (1977a). In studying the form of the equational specifications we realized that one viable course would be to allow conditional axioms in our specifications. The resulting approach was sketched in ADJ (1976). However, unanswered in that paper was the question of whether these new conditional specifications were in any sense more powerful than our earlier equational specifications. In ADJ (1978) we answer that question in the affirmative. However the answer opens the question of whether or not conditional specifications are too powerful in either the sense that they permit us to specify undesirable objects, or that, with such power, they are necessarily intractable from an applications point of view (for automatic implementations, for example).

In many cases it appeared to be necessary to include "hidden" or auxiliary operations in a specification in order to keep the specification finite. While we were working on this problem, Majster (1977, 1977a) announced a number of examples which she claimed could not be specified with a finite number of equations. Her examples have the merit that they are plausible (reasonable) data types, but, despite the fact that our intuition also tells us that these data types are not finitely specifiable, we do not find her proofs to be totally convincing. The problem is that there are just too many cases that have to be ruled out. In ADJ (1978) we and described the (unrealistic) data type ***toy-stack***, which is not finitely presentable but when equipped with an additional operation, is finitely (equationally) axiomatizable. This proves that "hidden functions" are necessary in some situations and provides a mathematical basis for the "hidden function question." (C.f. Liskov and Berzins (1977), p. 19, and the discussion of the operation "derive" in Burstall and Goguen (1977)).

A principal shortcoming of our earlier development was an inadequate treatment of "parameterized types". That is, it is clear intuitively that the two data types ***finite-sets-of-integers*** and ***finite-sets-of-characters*** are intimately related and are, indeed just different instances of the "parameterized type", ***finite-sets-of-( )***. The problem was to give a mathematically precise characterization of the kind of object ***finite-sets-of-( )*** is, and to show how to specify such objects. An answer is given in ADJ (1978).

We begin a summary of the techniques and ideas of ADJ (1975,1976,1976a,1978) with another definition.

Definition 9.1. Let Σ be a signature. A Σ-algebra is *minimal* iff it has no proper subalgebras. □

Definition 9.2. An *abstract data type* is the isomorphism class of a minimal Σ-algebra. □

In ADJ (1976,1976a), an abstract data type was defined to be the isomorphism class of an initial algebra in a category of Σ-algebras where the latter was interpreted to be a full subcategory of $\mathbf{Alg}_\Sigma$. The definition here is equivalent, because if A is a minimal Σ-algebra and E is the set of all pairs $<e_1,e_2>$ from T_Σ such that $(e_1)h_A = (e_2)h_A$,where h_A is the unique homomorphism from T_Σ to A, then A is initial in $\mathbf{Alg}_{\Sigma,E}$.

Definition 9.3. A *specification* of a data type is a pair $<\Sigma,E>$ consisting of a signature Σ and a set E of Σ-equations. The specification $<\Sigma,E>$ is *correct with respect to a* Σ-algebra A if and only if $T_{\Sigma,E}$ is isomorphic to A, or equivalently (Proposition 3.11.) A is initial in $\mathbf{Alg}_{\Sigma,E}$ □

Example 9.1. Let Ω be the $\{int,Bool\}$-sorted subsignature of Σ of Example 7.1 consisting of 0,Pr,Su,tt,ff,¬,and ∧ of the appropriate types. Let E be the following set of equations.

(A1) Pr(Su(x))=x

(A2) Su(Pr(x))=x

(A3) ¬(tt)=ff

(A4) ¬(ff)=tt

(A5) tt∧x=x

(A6) ff∧x=ff

Let A be the two-sorted algebra consisting of the integers and Boolean values with zero, successor, predecessor, truth value constants, 'not' and 'and'. Then the specification $<\Omega,E>$ is correct with respect to this algebra A. □

10. ORDERED ALGEBRAS.

Definition 10.1. Let Σ be an S-sorted signature. A (*strict*) *ordered* Σ*-algebra* A is a Σ-algebra in which each carrier A_s is equipped with a partial order $\sqsubseteq_s$ with minimum element $\perp_s$. The operations σ_A of A are monotonic:

$$a_i \sqsubseteq_{s_i} a'_i \text{ implies } (a_1,\dots,a_n)\sigma_A \sqsubseteq (a'_1,\dots,a'_n)\sigma_A. \qquad \square$$

Definition 10.2. A *homomorphism* of ordered Σ-algebras is a Σ-homomorphism which is monotonic and strict (preserving $\perp$). □

Let $\mathbf{Palg}_\Sigma$ be the category of ordered Σ-algebras with their homomorphisms.

Let X be a family of sets (generators) and write $X\cup\perp$ for the S-indexed family $<X_s\cup\{\perp_s\} \mid s\in S>$. Define $PT_\Sigma(X)$ to be the ordered Σ-algebra with carrier $T_\Sigma(X\cup\perp)$ ordered by the smallest partial order relation on $T_\Sigma(X\cup\perp)$ satisfying the following conditions.

(10.3.1) $\perp_s \sqsubseteq_s t$ for all $t \in PT_\Sigma(X)_s$

(10.3.2) Whenever $\sigma\in\Sigma_{s,s_1\dots s_n}$ and $u_i\sqsubseteq_{s_i}v_i$ for $i \in [n]$ and $u_i,v_i\in PT_\Sigma(X)_{s_i}$,

$$\sigma(u_1\dots u_n) \sqsubseteq_s \sigma(v_1\dots v_n).$$

Nivat (1973) shows (for the one-sorted case) that this ordering can be characterized in terms of the strings that make up $PT_\Sigma(X)$:

Proposition 10.4. For any t, t′ in $PT_\Sigma(X)$, $t \sqsubseteq t'$ iff t′ can be obtained from t by replacing zero or more occurrences of $\perp_s$ in t by strings of sort s, i.e., by elements of $PT_\Sigma(X)_s$. □

If B is an ordered Σ-algebra and $h:X\to B$, is a family of maps to the carrier of B, then $h_\perp:(X\cup\perp)\to B$ is also a family of (strict) maps taking $\perp_s$ to $\perp_s$ in B. Proposition 6.10 guarantees a unique Σ-algebra homomorphism $h^\#:T_\Sigma(X\cup\perp)\to B$ and it is easy to check that this Σ-homomorphism is a homomorphism of ordered algebras, i.e., that the order relation on $PT_\Sigma(X)$ is preserved. This outlines the proof of

Proposition 10.5. Let $I_X:X\to PT_\Sigma(X)$ be the family of maps taking the generators to the carrier of $PT_\Sigma(X)$. Then $<PT_\Sigma(X),I_X>$ is the free ordered Σ-algebra, i.e. it is free in the category **Palg** or ordered Σ-algebras: for any ordered Σ-algebra A and for any family of maps

$h:X \rightarrow B$, there exists a unique ordered Σ-algebra homomorphism $h^{\#}:PT_{\Sigma}(X) \rightarrow B$ such that $I_X h = h^{\#}$.

$$\begin{tikzcd}
X \arrow[r, "I_X"] \arrow[dr, "h"'] & PT_{\Sigma}(X) \arrow[d, "h^{\#}"] \\
 & A
\end{tikzcd}$$

□

Corollary 10.6. $PT_{\Sigma}(\emptyset) = PT_{\Sigma}$ is the initial algebra in the category of ordered Σ-algebras; for any ordered Σ-algebra A, there is a unique homomorphism of ordered Σ-algebras, $h_A:PT_{\Sigma} \rightarrow A$. □

Again we describe subcategories of $\mathbf{Palg}_{\Sigma}$, but this time we use "inequations." These are just pairs $e = <e_1,e_2>$ in $PT_{\Sigma}(X)$ and for such pairs we write $e_1 \sqsubseteq e_2$ because of the way we shall interpret these pairs. An ordered Σ-algebra, A, *satisfies* e iff for every assignment θ:var(e)$\rightarrow$A,

$$(e_1)\theta^{\#} \sqsubseteq (e_2)\theta^{\#}.$$

Again, when A satisfies e, we can also say $e_1 \sqsubseteq e_2$ is valid in A or that A is a model for e. An inequational system E is a set of pairs from $PT_{\Sigma}(X)$, and A *satisfies* E just in case A satisfies every $e \in E$. $\mathbf{Palg}_{\Sigma,E}$ is the category of ordered Σ-algebras satisfying E with their homomorphisms.

The free algebra in $\mathbf{Palg}_{\Sigma,E}$ is obtained in a manner analogous to that of Section 6, but there are slight complications.

Definition 10.7. A preorder $\underset{\sim}{\sqsubset}$ on a Σ-algebra A is said to be *admissible* iff for every $\sigma \in \Sigma_{s,s_1 \ldots s_n}$ and $a_i,b_i \in A_{s_i}$, if $a_i \underset{\sim}{\sqsubset} b_i$ for $i \in [n]$ then,

$$(a_1,\ldots,a_n)\sigma_A \underset{\sim}{\sqsubset} (b_1,\ldots,b_n)\sigma_A.$$

□

This generalizes Definition 8.1 of Section 6; we might have defined a congruence relation to be an "admissible equivalence relation," or an equivalence relation with the "substitution property." Likewise, Definition 10.7 gives us a preorder with the substitution property.

Recall from Section 2 that any preorder, $\underset{\sim}{\sqsubset}$, determines an equivalence relation $\sim$ given by:

$$a \sim b \text{ iff } a \underset{\sim}{\sqsubset} b \text{ and } b \underset{\sim}{\sqsubset} a.$$

Admissibility of $\underset{\sim}{\sqsubset}$ makes $\sim$ a congruence:

Proposition 10.8. If $\underset{\sim}{\sqsubset}$ is an admissible preorder on a Σ-algebra A, then the equivalence relation $\sim$ determined by $\underset{\sim}{\sqsubset}$ is a congruence relation. □

The steps for getting the free Σ-algebra satisfying an inequational system E now proceed just as they did in Section 6

Let A be a Σ-algebra and $\underset{\sim}{\sqsubset}$ an admissible preorder on A. Further, let $A/\sim$ be the quotient of A by the congruence (Proposition 10.8) $\sim$ determined by $\underset{\sim}{\sqsubset}$. By Proposition 8.2 $A/\sim$ is a Σ-algebra; we need the order relation on $A/\sim$. Define that partial order (see Section 2) $\sqsubseteq$ on $A/\sim$ by

$$[a] \sqsubseteq [b] \text{ iff } a \underset{\sim}{\sqsubset} b.$$

Proposition 10.9. If A is an ordered Σ-algebra and $\underset{\sim}{\sqsubset}$ is an admissible preorder then $A/\underset{\sim}{\sqsubset}$, which is the Σ-algebra $A/\sim$ with the ordering as defined above, is an ordered Σ-algebra, called the *quotient* of A by $\underset{\sim}{\sqsubset}$. □

Proposition 10.10. If A is a Σ-algebra and $\underset{\sim}{\sqsubset}$ is an admissible preorder on A, then the canonical map $h:A \rightarrow A/\underset{\sim}{\sqsubset}$ which sends each a to [a], its $\sim$-equivalence class, is a homomorphism of ordered Σ-algebras. Further, if $h:A \rightarrow B$ is an ordered Σ-algebra homomorphism and $\underset{\sim}{\sqsubset}_h$ is defined by

$$a \underset{\sim}{\sqsubset} a' \text{iff } (a)h \sqsubseteq_A (a')h,$$

is an admissible preorder and if h is surjective, then $A/\underset{\sim}{\sqsubset}_h \cong B$.□

Just as the congruences on a Σ-algebra form a complete lattice, (Proposition 8.4) so also the admissible preorders an an ordered Σ-algebra form a complete lattice.

Proposition 10.11. Let A be any ordered Σ-algebra. The admissible preorders of A form a complete lattice which we denote $\Theta(A)$, just as in the unordered case. □

Let E be an inequational system, i.e., a family of pairs from $PT_\Sigma(X)$, and let $E(PT_\Sigma(X))$ be the family of "substitution instances" of E;

$$E(PT_\Sigma(X))_s = \{<(e_1)\theta^{\#},(e_2)\theta^{\#}> \mid <e_1,e_2> \in E_s\}.$$

Then we can define $\underset{\sim}{\sqsubset}_E$ to be the least admissible preorder on $PT_\Sigma(X)$ containing $E(PT_\Sigma(X))$, i.e., the intersection in $\Theta(PT_\Sigma(X))$ of all admissible preorders containing $E(PT_\Sigma(X))$. Just as in the unordered case, define $PT_{\Sigma,E}(X)$ to be $PT_\Sigma(X)/\underset{\sim}{\sqsubset}_E$ with its canonical map $I_X:X \rightarrow PT_{\Sigma,E}$ sending each generator x to its equivalence class [x] determined by $\sim_E$.

Theorem 10.12. $<I_X,PT_{\Sigma,E}(X)>$ is the algebra freely generated by X in the category $\mathbf{Palg}_{\Sigma,E}$ of all ordered Σ-algebras satisfying E: for any ordered Σ-algebra A which satisfies E and for any family of maps $h:X \rightarrow A$, there exists a unique homomorphism of ordered Σ-algebras, $h^{\#}:PT_{\Sigma,E} \rightarrow A$ such that

$$\begin{tikzcd}
X \arrow[r, "I_X"] \arrow[dr, "h"'] & PT_{\Sigma,E}(X) \arrow[d, "h^{\#}"] \\
 & A
\end{tikzcd}$$

commutes in $\mathbf{Set}^S$. □

Corollary 10.13. $PT_{\Sigma,E} = PT_{\Sigma,E}(\emptyset)$ is initial in $\mathbf{Palg}_{\Sigma,E}$. □

At this point we introduce "partial trees"; they provide a particularly useful representation of the free continuous algebra to be described in Section 11. But they also are useful in visualizing both the free Σ-algebra and the free ordered Σ-algebra.

Definition 10.14. Let Σ be an S-sorted signature and X a family of generators or variables. Then an X-*ary* (*normalized, singly rooted, ordered, partial*) Σ-*tree of sort* s is a partial function $t:[\omega]^* \to \Sigma \cup X$ whose graph satisfies the following conditions.

(10.14.1) If $<\lambda,\xi> \in t$ then $\xi \in (\Sigma \cup X)_s$.

(10.14.2) For all $u \in [\omega]^*$ and $k \in [\omega]$, if $<uk,\xi> \in t$ for $\xi \in (\Sigma \cup X)_s$, then $<u,\eta> \in t$ for some $<w,s'>$ with $w_k=s$ and $\eta \in \Sigma_{w,s'}$. □

The first condition says that a non-empty tree of sort s must have its root labeled with a symbol of sort s. The second (and quite powerful) condition implies that the domain of definition of t be prefix closed, that is, if t is defined at uv, then t is defined at u. In addition, (10.14.2) forces the the successor labels of a node to be consistent with the arity of the operation symbol at that node. For example, if σ is of type <a,abc> and $<u,\sigma> \in t$ then $<ui,\xi> \in t$ implies i = 1,2, or 3 (nothing greater) and ξ must be of sort a, b, or c respectively.

