A UNIFORM THEORY OF AUTOMATA

by

T.S.E. Maibaum

Research Report CS-74-15

Department of Applied Analysis and
Computer Science

University of Waterloo
Waterloo, Ontario, Canada

October 1974

A UNIFORM THEORY OF AUTOMATA

T.S.E. Maibaum

Dept. of Applied Analysis
and Computer Science
University of Waterloo
Waterloo, Ontario.

0. INTRODUCTION

In this paper we set out to generalise certain conventional classes of automata (finite, pushdown, nested stack). We do this to prove classical theorems equating:

i) the class of recognisable sets and the class of sets generated by regular grammars;

ii) the class of sets recognised by pushdown automata and the class of context free sets;

iii) the class of sets recognised by nested stack automata and the class of indexed sets.

These theorems, moreover, are proved not only for classes of sets of strings but also in the generalised setting of classes of sets of terms over a many-sorted alphabet. At the same time, the apparent close relationship between the different classes of automata is made evident. It turns out that pushdown (and nested stack) automata are nothing but finite automata in disguise.

We can motivate our generalisation in the following way: Recall that a conventional finite automaton M over some string alphabet Σ is defined as a 4-tuple $\langle Q,\delta,q_0,Q^F\rangle$ where Q is the set of states, $\delta:Q\times\Sigma\to Q$ is the next state function, q_0 is the initial state, and $Q^F\subseteq Q$ is the set of final states. δ is extended to $\bar{\delta}:\Sigma^*\to Q$ by: $\bar{\delta}(\epsilon)=q_0$ (ϵ is the empty string) and $\bar{\delta}(aw) = \delta(\bar{\delta}(w),a)$. The set $U \subseteq \Sigma^*$ recognised by M is $\{w \in \Sigma^* | \bar{\delta}(w) \in Q^F\}$; i.e. U is the inverse image under $\bar{\delta}$ of the set of final states Q^F.

Büchi showed that we could regard M as a unary (monadic) algebra in the following way: The carrier of our algebra is Q, q_0 is a constant

(initial) symbol and for each $a \in \Sigma$, a unary (monadic) function $\delta_a: Q \to Q$ such that $\delta_a(q) = \delta(q,a)$. Thus the operator domain of our algebra has unary symbols Σ and a single nullary symbol (corresponding to the initial state q_0). We have only to specify what we choose our final states to be. Thus the set recognised by such an automaton becomes a function of the set of final states chosen. More precisely, define $\bar{\delta}: \Sigma^* \to Q$ by $\bar{\delta}(\epsilon)=q_0$, $\bar{\delta}(aw) = \delta_a(\bar{\delta}(w))$ for all $a \in \Sigma$. (Note that this definition of $\bar{\delta}$ is essentially equivalent to that above). Then we write $bh_M(Q^F) = \{w \in \Sigma^* | \bar{\delta}(w) \in Q^F\}$ and say $U \subseteq \Sigma^*$ is recognisable if and only if $U = bh_M(Q^F)$ for some choice of final states Q^F. Thatcher and Wright and Doner used this point of view to obtain the so-called generalised finite automaton by allowing non-unary symbols (and, so, transition functions) in their automata.

Our formalisation of a generalised finite automaton, though equivalent to those of Thatcher and Wright and Doner, paves the way to the definition of generalised pushdown and nested stack automata. The whole concept turns on the formalisation of the statement: "We feed our finite automaton a set of strings Σ^* and it accepts only those strings which leave the finite automaton in a final state." We view this "feeding" process as running a certain "program" on our automaton ("machine"). This program is the string-forming algebra. That is, it has as operations functions which (left) concatenate a symbol to a string. Our program then runs on our automaton by concatenating a symbol $a \in \Sigma$ to a string w while the automaton makes the corresponding transition $\delta_a(q)$ (with $\bar{\delta}(w)=q$). Thus, started out in the initial configuration pair of empty string and initial state, our program/machine pair will produce a

sequence of ordered pairs (w_0,q_0), $(w_1,q_1),\ldots,(w_n,q_n)$ where $w_{i+1}=aw_i$ and $\delta_a(q_i)=q_{i+1}$ for $0\leq i\leq n-1$. Then our automaton accepts a string w if and only if (w,q), (for q a final state) appears at the end of such a sequence. The "choosing" of our final states can be done by a map which picks out certain states in our automaton while the set accepted by an automaton is specified (as a function of the final states) by a "projection" map which selects the words from the ordered pairs with final states as the second element.

In Section 1, we proceed to introduce many-sorted alphabets and some basic algebraic concepts. In Section 2, we define different classes of grammars over many-sorted alphabets and state some results concerning them. In Section 3, we present an algebraic theory of programs and machines. In Section 4, we apply the ideas of Section 3 to those of Section 2 to define a generalised theory of automata. In Section 5, we summarise our results and suggest possible extensions of our work.

1. Many-sorted Structures

Let I be any set, called the set of sorts. A many-sorted alphabet (sorted by I) is an indexed family of sets $\Omega=\{\Omega_{<w,i>}\}_{<w,i>\in I^*\times I}$. $\Omega_{<w,i>}$ is the set of symbols of type $<w,i>$ while $f\in\Omega_{<w,i>}$ is said to be of type $<w,i>$, arity w, sort i, and rank $\ell(w)$.($\ell(w)$ is the length of the string w). A symbol of type $<\lambda,i>$ (λ the empty string) is called a constant (or nullary) symbol of sort i.

An Ω-structure A_Ω (or just A if the alphabet is obvious from the context) is an indexed family of sets $A=\{A_i\}_{i\in I}$ together with a

family of assignments $\alpha_{<w,i>}:\Omega_{<w,i>} \rightarrow [A^w \rightarrow A_i]$ from symbols in $\Omega_{<w,i>}$ to relations from $A^w=A_{w_0}\times\ldots\times A_{w_{n-1}}$ to A_i. ($[A^w\rightarrow A_i]$ is the set of relations from A^w to A_i). If $\alpha_{<w,i>}:\Omega_{<w,i>} \rightarrow (A^w\rightarrow A_i)$ (where $(A^w\rightarrow A_i)$ is the set of functions from A^w to A_i) then we call A_Ω an Ω-algebra. In either case, we denote $\alpha_{<w,i>}(f)$ for $f \in \Omega_{<w,i>}$ by f_A (or by f itself if it is obvious from the context which we mean). A is called the carrier of A_Ω.

Example 1: Let $I=\{S,\Omega,\Delta\}$. S is the sort "states", Ω is the sort "input symbols", and Δ is the sort "output symbols". Let $A=\{A_i\}_{i\in I}$ be a family of non-void sets such that A_Δ (the set of output symbols) is finite. We have $\Sigma_{<S\Omega,S>}=\{\delta\}$ and $\Sigma_{<S\Omega,\Delta>}=\{\S\}$ while all other $\Sigma_{<w,i>}=\phi$. We assign to δ the change of state operation δ_A of M and to § the output function $\S_A$ of M, where, of course, M is a sequential machine with states A_S, input symbols A_Ω, and output symbols A_Δ.

Analogously, we could describe a finite automaton as a Σ'-algegra B (although this is not the approach we will take in the remainder of this paper), with sorting set $I'=\{S,\Omega\}$, where B_S is a finite set of states and B_Ω is a finite input alphabet. Σ' consists of $\Sigma'_{<S\Omega,S>}=\{\delta\}$, called the next state function, and the set $\Sigma'_{<\lambda,S>}=\{q_0\}$, called the initial state of B.

Example 2: Let $\Omega=\{\Omega_n\}_{n\in\underline{N}}$ be a ranked alphabet (i.e. an alphabet indexed by the natural numbers $\underline{N}$). Let $I=\{1\}$. To each $f \in \Omega_n$ (an operator symbol of rank n) assign the type $<\underbrace{1\ldots1}_{n \text{ times}},1>$. So ranked alphabets are special cases of many-sorted alphabets: namely those sorted by an index set consisting of a single element. Algebras over ranked alphabets will thus have carriers consisting of an indexed family of sets containing exactly one set.

Fix the alphabet Ω. Let A and B be Ω-structures. A homomorphism $\phi:A \to B$ is an indexed set of functions $\{\phi_i:A_i \to B_i\}_{i\in I}$ which preserves the structure of the algebra A. That is, for any $f \in \Omega_{<w,i>}$ and $(a_0,\ldots,a_{n-1}) \in A^w$ (i.e., $a_j \in A_{w_j}$ for $0\leq j\leq n-1$), $\phi_i(f_A(a_0,\ldots,a_{n-1})) = f_B(\phi_{w_0}(a_0),\ldots,\phi_{w_{n-1}}(a_{n-1}))$. Homomorphisms which are injective, surjective or bijective are called <u>monomorphisms</u>, <u>epimorphisms</u>, and <u>isomorphisms</u>, respectively.

A <u>congruence</u> on A is an indexed family of equivalence relations $q=\{q_i\}_{i\in I}$, q_i defined on A_i, such that if $f\in\Omega_{<w,i>}$ and $a_j,b_j \in A_{w_j}$ for $0\leq j\leq n-1$, then the following substitution property holds: $a_j q_w b_j$ for $0\leq j\leq n-1$ implies aq_ib where $(a_0,\ldots,a_{n-1},a)$, $(b_0,\ldots,b_{n-1},b) \in f_A$.

