INFORMATION AND CONTROL 12, 331–345 (1968)

# Algebra Automata I: Parallel Programming as a Prolegomena to the Categorical Approach[1]

M. A. ARBIB AND Y. GIVE'ON

*Department of Electrical Engineering, Stanford University, Stanford, California 94305*

Wright, Thatcher and Mezei have built on the observation of Büchi that finite automata may be considered to be monadic algebras, to study non-monadic algebras from the viewpoint of automata theory, and have generalized the usual studies of regular sets and context-free languages in this context. We continue this work, but shift the emphasis from the use of algebra automata as acceptors to the dynamics of algebras with outputs. We show that the Nerode and Myhill approaches to state minimization and minimal dynamics can be carried through in the general case.

In Part I, we emphasize the interpretation of algebra automata as executing parallel programs. However, Eilenberg and Wright have shown that much of Wright, Thatcher and Mezei's work can be carried through in the context of categorical algebra, using the notion of algebraic theory introduced by Lawvere as a categorical explication of the notion of variety initiated by Birkhoff. Our results in Part I pave the way for the extension of this categorical framework to the treatment of algebra automata as dynamic systems in Part II [*Inform. Control* **13**, 346–370 (1968)].

## 1. COMPUTATION STRUCTURES FOR ALGEBRA AUTOMATA

Our approach in Part I is non-categorical, and is intended, *inter alia*, to provide a bridge between ordinary automata theory and computer science on the one hand, and the more abstract categorical approach of Part II on the other.

Our intuition will be based on the notion of a device which can execute highly parallel, but loop-free, programs. Given some set $Q$ of values for data within each register, each individual instruction causes the device

[1] This research was supported in part by the U. S. Air Force Office of Scientific Research, Information Sciences Directorate, under Grants No. AF-AFOSR-1198-67 and AF49(638)-1440.

to act on a tuple of Q-values (the states of an appropriate number of registers) to provide a new $Q$-value (place the result of a subcomputation in a register). The pattern of execution of a program is represented by a DOAG (directed ordreed acyclic graph), the vertices of which may then be labelled with elements of a suitable set $\Sigma$. The device is then capable of interpreting each element of $\Sigma$ as an actual command to manipulate the contents of $Q$-registers. We now proceed to formalise these notions, and explore their consequences.

(1.1) DEFINITION. By a *graded set* we mean a set $\Sigma$ with a map $\sigma: \Sigma \to N$. We denote by $\Sigma_p$ the set $\sigma^{-1}(p)$, and so the graded set is the sequence $(\Sigma_0, \Sigma_1, \cdots)$ of disjoint sets whose union is $\Sigma$. $\sigma$ is called the *grading map* of $\Sigma$. For $f \in \Sigma$, $\sigma(f)$ is called the *rank* of $f$. Given a graded set $\Sigma$ and an ungraded set $Q$, $\Sigma_Q$ shall denote the graded set obtained by adjoining $Q$ to $\Sigma_0$ :

$$(\Sigma_Q)_n = \begin{cases} \Sigma_n & \text{if } n > 0 \\ \Sigma_0 \cup Q & \text{if } n = 0 \end{cases}$$

(1.2) DEFINITION. A $\Sigma$-*algebra* $\mathfrak{A} = (Q, \alpha)$ consists of a set $Q$ and a map $\alpha$ such that for each $f \in \Sigma$, $\alpha_f$ is a map with domain a subset of $Q^*$ and codomain $Q$. If $\Sigma$ is graded, we further require that the domain of $\alpha_f$ is $Q^{\sigma(f)}$, and call $\mathfrak{A}$ a *uniform algebra*.

For our automata theory, we interpret an algebra as an automaton with registers which can hold any datum from the set $Q$. The maps $\alpha_f$ are then the instructions of the machine. However, to completely specify the automaton we must specify how the state determines an output.

(1.3) DEFINITION. A $\Sigma$-*automaton* $M$ is a quadruple $(Q, \alpha, Y, \lambda)$ where $(Q, \alpha)$ is a $\Sigma$-algebra ($Q$ is called the set of *states* of $M$, $\alpha$ the *transition function* of $M$), $Y$ is a set (the set of *outputs*) and $\lambda: Q \to Y$ is the output map.

In the paper of Thatcher and Wright (1966), automata are always used as acceptors, so $Y$ is implicit as $\{0, 1\}$; in actual fact, only $F = \lambda^{-1}(1) \subseteq Q$ is specified.