Let $Tr_\Sigma(X)$ be the S-indexed family of X-ary Σ-trees. We give $Tr_\Sigma(X)$ an algebraic structure by defining the operations, σ_{Tr}:

$$(10.14.3) \qquad (t_1,\ldots,t_n)\sigma_{Tr} = \{<\lambda,\sigma>\} \cup \bigcup_{i\in[k]}\{<iu,\xi> \mid <u,\xi> \in t\}.$$

The order relation on $Tr_\Sigma(X)$ is just set inclusion of the graphs of the partial functions that make up $Tr_\Sigma(X)$. It is an easy matter to check that the operations defined by (10.14.3) are monotonic. This gives

Proposition 10.15. For any signature Σ and indexed family of generators, X, $Tr_\Sigma(X)$ is an ordered Σ-algebra. □

More importantly, we shall see in Section 11 that $Tr_\Sigma(X)$, equipped with the injection $J_X:X \to Tr_\Sigma(X)$ which sends x to the tree $\{<\lambda,x>\}$, is the free continuous algebra generated by

X. Subalgebras of $Tr_\Sigma(X)$ will be of interest to us now. We define only the carriers of those algebras; one must verify that those carriers are closed under the operations given by (10.14.3).

(10.16.1) $TTr_\Sigma(X)$ is the set of all Σ-trees which are *total*, i.e., for all $<u,\xi>\in t$, if $\xi\in\Sigma_{s,s_1\ldots s_n}$, then there are $\eta_i\in(\Sigma\cup X)_{s_i}$ for $i=1,\ldots,n$, with $<ui,\eta_i>\in t$.

(10.16.2) $FTr_\Sigma(X)$ is the set of finite Σ-trees, those with their domain of definition, $\{w \mid <w,\xi>\in t$ for some $\xi\in\Sigma\cup X\}$, being finite.

For the next subalgebra of $Tr_\Sigma(X)$ we need the auxiliary notion of "subtree." For every $v\in[\omega]^*$ and Σ-tree, t, define t_v to be the Σ-tree consisting of all pairs $<u,\xi>$ such that $<vu,\xi>\in t$. Then t is a *subtree* of t' iff $t=t'_v$ for some $v\in[\omega]^*$. Sub(t) is the set of all subtrees of t,

$$Sub(t) = \{t_v \mid v\in[\omega]^*\}.$$

For finite trees and for trees with a "regular" pattern, Sub(t) is a finite set. This is how the next subalgebra is defined.

(10.16.3) $RTr_\Sigma(X)$ is the subset of Σ-trees t for which Sub(t) is finite. These are called the *rational* trees.

(10.16.4) $FTTr_\Sigma(X)$ is the set of finite total Σ-trees.

$$FTTr_\Sigma(X) = FTr_\Sigma(X)\cap TTr_\Sigma(X).$$

(10.16.5) $RTTr_\Sigma(X)$ is the set of rational total Σ-trees.

$$RTTr_\Sigma(X) = RTr_\Sigma(X)\cap TTr_\Sigma(X).$$

Proposition 10.17. $<FTTr_\Sigma(X),J_X>$ is the Σ-algebra freely generated by X; thus $<FTTr_\Sigma(X),J_X>$ and $<T_\Sigma(X),I_X>$ are (uniquely) isomorphic. □

Proposition 10.17 is a precise statement of the interchangeability of finite total trees and finite expressions. We get the corresponding result for ordered algebras.

Proposition 10.18. $<FTr_\Sigma(X),J_X>$ is the ordered Σ algebra freely generated X so that $<FTr_\Sigma(X),J_X>$ and $<PT_\Sigma(X),I_X>$ are uniquely isomorphic. □

11. CONTINUOUS ALGEBRAS.

Definition 11.1. Let Σ be an S-sorted signature. A *Z-continuous Σ-algebra* is a strict ordered Σ-algebra A, each carrier A_s of which is Z-complete and each operation of which is Z-continuous. A *homomorphism* of Z-continuous algebras is a homomorphism of ordered

Σ-algebras which is Z-continuous. Thus $h:A \to B$ must be a Σ-homomorphism which is strict, monotonic and Z-continuous. □

Let $\mathbf{ZAlg}_{\Sigma}$ be the category of Z-continuous Σ-algebras, with their homomorphisms. We are, for the most part, interested in $\Delta\mathbf{Alg}_{\Sigma}$ and $\mathbf{PCAlg}_{\Sigma}$ although other classes of algebras may turn out to be interesting.

In Section 10, we defined the algebra $Tr_{\Sigma}(X)$ of finite and infinite partial X-ary trees. This algebra plays the role of the algebra of expressions for the continuous case.

Theorem 11.2. The carriers of the algebra $Tr_{\Sigma}(X)$ are PC-complete and the operations are $\bigsqcup$-continuous. Thus $Tr_{\Sigma}(X)$ is a PC-continuous algebra. In fact, $<Tr_{\Sigma}(X),J_X>$ is the PC-continuous algebra freely generated by X. (Recall, J_X sends x to the finite (total) tree $\{<\lambda,x>\}$.) □

The Proof of Theorem 11.2 uses the fact that every tree in $Tr_{\Sigma}(X)$ is the least upper bound of an ω-chain of finite trees (approximations; say those of depth $n = 0,1,2,...$) and it therefore follows that $Tr_{\Sigma}(X)$ is the Σ-algebra freely generated by X in the category $\mathbf{ZAlg}_{\Sigma}$ of Z-continuous algebras for any subset system Z with $\omega \subseteq Z$ and $Z \subseteq PC$.

Again we will describe subcategories of $\mathbf{ZAlg}_{\Sigma}$ using inequalities as in Section 10, but things don't work quite as smoothly as they did for algebras or even ordered algebras. This can be indicated by a simple example. Consider a one-sorted signature Σ with $\Sigma_0=\{a\}$ and $\Sigma_1=\{f\}$. $Tr_{\Sigma}(X)$ has but one infinite tree which is, in effect, an ω-sequence of f's; the rest of $Tr_{\Sigma}(X)$ is $FTr_{\Sigma}(X)$ or its isomorphic copy $PT_{\Sigma}(X)$ if one wants to think of it that way. That one infinite tree is the least upper bound of the ω-chain of $f^n(\perp)$. Now if we impose the single inequation, $x \sqsubseteq f(x)$, we obtain new ω-chains, $f^n(a)$ and for each generator, $f^n(x)$. No simple quotient (equivalence classes, etc.) of $Tr_{\Sigma}(X)$ can give us a complete (say ω-complete) algebra because we need to have limits of all these new chains which do not exist in $Tr_{\Sigma}(X)$. In order to get analogs to Theorems 8.5 and 10.12 we must use the completion process described in Section 5 as was first realized by Courcelle and Nivat (1976) and independently by Bloom (1976).

12. ALGEBRAIC THEORIES.

Algebraic theories (Lawvere (1963)) are treated in ADJ (1976a, 1976b, 1977b, 1978a). There are many ways to formulate the concept of algebraic theory; a functorial approach similar to Lawvere's is found in ADJ (1975b). Like Ginali (1975), Elgot (1973) and Bloom and Elgot (1974), we employ an "equational" description because we believe that formulation to be more perspicuous and workable. Many of our earlier papers employed one-sorted (or conventional) algebraic theories. The definition of many-sorted (or S-sorted) algebraic theory to follow also appears in ADJ (1978a, 1978b).

Definition 12.1. Let S be a nonempty set (of *sorts*). An S-*sorted algebraic theory* consists of the following data.

(12.1.1) A family of sets, $\mathbf{T}(u,v)$, indexed by pairs of strings $u,v\in S^*$; the elements of $\mathbf{T}(u,v)$ are called *morphisms from* u *to* v. We write $\alpha:u\to v$ to mean $\alpha\in\mathbf{T}(u,v)$.

(12.1.2) An associative *composition* operation $\circ:\mathbf{T}(u,v)\times\mathbf{T}(v,w)$ defined for all u,v,w in S^*. That composition has two-sided identities, $1_v:v\to v$, for all $v\in S^*$, such that for all $\alpha:u\to v$,

$$1_u\circ\alpha = \alpha = \alpha\circ 1_v$$

(12.1.3) For each $n\in\omega$, $u\in S^n$ and $i\in[n]$, a *distinguished morphism* or *injection*, $x_i^u:u_i\to u$.

(12.1.4) For each $n\in\omega$, $u\in S^n$ and $v\in S^*$, an operation,

$$(\ ,\dots,\ )_{u,v}:\Pi_{i\in[n]}\mathbf{T}(u_i,v)\to\mathbf{T}(u,v)$$

which is called *source tupling.*

The case n=0 in (12.1.4) gives a unique morphism from λ to v for all $v\in S^*$, denoted $0_v:\lambda\to v$. We immediately drop the subscripts from the tupling operation as they can always be retrieved from context. The injections and source tupling operations are required to satisfy the following "coproduct conditions."

(12.1.5a) For all $n\in\omega$, $u\in S^n$, $v\in S^*$ and all families, $<\beta_i:u_i\to v\ \mid\ i\in[n]>$

$$x_i^u\circ(\beta_1,\dots,\beta_n) = \beta_i.$$

(12.1.5b) For all $n\in\omega$, $u\in S^n$, $v\in S^*$ and $\beta:u\to v$.

$$(x_1^u\circ\beta,\dots,x_n^u\circ\beta) = \beta.$$

For $\alpha\in\mathbf{T}(u,v)$, we say that α has *type* $<u,v>$, *source* u, *arity* or *target* v, and *rank* $|v|$; if u has length 1, then we say that α has *sort* $u\in S$. □

Comparing with the definition of category in Section 3, we see from (12.1.1,2), that an S-sorted algebraic theory is a category with objects S^*, but, as will be amplified below, Definition 12.1 is presenting an algebraic theory as a many $(S^*\times S^*)$ sorted algebra.

In the special case where S is a singleton, we refer to theories as one-sorted, rather than S-sorted, do not distinguish between different singleton sets, and thus identify S^* with the set $\omega = \{0,1,\dots\}$ of natural numbers. Morphisms then go $\alpha:n\to p$. We now give some examples of algebraic theories.

Example 12.1. (The theory $\mathbf{Sum}_A$, for a set A.) Let A be a set and for each $n,p\in\omega$ define $\mathbf{Sum}_A(n,p)$ to be the set of all *partial* functions from $A\times[n]$ to $A\times[p]$. Composition is function composition. Thus $\mathbf{Sum}_A$ is, in effect, the full subcategory of **Pfn** determined by the objects

$A\times[n]$ for $n\in\omega$. It is "in effect", because the objects of the full subcategory are actually the sets $A\times[n]$ whereas the objects of $\mathbf{Sum}_A$ are the corresponding natural numbers. This is a typical situation with algebraic theories. For each $n\in\omega$ and $i\in[n]$, the distinguished morphism $x_i^n:1\rightarrow n$ is the (total) function from $A\times[1]$ to $A\times[n]$ sending $<a,1>$ to $<a,i>$. Tupling of n functions, $f_i:1\rightarrow p$, gives $(f_1,...,f_n):n\rightarrow p$ defined by

$$<a,i>(f_1,...,f_n) = <a,1>f_i.$$

In $\mathbf{Sum}_A$ the object n corresponds to an n-fold disjoint union (or coproduct) of A with itself; the i^{th}copy is the set of $<i,a>$ for $a\in A$. Identifying A with $A\times[1]$, the i^{th} distinguished morphism sends A to the i^{th} copy of A in $A\times[n]$ and tupling of n functions defined on A (actually on $A\times[1]$) results in the i^{th} function working on its (the i^{th}) copy of A. The theory $\mathbf{Sum}_A$ is an extremely important algebraic theory for algebraic semantics; its formulation and importance being recognized in Elgot (1973); Elgot used the notation [A] for $\mathbf{Sum}_A$. □

Example 12.2. (The theory $\mathbf{Sum}_A$, for an S-indexed family, A.) The many-sorted version of $\mathbf{Sum}_A$ follows the comments in Section 2 on the way we can treat S-indexed families in very much the same way as we treat sets. Let S be the set of sorts and let $A = <A_s \mid s\in S>$ be an S-indexed family of sets. Then for $w\in S^*$, define $A\times[w]$ to be $\{<a,i> \mid a\in A_{w_i}\}$. $A\times[w]$ is the disjoint union (or coproduct), $A_{w_1}+...+A_{w_n}$, where $n=|w|$. Now we exactly repeat the definition of Example 12.1. $\mathbf{Sum}_A(v,w)$ consists of all *partial* functions $f:A\times[v]\rightarrow A\times[w]$. The distinguished morphism $x_i^w:u_i\rightarrow u$ is the injection of A_i in $A\times[w]$, sending $<a,1>$ to $<a,i>$. Tupling works just as above: $<a,i>(f_1,...,f_n) = <a,1>f_i$, where $i\in[n]$ and $f_i:A\times[w_i]\rightarrow A\times[v]$.

An equivalent formulation for the theory $\mathbf{Sym}_A$ is obtained using matrices. (See Elgot (1973, 1974) for, again, the one-sorted case.) Take for $\mathbf{Sym}_A(u,v)$ all "determinate" $|u|\times|v|$ matrices of partial functions where the (i,j)-th element is from A_{u_i} to A_{v_i}. Such a matrix is *determinate* if the domains of definition are disjoint along each row. Composition is the expected matrix multiplication (taking the join of the composites which exists because of determinateness) and tupling makes a matrix out of rows. □

Example 12.3. (The theory $\mathbf{Pow}_A$, for a set A.) Returning back to the one-sorted case for a preliminary version of this example, let A be a set and define $\mathbf{Pow}_A(n,p)$ to be the set of all *total* functions, $f:A^p\rightarrow A^n$ (note the reversal of the order of n and p). Composition in $\mathbf{Pow}_A$ is reversed function composition, i.e., given $f:n\rightarrow p$ and $g:p\rightarrow q$, then $f\circ g:n\rightarrow q$ in $\mathbf{Pow}_A$ is $gf:A^q\rightarrow A^n$. For each $i\in[n]$, the distinguished morphism $x_1^n:1\rightarrow n$ is the *projection* π_i from A^n to A sending $<a_1,...,a_n>$ to a_i. Tupling takes n morphisms, $f_i:1\rightarrow p$ $(f_i:A^p\rightarrow A)$, and produces (what is actually called the target tuple when working in **Set**) $(f_1,...,f_n):A^p\rightarrow A^n$ defined by $(<a_1,...,a_n>)(f_1,...,f_n) = <(a_1)f_1,...,(a_n)f_n> \in A^n$. □

Example 12.4. (The theory $\mathbf{Pow}_A$ for an S-indexed family of sets, A.) Let S be a set (of sorts) and A = $<A_s \mid s \in S>$ and S-indexed family of sets. Recall from Section 2, for $w \in S^*$, $A^w = A_{w_1} \times \ldots \times A_{w_n}$. Then $\mathbf{Pow}_A(v,w)$ is the set of all functions from A^w to A^v. $x_i^u : u_i \to u$ is the projection again, i.e., the function $A^u \to A_{u_i}$ sending $<a_1,\ldots,a_n>$ to a_i Tupling is just as in the previous example. □

Proposition 12.2. For all $n \in \omega$, $u \in S^n$, $v \in S^*$, $<\beta_i : u_i \to v>$, and $\gamma : u \to v$,

$$\text{if } x_i^u \circ \gamma = \beta_i \text{ for } i \in [n], \text{ then } \gamma = (\beta_1,\ldots,\beta_n) \qquad \square$$

Proposition 12.3. For each $n \in \omega$ and $u \in S^n$,

$$(x_1^u,\ldots,x_n^u) = 1_n \qquad \square$$

Proposition 12.2 says that a morphism in an algebraic theory is uniquely determined by its components, $x_i^u \circ \gamma$. Proposition 12.3 expresses (uniquely) the identity for u as a tuple of distinguished morphisms. We will call any tuple of distinguished morphisms a *map*. The reason for this is that any such tuple, say,

$$(x_{i_1}^v,\ldots,x_{i_n}^v) : u \to v$$

corresponds to a function $f:[\,|u|\,] \to [\,|v|\,]$ given by $(j)f = i_j$ and note that $(j)fu = (j)v$. Conversely, given any pair of strings, $u \in S^n$ and $v \in S^m$ and any function, $f:[n] \to [m]$ such that $fv = u$,

$$\begin{tikzcd}
{[n]} \arrow[d, "f"'] \arrow[dr, "u"] & \\
{[m]} \arrow[r, "v"'] & S
\end{tikzcd}$$

there is a corresponding tuple of distinguished morphisms $\hat{f}:u \to v$ where $\hat{f}=(x_{1f}^v,\ldots,x_{nf}^v):u \to v$. $\hat{f}$ can be thought as reflecting the corresponding substitution of variables; composition on the right with $\hat{f}$ results in simultaneously substituting x_{if} for x_i and by the restraints we have, both of these variables are of sort $u_i = v_{(i)f}$. $\hat{\ }$ is a mapping from Str_S (see Section 2) to $\mathbb{T}$ which is a functor (preserving identities by Proposition 12.3; preserving composition is easily checked). When $\hat{\ }$ is an injection, $\mathbb{T}$ is said to be *non-degenerate*; Str_S is called the *base category* for S-sorted algebraic theories. There are two degenerate theories that satisfy the conditions of Definition 12.1, one has a single morphism from n to p for every n and p and the other has a single morphism from n to p for every $n \geq 0$ and every $p > 0$. This corresponds to the situation for many-sorted algebras, that any system of equations has a model, or is satisfied by an algebra, which has at most one element in each carrier.