The <u>direct product</u> $A\times B$ of A and B is the structure with carrier $\{A_i \times B_i\}_{i\in I}$ and relations defined componentwise.

Let $X = \{X_i\}_{i\in I}$ be any indexed family of sets. Then $W_\Omega(X) = \{W_\Omega(X)_i\}_{i\in I}$, the indexed family of sets of <u>words</u> (or <u>terms</u>, or <u>expressions</u>) on the alphabet Ω and <u>generators</u> X is the least family of sets satisfying:

(0) $X_i \cup \Omega_{<\lambda,i>} \subseteq W_\Omega(X)_i$;

(i) For each $f \in \Omega_{<w,i>}$ and

$$(t_0,\ldots,t_{n-1}) \in W_\Omega(X)^w,\ ft_0\ldots t_{n-1} \in W_\Omega(X)_i.$$

If each $x_i=\phi$, we denote $W_\Omega(\{\phi\}_{i\in I})$ by W_Ω. We can make $W_\Omega(X)$ into an Ω-algebra (called the <u>word algebra</u> or <u>algebra of expressions</u>) by defining the assignment of operations for $f \in \Omega_{<w,i>}$ as follows: $f_{W_\Omega(X)}(t_0,\ldots,t_{n-1}) = ft_0\ldots t_{n-1}$. That is, $W_\Omega(X)$ is the "expression-forming algebra".

Example 3: Let $I=\{0,1\}$, $\Omega_{<\lambda,0>} = \{\lambda\}$, $\Omega_{<\lambda,1>} = \{a,b\}$, $\Omega_{<10,0>} = \{*\}$, and $\Omega_{<11,1>} = \{+\}$. Let $X_1 = \{x\}$ and $X_0 = \phi$. Then we may describe the set $W_\Omega(X)_0$ as terms of the form λ, $*a\lambda$, $*b\lambda$, $*+aa\lambda$, $*+ax\lambda$, etc. (generally terms of sort 0 are of the form $*+(\ )*(\ )$). Elements of $W_\Omega(X)_1$ are of the form a,b,x,+aa, +bb, +ab, +ba, +ax, +xa, +bx, +xb, ++aa+aa, etc. (generally terms of sort 1 are of the form $++(\ )+(\ )$). □

Let $X_w = \{x_{0,w_0},\ldots,x_{n-1,w_{n-1}}\}$ for some $w=w_0\ldots w_{n-1} \in I^*$. We say X_w is indexed by $w \in I^*$. As an I-sorted alphabet, we have $(X_w)_i = \{x_{j,i} \in X_w \mid j<n\}$. We shall denote $W_\Omega(\{(X_w)_i\}_{i\in I})$ by $W_\Omega(X_w)$. Similarly, we define $X_V = \{X_w \mid w \in V\}$ for some $V \subseteq I^*$. $W_\Omega(X_V)$ is then defined in the obvious way.

Example 4: Let Ω be as in Example 3 and let $w=10$. Then $X_w=\{x_{0,1},x_{1,0}\}$. $W_\Omega(X_w)_0 = \{\lambda,*a\lambda, *b\lambda, *x_{0,1}x_{1,0}, \text{etc.}\}$. $W_\Omega(X_w)_1$ is the set of expressions of the form $++(\ )+(\ )$ with only $x_{0,1}$ appearing in the expressions. □

Theorem 1: Let A be any Ω-algebra, X any family of generators, and $\psi = \{\psi\}_{i\in I}$ any indexed family of assignments $\{\psi_i : X_i \to A_i\}$. Then ψ extends in a unique way to a homomorphism $\bar{\psi} : W_\Omega(X) \to A$. In particular, there is a unique homomorphism from W_Ω to A. □

We now proceed to briefly explain how to obtain a number of different algebras closely related to a given algebra A_Ω. First, the raised algebra (or subset algebra or relational algebra) B(A) of an Ω-structure A: The carrier of B(A) is the indexed family of sets $\{2^{A_i}\}_{i\in I}$. The operations of B(A) are defined as follows: Given $(S_0,\ldots,S_{n-1}) \in B(A)^w$, $f \in \Omega_{<w,i>}$ define $f_{B(A)}(S_0,\ldots,S_{n-1}) = S_n \in B(A)_i$

if and only if $S_n = \{a_n | \exists a_0 \ldots \exists a_{n-1}$ such that $(a_0,\ldots,a_n) \in f_A$ and $a_j \in S_j$ for $0 \leq j \leq n-1\}$.

Let $e \in W_\Omega(X_w)_i$. We say e is an expression of <u>type</u> $\langle w,i\rangle$. e defines a function, called a <u>derived operation</u>, v_e on the carrier of an Ω-algebra A in the following way: For $(a_0,\ldots,a_{n-1}) \in A^w$, $v_e(a_0,\ldots,a_{n-1}) = \bar{\psi}(e)$ where $\bar{\psi}$ is the (unique) homomorphism generated by the assignments $\psi_{w_j}(x_{j,w_j}) = a_j$ for $0 \leq j \leq n-1$. v_e is called a derived operation because it is derived from the operations of the algebra by composition.

Consider the new I-sorted alphabet Ω' where $\Omega'_{\langle w,i\rangle} = W_\Omega(X_w)_i$ for w the arity of some symbol in Ω and $\Omega'_{\langle w,i\rangle} = \phi$ otherwise. We make the Ω-algebra A into an Ω'-algebra by having $e \in \Omega'_{\langle w,i\rangle}$ (i.e. $e \in W_\Omega(X_w)_i$) name the operation $v_e: A^w \to A_i$ as defined above. Denote this new algebra by $C(A)$. C(A) is called the <u>(initial) completion</u> of A.

Now let $D(I) = \{\langle w,i\rangle | w$ is the arity of some $f \in \Omega$ and $i \in I\}$. That is, the elements of D(I) are just those of the subset of $I^* \times I$ with the first argument an arity of a symbol in Ω. That is, the elements of D(I) are just the types of the operations in the algebra $C(\Omega)$. We use D(I) **to sort an alphabet $D(\Omega)$, called the <u>derived alphabet</u> of Ω, which is defined in** the following way:

(i) **If $f \in \Omega_{\langle w,i\rangle}$, then $f \in D(\Omega)_{\langle\lambda,\langle w,i\rangle\rangle}$.**
That is, f is a nullary of type $\langle\lambda,\langle w,i\rangle\rangle$ in $D(\Omega)$;

(ii) For each w an arity of an element in Ω, $\ell(w) = n > 0$, let $\delta^j_w \in D(\Omega)_{\langle\lambda,\langle w,w_{j-1}\rangle\rangle}$ for $1 \leq j \leq n$. These symbols are called <u>projection</u> symbols;

(iii) For each $\langle w,v,i\rangle \in I^+ \times I^* \times I$, let
$c_{\langle w,v,i\rangle} \in D(\Omega)_{\langle\langle w,i\rangle\langle v,w_0\rangle\ldots\langle v,w_{n-1}\rangle,\langle v,i\rangle\rangle}$.

These are called <u>composition</u> symbols.

We define an algebra D(A), called the <u>derived algebra</u> of A, as follows:

(i) The carrier of D(A) of sort $\langle w,i\rangle \in D(I)$ is the set of operations of type $\langle w,i\rangle$ of the algebra C(A). Or, equivalently, it is the set of derived operations of type $\langle w,i\rangle$ of the algebra A;

(ii) The operator domain of D(A) is $D(\Omega)$ and assignment of operations to $D(\Omega)$ is done by:

(a) assigning to $c_{\langle w,v,i\rangle}$ an operation of composition with first argument of sort $\langle w,i\rangle$, n arguments of sort $\langle v,w_j\rangle$ for $0 \leq j \leq n-1$ and result of sort $\langle v,i\rangle$. That is, $c_{\langle w,v,i\rangle}$ composes a derived operation of type $\langle w,i\rangle$ with n derived operations of type $\langle v,w_j\rangle$ $(0 \leq j \leq n-1)$ and the result is a derived operation of type $\langle v,i\rangle$;

(b) Assigning to $f \in D(\Omega)_{\langle\lambda,\langle w,i\rangle\rangle}$, where $f \in \Omega_{\langle w,i\rangle}$, the operation defined by f_A. That is, f is assigned the derived operation f_A; and

(c) Assigning to $\delta_w^j \in D(\Omega)_{\langle\lambda,\langle w,w_{j-1}\rangle\rangle}$ the operation of projection. That is, given $c_{\langle w,v,w_{j-1}\rangle}$ and $t_k \in D(A)_{\langle v,w_k\rangle}$ for $0 \leq k \leq n-1$, then $c_{\langle w,v,w_{j-1}\rangle}(\delta_w^j,t_0,\ldots,t_{n-1}) = t_{j-1}$.