This is an appropriate place to observe that an S-sorted algebraic theory is, in fact, an $(S^* \times S^*)$-sorted algebra with (infinite) signature given by the following.

$$\Sigma_{<u,w>,<u,v><v,w>} = \{\circ_{u,v,w}\}$$
$$\Sigma_{<u,u>,\lambda} = 1_u$$
$$\Sigma_{<u,u_i>,\lambda} = \{x_i^u\}, \quad \text{for } i \in [\,|u|\,]$$
$$\Sigma_{<u,v>,<u_1,u><u_2,u>\ldots<u_n,u>} = \{(\ ,\ldots,\ )_{u,v}\}$$
$$\Sigma_{X,Y} = \emptyset \text{ for all other } X \in S^* \times S^* \text{ and } Y \in (S^* \times S^*)^*.$$

The following is the infinite set of axioms which give us the equational class of S-sorted algebraic theories (as $(S^* \times S^*)$-sorted algebras). For completeness, we give those equations in their full regalia of subscripted operations; we hope you never have to see them that way again! In effect, they already appear in unsubscripted form in Definition 12.1.

(12.4.1) For all $\alpha{:}u \twoheadrightarrow v$, $\beta{:}v \twoheadrightarrow w$, and $\gamma{:}w \twoheadrightarrow z$,

$$(\alpha \circ_{u,v,w} \beta) \circ_{u,w,z} \gamma = \alpha \circ_{u,v,z} (\beta \circ_{v,w,z} \gamma)$$

(12.4.2) For all $\alpha{:}u \twoheadrightarrow v$,

$$(1_u \circ_{u,u,v} \alpha) = \alpha = (\alpha \circ_{u,v,v} 1_v)$$

(12.4.3) For all $u,v \in S^*$, $i \in [\,|u|\,]$ and $<\beta_i{:}u_i \twoheadrightarrow v>$,

$$x_i^u \circ_{u_i,u,v} (\beta_1,\ldots,\beta_n)_{u,v} = \beta_i$$

(12.4.4) For all $u,v \in S^*$ and $\beta{:}u \twoheadrightarrow v$,

$$(x_1^u \circ_{u_1,u,v} \beta,\ldots,x_n^u \circ_{u_n,u,v} \beta)_{u,v} = \beta$$

This is certainly a long-winded and notationally cumbersome (if not embarrassing) way to write out the equations of Definition 12.1, but it is an important exercise for it means that results on many-sorted algebras carry over to many-sorted algebraic theories.

In particular we know what homomorphisms (morphisms) of S-sorted theories are and we use the notation $\mathbf{Th}_S$ for the category of S-sorted theories. This is in fact the category $\mathbf{Alg}_{\Sigma,E}$ for the $(S^* \times S^*)$-sorted signature listed above with E, the axioms listed in 12.4.1 through 12.4.4. We also want to make it clear that the material on S-sorted algebras also tells us what subtheories are and how to define equational classes *of theories*.. We can almost infer what ordered (Section 10) and continuous (Section 11) S-sorted theories are, but there is a slight twist in those definitions which we postpone for a while.

From Theorem 8.5 we know of the existence (and construction) of free S-sorted algebraic theories generated by $(S^* \times S^*)$-indexed families of sets. This is at times useful (see ADJ (1977a)) but it is really more than necessary. Note than any generator y in the carrier (hom set) $\mathbb{T}(u,v)$ determines $|u|$ elements $x_i^u \circ y$ in the carriers $\mathbb{T}(u_i,v)$ and y in turn is uniquely determined (Proposition 12.2) by those elements. Thus it is sufficient to consider generators indexed by $S\times S^*$, but such indexed families are exactly S-sorted signatures (see Definition 6.1).[†] Thus we know from Theorem 8.5 that for any S-sorted signature Σ, there is an S-sorted algebraic theory, $\mathbb{T}_\Sigma$, freely generated by Σ.

We actually go much further in using the results of Section 6 and construct $\mathbb{T}_\Sigma$ using Theorem 6.10, rather than the quotient construction of Theorem 8.5.

First we pick out a canonical family of variables, X_v for each $v\in S^*$, assumed to be disjoint from any signature Σ. As a set, $X_v=\{x_{v_1,1},\ldots,x_{v_n,n}\}$ which, as a S-indexed family, is

$$(X_v)_s = \{x_{v_i,i} \mid v_i=s\}.$$

Next, define an S-sorted algebraic theory $\mathbb{T}_\Sigma$ by

$$\mathbb{T}_\Sigma(u,v) = T_\Sigma(X_v)^u = T_\Sigma(X_v)_{u_1}\times\ldots\times T_\Sigma(X_v)_{u_n}$$

where n is the length of u. Thus the morphisms in $\mathbb{T}_\Sigma(u,v)$ are "u-tuples of expressions in the variables X_v." Informally composition is substitution, i.e., if $<t_1,\ldots,t_n>\in\mathbb{T}_\Sigma(u,v)$ and $<t'_1,\ldots,t'_m>\in\mathbb{T}_\Sigma(v,w)$ then the i^{th} component of the composite ($i\in[n]$) is the result of simultaneously substituting t'_j for $x_{v_j,j}$ in t_i.

We need to give a mathematical (i.e. algebraic) definition of this composition operation to replace the informal one. For this, let us identify the set $T_\Sigma(X_v)^u$ with the set of S-indexed families of functions from X_u to $T_\Sigma(X_v)$. Notice that this is exactly like the identification of A^n with the set of functions from [n] to A discussed in Section 2. Given $t{:}X_u\rightarrow T_\Sigma(X_v)$, for each $i\in[\,|u|\,]$, $(x_{u_i,i})t\in T_\Sigma(X_v)_{u_i}$, that is, the ith component of t is of sort u_i as required.

Now given $t{:}u\rightarrow v$ and $t'v\rightarrow w$ in $\mathbb{T}_\Sigma$, Theorem 6.10 gives us a unique homomorphism $(t')^\#{:}T_\Sigma(X_v)\rightarrow T_\Sigma(X_w)$. $t\circ t'$ is the composite $t(t')^\#$ (in Set^S), or keeping the tuple view of t , $<t_1,\ldots,t_n>\circ t' = <(t_1)(t')^\#,\ldots,(t_n)(t')^\#>$. (This is all worked out in detail for the one-sorted case in ADJ (1977b).)

[†] In all our earlier work (see especially ADJ (1975a, 1976d)) signatures were defined to be $(S^*\times S)$-indexed families of sets. There is good pedagogical reason for this. An operation symbol σ from $\Sigma_{s,s_1\ldots s_n}$ takes arguments of sorts $s_1,\ldots,s_n$, and gives a value of sort s. One tends to read it that way and the indexing, "$s,s_1\ldots s_n$" seems backwards. But as we are just seeing, that "backwards" indexing is the natural one for S-sorted theories; every theory $\mathbb{T}$ gives rise to an $(S\times S^*)$-indexed family, $<\mathbb{T}(s,w) \mid s\in S \text{ and } w\in S^*>$, called the *underlying signature* of $\mathbb{T}$.

The distinguished morphism from s to v in T_Σ is simply the variable $x_{s,1}$ (well, actually it is the corresponding one-tuple) and tupling in T_Σ is exactly that, tupling.

We equip T_Σ with the obvious injection of the generators, $I_\Sigma:\Sigma \to T_\Sigma$, sending $\sigma \in \Sigma_{s,s_1 \ldots s_n}$ to the expression $\sigma(x_{s_1,1} \ldots x_{s_n,n}) \in T_\Sigma(X_{s_1 \ldots s_n})$. Then we have

Theorem 12.5. $<T_\Sigma, I_\Sigma>$ is the S-sorted algebraic theory freely generated by the S-sorted signature Σ. Thus, let T be any S-sorted algebraic theory and let $H:\Sigma \to T$ be any $(S \times S^*)$-indexed family of maps (this is just the many-sorted algebra situation except that we are only looking at the $(S \times S^*)$-indexed part of T, $<T(s,u) \mid s \in S \text{ and } u \in S^*>$). Then there exists a unique morphism of algebraic theories, $H^\#:T_\Sigma \to T$ that extends H in the sense that $I_\Sigma H^\# = H$. □

We are going to jump ahead and define another algebraic theory, one whose morphisms are tuples of partial finite and infinite trees as described in Section 10. It is jumping ahead because this theory, CT_Σ, is the free continuous S-sorted algebraic theory and it has, as subtheories, the free ordered, rational and iterative algebraic theories as well a theory isomorphic to T_Σ described above. The reason for giving the construction now (before the definitions of the other classes of theories) is because that construction is exactly like the one we just went through for T_Σ. We will change notation slightly so as to incorporate the identification of tuples and functions alluded to there.

Recall from Section 2 that if A and B are S-indexed families, then A^B is the set of $f:A \to B$, i.e., the set of S-indexed families of functions, $<f_s:A_s \to B_s \mid s \in S>$.

Define the S-sorted algebraic theory CT_Σ by

$$CT_\Sigma(u,v) = Tr_\Sigma(X_v)^{X_u}.$$

Given $\alpha:u \to v$ and $\beta v \to w$, $\alpha \circ \beta$ is $\alpha\beta^\#$ where $\beta^\#$ is the unique (continuous) Σ-homomorphism from $Tr_\Sigma(X_v)$ to $Tr_\Sigma(X_w)$ guaranteed by Theorem 11.2.

$$\begin{tikzcd}
 & X_v \arrow[d, "J"] \arrow[dr, "\beta"] & \\
X_u \arrow[r, "\alpha"] & Tr_\Sigma(X_v) \arrow[r, "\beta^\#"'] & Tr_\Sigma(X_w)
\end{tikzcd}$$

The distinguished morphism $\overset{u}{x_i}:u_i \to u$ is the map from X_{u_i} to $Tr_\Sigma(X_u)$ sending $x_{u_i,1}$ to $x_{u_i,i}$. Tupling of n morphisms $\beta_i:u_i \to v$ gives $\beta:X_u \to Tr_\Sigma(X_v)$ given by $(x_{u_i,i})\beta = (x_{u_i,1})\beta_i$. We check the tupling equations:

$$(x_{u_i,1})\overset{u}{x_i} \circ (\beta_1,\ldots,\beta_n) = (x_{u_i,i})(\beta_1,\ldots,\beta_n) = (x_{u_i,1})\beta_i,$$

so that 12.1.5a is satisfied. Next,

$$(x_{u_i,i})(x_1^u \circ \beta,\ldots,x_n^u \circ \beta) = (x_{u_i,1})(x_i^u \circ \beta) = (x_{u_i,i})\beta,$$

and this shows that 12.1.5b is satisfied. To be completely precise, the injection of the generators is $J_X:\Sigma \to CT_\Sigma$ given by

$$(x_{s,1})(\sigma J_X) = \{<\lambda,\sigma>,<1,x_{s_1,1}>,\ldots,<n,x_{s_n,n}>\}$$

for $\sigma \in \Sigma_{s,s_1\ldots s_n}$.

Proposition 12.6. CT_Σ, as defined above is an S-sorted algebraic theory. □

CT_Σ is truly a remarkable algebraic theory as is its algebraic cousin $Tr_\Sigma(X)$. It has subtheories corresponding to the various subalgebras of $Tr_\Sigma(X)$ given in 10.16.1 through 10.16.5. All but the first of these are of particular interest to us and we give them names as follows.

(12.7.2) PT_Σ is the subtheory of CT_Σ determined by tuples of trees from $FTr_\Sigma(X_V)$, i.e., tuples of finite partial X_V-ary Σ-trees (10.16.2).

(12.7.3) RT_Σ is the subtheory of CT_Σ determined by tuples of trees from $RTr_\Sigma(X_V)$, i.e., tuples of rational trees (10.16.3).

(12.7.4) FT_Σ is the subtheory of CT_Σ (and of PT_Σ and RT_Σ) determined by tuples of finite total trees (10.16.4).

(12.7.5) IT_Σ is the subtheory of CT_Σ (and of RT_Σ) determined by tuples of total rational trees (10.16.5).

For each σ, σJ_Σ is in each of the subtheories defined above because it is total and finite. We will use the same notation, J_Σ for the target restriction of $J_\Sigma:\Sigma \to CT_\Sigma$ to each of the subtheories PT_Σ, RT_Σ,FT_Σ, and IT_Σ.

Theorem 12.8. For any S-sorted signature Σ, $<FT_\Sigma,J_\Sigma>$ is the S-sorted algebraic theory freely generated by Σ. Thus $<FT_\Sigma,J_\Sigma>$ and $<T_\Sigma,I_\Sigma>$ are (uniquely) isomorphic.

Definition 12.9. An *ordered* S-sorted algebraic theory T is an S-sorted algebraic theory which, as an $(S^* \times S^*)$-sorted algebra is ordered, and in which the composition operation is *left strict*, that is, for all $u,v,w \in S^*$ and $\alpha:v \to w$,

$$\perp_{u,v} \circ \alpha = \perp_{u,w}. \qquad \square$$

We said earlier in this section that viewing algebraic theories as many-sorted algebras led us to "almost" infer what ordered theories were. The almost comes in the "twist" that we want the composition operation of the ordered theory to be left strict. There are algebraic reasons for this, but the computational explanation is convincing. When the morphisms of the theory are viewed as some kind of computations and $\perp$ stands for the *totally* undefined never halting computation, then any computation following $\perp$ must be $\perp$ (totally undefined and never halting). The opposite direction does not work the same way for a computation followed by the totally undefined computation may have "some definition", for instance the first computation might have had output which would be there whether or not the subsequent computation terminated.