Thus the derived algebra D(A) of an algebra A has as carrier the derived operations of the algebra A and has as operations composition and projection functions. We will be mainly interested in the algebra W_Ω, its derived algebra $D(W_\Omega)$ and the word algebra on the derived alphabet

$D(\Omega)$, namely $W_{D(\Omega)}$. Note that if Ω is finite, then so is $D(\Omega)$. Similarly, if A is finite, so is D(A). Let YIELD:$W_{D(\Omega)} \rightarrow D(W_{\Omega})$ be the unique homomorphism.

Example 5: Consider the unary alphabet Δ where $\Delta_0 = \{\lambda\}$ and $\Delta_1 = \{a,b\}$. (That is, we are considering the string alphabet {a,b}). From Example 2, we know that Δ is a many-sorted alphabet sorted by the set I = {i}, for example. That is, $\Delta_0 = \Delta_{<\lambda,i>}$ and $\Delta_1 = \Delta_{<i,i>}$. So $D(I) = \{<\lambda,i>,<i,i>\}$. $D(\Delta)$ is given as follows: $D(\Delta)_{<\lambda,<\lambda,i>>} = \{\lambda\}$; $D(\Delta)_{<\lambda,<i,i>>} = \{a,b,\delta^1_i\}$;

$D(\Delta)_{<<i,i><i,i>,<i,i>>} = \{+\}$; and $D(\Delta)_{<<i,i><\lambda,i>,<\lambda,i>>} = \{*\}$. If we represent $<\lambda,i>$ by 0 and $<i,i>$ by 1, then $D(I) = \{0,1\}$ and $D(\Delta)_{<\lambda,0>} = \{\lambda\}$, $D(\Delta)_{<\lambda,1>} = \{a,b,\delta^1_1\}$, etc. Note the close relationship of this alphabet $D(\Delta)$ to the alphabet Ω of Example 3.

Now let us start with $D(\Delta)$ and try to obtain $D(D(\Delta))$. $D(D(I)) = \{<\lambda,0>,<\lambda,1>,<11,1>\ <10,1>,<11,0>,<10,0>\}$.

$$D(D(\Delta))_{<\lambda,<\lambda,0>>} = \{\lambda\};$$

$$D(D(\Delta))_{<\lambda,<\lambda,1>>} = \{a,b,\delta^1_1\};$$

$$D(D(\Delta))_{<\lambda,<11,1>>} = \{\delta^1_{11}, \delta^2_{11}, +\};$$

$$D(D(\Delta))_{<\lambda,<10,1>>} = \{\delta^1_{10}\};$$

$$D(D(\Delta))_{<\lambda,<11,0>>} = \phi;$$

$$D(D(\Delta))_{<\lambda,<10,0>>} = \{\delta^2_{10}, *\} \text{ and}$$

$$c_{<w,v,i>} \in D(D(\Delta))_{<<w,i><v,w_0>\ldots<v,w_{n-1}>,<v,i>>}$$

for each $(w,v,i) \in \{10,11\} \times \{\lambda,10,11\} \times D(I)$. □

Remark: If we are given two I-sorted alphabets Ω and Σ such that $\Omega \subseteq \Sigma$ (that is, $\Omega_{<w,i>} \subseteq \Sigma_{<w,i>}$ for each $<w,i> \in I^* \times I$), then we will adopt the following convention for the meaning of $D(\Omega)$: We will consider $D(\Omega)$ to be a subset (in the extended sense above) of $D(\Sigma)$ by identifying it with the following (extended) subset of $D(\Sigma)$: each f in Ω is in $D(\Omega)$, each δ_w^j is in $D(\Omega)$ for w an arity of some symbol in Ω and $1 \leq j \leq n$, and each $c_{<w,v,i>}$ is in $D(\Omega)$ for w,v arities of symbols in Ω. Thus if $\Omega \subseteq \Sigma$, then we are able to say $D(\Omega) \subseteq D(\Sigma)$. (Note that $D(\Omega)$, as defined according to the rules for obtaining the derived alphabet of Ω, is not in general a subset of $D(\Sigma)$, even if $\Omega \subseteq \Sigma$).

2. Formal Languages

In this section we summarise some of the definitions and results of Maibaum (2), (3). Let $A = \{A_i\}$ and $B = \{B_i\}$ be two families of indexed sets. We extend all set theoretic operations from sets to indexed families of sets componentwise. For example, $A \cup B = \{A_i \cup B_i\}$ and $A \times B = \{A_i \times B_i\}$. A many-sorted alphabet is finite if the disjoint union of the sets of different types in the alphabet is finite and I is finite. A many-sorted structure (algebra) is finite if the alphabet is finite and the carrier of each sort is a finite set.

Let Ω be a finite alphabet. A finite Ω-automaton is a finite Ω-structure A. If A is an algebra, we say A is a deterministic automaton; otherwise A is said to be non-deterministic. An indexed family of sets $U \subseteq W_\Omega$ is said to be recognisable if there exists an Ω-automaton A (deterministic or non-deterministic) and a choice of an indexed family of final

states $A^F \subseteq A$ such that $bh_A(A^F) = U$. $bh_A(A^F) = \{t \in (W_\Omega)_i | h_i^A(t) \in A_i^F\}_{i \in I}$ is the behaviour of the automaton A with respect to the choice of the family of final states A^F and h^A is the unique homomorphism from W_Ω to A. We can prove a number of classical results: the equivalence of the classes of deterministic and non-deterministic automata, the closure of the class of sets accepted by finite automata under the boolean operations, the decidability of the emptyness, finiteness, and equivalence problems, etc.

Let Σ be a many-sorted alphabet and $\Omega \subseteq \Sigma$. A <u>semi-Thue relation</u> (or <u>production</u>) on terms (over Σ) is a pair $p = (s,s')$ usually written $s \to s'$, where $s,s' \in W_\Sigma(X_w)_i$, some $i \in I$, and no more elements of X_w appear in s' than in s (although the variables that do appear in s can be repeated or omitted in s').

We define the relation $\overset{=}{p}>$ on W_Σ as follows for $t,t' \in (W_\Sigma)_j$, some $j \in I$:

$t \overset{=}{p}> t'$ if and only if there exists $\bar{t} \in W_\Sigma(X_i)_j$ (here i is considered to be the string of length one consisting of the symbol i) and $(t_0,\ldots,t_{n-1}) \in (W_\Sigma)^W$ such that $Sub_i(\bar{t}; Sub_w(s;t_0,\ldots,t_{n-1})) = t$ and $Sub_i(\bar{t}; Sub_w(s';t_0,\ldots,t_{n-1})) = t'$.

$Sub_v(-; t_0,\ldots,t_{n-1}): W_\Sigma(X_w) \to W_\Sigma$ is the homomorphism generated by the assignments $\phi_{w_j}: x_{j,w_j} \to t_j$ for $0 \leq j \leq n-1$. Intuitively, $t \overset{=}{p}> t'$ if t' can be obtained from t by replacing a subterm of t of the form $Sub_w(s;t_0,\ldots,t_{n-1})$ by a subterm of the form $Sub_w(s';t_0,\ldots,t_{n-1})$.

A <u>semi-Thue grammar</u> is a 4-tuple $G = \langle \Omega, \Sigma, P, Z \rangle$ where:

(i) Σ is a finite, many-sorted alphabet sorted by I;

(ii) $\Omega \subseteq \Sigma$ is a distinguished subset called the terminal alphabet. We call $N = \Sigma - \Omega$ the non-terminal (or auxiliary) alphabet;

(iii) P is a finite set of semi-Thue productions;

(iv) Z is the axiom and $Z \in N_{<\lambda,i>}$ for each $i \in I$.

Define the relation $\underset{G}{\Rightarrow}$ on W_Σ as follows for $t,t' \in (W_\Sigma)_i$: $t \underset{G}{\Rightarrow} t'$ if and only if there is some $p \in P$ such that $t \underset{p}{=>} t'$. Let $\overset{*}{\underset{G}{=>}}$ be the reflexive, transitive closure of $\underset{G}{=>}$. The relation $\overset{*}{\underset{G}{=>}}$ is called derivation and we will drop the G if it is obvious which grammar we mean. The indexed family of sets generated by the grammar $G = <\Omega, \Sigma, P, Z>$ is $L(G) = \{t \in (W_\Omega)_i \mid Z \overset{*}{\underset{G}{=>}} t\}_{i \in I}$.

In this paper, we will study two restrictions on the types of productions allowed in a semi-Thue grammar. A grammar $G = <\Omega, \Sigma, P, Z>$ is called regular if:

(i) $\Sigma_{<w,i>} = \Omega_{<w,i>}$ for $\ell(w)>0$. i.e., we have only nullary non-terminal symbols;

and (ii) $p \in P$ is of the form $A \to fB_0 \ldots B_{n-1}$ where $A \in N_{<\lambda,i>}$, $f \in \Omega_{<w,i>}$ and $B_j \in N_{<\lambda,w_j>}$ for $0 \leq j \leq n-1$.

A grammar $G = <\Omega, \Sigma, P, Z>$ is called context free if all productions $p \in P$ are of the form $A(x_{0,w_0},\ldots,x_{n-1,w_{n-1}}) \to t$ where $A \in N_{<w,i>}$ and $t \in W_\Sigma(X_w)_i$. (See also Rounds (1) and (2)).

Theorem 1: The class of indexed families of sets recognised by finite

Ω-automata is the same as the class generated by regular grammars over Ω. □

Example 1: Let I = {0,1}; $\Omega_{<\lambda,0>} = \{\lambda\}$, $\Omega_{<\lambda,1>} = \{a,b\}$, $\Omega_{<10,0>} = \{*\}$, $\Omega_{<11,1>} = \{+\}$; $\Sigma_{<\lambda,0>} - \Omega_{<\lambda,0>} = \{Z,E\}$, $\Sigma_{<\lambda,1>} - \Omega_{<\lambda,1>} = \{A,B,C,D,F,G\}$.

P = {$Z \to \lambda$, $Z \to *FE$, $Z \to *GE$, $C \to +AD$, $C \to +AB$, $D \to +CB$, $E \to \lambda$, $F \to +AB$, $G \to +AD$, $A \to a$, $B \to b$}.

Then G = $<\Omega, \Sigma, P, Z>$ is a regular grammar and L(G) = {{λ,* + abλ,* +a++abbλ, *+a++a++abbbλ, etc},ϕ}. (Note L(G) is an indexed family of sets; in this case, the second element is empty).

If we change G to G' by allowing $Z \in \Sigma_{<\lambda,1>} - \Omega_{<\lambda,1>}$ and add $Z \to +AB$, $Z \to +AD$ to P, then L(G') = {{λ,*+abλ, *+a++abbλ,...},{+ab,+a++abb, +a++a++abbb,...}}. Note that YIELD (L(G)) = {{$a^n b^n \lambda \mid n \geq 0$},$\phi$} and YIELD (L(G')) = {{$a^n b^n \lambda \mid n \geq 0$},{$a^n b^n y \mid n > 0$}} for $y \in X_1$.□

Example 2: Let I = {0,1}; $\Omega_{<\lambda,0>} = \{\lambda\}$, $\Omega_{<\lambda,1>} = \{a\}$, $\Omega_{<10,0>} = \{*\}$, $\Omega_{<11,1>} = \{+\}$;

$\Sigma_{<\lambda,0>} - \Omega_{<\lambda,0>} = \{Z\}$, $\Sigma_{<1,1>} = \{B\}$; P = {$Z \to *a\lambda$, $Z \to *B(a)\lambda$, $B(x) \to B(+xx)$, $B(x) \to +xx$}.

Then G = $<\Omega, \Sigma, P, Z>$ is a context free grammar and L(G) = {*aλ,*+aaλ, *++aa+aaλ, etc},ϕ}.

If we change G to G' by allowing $Z \in \Sigma_{<\lambda,1>} - \Omega_{<\lambda,1>}$ and adding $Z \to B(a)$ and $Z \to a$ to P , then L(G') = {{*aλ, *+aaλ, *++aa+aaλ,...}, {a,+aa,++aa+aa,...}}. Then YIELD (L(G)) = {{$a^{2^n} \lambda \mid n \geq 0$},$\phi$} and YIELD (L(G')) = {{$a^{2^n} \lambda \mid n \geq 0$},{$a^{2^n} y \mid n \geq 0$}} for $y \in X_1$.□

Remark: If, as for L(G) above and in the previous example, one of the elements of the indexed family of sets L(G) is empty, we will omit any

Theorem 3: Given a context free grammar G, we can effectively find another grammar G' in Chomsky normal form such that L(G) = L(G'). □

Example 3:

Let G be the context free grammar of Example 2. Consider the alphabet $\Sigma' \supseteq \Omega$ such that $\Sigma'_{<\lambda,0>} - \Omega_{<\lambda,0>} = \{Z,L\}$, $\Sigma'_{<\lambda,1>} - \Omega_{<\lambda,1>} = \{A,C\}$, $\Sigma'_{<1,1>} = \{B,D\}$, $\Sigma'_{<10,0>} - \Omega_{<10,0>} = \{S\}$ and P'= {Z → S(A,L), Z → S(C,L), C → B(A), S(x,y) → *xy, B(x) → B(D(x)), B(x) → +xx, D(x) → +xx, A → a, L → λ}. Then G' = <Ω, Σ', P', Z> is a context free grammar in Chomsky normal form and L(G') = L(G) (and YIELD (L(G)) = YIELD(L(G'))).□

The following is a generalisation of the concepts of yield, leaf profile, frontier, etc. (See Rounds (1), (2); Brainerd; Thatcher).

Theorem 4: Given a context free grammar G over the alphabet $\Omega \subseteq \Sigma$, one can effectively find a regular grammar G' over D(Ω) such that YIELD (L(G')) = L(G) and conversely. □

Let R(T) be the unary ranked alphabet corresponding to the string alphabet T (see Thatcher and Wright, for example).

Theorem 5: Given an indexed language U over a string alphabet T, one can effectively find a context free grammar G over D(R(T)) such that U=YIELD(L(G)) and conversely.

Corollary: Given an indexed language U over a string alphabet T,

one can effectively find a regular grammar G over D(D(R(T))) such that U = YIELD (YIELD (L(G))) and conversely. □

(Note that the use of "indexed" here is not the same as that in "an indexed family of...". The former refers to a class of languages defined by Aho).

Motivated by the above theorems, we define an indexed family of indexed languages over the many-sorted alphabet Ω to be any indexed family of sets U = YIELD (YIELD(U')) for a recognizable set U' over D(D(Ω)).

3. Machine/Program Structures

In this section we review the concept of a computation viewed as the generation of a subalgebra in a direct product of algebraic structures. The definitions, results, and examples given are versions of those of Landin (1970) and the reader is referred to that work for the details and clarification.

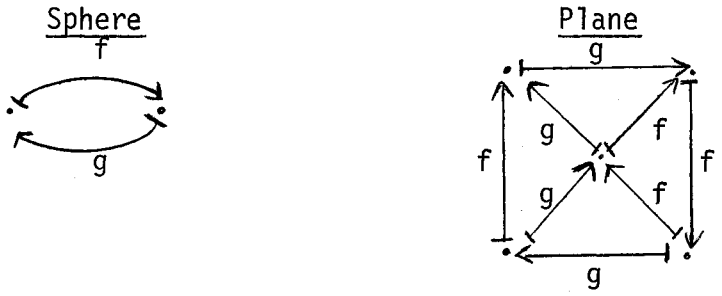
We begin with an informal view of what the following definitions will formalise. Suppose we are given a set of "labels" and we have two directed graphs with edges labelled by this set. One directed graph is drawn on a plane and the other on a sphere. Let us roll, swivel, and slip the sphere on the plane in such a way that the point of contact follows a path through both graphs, synchronising the vertices and matching labels.

If we start rolling the sphere on the plane at a particular vertex-pairing, the device might be unable to move, or it might be able to make exactly one move, or it might have a choice. The latter two possibilities

lead to another vertex-pairing where we again have the same set of three possibilities. If no choice arises, the "path" traced out by our device is given by a string of vertex-pairs. This might cycle on the plane or on the sphere or on both. In the last case, the cycles need not synchronize. Cycling is avoided only because of a halt or because the graph is infinite.

Example 1: The "label" set has a single member, the plane is a 3-cycle and the sphere is a 2-cycle. At whatever vertex-pair the device is started, the "path" will be a 6-cycle going through all possible vertex-pairings.

Example 2: The "label" set is {f,g}.



Here, any vertex-pair leads to a unique path, some halting and some cycling, but none going through all ten possible pairs.

We introduce some terminology:

(i) Graphs: By a graph we always mean a directed graph with labelled edges. Each edge has a label, a source vertex, and a target vertex;

(ii) Plane/Machine: The plane vertices are the states;

(iii) Sphere/Program: The sphere vertices are instructions;

(Thus we view a program as having an operation between (not at) instructions. In terms of an ALGOL 60 program, the occurrences of semi-colons (roughly the contexts which admit labels) are our instructions, the statements themselves are the labels. The plane must then have the transition diagram of the states of some "ALGOL 60 machine").

Executing different programs on the same machine corresponds to rolling different balls on the same plane. In each such execution of a program, the path is an (ordered) set of instruction-state pairs. The set of pairs that touch is a subset of the direct product of the two graphs. It is also a relation that associates with each instruction zero or more states. But the asymmetry suggested by sphere and plane is, of course, spurious. The situation is quite symmetrical. The above set of pairs could just as easily be viewed as a relation which associates with each machine vertex zero or more program vertices.

The intuitive description we have so far given for the running of a program on a machine is inadequate in four ways:

(i) Non-finiteness: We allow the possibility of an infinite set of instructions or states. We "need" this for programs to allow for indefinitely many substitutions to "eliminate" a self-referential definition. The reason we need non-finiteness in this paper is because we want to consider as a program the set of all programs of a certain kind, e.g. the set of strings on terms in a given alphabet (the label set);

(ii) Non-determinateness: We are interested in all possible executions for our program/machine structures. The possibility of more than one path can correspond to a parallel computation (as in the evaluation of a term), or look-ahead (back-up) (as when two computations must be pursued to see which is eventually fruitless), or giving more than one answer;

(iii) Start and Finish: We can mark begin vertices and end vertices on the program (or symmetrically on the machine) and say the programme "computes" a certain relation over the set of vertices on the machine (or symmetrically on the program);

(iv) Polyadicity: A label in a directed graph determines a partial (non-deterministic) function (relation) over the set of vertices. This partial (non-deterministic) function is necessarily unary in the directed graphs we have used so far. We want to relax this condition and allow polyadic (including nullary or constant) operations. To restore the desired symmetry, we allow an operation to have several results (and so to non-deterministically produce several sets of results).