We can still infer from Theorem 10.12, the existence of a free S-sorted ordered algebraic theory because the left strict condition is equational (and thus inequational), but as in the unordered case, we do better by using the free ordered algebra to obtain the free ordered theory.

Theorem 12.10. Let Σ be any S-sorted signature. Then $\mathbf{PT}_\Sigma$ is an S-sorted ordered algebraic theory and in particular, $<\mathbf{PT}_\Sigma, J_X>$ is the S-sorted ordered algebraic theory freely generated by Σ. □

The "twist" for continuous theories is that we do not require that the composition operation be continuous, only continuous by components (see Section 5).

Definition 12.11. Let Z be a subset system. A Z-*continuous* algebraic theory is an ordered algebraic theory in which the carriers, $\mathbf{T}(u,v)$, are all Z-complete posets and in which composition is Z-continuous *by components*, i.e., for all Z-sets $X \subseteq \mathbf{T}(u,v)$ and $Y \subseteq \mathbf{T}(v,w)$,

$$(\bigsqcup X) \circ (\bigsqcup Y) = \bigsqcup\{x \circ y \mid x \in X \text{ and } y \in Y\}. \qquad \square$$

Theorem 12.12. For any S-sorted signature Σ, $\mathbf{CT}_\Sigma$ is a PC-continuous algebraic theory. Further, if Z is any subset system with $\omega \subseteq Z$ and $Z \subseteq PC$, then $<\mathbf{CT}_\Sigma, J_{\Sigma'}>$ is the Z-continuous algebraic theory freely generated by Σ. □

13. SOLVING EQUATIONS: ITERATIVE AND RATIONAL THEORIES.

A principal interest in algebraic theories comes from the process of "solving equations" within theories. This process is well illustrated in the one-sorted case by a theory for context-free sets and grammars first described in ADJ (1976d).

Let T be a set of *terminal* symbols and let $X = \{A_1, A_2, \ldots\}$ be a countable set of *non-terminal* symbols with $X_n = \{A_1, \ldots, A_n\}$.

For each n,p$\in\omega$, let $\mathbf{CF}_T(n,p)$ be the complete lattice ($\bigsqcup$-complete poset) of all n-tuples of subsets of $(T\cup X_p)^*$ under component-wise set-theoretic inclusion. The minimum element is $\perp_{n,p} = <\emptyset,...,\emptyset>$. For $U=<U_1,...,U_n>\in\mathbf{CF}_T(n,p)$ and $V=<V_1,...,V_p>\in\mathbf{CF}_T(p,q)$, define the composite $U\circ V$ to be $W = <W_1,...,W_n>\in\mathbf{CF}_T(n,q)$, where W_i consists of all words $w\in(T\cup X_q)^*$ for which there exists $u\in U_i$ such that w results from simultaneously replacing each occurrence of A_j ($j\in[p]$) by some element of V_j (where distinct occurrences of A_j may be replaced by distinct elements of V_j). This composition is associative, left strict and Δ-continuous. (Note, composition is not $\bigsqcup$-continuous by components as analysis of the composite, $\{A_1A_1\}\circ V$ shows with a non-trivial $V = \bigsqcup\{\{v\}\mid v\in V\}$.) The identity for $\circ$ in $\mathbf{CF}_T(n,n)$ is $<\{A_1\},...,\{A_n\}>$. The distinguished morphism x_i^n in $\mathbf{CF}_T(1,n)$ is just $<\{A_i\}>$ and $<\{A_i\}>\circ(U_1,...,U_n)=U_i$ and $<<\{A_1\}>\circ U,...,<\{A_n\}>\circ U>=U$, so that the tupling equations (12.1.5a, 12.1.5b) are satisfied.

Proposition 13.1. For any set T (of terminal symbols), $\mathbf{CF}_T$ is an Δ-continuous algebraic theory. □

For example we can take $T = \{a,b\}$, $U=<\{aA_1a,A_2\},\{b\}>:2\to 2$, and $V=<\{aa,ab,ba\},\{\lambda\}>:2\to 0$. Then $U\circ V=<\{aaaa,aaba,abaa,\lambda\},\{b\}>$. More interesting is the sequence of composites $U^k\circ\perp_{2,0}$ for $k\geq 0$. These are:

$$\begin{aligned}
\perp_{2,0} &= <\emptyset,\emptyset> \\
U\circ\perp_{2,0} &= <\emptyset,\{b\}> \\
U^2\circ\perp_{2,0} &= <\{b\},\{b\}> \\
U^3\circ\perp_{2,0} &= <\{aba,b\},\{b\}> \\
U^4\circ\perp_{2,0} &= <\{aabaa,aba,b\},\{b\}> \\
&\dots
\end{aligned}$$

This sequence of morphisms from 2 to 0 forms a chain, the kth element of which is

$$<\{a^jba^j \mid 0\leq j\leq k-2\},\{b\}>$$

and the union of which is

$$U^\dagger \;=\; <\{a^jba^j \mid 0\leq j\},\{b\}>.$$

Now $U^\dagger$ has the property that $U^\dagger = U\circ U^\dagger$, i.e., $U^\dagger$ is a *solution* (for ξ) in the equation $\xi = U\circ\xi$. In fact it is the least solution in the sense that if $V = U\circ V$ then $U^\dagger\subseteq V$.

The morphism U in this discussion is a representation of the context-free grammar with non-terminals $\{A_1,A_2\}$, terminals {a,b} and productions, $\{A_1\to aA_1a,\ A_1\to A_2,\ A_2\to b\}$. The "solution" process we've gone through is that of finding the terminal sets (languages) for both the non-terminals A_1 and A_2 simultaneously, a familiar process in not such a familiar notation.

Any morphism $U=<U_1,...,U_n>:n\rightarrow n$ (with U_i finite) can be viewed in this way as a context-free grammar in non-terminals $\{A_1,...,A_n\}$. The solution $U^\dagger:n\rightarrow 0$ consists of an n-tuple of subsets of $T^* = (T\cup X_0)^*$, the ith component $(x_i^n\circ U^\dagger)$ of which is the context-free set generated from the non-terminal A_i. It is the minimum solution to the equation, $\xi=U\circ\xi$.

Now we make the step of introducing equations with "parameters," an essential ingredient in the study of iteration and recursion first recognized by Wagner (1971a). It is essential in that it permits the study of the interaction of composition with the iteration operation. For the context-free case, a morphism $U:n\rightarrow n+p$ is an n-tuple of sets of strings involving the non-terminals $A_1,...,A_n$, and $A_{n+1},...,A_{n+p}$, the latter being called "parameters." The solution $U^\dagger:n\rightarrow p$ will be the n-tuple of context-free sets obtained by viewing the parameters *as if* they were terminals and then replacing them with the "first" p non-terminals, $A_1,...,A_p$. This shift of "variables" may be confusing at first, but it makes things work beautifully. $U^\dagger$ is the solution to the (rational) equation,

$$\xi = U\circ(\xi_1,...,\xi_n,\{A_1\},...,\{A_n\}),$$

where $\xi_i=x_i^n\circ\xi$.[†] That solution is obtained as the least upper bound of the sequence $U^{(k)}$ where:

$$\begin{aligned} U^{(0)} &= <\emptyset,...,\emptyset> \\ U^{(k+1)} &= U\circ(x_1^n\circ U^{(k)},...,x_n^n\circ U^{(k)},\{A_1\},...,\{A_n\}). \end{aligned}$$

To formulate the general process illustrated by the context-free sets example, we need some additional concepts and notation for algebraic theories in general. Source tupling (12.1.4) puts morphisms $\beta_i:s_i\rightarrow w$ together to get $\beta=(\beta_1,...,\beta_n):s_1...s_n\rightarrow w$. This process is generalized to *source pairing* where we put $\alpha:u\rightarrow w$ and $\beta:v\rightarrow w$ together to get $(\alpha,\beta):u+v\rightarrow w$. Here we have written u+v for the concatenation uv of the strings u and v. The justification, incidentally, for using this notation is that uv is the coproduct (or sum) object for u and v in the category $\mathbf{Str}_S$ or in any S-sorted algebraic theory $\mathbb{T}$. In the one-sorted case that sum is in fact the sum of natural numbers. Pairing is defined in terms of tupling and the distinguished morphisms:

$$(13.2) \qquad (\alpha,\beta) = (x_1^u\circ\alpha,...,x_n^u\circ\alpha,x_1^v\circ\beta,...,x_m^v\circ\beta)$$

where $n=|u|$ and $m=|v|$. Thus pairing is just tupling the components of α and β. Corresponding to the role of the distinguished morphisms x_i^u which pick out the components of a tuple, there are "coproduct injections,"

[†] This particular equational view of fixed-point solutions derives directly form Elgot (1973).

(13.3.1) $$x_{(1)}^{u+v} = (x_1^{u+v},\ldots,x_n^{u+v}):u\to u+v$$

(13.3.2) $$x_{(2)}^{u+v} = (x_{n+1}^{u+v},\ldots,x_{n+m}^{u+v}):v\to u+v$$

which pick out the components of a pair:

(13.3.3) $$x_{(1)}^{u+v}\circ(\alpha,\beta) = \alpha \quad x_{(2)}^{u+v}\circ(\alpha,\beta) = \beta.$$

The following is a collection of important properties of source pairing relative to composition and the special morphisms $0_v:\lambda\to v$. (Compare with Elgot (1973), equations 5.1 through 5.4.)

Proposition 13.4. Let T be any S-sorted algebraic theory with $\alpha:u\to w$ and $\beta:v\to w$.

(13.4.1) $((\alpha,\beta),\gamma) = (\alpha,(\beta,\gamma))$ for $\gamma:v\to w$

(13.4.2) $(0_u,\alpha) = \alpha = (\alpha,0_w)$

(13.4.3) $(\alpha,\beta)\circ\gamma = (\alpha\circ\gamma,\beta\circ\gamma)$ for $\gamma:w\to r$

(13.4.4) $0_u\circ\alpha = 0_w$ □

Whereas pairing of morphisms is tupling of their respective components, *summing* can be viewed as pairing with a change of "variables" to avoid conflict. The idea is best illustrated (informally and formally) with flowcharts. Pairing lays two flowcharts side-by-side *identifying* their corresponding outputs, whereas summing lays them side-by-side keeping their outputs disjoint. The sum of $\alpha:u\to w$ and $\beta:v\to r$ is $\alpha+\beta:u+v\to w+r$ defined by

(13.5) $$\alpha+\beta = (\alpha\circ x_{(1)}^{w+r},\beta\circ x_{(2)}^{w+r}).$$

Now we can give the general formulation of the solution process informally discussed above. The basic form of a rational equation in an S-sorted algebraic theory T is

(13.6) $$\xi = \alpha\circ(\xi,1_p),$$

where $\alpha:u\to u+v$ is a morphism of T, and we are solving for $\xi:u\to v$. A fixed-point view of (13.6) is that each $\alpha:u\to u+v$ defines a function $\hat{\alpha}:T(u,v)\to T(u,v)$ defined by $(\eta)\hat{\alpha} = \alpha\circ(\eta,1_p)$. ξ is a solution to (13.6) if ξ is a fixed-point of $\hat{\alpha}$. When the theory is ordered, then ξ is the *minimum* solution if it is a solution and whenever $(\eta)\hat{\alpha}\sqsubseteq\eta$, then $\xi\sqsubseteq\eta$.

The concept of an algebraic theory having unique solutions for rational equations is abstracted in the definition of "iterative theory" given by Elgot (1973). (Our definition differs from his only in that we are considering S-sorted theories.)

Definition 13.7. Let $\mathbb{T}$ be an S-sorted algebraic theory. A morphism $\alpha:u \twoheadrightarrow v$ is *ideal* iff for each $i\in[\,|u|\,]$, $x_i^u \circ \alpha$ is *not* a distinguished morphism. $\mathbb{T}$ is an *ideal* theory iff $\alpha:u\twoheadrightarrow v$ ideal implies $\alpha\circ\beta$ ideal for all $\beta:v\twoheadrightarrow w$. The theory $\mathbb{T}$ is *iterative* if it is ideal and for every *ideal* morphism $\alpha:u\twoheadrightarrow u+v$ there is a *unique* solution $\xi=\alpha\circ(\xi,1_v):u\twoheadrightarrow v$. ξ is called the iterate of α and denoted $\alpha^\dagger$. □

Theorem 13.8. Let Σ be any S-sorted signature. The theories $\mathbb{CT}_\Sigma$, $\mathbb{RT}_\Sigma$, and $\mathbb{IT}_\Sigma$ are all iterative theories. $<\mathbb{IT}_\Sigma,J_\Sigma>$ is the iterative theory freely generated by Σ. □

Note that the existence of a free iterative theory (see Bloom and Elgot (1974)) does not follow from Theorem 8.5 because of the fact that the iteration operation is *partial*, being defined only on ideal morphisms. With an ordered theory approach we get away from this restriction and have a totally defined iteration operation. The concept of ordered algebraic theories having solutions to rational equations is abstracted in our definitions of rationally closed and rational theories.

Definition 13.9. A *rationally closed* S-sorted algebraic theory $\mathbb{T}$ is an ordered theory equipped with a function $^\dagger:T(u,u+v)\twoheadrightarrow T(u,v)$ for all $u,v\in S^*$. $\alpha^\dagger$ is called the *iterate of* or *minimum solution for* α and must satisfy the following for all $\eta:u\twoheadrightarrow v$ and $\tau:v\twoheadrightarrow w$.

(13.9.1) $\alpha\circ(\alpha^\dagger,1_v) = \alpha^\dagger$

(13.9.2) If $\alpha\circ(\eta,1_v)\sqsubseteq\eta$ then $\alpha^\dagger\sqsubseteq\eta$

(13.9.3) $(\alpha\circ(1_u+\tau))^\dagger = \alpha^\dagger\circ\tau$

$\mathbb{T}$ is a *rational* theory if $\mathbb{T}$ is rationally closed and

(13.9.4) $$\alpha^\dagger = \bigsqcup\alpha^{(k)}$$

where,

(13.9.5) $$\alpha^{(0)} = \bot_{u,v}$$

(13.9.6) $$\alpha^{(k+1)} = \alpha\circ(\alpha^{(k)},1_v).$$ □

The first two conditions on the dagger operation in a rationally closed theory require that $\alpha^\dagger$ be a a minimum solution to the rational equation (13.6). The third condition (and the most interesting one) says that the parameters have to indeed behave like parameters; one can read (13.9.3) as saying that if τ is substituted for the parameters and the result is solved or iterated, then that is the same as solving first and then substituting τ for the parameters.

Theorem 13.10. For any subset system Z, if $\omega\subseteq Z$ and $\mathbb{T}$ is a Z-continuous theory, then $\mathbb{T}$ is rational. □

Theorem 13.11. For any S-sorted signature Σ, $\mathbf{RT}_\Sigma$ is a rational theory; in particular, $<\mathbf{RT}_\Sigma,J_\Sigma>$ is the rational theory freely generated by Σ. □

14. FLOWCHARTS AND BEHAVIORS.

We revert to considering one-sorted theories. The following definition was obtained independently by Elgot (1973) and Wagner (1974). Elgot's definition exactly parallels ours though he then goes on to consider only *ideal* (the "body" is ideal in the sense of Definition 13.7) "normal descriptions." Wagner defines "abstract recursive definitions" where the "begin" is not restricted to being a map.