In spite of these inadequacies, the intuitive description of the unary case above will illustrate in an illuminating way the properties discussed below. (The concept of polygraphs described in Landin (4) can be used to encompass the above four points but its value as an expository device in this paper is much less direct than the unary case).

We now proceed to formalise our discussion with the following definitions:

For $x \in A$ and $f \in [A \to B]$, let $Im_f(x) = \{y \in B | (x,y) \in f\}$. If $A = \{A_i\}_{i \in I}$, $B = \{B_i\}_{i \in I}$, $x = \{x_i\}_{i \in I}$ with $x_i \in A_i$, and $f = \{f_i\}_{i \in I}$ with $f_i \in [A_i \to B_i]$, then $Im_f(x) = \{Im_f(x)_i\}_{i \in I} = \{y \in B_i | (x_i, y) \in f_i\}_{i \in I}$.

That is, we can extend our definition from sets to indexed families of sets by defining things componentwise. From now on, all our definitions will be for indexed families of sets (usually indexed by some fixed set I).

The dot product $g \cdot f$ of two relations is defined by $(x,z) \in g \cdot f$ iff $\exists y.(x,y) \in f$ and $(y,z) \in g$ (i.e. iff $\exists y = \{y_i\} \cdot (x_i,y_i) \in f_i$ and $(y_i,z_i) \in g_i$ for each $i \in I$). If f and g are functions this reduces to the usual composition of functions.

The sum $f+g$ of two functions producing (indexed families of) sets is defined by $(f+g)x = f(x) \cup g(x) = \{f_i(x) \cup g_i(x)\}_{i \in I}$. (Note that $f+g = f \cup g$ if and only if f and g have disjoint domains or agree on the overlap; otherwise both are defined but $f+g$ is a function while $f \cup g$ is not).

The dagger f^+ of a function on (indexed families of) sets is defined by $f^+(x) = \bigcup_{n \geq 0} f^i(x)$. A function f on (indexed families of) sets is increasing if $x \subseteq f(x)$ for all x. I_S is the identity relation on the (indexed family of) set(s) S.

A function f on (indexed families of) sets is additive if $f(x \cup y) = f(x) \cup f(y)$. An (indexed family of) set(s) x is f-reduced if $f(x) \subseteq x$.

An indexed family of sets of subsets is a closure system if it is closed under arbitrary intersections. f is monotonic if $x \subseteq y$ implies $f(x) \subseteq f(y)$. f is a closure operation if it is increasing, monotonic, and $f^2(x) \subseteq f(x)$ for all x. If f is monotonic, the indexed family of sets of f-reduced subsets is a closure system. So, for monotonic f, let J_f be the closure operation associated with the closure system consisting of the indexed family of sets of f-reduced subsets. f is continuous if for any ascending chain $x^0 \subseteq x^1 \subseteq \ldots$,

$$f(\cup_{n\geq 0} x^n) = \cup_{n\geq 0} f(x^n).$$

Theorem 1: For an increasing, monotonic, and continuous function f, $f^+ = J_f$.

Theorem 2: If f and g are monotonic and continuous, $J_{f+g} = (f^+ \cdot g^+)^+$.

Corollary: $(f+g)^+ = (f^+ \cdot g^+)^+$.

We shall now use these definitions and results to study many-sorted Σ-structures, for some fixed Σ sorted by I.

The immediate accessability function Immac_A of a Σ-structure A is defined by:

$$\mathrm{Immac}_A(x)_i = \{a_n \in A_i \mid \exists f \in \Sigma_{\langle w,i\rangle} \text{ and } \langle a_0,\ldots,a_{n-1}\rangle \in x^w \text{ and } \langle a_0,\ldots,a_{n-1},a_n\rangle \in f_A\} \text{ for each } i \in I.$$

(Recall that $x^w = x_{w_0} \times \ldots \times x_{w_{n-1}}$). Intuitively, $\mathrm{Immac}_A(x)$ just gives us

the indexed family of sets accessible from $\{x_i\}$ by just one application of some relation of A operating on elements of x. $Immac_A$ is not necessarily an increasing function but $I_A + Immac_A$ is.

An indexed family of sets x (together with the relations assigned to Σ) is a <u>substructure</u> of A if it is $Immac_A$-reduced. This is equivalent to the usual definition of substructure. Define $Ac_A = J_{Immac_A}$. This definition is valid as the intersection of any set of substructures is a substructure. In particular, $Ac_A(\phi)$ is the minimal substructure of A (in the lattice of substructures of A). It will be particularly important in our later discussion.

The concept of "minimal substructure" is good enough in another sense. This is because it is possible to consider $Ac_A(x)$ as the minimal substructure of a structure obtained from A by adding nullaries. In actual fact, we are changing the alphabet Σ and <u>not</u> the underlying sets of our structures, but we prefer to ambiguously present this change as an operation on the structure A. Thus let $\Phi = \{\Phi_i\}_{i \in I}$ be a relation from $B = \{B_i\}$ to A ($B \cap \Sigma = \phi$). Then $Nulls_\Phi$ (A) is the structure obtained from A by adding nullary relations, one for each $b \in B_i$, each i, and having as results (i.e. denoting the elements of A_i) $\Phi_i(b)$. Thus $Ac_A(x) = Ac_{Nulls_{I_x} A}\{(\phi)\}$.

The minimal substructure $Ac_A(\phi)$ is not merely $Immac_A$-reduced but is also a fixed point of $Immac_A$. If $Immac_A$ is increasing, then every subalgebra is a fixed point, but not in general otherwise. Since all relations of A are finitary, $Immac_A$ is continuous. If $Immac_A$ is additive, then $Immac_A = Ac_A$. This is the case, for instance, if A is unary or

if $Immac_A$ is increasing. However, the main result we will need is that, for any structure A, $Immac_A^+\phi = Ac_A\phi$.

In general, we have the important

Theorem 3: For any A, $(I_A+Immac_A)^+ = Ac_A$.

Now, let A and B be Σ-structures. The computation $Comp_x(A,B)$ of A on B generated by $x \in [A \to B]$ (and we ambiguously use $x \subseteq A\times B$) is the substructure $Ac_{A\times B}(x)$ of $A\times B$ generated by x. If $(a,b) \in Comp_\phi(A,B)_i$, we say that the instruction attains the state b; we also say that the state b accepts the instruction a. The set generated at the instruction $a \in A_i$ is the set of states attained at a, i.e. $Im_{\{Ac_{A\times B}(\phi)\}_i}(\{a\})$. The set recognised at the state $b \in B_i$ is the set of instructions accepted by b, i.e.

$$Im_{\{Ac^{-1}_{A\times B}\phi\}_i}(\{b\}).$$

We generalise these concepts to allow for unions of recognisable sets or sets generated as follows: The result Gen(R,C) generated at the "exit map" $R \in [r \to A]$ by a computation $C \subseteq A\times B$ is C;R; i.e. a set of union-sets of states indexed by R_i for each $i\in I$. If r_i is a singleton, this corresponds to a set of terminal instructions. If R_i is a singleton, it reduces to the "set generated by" (as above). The argument Recog (R,C) recognised the output may $R \in [r \to B]$ by a computation $C \subseteq A\times B$ is C^{-1};R; i.e. a set of union sets of instructions indexed by r_i, for each $i\in I$.

Example 3: Let Σ be some finite (string) alphabet and let $M = \langle Q, \{\delta_a\}_{a\in\Sigma}, q_0\rangle$ be a finite automaton over Σ with initial state q_0 and (unary) transition functions δ_a for each $a \in \Sigma$ (see the introduction). Let W_Σ be the algebra which forms the strings over Σ. That is, it has as a constant (initial) symbol λ, the empty string, and for each $a \in \Sigma$, **and operation of left concatenation by a.**

Now consider $\text{Comp}_{\phi}(W_{\Sigma},M) = \text{Ac}_{W_{\Sigma}\times M}(\phi)$

$= \text{Immac}^{+}_{W_{\Sigma}\times M}(\phi)$

$= \cup_{i\geq 0} \text{Immac}^{i}_{W_{\Sigma}\times M}(\phi)$.