Definition 14.1. By a *normal description* D *over* a theory $\mathbf{T}$ *from* n *to* p *of weight* s we mean a pair $<a,\alpha>$ where the *begin*, $a{:}n \twoheadrightarrow s+p$ is a map and the *body* $\alpha{:}s \twoheadrightarrow s+p$ is arbitrary in $\mathbf{T}$.

(14.1.1) The *identity* normal description from n to n is also denoted 1_n; it has weight 0 (c.f. Definition 4.10):

$$1_n = <1_n,0_n>$$

where the begin (1_n) is the identity for n in $\mathbf{T}$.

(14.1.2) The *composite* of two normal descriptions, $<a,\alpha>$ from n to p of weight s and $<b,\beta>$ from p to q of weight s′ is the normal description $<a,\alpha>\circ<b,\beta>$ from n to q of weight s+s′ given by (c.f. Definition 4.11):

$$<a,\alpha>\circ<b,\beta> = <a\circ f,\ (\alpha\circ f,\beta\circ g)>$$

where $f=1_s+b$ and $g=0_s+1_{s'+q}$ in $\mathbf{T}$.

(14.1.3) The *pairing* or *coalesced sum* of two normal descriptions $<a,\alpha>$ from n to p of weight s and $<b,\beta>$ from n′ to p of weight s′ is the normal description $(<a,\alpha>,<b,\beta>)$ from n+n′ to p of weight s+s′ given by (c.f. Definition 4.13):

$$(<a,\alpha>,<b,\beta>) = <(a\circ f,b\circ g),\ (\alpha\circ f,\beta\circ g)>$$

where $f=1_s+0_{s'}+1_p$ and $g=0_s+1_{s'+p}$ in $\mathbf{T}$.

(14.1.4) The *iterate* of a normal description $<a,\alpha>$ from n to n+p of weight s is $<a,\alpha>^\dagger$ from n to p of weight n+s given by (c.f. Definition 4.15):

$$<a,\alpha>^\dagger = <a,(\alpha,a)>.$$

(14.1.5) For any map $f{:}n \to p$ in T is the corresponding normal description $<f,0_p>$ from n to p of weight 0 (c.f. Definition 4.16). □

Theorem 14.2. $\mathbf{ND}_T$ is a category whose objects are the natural numbers and whose morphisms from n to p are all normal descriptions from n to p; composition is given by 14.1.2 and identities are given by 14.1.1. □

The relationship between the the category of flowcharts given in Section 4 and the category of normal descriptions is given by the following

Theorem 14.3. Let I be any interpretation of Σ into a rational theory $\mathbf{T}$, i.e., I is an indexed family of maps, $<I_n{:}\Sigma_n \to T(1,n)>$. Extend I to $\Sigma_\perp$ with $(\perp)I=\perp{:}1 \to 1$. Then define $\overline{I}{:}\mathbf{Fl}_{\Sigma_\perp} \to \mathbf{ND}_T$ by

$$<b,\tau,\ell>\overline{I} = <b,\beta> \text{ where } \beta_i=((i\ell)I)\circ(i)\tau.$$

Then $\overline{I}$ is a functor that also preserves maps, pairing and iteration. □

The final connection in interpreting flowcharts is given in the following

Theorem 14.4. Let T be any rational theory and define $|\ |{:}\mathbf{ND}_T \to T$ by

$$|<a,\alpha>| = a\circ(\alpha^\dagger,1_p).$$

Then $|\ |$ is a functor which preserves maps, pairing and iteration. □

Example 14.1. We present a signature (ranked alphabet) Ω which we use to construct $\Omega_\perp$-flowcharts. In that alphabet we include some of the symbols from the $\{int,Bool\}$-sorted signature Σ of Example 7.1.

$$\Omega_1 = \{\text{load}_x, \text{store}_x \mid x\in X\} \cup \{\text{switch}\} \cup \textstyle\bigcup_{w\in\{int\}^*} \Sigma_{int,w}$$
$$\Omega_2 = \textstyle\bigcup_{w\in\{int\}^*} \Sigma_{Bool,w}$$
$$\Omega_n = \emptyset,\ n = 0,3,4,\ldots .$$

This signature determines the category $\mathbf{Fl}_{\Omega_\perp}$ of $\Omega_\perp$-flowcharts via Definition 4.9 and Theorem 4.12. We provide an interpretation of Ω in $\mathbf{Sum}_A$ where A = Stk×Env (stacks cross environments):

$$\text{Stk} = [\omega \to \mathbb{Z}].$$
$$\text{Env} = [X \to \mathbb{Z}].$$

Note that we have taken stacks to be infinite to make the definitions simpler. For example we will write $v_1\bullet v_2\bullet\ldots\bullet v_n\bullet\rho$ where $v_i\in\mathbb{Z}$ and $\rho\in\text{Stk}$ to denote the stack whose first n elements are $v_1,\ldots,v_n$, and whose "rest" is ρ.

With the identification of A with $A \times [1]$, the interpretation, $I: \Omega \to \mathbf{Sum}_A$, is given as follows.

(I1) $\langle \rho, e \rangle (\mathrm{load}_x I) = \langle (x)e \bullet \rho, e \rangle$ For $x \in X$

(I2) $\langle v \circ \rho, e \rangle (\mathrm{store}_x I) = \langle \rho, e[x/v] \rangle$

(I3) $\langle v_1 \bullet v_2 \bullet \rho, e \rangle (\mathrm{switch} I) = \langle v_2 \bullet v_1 \bullet \rho, e \rangle$

(I4) $\langle \rho, e \rangle (cI) = \langle c_S \bullet \rho, e \rangle$ For $c \in \Sigma_{int,\lambda}$

(I5) $\langle v \bullet \rho, e \rangle (\mathrm{aop1} I) = \langle (v)\mathrm{aop1}_S \bullet \rho, e \rangle$ For $\mathrm{aop1} \in \Sigma_{int,int}$

(I6) $\langle v_1 \bullet v_2 \bullet \rho, e \rangle (\mathrm{aop2} I) = \langle (v_1, v_2)\mathrm{aop2}_S \bullet \rho, e \rangle$ For $\mathrm{aop2} \in \Sigma_{int,int\ int}$

(I7) $\langle \rho, e \rangle (\mathrm{bc} I) = \langle \langle \rho, e \rangle, \mathrm{bc}_S \rangle$ For $\mathrm{bc} \in \Sigma_{Bool,\lambda}$

(I8) $\langle v \bullet \rho, e \rangle (\mathrm{prop} I) = \langle \langle \rho, e \rangle, (v)\mathrm{prop}_S \rangle$ For $\mathrm{prop} \in \Sigma_{Bool,int}$

(I9) $\langle v_1 \bullet v_2 \bullet \rho, e \rangle (\mathrm{rel} I) = \langle \langle \rho, e \rangle, (v_1, v_2)\mathrm{rel}_S \rangle$ For $\mathrm{rel} \in \Sigma_{Bool,int\ int}$

Now taking $T_{<ae>} = T_{<st>} = \mathbf{Fl}(1,1)$ and $T_{<be>} = \mathbf{Fl}(1,2)$, we will make T into a G-algebra where G is the context-free grammar of Example 7.2 and we do that by defining operations on $\Omega_\perp$-flowcharts corresponding to each of the seventeen productions of G.

(T1) $\mathrm{continue}_T = 1_1$

(T2) $(F)x{:=}_T = F \circ \mathrm{store}_x$

(T3) $(P, F_1, F_2)\mathrm{ifthenelse}_T = P \circ (F_1, F_2)$

(T4) $(F_1, F_2);_T = F_1 \circ F_2$

(T5) $(P, F)\mathrm{whiledo}_T = (P \circ (F \oplus 1_1))^{\dagger}$

(T6) $c_T = c$

(T7) $x_T = \mathrm{load}_x$

(T8) $(F)\mathrm{aop1}_T = F \circ \mathrm{aop1}$

(T9) $(F_1, F_2)\mathrm{aop2}_T = F_1 \circ F_2 \circ \mathrm{aop2}$

(T10) $(P, F_1, F_2)\mathrm{cond}_T = P \circ (F_1, F_2)$

(T11) $(F_1, F_2)\mathrm{result}_T = F_1 \circ F_2$

(T12) $(F_1, F_2)\mathrm{letx}_T = \mathrm{load}_x \circ F_1 \circ \mathrm{store}_x \circ F_2 \circ \mathrm{switch} \circ \mathrm{store}_x$

(T13) $\mathrm{bc}_T = \mathrm{bc}$

(T14) $(F)\mathrm{prop}_T = F \circ \mathrm{prop}$

(T15) $(F_1, F_2)\mathrm{rel}_T = F_1 \circ F_2 \circ \mathrm{rel}$

(T16) $(P)\neg_T = P \circ (x_2^2, x_1^2)$

(T17a) $(P_1, P_2)\wedge_T = P_1 \circ (P_2, x_2^2)$

(T17b) $(P_1, P_2)\vee_T = P_1 \circ (x_1^2, P_2)$ □

BIBLIOGRAPHY.

Andreka and Nemeti (1978,79) have an excellent survey and bibliography of applications of universal algebra outside pure mathematics. We have, without reservation, used their bibliography to enhance ours, but by no means have we included all their (up-to-date) references.

ADJ (Authors: J. A. Goguen, J. W. Thatcher, E. G. Wagner and J. B. Wright)

(1973) (JAG, JWT, EGW, JBW) "A Junction between computer science and category theory: I, Basic definitions and examples," Part 1, IBM Research report RC-4526, Sept 1973.

(1975) (JAG, JWT, EGW, JBW) "Abstract data types as initial algebras and correctness of data representations," *Proceedings*, Conference on Computer Graphics, Pattern Recognition and Data Structure, May 1975, pp. 89-93.

(1975a) (JAG, JWT, EGW, JBW) "Initial algebra semantics and continuous algebras," IBM Research Report RC-5701. November 1975. *JACM 24* (1977) pp. 68-95.

(1975b) "Introduction to categories, algebraic theories and algebras," IBM Research Report RC 5369, April, 1975.

(1975c) "Parallel realization of systems using factorizations and quotients in categories," IBM Research Report RC-5668, October, 1975. *J. Franklin Inst. 301* (1976) 541-558.

(1976) (JWT, EGW, JBW) "Specification of abstract data types using conditional axioms," IBM Research Report RC-6214, September 1976.

(1976a) (JAG, JWT, EGW) "An initial algebra approach to the specification, correctness, and implementation of abstract data types," IBM Research Report RC-6487, October 1976. *Current Trends in Programming Methodology, IV: Data Structuring* (R. Yeh, Ed.) Prentice Hall, New Jersey (1978) 80-149.

(1976b) (EGW, JBW, JAG, JWT) "Some fundamentals of order algebraic semantics," *Lecture Notes in Computer Science 45* (Mathematical Foundations of Computer Science 1976), Springer-Verlag, pp153-168; IBM Research Report RC 6020, May 1976.

(1976c) (JAG, JWT, EGW, JBW) "A junction between computer science and category theory:I, Basic definitions and examples," Part 2, IBM Research Report RC 5908, March 1976.

(1976d) (JBW, JWT, EGW, JAG) "Rational algebraic theories and fixed-point solutions," *Proceedings* 17th IEEE Symposium on Foundations of Computing, Houston, Texas, October, 1976, pp. 147-158.

(1977) (JBW, EGW, JWT) "A uniform approach to inductive posets and inductive closure." *Lecture Notes in Computer Science 53* (Mathematical Foundations of Computer Science 1977), pp. 192-212. IBM Research Report RC-6817, October 1977. *Theoretical Computer Science 7* (1978) 57-77.

(1977a) (JAG) "Abstract errors for abstract data types,"UCLA Semantics Theory of Computation Report 6, February 1977. *Proceedings* IFIP Working Conference on Formal Description of Programming Concepts, St. Andrews, New Brunswick, pp. 21.1-21.32, August, 1977.

(1977b) (EGW, JWT, JBW) "Free continuous theories," IBM Research Report RC 6906, December, 1977. To appear, *Fundamenta Informaticae.*

(1978) (JWT, EGW, JBW) "Data type specification: parameterization and the power of specification techniques," *Proceedings*, Tenth ACM SIGACT Symposium on Theory of Computing, San Diego CA, May, 1978, pp. 119-132.

(1978a) (EGW, JWT, JBW) "Programming languages as mathematical objects," *Proceedings*, Symposium on Mathematical Foundations of Computer Computer Science, Zacopane, Poland, September, 1978. *Lecture Notes in Computer Science 64* (J. Winkowski, Ed.)

(1979) (JWT, EGW, JBW) "Many-sorted and ordered algebraic theories," IBM Research Report RC 7595, April 1979.

(1979a) "More on advice on structuring compilers and proving them correct," IBM research Report RC 7588, April, 1979. To appear, *Proceedings* ICALP ‘79, Graz, Austria.

Aiello, L., Attardi, G. and Prini, G

(1977) "Towards a more declarative programming style," *Proceedings*, IFIP Working Conference on Formal Description of Programming Concepts, St. Andrews, New Brunswick, Canada, August, 1977, pp.5.1-5.16.

Andreka, Hajnal and Nemeti, Istvan

(1978) "A survey and bibliography of some applications of universal algebra outside pure mathematics," Preprint 12/1978, Mathematical Institute of the Hungarian Academy of Sciences, January, 1978.

(1979) "Applications of universal algebra in computer science" (Review), Manuscript, January 1979.

Arbib, M.A. and Giveon, Y.

(1968) "Algebra automata I: Parallel programming as a prolegomena to the categorical approach," *Information and Control 12* (1968) 331-345.

Arnold, A.

(1977) "Systems d'équations dans le magmoid. Ensembles rationnels et algébriques d'arbes," These d'État, Lille, 1977.

Arnold, A. and Nivat, M.

(1977) "Non deterministic recursive program schemes," Fundamentals of Computer Science 1977, *Lecture Notes in Computer Science 56* (1977)12-22.

Banaschewski, Bernhard and Nelson, Evelyn

(1979) "Completions of partially ordered sets as reflections," McMaster University Technical Report 79-CS-6, 1979.

Bekič, H

(1969) "Definable operations in general algebra and the theory of automata and flowcharts," IBM Vienna Report, 1969.

Berry, G. and Courcelle, B.

(1976) Program equivalence and canonical forms in stable discrete interpretations," *Automata Languages and Programming*, Third International Colloquium, (S. Michaelson and R,. Milner, Eds.) (1976) 168-189.

Birkhoff, G.

(1933) "On the combination of subalgebras," *Proceedings* Cambridge Phil. Soc. *29* (1933) 441-464.

(1935) "On the structure of abstract algebras," *Proceedings* Cambridge Phil. Soc. *31* (1935) 433-454.

(1967) *Lattice Theory* Amer. Math. Soc. Coloq. Pub. *25*, New York (1948). Revised edition, 1967.

Birkhoff, G. and Lipson, J.D.

(1970) "Heterogeneous algebras," *J. Combinatorial Theory 8* (1970) 115-133.

Blikle, A.

(1972) "Equational languages," *Information and Control* (1972) 134-147.

(1973a) "An algebraic approach to programs and their computations," *Proceedings*, Symposium and Summer School on the Mathematical Foundations of Computer Science, High Tatras, Czechoslovakia, 1973.

(1973b) "An algebraic approach to mathematical theory of programs," PRACE CO PAN. PAS REPORTS 119, 1973, Warsaw.

Blikle, A. and Mazurkiewicz, A.