Now $\text{Immac}^{0}_{W_{\Sigma}\times M}(\phi) = \{(\lambda,q_0)\}$. That is, the only ordered pair immediately accessible from the empty set $\phi \subseteq W_{\Sigma}\times M$ is the ordered pair of constant (initial) symbols (λ,q_0). Then $\text{Immac}^{1}_{W_{\Sigma}\times M}(\phi)=$ $\{(\lambda,q_0)\} \cup \{(a,q) | a \in \Sigma \text{ and } \delta_a(q_0) = q\}$. That is, Immac^{1} gives us all the strings of length one and pairs these strings with the states of M which are specified by the corresponding transition function. Then $\text{Immac}^{2}_{W_{\Sigma}\times M}(\phi) = \text{Immac}^{1}_{W_{\Sigma}\times M}(\phi) \cup \{(w,q) | \ell(w)=2 \text{ and } w=aw' \text{ for some } a \in \Sigma, (w',q') \in \text{Immac}^{1}_{W_{\Sigma}\times M}(\phi) \text{ and } \delta_a(q') = q\}$. By this iteration process we obtain $\text{Immac}^{n}_{W_{\Sigma}\times M}(\phi)$ for any $n \in \underline{N}$ and so obtain $\text{Immac}^{+}_{W_{\Sigma}\times M}(\phi)$ as the union of all these sets.

So $\text{Comp}_{\phi}(W_{\Sigma},M)$ gives us a set of ordered pairs (w,q) such that w is a string over Σ and q is the state reached by the automaton after processing w. Then if we specify an output map R for some set of final states $Q^F \subseteq Q$ by the inclusion map, $\text{Recog}\ (R, \text{Comp}_{\phi}(W_{\Sigma},M)) = \text{Comp}^{-1}_{\phi}\ (W_{\Sigma},M)\cdot R$ □.

4. Generalised Automata Theory

In this section we provide a generalisation of some conventional classes of automata. In the case of finite automata, this generalisation is equivalent to those of Büchi, Thatcher and Wright, and Doner. On

the other hand, our formulation lends itself to simple generalisations of pushdown and nested stack automata. Also, the relation between these classes of automata becomes transparent.

Let Ω be a (finite) many-sorted alphabet.

Theorem 1: $U \subseteq W_\Omega$ is recognisable (by a finite automaton) if and only if $U = \text{Recog}(R, \text{Comp}_\phi (W_\Omega, A))$ for some finite Ω-automaton A and output map R.

Proof: Since U is recognisable, there exists a finite Ω-automaton A (in the sense of Section 2) and a choice of final states $A^F \subseteq A$ such that $bh_A(A^F) = U$. Let $R: A^F \to A$ be the injection map. It is then easy to check that $bh_A(A^F) = \text{Recog}(R, \text{Comp}_\phi (W_\Omega, A))$ by a simple induction argument. The converse is also obvious.□

Theorem 2: $U \subseteq W_\Omega$ is context free if and only if $U = \text{Recog}(R, \text{Comp}_\phi (D(W_\Omega), B))$ for some finite $D(\Omega)$-automaton B and output map R.

Proof: We give a sketch of the proof. Let $G = \langle \Omega, \Sigma, P, Z \rangle$ be a Chomsky normal form grammar for U. We construct the finite $D(\Omega)$-automaton B as follows. (Note that we are here using the convention of Section 1 in assuming $D(\Omega) \subseteq D(\Sigma)$):

(i) Our $D(\Omega)$-automaton B will have as carrier of sort $\langle w,i \rangle$ the set $B_{\langle w,i \rangle} = \{A' \mid A \in D(\Sigma)_{\langle \lambda, \langle w,i \rangle \rangle} - \Omega_{\langle w,i \rangle}\}$.

That is, a symbol corresponding to each nonterminal of type $\langle w,i \rangle$ in Σ and a symbol corresponding to each projection operator δ_w^j in $D(\Sigma)_{\langle \lambda, \langle w, w_{j-1} \rangle \rangle}$;

(ii) For each production of the form $A(x_0, w_0, \ldots, x_{n-1}, w_{n-1}) \to G(H_0(x_0, w_0, \ldots, x_{n-1}, w_{n-1}), \ldots, H_{m-1}(x_0, w_0, \ldots, x_{n-1}, w_{n-1}))$

with $A \in \Sigma_{<w,i>}$, $G \in \Sigma_{<v,i>}$ and $H_j \in \Sigma_{<w,v_j>}$ for $0 \le j \le n-1$, let $<G',H'_0,\ldots,H'_{m-1}> \in (c_{<v,w,i>})_B$;

(iii) For each production of the form $A(x_0,w_0,\ldots,x_{n-1},w_{n-1}) \to fx_{i_0};w_{i_0}\ldots x_{i_{m-1}};w_{i_{m-1}}$ with $A \in \Sigma_{<w,i>}$, $f \in \Omega_{<v,i>}$ and $0 \le i_j \le n-1$ (for $0 \le j \le m-1$), we let $<f_B,(\delta_w^{i_0+1})',\ldots,(\delta_w^{i_{m-1}+1})', A' > \in (c_{<v,w,i>})_B$ and $(\delta_w^{i_j+1})' \in (\delta_w^{i_j+1})_B$ for each $0 \le j \le m-1$;

(iv) (This is really included in (iii)). **For each production of the** form $A \to a$ with $A \in \Sigma_{<\lambda,i>}$ and $a \in \Omega_{<\lambda,i>}$, we let $A' \in a_B$.

Let $R:Z \to B$ be defined by $Z' \mapsto Z'$ for each $<\lambda,i> \in D(I)$. Then we prove $U = \text{Recog}(R, \text{Comp}_\phi (D(W_\Omega),B))$ by showing that Comp_ϕ "provides a parse" for any term in $D(W_\Omega)$ while R "picks out" the ones which have Z as the "root of the parse tree". This is an easy induction proof and is left as an exercise for the interested reader.

Conversely, let $U=\text{Recog}\ (R, \text{Comp}_\phi\ (D(W_\Omega),B))$ for some R and B. Define the context free grammar $G = <\Omega, \Sigma, P\ \ Z>$ where:

(i) $\Sigma_{<w,i>} - \Omega_{<w,i>} = B_{<w,i>}$ for each $<w,i> \in I^* \times I$;

(ii) P is given as follows:

(a) If $A \in B_{<\lambda,i>}$ and $a \in \Omega_{<\lambda,i>}$ is such that $A \in a_B$, then let $A \to a$ be in P;

(b) If $A \in B_{<w,w_{j-1}>}$ some $1 \le j \le n$ and $\delta_w^j \in D(\Sigma)_{<\lambda,<w,w_{j-1}>>}$ is

such that $A \in (\delta_w^j)_B$, then let $A(x_{0,w_0},\ldots,x_{n-1,w_{n-1}}) \to x_{j-1,w_{n-1}}$ be in P;

(c) If $A \in B_{<w,i>}$ and there exist $f \in \Omega_{<v,i>}$ and $\delta_w^{i_j} \in \Sigma_{<\lambda,<w,w_{i_j-1}>>}$ for $0 \leq j \leq m-1$ $(\ell(v)=m)$ such that $(f_B, (\delta_w^{i_0})_B,\ldots, (\delta_w^{i_{m-1}})_B, A) \in (c_{<v,w,i>})_B$, then let $A(x_{0,w_0},\ldots,x_{n-1,w_{n-1}}) \to fx_{i_0,w_{i_0}},\ldots,x_{i_{m-1},w_{m-1}}$ be in P;

(d) If $<G,H_0,\ldots,H_{m-1},A> \in (c_{<v,w,i>})_B$ then let $A(x_{0,w_0},\ldots,x_{n-1,w_{n-1}}) \to G(H_0(x_{0,w_0},\ldots x_{n-1,w_{n-1}}),\ldots,H_{m-1}(x_{0,w_0},\ldots,x_{n-1,w_{n-1}}))$ be in P; and

(iii) For R:r → B the output map, designate the image, under R_i, of r_i to be axioms. If $R_i(r_i)$ is a singleton, then denote it by Z, for each $i \in I$. Otherwise we have a grammar which has more than one axiom of each sort. This can easily be converted to a grammar with a single axiom of each sort, denoted by Z, by well known methods.

It is simple matter to check by induction that the language generated by G is exactly U. □

The above proof could have been done more indirectly by invoking Theorem 4 of Section 2. According to this theorem, to an indexed family of context free sets $U \subseteq W_\Omega$, there corresponds a regular language U' over $D(\Omega)$ such that YIELD(U')=U. Then there exists some finite $D(\Omega)$-automaton B and a choice of final states $R:B^F \subseteq B$ such that $bh_B(B^F)=U'$. Then, clearly, $U=Recog(R,Comp_\phi(D(W_\Omega),B))$. Conversely, we can obtain a context free grammar G over Ω from some finite $D(\Omega)$-automaton B and output map R by considering a regular grammar obtainable from B and R for the set of derivation trees of U.

Theorem 3: $U \subseteq W_\Omega$ is indexed if and only if $U = \text{Recog}(R, \text{Comp}_\phi(D(D(\Omega)),C))$ for some finite $D(D(\Omega))$-automaton C and output map R.

Proof: The simplest (indirect) proof is motivated by Theorem 5 and its corollary of Section 2. The proof is similar to the informal one given above and is left as an exercise for the interested reader. □

Example 1: Consider the alphabet $\Omega \subseteq \Sigma'$ of Example 3, Section 2. $D(\Omega)$ is then described as follows:

$$D(I) = \{<\lambda,0>,<\lambda,1>,<1,0>,<1,1>,<10,1>,<10,0>, <11,1>,<11,0>\}.$$

$$D(\Omega)_{<\lambda\ <\lambda,0>>} = \{\lambda\};$$

$$D(\Omega)_{<\lambda,<\lambda,1>>} = \{a\};$$

$$D(\Omega)_{<\lambda,<1,0>>} = \phi.$$

$$D(\Omega)_{<\lambda,<1,1>>} = \{\delta^1_1\}.$$

$$D(\Omega)_{<\lambda,<10,0>>} = \{\delta^2_{10},*\}.$$

$$D(\Omega)_{<\lambda,<10,1>>} = \{\delta^1_{10}\}.$$

$$D(\Omega)_{<\lambda,<11,0>>} = \phi\ .$$

$$D(\Omega)_{<\lambda,<11,1>>} = \{\delta^1_{11}, \delta^2_{11}, +\ \}.$$

$$D(\Omega)_{<<w,i><v,w_0>\ldots<v,w_{n-1}>,<v,i>>} = \{c_{<w,v,i>}\}$$

for each $(w,v,i) \in \{1,10,11\} \times \{\lambda,1,10,11\} \times I$.

Let $M = \{M_i\}_{i \in D(I)}$ be a $D(\Omega)$-automaton such that:

(i) $M_{<\lambda,0>} = \{L,Z\}.$

$$M_{<\lambda,1>} = \{A,C\}.$$

$M_{<1,0>} = \phi$.

$M_{<1,1>} = \{H,B,D\}$.

$M_{<10,0>} = \{E,F_2,S\}$.

$M_{<10,1>} = \{F_1\}$.

$M_{<11,0>} = \phi$.

$M_{<11,1>} = \{F\}$;

(ii) The relations of M are:

$L \in \lambda_M$.

$A \in a_M$.

$H \in (\delta^1_1)_M$.

$F_1 \in (\delta^1_{10})_M$.

$F_2 \in (\delta^2_{10})_M$.

$F \in +_M$.

$E \in *_M$.

(S,A,L,Z), $(S,C,L,Z) \in c_{<10,\lambda,0>}$ in M.

$(B,A,C) \in c_{<1,\lambda,1>}$ in M.

$(B,D,B) \in c_{<1,1,1>}$ in M.

$(E,F_1,F_2,S) \in c_{<10,10,0>}$ in M.

(F,H,H,B), $(F,H,H,D) \in c_{<11,1,1>}$ in M.

Let $r = \{r_i\}_{i \in D(I)}$ be defined by $r_{<\lambda,0>} = \sigma$ and $r_i = \phi$ for $i \neq <\lambda,0>$. Define the output relation $R: r \to M$ by $(\sigma, Z) \in R_{<\lambda,0>}$. ($R_i$ is empty for $i \neq <\lambda,0>$). Then $U = Recog\ (R, Comp_\phi\ (W_{D(\Omega)}, M)) =$

$\{c_{<10,\lambda,0>}\ c_{<10,10,0>}{}^{*}\delta^1_{10}\ \delta^2_{10}\ a\lambda,$

$c_{<10,\lambda,0>}\ c_{<10,10,0>}{}^{*}\delta^1_{10}\ \delta^2_{10}\ c_{<1,\lambda,1>}\ c_{<11,1,1>} + \delta^1_1\ \delta^1_1\ a\lambda$, etc$\}$.

On the other hand, $Recog(R, Comp_\phi\ (D(W_\Omega), M)) = \{{*}a\lambda, {*}{+}aa\lambda, {*}{+}{+}aa{+}aa\lambda$, etc.$\} = U'$. Note that $YIELD\ (U) = U'$.

We note that we can consider the alphabet Ω to be the derived alphabet of some unary alphabet $\Delta \subseteq \Gamma$ with $\Delta_0 = \{\lambda\}$ and $\Delta_1 = \{a\}$. Thus $\Omega = D(\Delta)$ and $D(\Omega) = D^2(\Delta)$; $W_\Omega = W_{D(\Delta)}$ and $D(W_\Omega) = D(W_{D(\Delta)})$. Thus we can run $D^2(W_\Delta)$ on M and $U'' = Recog\ (R, Comp_\phi\ (D^2(W_\Delta), M)) = \{a\lambda, aa\lambda, aaaa\lambda, aaaaaaaa\lambda$, etc.$\} = \{a^{2^n}\ \lambda \mid n \geq 0\}$. If we interpret λ to be the empty string, then we get $U'' = \{a^{2^n} \mid n \geq 0\}$. Note also that $YIELD^2(U) = YIELD(U') = U''$. Thus we have used the same automaton M to recognise an indexed set (in Aho's sense) U" over some alphabet Δ; its set of derivation trees, a context free set U' over $D(\Delta)$; and its set of "derivation^2 trees", a recognisable set U over $D^2(\Delta)$. That is, we have used the same automaton M in the nested stack, pushdown, and finite automata modes by running different programs on it.

It is left as an exercise for the reader to provide a deterministic automaton M' which recognises the same sets as M. □

The above example illustrates the remark made in the introduction about getting different sets as results of running different programs on a given machine (automaton). If we run the program $W_{D(D(\Omega))}$ on a finite $D(D(\Omega))$- automaton we get a recognisable set over $D(D(\Omega))$; if we run the program $D(D(W_\Omega))$ on it we get an indexed set over Ω. Motivated by this

and the above theorems, we make the following definitions: A pushdown automaton over Ω is a finite $D(\Omega)$-automaton (with program $D(W_\Omega)$). A nested stack automaton over Ω is a finite $D(D(\Omega))$-automaton (with program $D(D(W_\Omega))$).

It is a remarkable result of this theory that we do not "need" nondeterminism in the conventional sense of non-determinism: producing more than one answer). Our automata can be restricted to finite algebras (that is, have operations which are totally defined on all possible arguments and which produce exactly one result for a given list of arguments) and still recognise the same class of sets.

Theorem 4: The class of sets recognised by non-deterministic automata (with a fixed program W) is the same as the class recognised by deterministic automata (with the same fixed program W). (Note that this result is independent of the program in the sense that the theorem is true for any program W compatible with the class of automata being considered. Thus the theorem states that the class of sets recognised by finite (pushdown, nested stack) automata is independent of the determinism or non-determinism of the automata. This result is well known for finite automata; for pushdown automata in the conventional formulation, we know this result is not true; and for nested stack automata in the conventional sense, the result is still an open question).

Proof: Since we have defined different classes of automata over some alphabet Ω in terms of finite automata over related alphabets, it is not surprising that we can use the conventional notion of "subset" automaton to prove our result. Thus, let U = Recog (R, Comp_ϕ (W,A)) for some program W and automaton A and assume A is non-deterministic. Consider

the finite automaton B(A) and the output map R' given by $R'_i : r_i \rightarrow S_i$ where $S_i = \{s \in A_i \mid (r_i,s) \in R_i\}$ for each $i \in I$. It is a simple matter to check that Recog (R, Comp_ϕ (W,A)) = Recog (R', Comp_ϕ (W,B(A)) = U. Thus the class of sets recognised by non-deterministic automata is a subclass of the class recognised by deterministic automata. The converse is trivially true and our result follows. □

Conventional classes of automata work essentially by trying to parse a given string and can thus be non-deterministic for two reasons: Firstly because the particular parse the automaton is trying to construct may not be a valid parse for the given string and, secondly, because a given string may have more than one distinct parse. By the very construction of our automata, they are constrained to try only valid parses and if a given term has more than one distinct parse, our automaton "identifies" them. (That is, it considers the class of parse trees for a given term to be a single entity). This identification reflects the associativity of composition in our derived algebras. In the sphere-plane terminology, we identify all valid paths from one vertex pair to another vertex pair.

At this point we can perhaps mention some comparisons with other tree manipulation systems (Thatcher, Rounds (3), Baker (2)). In traditional systems there is no obvious reason to consider top-down manipulation (i.e. from root to leaves) as being somehow more natural or more illuminating than bottom-up (i.e. from leaves to root) manipulation or vice versa. In the systems just defined, we are generating subalgebras and thus, in effect, working in a bottom-up mode. Our programme forms expressions (constricted by our machine) by starting at a constant and forming an expression, then using

these to form other expressions, etc. This is the only natural way of looking at this system. There is no immediately obvious way of defining any top-down manipulation.

$$\text{Let } D^{i}(W_{D}{}^{j}{}_{(\Omega)}) \equiv \underset{\text{i-times}}{D(\ldots(D(}W_{\underset{\text{j-times}}{D(\ldots(D(\Omega)\ldots)}})\ldots)$$

for $0 \leq i,j \leq \omega$.

Lemma 1: There is a unique homomorphism from $D^{n-i}(W_{D}{}^{i}{}_{(\Omega)})$ to $D^{n-j}(W_{D}{}^{j}{}_{(\Omega)})$ for any $0 \leq j \leq i \leq n$.

Proof: (0) If j=i, the required homomorphism is the identity.