(1972) "An algebraic approach to the theory of programs, algorithms languages and recursiveness. A concise text," Computation Centre, Polish Academy of Sciences, Warsaw, PKiN P.O. Box 22, August 1972.

Bloom, S.L.

(1976) "Varieties of ordered algebras," *JCSS 13* (1976) 200-212.

Bloom, S.L. and Elgot, C.C.

(1974) "The existence and construction of free iterative theories," IBM Research Report RC 4937 (1974). *JCSS 12* (1976) 305-318.

Bloom, S.L., Elgot, C.C. and Wright, J.B.

(1978) Solutions of the iteration equation and extensions of the scalar iteration operation," IBM Research Report RC 7029, March 1978.

Blum, E.K.

(1969) "Towards a theory of semantics and compilers for programming languages," *JCSS 3* (1969) 248-274.

Blum, E.K. and Estes, D.R.

(1977) "A generalization of the homomorphism concept," *Algebra Universalis 7* (1977) 143-161.

Brand, D.

(1978) "A note on data abstractions," *SIGPLAN Notices 13* (1978) 21-24.

Burstall, R.M.

(1969) "Proving properties of programs by structural induction," *Computer Journal* 1969.

(1972) "Some techniques for proving correctness of programs which alter data structures," *Machine Intelligence 7* (Eds. B. Meltzer and D. Michie) Edinburgh University Press (1972) 23-50.

(1972a) "An algebraic description of programs with assertions, verification and simulation," *Proceedings*, ACM Conference on Proving Assertions about Programs, Las Curces, New Mexico (1972) 7-14.

Burstall, R.M. and Landon, P.J.

(1969) "Programs and their proofs: an algebraic approach," *Machine Intelligence 4* (M. Meltzer and D. Michie, eds.) Edinburgh University Press(1969) 17-43.

Burstall, R. M. and Goguen, J. A.

(1977) "Putting Theories together to make Specifications," *Proceedings*, 1977 IJCAI, MIT, Cambridge, MA., August, 1977.

Burstall, R.M., and Thatcher, J.W.

(1974) "The algebraic theory of recursive program schemes," *Proceedings* AAAS Symposium on Category Theory Applied to Computation and Control, U. Mass. Press, Amherst (1974). *Lecture Notes in Computer Science 25* (1975) 126-131, Springer-Verlag.

Cohn, P.M.

(1965) *Universal Algebra*, Harper and Row, New York (1965).

Courcelle, B.

(1978) "Equational theories and equivalences of programs," IRIA Technical report, 1978.

Courcelle, B. and Guessarian, I.

(1977) "On some classes of interpretations," Rapport de Recherce No 253, IRIA, September 1977.

Courcelle, B. and Nivat, M.

(1976) "Algebraic families of interpretations," *Proceedings* 17th Annual IEEE Symposium on Foundations of Computing, Houston, Texas, October, 1976, pp. 137-146.

(1978) "The algebraic semantics of recursive program schemes," Proceedings 7th Symposium on Mathematical Foundations of Computer Science, *Lecture Notes in Computer Science 64* (1978) 16-30.

Courcelle, B. and Raoult, J-C.

(1978) "Completions of ordered magmas," Technical Report, IRIA, 1978.

Courcelle, B. and Vuillemin, J.

(1974) "Semantics and axiomatics of a simple recursive language," *Proceedings*, 6th ACM Symposium on Theory of Computing, Seattle, Wash. (1974) 13-26.

Damm, W.

(1977) "Higher type program schemes and their tree languages," Proceedings, Third G.I. Conference on Theoretical Computer Science, *Lecture Notes in Computer Science 48*, (1977)51-72.

Damm, W. and Fehr, E.

(1978) "On the power of self-application and higher type recursion," Proceedings 5th ICALP, *Lecture Notes in Computer Science 62* (1978) 177-191.

Damm, W., Fehr, E. and Indermark, K.

(1978) "Higher type recursion and self application as control structures," Formal Description of Programming Concepts, IFIP 1977.

de Bakker, J.W.

(1976) "Semantics and termination of non-deterministic recursive programs," *Automata languages and programming*, Third International Colloquium, University of Edinburgh (S. Michaelson and R. Milner, Eds.), Edinburgh University Press (1976) 435-478.

Doner, J. E.

(1970) "Tree acceptors and some of their applications," *JCSS 4* (1970) 406-451.

Dubinsky, A.

(1975) "Computation on arbitrary algebras," Queen Mary College, University of London. *Lecture Notes in Computer Science 37* (1975) 319-341.

Egli, H. and Constable, R.L.

(1976) "Computability concepts for programming language semantics," *Theoretical Computer Science 2* (1976) 133-145.

Ehrich, H.D.

(1977) "Algebraic semantics of type definitions and structured variables," Fundamentals of Computer Science, *Lecture Notes in Computer Science 56* (1977) 84-98.

(1978) "Extensions and implementations of abstract data type specifications," Abteilung Informatik, Universitat Dortmund.

Ehrich, H.D. and Lohberger V.G.

(1978) "Parametric specification of abstract data types, parameter substitution, and graph replacement," *Proceedings,* Workshop Graphentheoretische Konzepte om der Informatik, Hanser-Verlag, München, 1978.

Ehrig, H. and Kreowski, H. J.

(1977) "Some remarks concerning correct specification and implementation of abstract data types," Technical University of Berlin, Report 77-13, August 1977.

Ehrig, H., Kreowski, H. J., and Padawitz, P.

(1977) "Spezifikation, Korrektheit, und Implementierung von abstrakten Datenstrukturen," Einfurhungsskript zur LV Theorie von Datenstrukturen an der TU Berlin (SS 1977).

(1977a) "Stepwise specification and implementation of abstract data types," Technical University of Berlin, Report, November 1977.

Eilenberg, S. and Wright, J.B.

(1967) "Automata in general algebras," *Information and Control 11* (1967) 52-70.

Elgot, C.C.

(1969) "The external behavior of machines," IBM Research Report RC 2740, December, 1969. Presented, Third Hawaii International Conference on System Science, January 14016, 1970.

(1970) "The common algebraic structure of exit-automata and machines," *Computing 6* (1970) 349-370.

(1971) "Algebraic theories and program schemes," *Proceedings* Symp. on semantics of Algorithmic Languages, (Ed. E Engler) Springer-Verlag(1971) 71-88.

(1972) "Remarks on one-argument program schemes," *Formal Semantics of Programming Languages*, (Ed. R. Rustin) Prentice-Hall, New Jersey (1972) 59-64.

(1973) "Monadic computation and iterative algebraic theories," IBM Research Report RC 4564, October 1973. *Proceedings*, Logic Colloquium 1973, North Holland (1975)175-230.

(1974) "Matricial theories," IBM Research Report RC 4833, May 1974. *Journal of Algebra 42* (1976) 391-421.

(1976) "Structured programming with or without GO TO statements," *IEEE Transactions on Software Engineering SE-2* (1976) 41-53. Erratum and Corrigendum, *IEEE Transactions on Software Engineering*, September, 1976.

(1977) "Some geometrical categories associated with flow chart schemes," IBM Research Report RC 6534, May 1977. *Proceedings*, Conference on Fundamentals of Computation Theory, Poznan-Kornik, Poland, 1977.

(1978) "A representative strong equivalence class for accessible flowchart schemes," *Proceedings,* International Conference on Mathematical Studies of Information Processing, Kyoto, Japan (1978)

Elgot, C.C., Bloom, S.L. and Tindell, R.

(1978) "On the algebraic structure of rooted trees," *JCSS 16* (1978) 362-399.

Elgot, C.C. and Shepherdson, J.C.

(1977) "A semantically meaningful characterization of reducible flowchart schemes," IBM Research Report RC 6656, July, 1977.

Elgot, C.C. and Snyder, L.

(1977) "On the many facets of lists," *Theoretical Computer Science 5* (1977) 275-305.

Engelfreit, J. and Schmidt, E.M.

(1978) "IO and OI," *JCSS 16* (1978) 67-99.

Feferman, S.

(1969) "Set-theoretic foundations of category theory," Reports of the Midwest Category Seminar III (Ed. S. Mac Lane) *Lecture Notes in Mathematics 106* Springer-Verlag (1969) 201-247.

Gallier, J.H.

(1977) "Semantics and correctness of classes of deterministic recursive schemes," Ph.D. Dissertation, UCLA, 1977.

(1978) "Semantics and correctness of nondeterministic flowchart programs with recursive procedures," Preliminary Report, Department of Mathematics and Computer Science, USC, 1978.

Gécseg, F

(1977) "Universal algebras and tree automata," Fundamentals on Computer Science, *Lecture Notes on Computer Science 56* (1977) 98-113.

Gécseg, F and Horváth, G.

(1976) "On representation of trees and context-free languages by tree automata," Foundations of Control Engineering (1976) 161-168.

Gécseg, F. and Tóth, E.P.

(1977) "Algebra and logic in theoretical computer science," Mathematical Foundations of Computer Science, *Lecture Notes in Computer Science 53* (1977) 78-93.

Giarratana, V., Gimona, F. and Monianari, U.

(1976) "Observability concepts in abstract data type specification," *Lecture Notes in Computer Science 45* (Mathematical Foundations of Computer Science 1976, A. Mazurkiewicz, Ed.) (1976) 576-587

Ginali, S.M.

(1975) "A concrete introduction to algebraic automata theory," Quarterly Report #44, Institute for Computer Research, University of Chicago, February 1975.

(1976) "Iterative algebraic theories, infinite trees and program schemata." Dissertation, Department of Mathematics, University of Chicago, June 1976.

Ginsburg, S.

(1976) *The Mathematical Theory of Context-Free Languages*, McGraw Hill, New York, 1966.

Goguen, J.A.

(1972) "On mathematics in Computer Science education," IBM Research Report RC 3899, 1972.

(1974) "On homomorphisms, correctness, termination, unfoldments, and equivalence of flow diagram programs," *JCSS 8* (1974) 333-365.

(1974a) "Semantics of Computation," *Lecture Notes in Computer Science 25* (1975) 151-163, Springer-Verlag.

(1977) "Abstract errors for abstract data types," *Proceedings*, IFIP Working Conference on Formal Description of Programming Concepts, MIT (1977) 21.1-21.32.

(1978) "Some design principles and theory for OBJ-0, a language for expressing and executing algebraic specifications of programs," *Proceedings*, International Conference on Mathematical Studies of Information Processing, Kyoto, Japan (1978) 429-475.

(1979) "Some ideas in algebraic semantics," Department of Computer Science, UCLA Preprint, 1979.

Goguen, J. A. and Burstall, R.M.

(1978) "Some fundamental properties of algebraic theories: A tool for semantics of computation," Preprint 1978. Submitted to *Theoretical Computer Science*.

Goguen, J.A. and Meseguer, J.

(1977) "Correctness of recursive flow diagram programs," Semantics and Theory of Computation Report No. 8, Department of Computer Science, UCLA, July, 1977. *Lecture Notes in Computer Science 53* Springer-Verlag pp.580-595.

Goguen, J. A., and Tardo, J.

(1977) "OBJ-0 preliminary users manual," UCLA, Los Angeles, CA. 1977.

Graetzer, G.

(1968) *Universal Algebra*, Van Nostrand, Princeton (1968). Gussarian, Irene

(1976) "Semantic equivalence of program schemes and its syntactic characterization," *Automata, Languages and Programming*, Proceedings Third ICALP, University of Edinburgh, July 1976, pp189-201.

(1978) "Some applications of algebraic semantics," Mathematical Foundations of Computer Science 1978 (J. Winkowski, Ed.) *Lecture Notes in Computer Science 64* (1978) 257-266.

Guttag, J. V.

(1975) "The specification and application to programming of abstract data types," Univ. of Toronto, Computer Systems Research Group, Technical Report CSRG-59, September, 1975.

(1976) "Abstract data types and the development of data structures," supplement to Proc. Conf. on Data Abstraction, Definition, and Structure, *SIGPLAN Notices 8*, March, 1976.

(1977) "The algebraic specification of abstract data types," USC Computer Science Department, Draft Manuscript, April, 1977.

Guttag, J. V., Horowitz, E., and Musser, D. R.

(1976) "Abstract data types and software validation," Information Sciences Institute, Report RR-76-48 (Marina del Rey, Calif.).

(1976a) "The design of data type specifications," Information Sciences Institute, Report RR-76-48 (Marina del Rey, Calif.).

(1977) "Some extensions to algebraic specifications," Proceedings of an ACM Conference on language design for reliable software, *SIGPLAN Notices 12*, March 1977, pp 63-67.

Hatcher, W.S. and Rus, T.

(1976) "Context-free algebra," Submitted to *J. Cybernetics*.

Herrlich, H. and Strecker, G.E.

(1973) *Category Theory*, Allyn and Bacon (1973).

Higgins, P.J.

(1963) "Algebras with a scheme of operators," *Math. Nachr. 27* (1963) 115-132.

Hilfinger, Paul N.

(1978) Correspondence from Members, *SIGPLAN Notices 13*, (1978) 11-12.

Hoare, C. A. R.

(1972) "Proof of Correctness of Data Representations," *Acta Informatica 1* (1972) pp. 271-281.

Horowitz, Ellis and Sahni, Sartaj

(1976) *Fundamentals of Data Structures*, Computer Science Press, Inc., 1976.

Irons, E.T.

(1961) "A syntax directed compiler for ALGOL 60," *CACM 4* (1961) 51-55.

Kamin, S

(1979) "Rationalizing many-sorted algebraic theories," SUNY, Stony Brook, Draft manuscript. IBM Research Report RC 7574, March 1979.

Kapengst, H. and Reichel, H.

(1977) "Initial algebraic semantics for non-context-free languages," Fundamentals of Computer Science, *Lecture Notes in Computer Science 56* (1977) 120-127.

Karp, R.M.

(1959) "Some applications of logical syntax to digital computer programming," Harvard University Thesis (1959).

Knuth, D.E.

(1968) "Semantics of context-free languages," *Math. Sys. Th. 2* (1968) 127-145.

Kotov, V.E.

(1978) "An algebra for parallelism based on Petri nets," Mathematical Foundations of Computer Science, 1978, *Lecture Notes in Computer Science 64* (1978) 39-55.

Kühnel, W., Meseguer, J., Pfender, M. and Sols, I.

(1977) "Primitive recursive algebraic theories and program schemes," *Bulletin Austral. Math. Soc. 17* (1977) 207-233.

Landon, P.J.

(1970) A program machine symmetric automata theory," *Machine Intelligence 5* (Eds. B. Meltzer and D. Michie) Edinburgh University Press (1970) 99-120.

Lawvere, F.W.

(1963) "Functorial semantics of algebraic theories," *Proceedings*, Nat'l Acad. Sci. *50* (1963) 869-872.

Lehmann, D. J.

(1976) "Categories for fixpoint semantics," Theory of Computation Report No. 15, Department of Computer Science, University of Warwick. Also: *Proceedings* 17th Annual Symposium on Foundations of Computer Science (1976) 122-126.

Lehmann, D. J. and Smyth, M. B.

(1977) "Data types," University of Warwick, Department of Computer Science Report 19, May 1977.

(1977a) "Data Types," *Proceedings* 18th IEEE Symposium on Foundations of Computing, Providence, R.I., November 1977, pp. 7-12.

Levy, M.R.

(1978) "Data types with sharing and circularity," Ph.D. Thesis, Faculty of Mathematics, University of Waterloo; Report CS-78-26.

Lewis, C.H. and Rosen, B.K.

(1973) "Recursively defined data types, Pt. 1," *Proceedings*, ACM Symposium on Principles of Programming languages (1973) 125-138; Part 2, IBM Research Report, RC 4713 (1974).

Liskov, B. H. and Berzins, V.

(1977) "An appraisal of program specification," MIT, Laboratory for Computer Science, Computation Structures Memo 141-1, April, 1977.

Liskov, B.H. and Zilles, S.N.

(1974) "Programming with abstract data types," *Proceedings* ACM Symp. pn Very High Level Languages, *SIGPLAN Notices 9* (1974) 50-59.

Littrich, G. and Merzenich, W.

(1977) "Nets over many-sorted operator domains and their semantics," Fundamentals of Computer Science, *Lecture Notes on Computer Science 56* (1977) 240-245.

Lloyd, C

(1972) "Some concepts of universal algebra and their application to computer science," Report CSWP-1, Computing Centre, University of Essex, 1972.

McCarthy, J.

(1962) "Towards a mathematical science of computation," *Proceedings*, IFIP Congress, Amsterdam (1962) 21-28.

Mac Lane, S

(1971) *Category Theory for the Working Mathematician*, Springer-Verlag (1971)

Markowski, G.

(1974) "Categories of chain-complete posets," IBM Research Report RC 5100, October, 1974.

Markowski, G. and Rosen, B.K.

(1976) "Bases for chain complete posets," *IBM J. Res. Dev. 20* (1976) 138-147.

Maibaum, T. S. E.

(1972) "The characterization of derivation trees of context-free sets of terms as regular sets," *Proceedings*, 13th IEEE Symposium on Switching and Automata (1972) 224-230.

(1973) "Generalized grammars and homomorphic images of regular sets," U. of Waterloo Res. Report CS-73-30 (1973).

Majster, M. E.

(1977) "Data types, abstract data types and their specification problem," Technical University of Munich, Report TUM-INFO-7740, August 1977.

(1977a) "Limits of the algebraic specification of data types," *SIGPLAN Notices 12* (1977) 37-42.

(1978) Correspondence from Members, *SIGPLAN Notices 13* (1978) 8-10.

Manes, E.

(1976) *Algebraic Theories*, Graduate Texts in Mathematics 26, Springer-Verlag, 1976

Meseguer, Jose

(1977) "On order-complete universal algebra and enriched functorial semantics," Fundamentals on Computer Science, *Lecture Notes on Computer Science 56* (1977) 294-302.

(1978) "Completions, factorizations, and colimits for ω-posets," Semantics and Theory of Computation Report No. 13 (1979) UCLA.

(1979) "Ideal monads and Z-posets," Manuscript, University of California at Berkeley, Department of Mathematics, 1979.

Mezei, J. and Wright, J.B.

(1967) "Algebraic Automata and context-free sets," *Information and Control 11* (1967) 3-29.

Milner, R.

(1971) "An algebraic definition of simulation between programs," Stanford A.I. Memo AIM-142 and Computer Science Department report CS 205, 1971.

(1976) "Program semantics and mechanized proof," Mathematical Centre Tracts 82 (K.R. Apt and J.W. de Bakker (Eds.), Mathematisch Centrum, Amsterdam, 1976, pp. 3-44.

(1977) "Concurrent processes and their syntax," University of Edinburgh, Department of Computer Science, Internal Report CSR-2-77, May 1977. To appear *JACM*.

(1977a) "Flowgraphs and flow algebras," University of Edinburgh, Department of Computer Science, Internal Report, 1977. To appear *JACM*.

Milner, R. and Weyrauch, R

(1972) "Proving compiler correctness in a mechanized logic," *Machine Intelligence 7* (B. Meltzer and D. Michie, Eds.), Edinburgh University Press (1972) 51-72.

Minsky, M.

(1967) *Computation: Finite and Infinite Machines*, Prentice-Hall, New Jersey (1967).

Mitchell, B.

(1965) *Theory of Categories*, Academic Press, New York (1965).

Morris, F. L.

(1972) "Correctness of translations of programming languages," Stanford Computer Science Memo CS 72-303 (1972).

(1973) "Types are not sets," *Proceedings*, ACM Symposium on the Principles of Programming Languages, October 1973, pp. 120-124.

(1973a) "Advice on structuring compilers and proving them correct," *Proceedings*, ACM Symposium on Principles of Programming Languages, Boston (1973) 144-152.

Mosses, P

(1977) "Making denotational semantics less concrete,"Department of Computer Science, Aarhus University, 1977.

Nivat, M.

(1973) "Languages algebriques sur le magma libre et sémantique des schémas de programme," *Automata, Languages and Programming* (Ed. M. Nivat) North Holland, 1973.

(1975) "On the interpretation of polyadic recursive program schemes," Symposia Mathematica, Vol XV, Instituto Nationale di Alta Mathematica, Italy, 1975.

Obtulowicz, A.

(1977) "Functorial semantics of the type calculus," Fundamentals on Computer Science, *Lecture Notes on Computer Science 56* (1977) 302-308.

Pasztor, A.

(1979) "Surjections in the category of chain-complete posets, continuous universal algebras and related structures used in algebraic semantics of programming," Submitted to FCT 79 Berlin.

Plotkin, G.D. and Smyth, M.

(1977) "The category theoretic solution of recursive domain equations," Foundations of Computer Science, 1977. Extended version, University of Edinburgh Department of Artificial Intelligence Report No. 60, 1978.

Rabin, M.P., and Scott, D.

(1959) "Finite automata and their decision problems," *IBM J. Res. Dev. 3* (1959) 114-125.

Rattray, C.M.I. and Rus, T.

(1977) "Has-hierarchy, a mathematical device for computer system modeling," *Proceedings*, 1st International Symposium on Mathematical Modelling, Missouri (1977) 1-15.

Reichel, H

(1978) "Fundamentals of an algebraic theory of computation (Summary)," Preprint, Banach Center of Mathematics, Warsaw, May, 1978.

Reiterman, J.

(1977) "A more categorical model of universal algebra," Fundamentals on Computer Science, *Lecture Notes on Computer Science 56* (1977) 308-314.

Reynolds, J.C.

(1977) "Semantics of the domain of flow diagrams," *JACM 24* (1977) 484-503.

Rine, D.C.

(1974) "A category theory for programming languages," *MST 7* (1974) 304-317.

Rosen, B.K.

(1974) "Program equivalence and Context-free sets," *Proceedings* 13th IEEE Symp. on Switching and Automata Theory (1972) 7-18. Revised as IBM Research Report RC 4822 (1974).

Rus, Teodor

(1974) *Structuri de Date si Sisteme Operative*, Editura Academiei, Bucharest, Hungary, 1974.

(1976) "Context-free algebra: A mathematical device for compiler specification," *Lecture Notes in Computer Science 45* 1976.

(1979) *Data Structures and Operating Systems*, John Wiley and Sons, Chichester, 1979. (Updated translation of Rus (1974).)

Scott, Dana

(1970) "Outline of a mathematical theory of computation," *Proceedings*, 4th Annual Princeton Conference on Information Sciences and Systems (1970) 169-176.

(1971) "The lattice of flow diagrams," *Lecture Notes in Mathematics 182*, Semantics of Algorithmic Languages (E.Engler, Ed.) Springer-Verlag (1971) 311-366.

(1972) "Continuous Lattices," *Lecture Notes in Mathematics 274* (1972) 97-136.

(1972a) "Data types as lattices," unpublished notes, Amsterdam (1972); Oxford (1974). *SIAM J. Computing 5* (1976), 522-587.

Scott, D. and Strachey, C.

(1971) "Toward a mathematical semantics for computer languages," Technical Monograph PRG-6, Oxford University Computing Laboratory, Programming Reasearch Group, 1971.

Spitzen, J. and Wegbreit, B.

(1975) "The verification and synthesis of data structures," *Acta Informatica 4* (1975) 127-144.

Tarski, A.

(1968) "Equational logic and equational theories of algebras," *Contributions to Mathematical Logic* (K. Schütte, Ed), North Holland, Amsterdam, 1968.

Thatcher, J.W.

(1967) "Characterizing derivation trees of context-free grammars through a generalization of finite automata theory," *JCSS 1* (1967) 317-322.

(1973) "Tree automata: an informal survey," *Currents in Computing* (A. Aho, Ed.) Prentice-Hall (1973) 143-172.

Tiuryn, Jerzy

(1976) "Regular algebras," (Extended Abstract) Warsaw University (1976).

(1977) "Fixed points and algebras with infinitely long expressions, Part I: Regular algebras," *Lecture Notes in Computer Science 53* (1977) 513-523. Part II, *Lecture Notes in Computer Science 56* (1977) 332-340.

(1979) "Connection between regular algebras and rational algebraic theories," *Proceedings,* 2nd Workshop on Categorical and Algebraic Methods in Computer Science and Systems Theory, Dortmund, 1979.

Wagner, E.G.

(1971) "Languages for defining sets in arbitrary algebras," *Proceedings* 12th IEEE Symp. on Switching and Automata Theory, East Lansing, Mich. (1971).

(1971a) "An algebraic theory of recursive definitions and recursive languages," *Proceedings*, 3rd ACM Symp. of Theory of Computing, Shaker Heights, Ohio (1971).

(1973) "From algebras to programming languages," *Proceedings*, 5th ACM Sym. on Theory of Computing, Austin, Texas (1973).

(1974) "Notes on categories, algebras, and programming languages," Lecture Notes, Queen Mary College, London, England, Spring, 1974.

Wand, M.

(1972) "A concrete approach to abstract recursive definitions," *Automata, Languages and Programming* (M. Nivat, Ed.) North Holland (1972) 331-341.

(1974) "An algebraic formulation of the Chomsky hierarchy," *Lecture Notes in Computer Science 25* (Category Theory Applied to Computation and Control) Springer-Verlag (1974) 209-213.

(1977) "Final algebra semantics and data type extensions," Technical Report 65, Computer Science Department, Indiana University, 1977.

Wegner, P.

(1972) "A view of computer science education," *Amer. Math. Monthly*, February (1972) 168-172.

Whitehead, A.N.

(1898) "A Treatise on Universal Algebra," Cambridge at the University Press (1898).

Wojdylo, B.

(1977) "Many-sorted algebras and their application in computer science," Institute of Mathematics, Nicholas Copernicus University, Torun, Poland, 1977.

van Wijngaarden, E.

(1969) "Report on the algorithmic language ALGOL 68," *Numer. Math. 14* (1969) 79-218.

Zilles, S. N.

(1974) "Algebraic specification of data types," Computation Structures Group Memo 119, MIT, Cambridge, Mass. (1974) 28-52.

(1975) "An introduction to data algebras," working draft paper, IBM Research, San Jose, September, 1975.

TITLES IN THE SERIES MATHEMATICAL CENTRE TRACTS

(An asterisk before the MCT number indicates that the tract is under preparation).

A leaflet containing an order form and abstracts of all publications mentioned below is available at the Mathematisch Centrum, Kruislaan 413, 1098 SJ Amsterdam, The Netherlands. Orders should be sent to the same address.

MCT 1 T. VAN DER WALT, *Fixed and almost fixed points,* 1963. ISBN 90 6196 002 9.

MCT 2 A.R. BLOEMENA, *Sampling from a graph,* 1964. ISBN 90 6196 003 7.

MCT 3 G. DE LEVE, *Generalized Markovian decision processes, part I: Model and method,* 1964. ISBN 90 6196 004 5.

MCT 4 G. DE LEVE, *Generalized Markovian decision processes, part II: Probabilistic background,* 1964. ISBN 90 6196 005 3.

MCT 5 G. DE LEVE, H.C. TIJMS & P.J. WEEDA, *Generalized Markovian decision processes, Applications,* 1970. ISBN 90 6196 051 7.

MCT 6 M.A. MAURICE, *Compact ordered spaces,* 1964. ISBN 90 6196 006 1.

MCT 7 W.R. VAN ZWET, *Convex transformations of random variables,* 1964. ISBN 90 6196 007 X.

MCT 8 J.A. ZONNEVELD, *Automatic numerical integration,* 1964. ISBN 90 6196 008 8.

MCT 9 P.C. BAAYEN, *Universal morphisms,* 1964. ISBN 90 6196 009 6.

MCT 10 E.M. DE JAGER, *Applications of distrubutions in mathematical physics,* 1964. ISBN 90 6196 010 X.

MCT 11 A.B. PAALMAN-DE MIRANDA, *Topological semigroups,* 1964. ISBN 90 6196 011 8.

MCT 12 J.A.Th.M. VAN BERCKEL, H. BRANDT CORSTIUS, R.J. MOKKEN & A. VAN WIJNGAARDEN, *Formal properties of newspaper Dutch,* 1965. ISBN 90 6196 013 4.

MCT 13 H.A. LAUWERIER, *Asymptotic expansions,* 1966, out of print; replaced by MCT 54.

MCT 14 H.A. LAUWERIER, *Calculus of variations in mathematical physics,* 1966. ISBN 90 6196 020 7.

MCT 15 R. DOORNBOS, *Slippage tests,* 1966. ISBN 90 6196 021 5.

MCT 16 J.W. DE BAKKER, *Formal definition of programming languages with an application to the definition of ALGOL 60,* 1967. ISBN 90 6196 022 3.

MCT 17 R.P. VAN DE RIET, *Formula manipulation in ALGOL 60, part 1,* 1968.
ISBN 90 6196 025 8.

MCT 18 R.P. VAN DE RIET, *Formula manipulation in ALGOL 60, part 2,* 1968.
ISBN 90 6196 038 X.

MCT 19 J. VAN DER SLOT, *Some properties related to compactness,* 1968.
ISBN 90 6196 026 6.

MCT 20 P.J. VAN DER HOUWEN, *Finite difference methods for solving partial differential equations,* 1968. ISBN 90 6196 027 4.

MCT 21 E. WATTEL, *The compactness operator in set theory and topology,* 1968.
ISBN 90 6196 028 2.

MCT 22 T.J. DEKKER, *ALGOL 60 procedures in numerical algebra, part 1,* 1968.
ISBN 90 6196 029 0.

MCT 23 T.J. DEKKER & W. HOFFMANN, *ALGOL 60 procedures in numerical algebra, part 2,* 1968. ISBN 90 6196 030 4.

MCT 24 J.W. DE BAKKER, *Recursive procedures,* 1971. ISBN 90 6196 060 6.

MCT 25 E.R. PAËRL, *Representations of the Lorentz group and projective geometry,* 1969. ISBN 90 6196 039 8.

MCT 26 EUROPEAN MEETING 1968, *Selected statistical papers, part I,* 1968.
ISBN 90 6196 031 2.

MCT 27 EUROPEAN MEETING 1968, *Selected statistical papers, part II,* 1969.
ISBN 90 6196 040 1.

MCT 28 J. OOSTERHOFF, *Combination of one-sided statistical tests,* 1969.
ISBN 90 6196 041 X.

MCT 29 J. VERHOEFF, *Error detecting decimal codes,* 1969. ISBN 90 6196 042 8.

MCT 30 H. BRANDT CORSTIUS, *Exercises in computational linguistics,* 1970.
ISBN 90 6196 052 5.

MCT 31 W. MOLENAAR, *Approximations to the Poisson, binomial and hypergeometric distribution functions,* 1970. ISBN 90 6196 053 3.