(i) If i-j=1, the homomorphism is given by interpreting the non-nullaries of $D^{i}(\Omega)$ as composition and the projection symbols of $D^{i}(\Omega)$ as projection operations. Thus YIELD:$W_{D(\Omega)} \rightarrow D(W_{\Omega})$ is a particular example of the above with n=i=1 and j=0.

(ii) If i-j>1, then we get our homomorphism in the obvious way by composing i-j homomorphisms of the type used in step (i).

The uniqueness of the homomorphisms in cases (0) and (i) is obvious while that in (ii) is easy to check because of the constructive nature of the homomorphism. □

Motivated by this lemma, let YIELD be the unique homomorphism from $D^{n-i}(W_{D}{}^{i}{}_{(\Omega)})$ to $D^{n-j}(W_{D}{}^{j}{}_{(\Omega)})$ for i-j=1. (Thus the previous definition of YIELD is the special case where n=i=1 and j=0). Then for any $0 \leq j \leq i \leq n$ let $\text{YIELD}^{i-j}: D^{n-i}(W_{D}{}^{i}{}_{(\Omega)}) \rightarrow D^{n-j}(W_{D}{}^{j}{}_{(\Omega)})$ be the unique homomorphism. YIELD^{0} is then the identity in all cases. (Note that we have purposely defined YIELD^{i-j} ambiguously in the sense that it is unique only in a given context).

We can now state the following easily proved result:

Theorem 5: Suppose we are given a finite alphabet Ω, a finite $D^n(\Omega)$-automaton A and an output map R. Then

$$\begin{aligned} & YIELD^{i-j}\ (Recog(R,\ Comp_\phi(D^{n-i}(W_D i_{(\Omega)}),A))) \\ = \quad & Recog(R,\ Comp_\phi\ (YIELD^{i-j}(D^{n-i}(W_D i_{(\Omega)})),A)) \\ = \quad & Recog(R,\ Comp_\phi\ (D^{n-j}(W_D j_{(\Omega)}),A)) \end{aligned}$$

for $0 \le j \le i \le n$. □

Suppose U is context free over Ω. That is $U = Recog(R, Comp_\phi(D(W_\Omega),A))$ for some $D(\Omega)$-automaton A and output map R. Then the fact that there may be more than one path from the initial vertex pair (the empty set in this case) to a final vertex pair (with one of its components a final state in A) corresponds to the fact that context free grammars (or languages) may be ambiguous. That is, the associativity of composition gives us the ambiguity of context free grammars. Can we state this intuitive result in more precise terms? We know that a context free language generated by a given grammar is unambiguous if and only if there is a one to one relationship between terms in a context free set and its set of derivation trees. That is, the homomorphism

$$\psi: Comp_\phi(W_{D(\Omega)},A) \to Comp_\phi\ (D(W_\Omega),A)$$

is one-one, onto (i.e. an isomorphism). (Recall that $Comp_\phi\ (A,B)$ is the least subalgebra of $A \times B$ generated by ϕ). We can extend this idea to define unambiguous indexed languages to be those for which

$\psi \circ \psi'$: $Comp_\phi\ (W_{D^2(\Omega)},A) \to Comp_\phi(D^2(W_\Omega),A)$ is an isomorphism (with ψ

as above and ψ': $\mathrm{Comp}_{\phi}(W_{D^2(\Omega)},A) \rightarrow \mathrm{Comp}_{\phi}(D(W_{D(\Omega)}),A))$. Thus there are two levels of ambiguity depending on whether one or more of ψ and ψ' are isomorphisms. This double ambiguity needs to be explored further.

Finally, we make the observation that decision problems for classes of languages recognized by certain classes of automata depend solely on the corresponding decision problems for finite automata and the properties of the unique homomorphism YIELD. For example, the emptiness and finiteness problems for context free (and indexed) sets are solvable because of the solvability of the corresponding problems for finite automata and because of the fact that YIELD preserves unions (i.e. YIELD $(\cup_i V_i) = \cup_i \mathrm{YIELD}(V_i)$). This has been noted in a number of places in different contexts from ours (see, for example, Thatcher 4). On the other hand, the equivalence problem for context free (indexed) sets is not solvable because YIELD does not preserve intersections (i.e. YIELD $(\cap_i V_i)$ is not in general the same as $\cap_i \mathrm{YIELD}(V_i)$).

Using the above fact about intersections and the fact that context free and indexed sets can be defined as homomorphic images of unions of classes of a finite congruence (i.e. recognisable sets), we can give an algebraic proof (i.e. one not depending on the generation of particular counterexamples) of the fact that the class of context free (indexed) sets is not closed under intersection. Using the same techniques we can easily show that the class of indexed sets is not closed under intersection with context free sets.

5. Conclusion

At the end of Maibaum (3), we expressed the need for providing a uniform theory of automata to accompany our generalisation of formal language theory. We believe the ideas expounded in this paper fit the bill. There are a number of surprising but pleasant results provided by the theory: the equivalence of classes of deterministic and non-deterministic automata, the nice algebraic characterisation of ambiguous languages in terms of isomorphisms, the trade-off between the complexity/simplicity of our programs and the simplicity/complexity of our YIELD operations, and the close relationship between different classes of automata.

But this study is only a beginning. We must study the ambiguity problem and parsing methods in light of the above algebraic characterisations. Also, we have only used program/machine structures with no input (i.e. $Comp_{\phi}(A,B)$). We must study these structures with input. This, we believe, will lead to generalisations of finite state transducers, pushdown transducers, etc.

This study was also confined to running only certain special types of programs on our machines: specifically word algebras and derived word algebras. This restriction was natural if we wanted to study classical automata and formal language theory. On the other hand, there is no mathematical reason for such a restriction and so we propose definitions of recognisable, context free, and indexed sets in the carriers of arbitrary structures (or algebras) as follows (for an Ω-structure A):

(a) $U \subseteq A$ is recognisable if and only if $U = Recog(R, Comp_{\phi}(A,B))$ for some finite Ω-automaton B and output map R;

(b) $U \subseteq A$ is context free if and only if $U = \text{Recog}(R, \text{Comp}_{\phi}(D(A),B))$ for some finite $D(\Omega)$-automaton B and output map R;

(c) $U \subseteq A$ is indexed if and only if $U = \text{Recog}\ (R, \text{Comp}_{\phi}\ (D^2(A),B))$ for some finite $D^2(\Omega)$-automaton B and output map R.

Further, there is no reason to assume that one could not continue the obvious iteration in definitions (a), (b), (c) for any finite number of steps. This could be done for any Ω-structure A, including W_{Ω}. This suggests the existence of an infinite hierarchy of classes of sets beginning with recognisable, context free and indexed. In the case of W_{Ω}, this hierarchy would correspond to the one given at the end of Maibaum (3) and in Turner. We suggest that because of this, it is a proper (infinite) hierarchy.

BIBLIOGRAPHY

Aho, A.V., Indexed Grammars - an Extension of Context Free Grammars, "Journal of the ACM", Vol. 15, No. 4.

Baker, B.S., Tree Transductions and Families of Tree Languages, Technical Report TR9-73, Harvard University.

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

Brainerd, W.S., Tree Generating Regular Systems, "Information and Control", 14 (1969), 217-231.

Büchi, J.R., Weak Second-order Arithmetic and Finite Automata, "Z. Math. Logik Grundlagen Math.", Vol. 6, pp. 66-92.

Cohn, P.M., Universal Algebra", Harper & Row, 1965.

Doner, J., Tree Acceptors and some of their applications, "J. Comput. Syst. Sci.", 4 (1969), 406-451.

Eilenberg, S. and Wright, J.B., Automata in General Algebras, "Information and Control", 11 (1967), 452-470.

Higgins, P.J., Algebras with a scheme of operators, Math. Nachrichten, 27 (1963), 115-132.

Landin, P.J., A Program Machine Symmetric Automata Theory, "Machine Intelligence 5", Edinburgh University Press, 1970.

Maibaum, T.S.E. (1) The Characterisation of the Derivation Trees of Context Free Sets of Terms as Regular Sets, "Proceedings of the 13th IEEE Symposium on Switching and Automata Theory", 1972.

Maibaum, T.S.E. (2) A Generalised Approach to Formal Languages,"J. Comput. Syst. Sci.", Vol. 8, No. 3, pp. 409-439.

Maibaum, T.S.E. (3) Generalised Grammars and Homomorphic Images of Recognisable Sets, Doctoral Dissertation, University of London, 1973.

Mezei, J. and Wright, J.B., Algebraic Automata and Context-Free Sets, "Information and Control", Edinburgh University Press, 1970.

Rounds, W.C., (1) Mappings and grammars on trees, "Math. Systems Theory", 4 (1970) 257-287.

Rounds, W.C., (2) Tree-oriented proofs of some theorems on context free and indexed languages, "Proc. ACM Symp. on Theory of Computing", 1970.

Thatcher, J.W., Characterizing derivation trees of context-free grammars through a generalisation of finite automata theory. "J. Comput. Syst. Sci.", 1 (1967), 317-322.

Thatcher, J.W. and Wright, J.B., Generalised finite automata theory with an application to a decision problem of second order logic, "Math Systems Theory", 2 (1968), 57-81.