MCT 32 L. DE HAAN, *On regular variation and its application to the weak convergence of sample extremes,* 1970. ISBN 90 6196 054 1.

MCT 33 F.W. STEUTEL, *Preservation of infinite divisibility under mixing and related topics,* 1970. ISBN 90 6196 061 4.

MCT 34 I. JUHÁSZ, A. VERBEEK & N.S. KROONENBERG, *Cardinal functions in topology,* 1971. ISBN 90 6196 062 2.

MCT 35 M.H. VAN EMDEN, *An analysis of complexity,* 1971. ISBN 90 6196 063 0.

MCT 36 J. GRASMAN, *On the birth of boundary layers,* 1971. ISBN 90 6196 064 9.

MCT 37 J.W. DE BAKKER, G.A. BLAAUW, A.J.W. DUIJVESTIJN, E.W. DIJKSTRA, P.J. VAN DER HOUWEN, G.A.M. KAMSTEEG-KEMPER, F.E.J. KRUSEMAN ARETZ, W.L. VAN DER POEL, J.P. SCHAAP-KRUSEMAN, M.V. WILKES & G. ZOUTENDIJK, *MC-25 Informatica Symposium* 1971.
ISBN 90 6196 065 7.

MCT 38 W.A. VERLOREN VAN THEMAAT, *Automatic analysis of Dutch compound words*, 1971. ISBN 90 6196 073 8.

MCT 39 H. BAVINCK, *Jacobi series and approximation*, 1972. ISBN 90 6196 074 6.

MCT 40 H.C. TIJMS, *Analysis of (s,S) inventory models*, 1972. ISBN 90 6196 075 4.

MCT 41 A. VERBEEK, *Superextensions of topological spaces*, 1972. ISBN 90 6196 076 2.

MCT 42 W. VERVAAT, *Success epochs in Bernoulli trials (with applications in number theory)*, 1972. ISBN 90 6196 077 0.

MCT 43 F.H. RUYMGAART, *Asymptotic theory of rank tests for independence*, 1973. ISBN 90 6196 081 9.

MCT 44 H. BART, *Meromorphic operator valued functions*, 1973. ISBN 90 6196 082 7.

MCT 45 A.A. BALKEMA, *Monotone transformations and limit laws* 1973. ISBN 90 6196 083 5.

MCT 46 R.P. VAN DE RIET, *ABC ALGOL, A portable language for formula manipulation systems, part 1: The language*, 1973. ISBN 90 6196 084 3.

MCT 47 R.P. VAN DE RIET, *ABC ALGOL, A portable language for formula manipulation systems, part 2: The compiler*, 1973. ISBN 90 6196 085 1.

MCT 48 F.E.J. KRUSEMAN ARETZ, P.J.W. TEN HAGEN & H.L. OUDSHOORN, *An ALGOL 60 compiler in ALGOL 60, Text of the MC-compiler for the EL-X8*, 1973. ISBN 90 6196 086 X.

MCT 49 H. KOK, *Connected orderable spaces*, 1974. ISBN 90 6196 088 6.

MCT 50 A. VAN WIJNGAARDEN, B.J. MAILLOUX, J.E.L. PECK, C.H.A. KOSTER, M. SINTZOFF, C.H. LINDSEY, L.G.L.T. MEERTENS & R.G. FISKER (Eds), *Revised report on the algorithmic language ALGOL 68*, 1976. ISBN 90 6196 089 4.

MCT 51 A. HORDIJK, *Dynamic programming and Markov potential theory*, 1974. ISBN 90 6196 095 9.

MCT 52 P.C. BAAYEN (ed.), *Topological structures*, 1974. ISBN 90 6196 096 7.

MCT 53 M.J. FABER, *Metrizability in generalized ordered spaces*, 1974. ISBN 90 6196 097 5.

MCT 54 H.A. LAUWERIER, *Asymptotic analysis, part 1*, 1974. ISBN 90 6196 098 3.

MCT 55 M. HALL JR. & J.H. VAN LINT (Eds), *Combinatorics, part 1: Theory of designs, finite geometry and coding theory*, 1974. ISBN 90 6196 099 1.

MCT 56 M. HALL JR. & J.H. VAN LINT (Eds), *Combinatorics, part 2: Graph theory, foundations, partitions and combinatorial geometry*, 1974. ISBN 90 6196 100 9.

MCT 57 M. HALL JR. & J.H. VAN LINT (Eds), *Combinatorics, part 3: Combinatorial group theory*, 1974. ISBN 90 6196 101 7.

MCT 58 W. ALBERS, *Asymptotic expansions and the deficiency concept in statistics*, 1975. ISBN 90 6196 102 5.

MCT 59 J.L. MIJNHEER, *Sample path properties of stable processes*, 1975. ISBN 90 6196 107 6.

MCT 60 F. GÖBEL, *Queueing models involving buffers*, 1975. ISBN 90 6196 108 4.

*MCT 61 P. VAN EMDE BOAS, *Abstract resource-bound classes, part 1*, ISBN 90 6196 109 2.

*MCT 62 P. VAN EMDE BOAS, *Abstract resource-bound classes, part 2*, ISBN 90 6196 110 6.

MCT 63 J.W. DE BAKKER (ed.), *Foundations of computer science*, 1975. ISBN 90 6196 111 4.

MCT 64 W.J. DE SCHIPPER, *Symmetric closed categories*, 1975. ISBN 90 6196 112 2.

MCT 65 J. DE VRIES, *Topological transformation groups 1 A categorical approach*, 1975. ISBN 90 6196 113 0.

MCT 66 H.G.J. PIJLS, *Locally convex algebras in spectral theory and eigenfunction expansions*, 1976. ISBN 90 6196 114 9.

*MCT 67 H.A. LAUWERIER, *Asymptptic analysis, part 2*, ISBN 90 6196 119 X.

MCT 68 P.P.N. DE GROEN, *Singularly perturbed differential operators of second order*, 1976. ISBN 90 6196 120 3.

MCT 69 J.K. LENSTRA, *Sequencing by enumerative methods*, 1977. ISBN 90 6196 125 4.

MCT 70 W.P. DE ROEVER JR., *Recursive program schemes: Semantics and proof theory*, 1976. ISBN 90 6196 127 0.

MCT 71 J.A.E.E. VAN NUNEN, *Contracting Markov decision processes*, 1976. ISBN 90 6196 129 7.

MCT 72 J.K.M. JANSEN, *Simple periodic and nonperiodic Lamé functions and their applications in the theory of conical waveguides*, 1977. ISBN 90 6196 130 0.

MCT 73 D.M.R. LEIVANT, *Absoluteness of intuitionistic logic*, 1979. ISBN 90 6196 122 X.

MCT 74 H.J.J. TE RIELE, *A theoretical and computational study of generalized aliquot sequences*, 1976. ISBN 90 6196 131 9.

MCT 75 A.E. BROUWER, *Treelike spaces and related connected topological spaces*, 1977. ISBN 90 6196 132 7.

MCT 76 M. REM, *Associations and the closure statement*, 1976. ISBN 90 6196 135 1.

MCT 77 W.C.M. KALLENBERG, *Asymptotic optimality of likelihood ratio tests in exponential families*, 1977. ISBN 90 6196 134 3.

MCT 78 E. DE JONGE & A.C.M. VAN ROOIJ, *Introduction to Riesz spaces*, 1977. ISBN 90 6196 133 5.

MCT 79 M.C.A. VAN ZUIJLEN, *Empirical distributions and rank statistics,* 1977. ISBN 90 6196 145 9.

MCT 80 P.W. HEMKER, *A numerical study of stiff two-point boundary problems,* 1977. ISBN 90 6196 146 7.

MCT 81 K.R. APT & J.W. DE BAKKER (Eds), *Foundations of computer science II,* part 1, 1976. ISBN 90 6196 140 8.

MCT 82 K.R. APT & J.W. DE BAKKER (Eds), *Foundations of computer science II,* part 2, 1976. ISBN 90 6196 141 6.

MCT 83 L.S. BENTHEM JUTTING, *Checking Landau's "Grundlagen" in the AUTOMATH system,* 1979. ISBN 90 6196 147 5.

MCT 84 H.L.L. BUSARD, *The translation of the elements of Euclid from the Arabic into Latin by Hermann of Carinthia (?) books vii-xii,* 1977. ISBN 90 6196 148 3.

MCT 85 J. VAN MILL, *Supercompactness and Wallman spaces,* 1977. ISBN 90 6196 151 3.

MCT 86 S.G. VAN DER MEULEN & M. VELDHORST, *Torrix I, A programming system for operations on vectors and matrices over arbitrary fields and of variable size.* 1978. ISBN 90 6196 152 1.

*MCT 87 S.G. VAN DER MEULEN & M. VELDHORST, *Torrix II,* ISBN 90 6196 153 X.

MCT 88 A. SCHRIJVER, *Matroids and linking systems,* 1977. ISBN 90 6196 154 8.

MCT 89 J.W. DE ROEVER, *Complex Fourier transformation and analytic functionals with unbounded carriers,* 1978. ISBN 90 6196 155 6.

*MCT 90 L.P.J. GROENEWEGEN, *Characterization of optimal strategies in dynamic games,* . ISBN 90 6196 156 4.

MCT 91 J.M. GEYSEL, *Transcendence in fields of positive characteristic,* 1979. ISBN 90 6196 157 2.

MCT 92 P.J. WEEDA, *Finite generalized Markov programming,* 1979. ISBN 90 6196 158 0.

MCT 93 H.C. TIJMS & J. WESSELS (eds), *Markov decision theory,* 1977. ISBN 90 6196 160 2.

MCT 94 A. BIJLSMA, *Simultaneous approximations in transcendental number theory,* 1978. ISBN 90 6196 162 9.

MCT 95 K.M. VAN HEE, *Bayesian control of Markov chains,* 1978. ISBN 90 6196 163 7.

MCT 96 P.M.B. VITÁNYI, *Lindenmayer systems: Structure, languages, and growth functions,* 1980. ISBN 90 6196 164 5.

*MCT 97 A. FEDERGRUEN, *Markovian control problems; functional equations and algorithms,* . ISBN 90 6196 165 3.

MCT 98 R. GEEL, *Singular perturbations of hyperbolic type,* 1978. ISBN 90 6196 166 1.

MCT 99 J.K. LENSTRA, A.H.G. RINNOOY KAN & P. VAN EMDE BOAS, *Interfaces between computer science and operations research,* 1978. ISBN 90 6196 170 X.

MCT 100 P.C. BAAYEN, D. VAN DULST & J. OOSTERHOFF (Eds), *Proceedings bicentennial congress of the Wiskundig Genootschap, part 1,* 1979. ISBN 90 6196 168 8.

MCT 101 P.C. BAAYEN, D. VAN DULST & J. OOSTERHOFF (Eds), *Proceedings bicentennial congress of the Wiskundig Genootschap, part 2,* 1979. ISBN 90 6196 169 6.

MCT 102 D. VAN DULST, *Reflexive and superreflexive Banach spaces,* 1978. ISBN 90 6196 171 8.

MCT 103 K. VAN HARN, *Classifying infinitely divisible distributions by functional equations,* 1978. ISBN 90 6196 172 6.

MCT 104 J.M. VAN WOUWE, *Go-spaces and generalizations of metrizability,* 1979. ISBN 90 6196 173 4.

*MCT 105 R. HELMERS, *Edgeworth expansions for linear combinations of order statistics,* . ISBN 90 6196 174 2.

MCT 106 A. SCHRIJVER (Ed.), *Packing and covering in combinatorics,* 1979. ISBN 90 6196 180 7.

MCT 107 C. DEN HEIJER, *The numerical solution of nonlinear operator equations by imbedding methods,* 1979. ISBN 90 6196 175 0.

MCT 108 J.W. DE BAKKER & J. VAN LEEUWEN (Eds), *Foundations of computer science III,* part 1, 1979. ISBN 90 6196 176 9.

MCT 109 J.W. DE BAKKER & J. VAN LEEUWEN (Eds), *Foundations of computer science III,* part 2, 1979. ISBN 90 6196 177 7.

MCT 110 J.C. VAN VLIET, *ALGOL 68 transput,* part I: *Historical review and discussion of the implementation model,* 1979. ISBN 90 6196 178 5.

MCT 111 J.C. VAN VLIET, *ALGOL 68 transput,* part II: *An implementation model,* 1979. ISBN 90 6196 179 3.

MCT 112 H.C.P. BERBEE, *Random walks with stationary increments and renewal theory,* 1979. ISBN 90 6196 182 3.

MCT 113 T.A.B. SNIJDERS, *Asymptotic optimality theory for testing problems with restricted alternatives,* 1979. ISBN 90 6196 183 1.

MCT 114 A.J.E.M. JANSSEN, *Application of the Wigner distribution to harmonic analysis of generalized stochastic processes,* 1979. ISBN 90 6196 184 X.

MCT 115 P.C. BAAYEN & J. VAN MILL (Eds), *Topological Structures II,* part 1, 1979. ISBN 90 6196 185 5.

MCT 116 P.C. BAAYEN & J. VAN MILL (Eds), *Topological Structures II,* part 2, 1979. ISBN 90 6196 186 6.

MCT 117 P.J.M. KALLENBERG, *Branching processes with continuous state space,* 1979. ISBN 90 6196 188 2.

MCT 118 P. GROENEBOOM, *Large deviations and asymptotic efficiencies*, 1980. ISBN 90 6196 190 4.

MCT 119 F.J. PETERS, *Sparse matrices and substructures, with a novel implementation of finite element algorithms*, 1980. ISBN 90 6196 192 0.

MCT 120 W.P.M. DE RUYTER, *On the asymptotic analysis of large-scale ocean circulation*, 1980. ISBN 90 6196 192 9.

MCT 121 W.H. HAEMERS, *Eigenvalue techniques in design and graph theory*, 1980. ISBN 90 6196 194 7.

MCT 122 J.C.P. BUS, *Numerical solution of systems of nonlinear equations*, 1980. ISBN 90 6196 195 5.

MCT 123 I. YUHÁSZ, *Cardinal functions in topology - ten years later*, 1980. ISBN 90 6196 196 3.

MCT 124 R.D. GILL, *Censoring and stochastic integrals*, 1980. ISBN 90 6196 197 1.

MCT 125 R. EISING, *2-D systems, an algebraic approach*, 1980. ISBN 90 6196 198 X.

MCT 126 G. VAN DER HOEK, *Reduction methods in nonlinear programming*, 1980. ISBN 90 6196 199 8.

MCT 127 J.W. KLOP, *Combinatory reduction systems*, 1980. ISBN 90 6196 200 5.

MCT 128 A.J.J. TALMAN, *Variable dimension fixed point algorithms and triangulations*, 1980. ISBN 90 6196 201 3.

MCT 129 G. VAN DER LAAN, *Simplicial fixed point algorithms*, 1980. ISBN 90 6196 202 1.

MCT 130 P.J.W. TAN HAGEN et al., *ILP Intermediate language for pictures*, 1980. ISBN 90 6196 204 8.

MCT 131 R.J.R. BACK, *Correctness preserving program refinements: Proof theory and applications*, 1980. ISBN 90 6196 207 2.

MCT 132 H.M. MULDER, *The interval function of a graph*, 1980. ISBN 90 6196 208 0.

MCT 133 C.A.J. KLASSEN, *Statistical performance of location estimators*, 1981. ISBN 90 6196 209 9.