Our interest here is in specifying the most general loop-free computations of such automata. To do this we must introduce various graphical structures to specify parallel computations.

(1.4) DEFINITION. A *directed ordered*[2] *graph* $\Gamma$ (DOG) is a map $\Gamma: V \to V^*$. $V$ is called the set of *vertices* of $\Gamma$. We often write $({}^1\mathbf{v}, {}^2\mathbf{v}, \cdots {}^{n(v)}\mathbf{v})$ for $\Gamma(v)$ and call any ${}^i\mathbf{v}$ an *input node for* $v$.

[2] Here ordered refers *not* to the set of vertices (= nodes), but to each set of nodes leading to a given node.

$v$ is called *initial* if $\Gamma(v) = \Lambda$;

$v$ is called *terminal* if $v$ is not an input node for any $v' \in V$.

We say $v < v'$ if there is a sequence $v = v_1, v_2, \cdots, v_n = v'(n > 1)$ with each $v_i$ an input node for $v_{i+1}$.

Henceforth, all DOGs considered will be finite.

(1.5) DEFINITION. A *loop* of a DOG $\Gamma$ is a sequence $v_1, \cdots, v_n$, $v_{n+1} = v_1$ of vertices of $\Gamma$ such that $v_i$ is an input node of $v_{i+1}$, $i = 1, \cdots, n$. A *directed ordered acyclic graph* (DOAG) is a DOG without any loops. Thus a DOG is a DOAG if $v < v$ for *no* vertex $v$. A tree is a DOAG for which a node is input to at most one other node and there is only one terminal node.

Our idea of a program structure is then easily specified.

(1.6) DEFINITION. A $\Sigma$-DOAG is a pair $(\Gamma, h)$ where $\Gamma: V \to V^*$ is a DOAG, and $h: V \to \Sigma$. If $\Sigma$ is graded, we further require that $n(v) = \sigma(h(v))$.

As we shall make explicit in Definitions 1.14 and 1.15, this will represent a program in which all initial data are specified. The simple extension to programs which allow different intial data for each "run" will be made in Section 2.

(1.7) DEFINITION. Given a DOAG $\Gamma$, we associate a *level* with each node as follows:

level $(v) = 0$ iff $v$ is initial;

level $(v) \leqq k + 1$ iff level $({}^j\mathbf{v}) \leqq k$ for $1 \leqq j \leqq n(v)$.

Thus $v < v' \Rightarrow$ level $(v) <$ level $(v')$, but the converse is not true, even in trees.

(1.8) LEMMA. *For a finite* DOAG $\Gamma$, *there is a finite number* $d_\Gamma$, *the* ***depth*** *of* $\Gamma$, *such that each node of* $\Gamma$ *has level* $k$ *for some* $k \in \{0, 1, \cdots, d_\Gamma\}$; *for every such* $k$ *there is a node with that level; and* $v$ *is terminal if* (*but not only if, in general*) *it has level* $d_\Gamma$.

*Proof.* By induction, noting that if we have nodes of all levels up to $k \geqq -1$, the absence of nodes of level k + 1 would imply a cycle among the remaining nodes. Q.E.D.

(1.8) Given a DOAG $\Gamma$ and a node $v$ of $\Gamma$, we may define the tree ${}^v\Gamma$ obtained by "*unfolding* that part of $\Gamma$ which feeds into $v$". More formally, we work by induction:

(1.9) Basis: For an initial node $v$, the tree ${}^v\Gamma$ is given by the function $\{v\} \to \{v\}^*: v \mapsto \Lambda$,

(1.10) Induction Step: If $v$ is a node of level $k \geqq 1$ of a DOAG $\Gamma$ then each ${}^j\mathbf{v}$ is of level $< k$, and so by induction we assume ${}^{j\mathbf{v}}\Gamma$ is already defined, say as $\Delta: W \to W^*$. For each $w \in W$ introduce a new symbol $w_j{}^v$

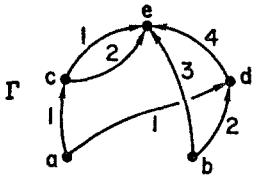
to obtain the set $W_j^v$. Let ${}^vV = \{v\} \cup \bigcup_j W_j^v$ and define ${}^v\Gamma : {}^vV \to {}^vV^*$ as follows:

(i) ${}^v\Gamma(v) = ({}^1v_1^v\ {}^2v_2^v, \cdots, {}^{n(v)}v_{n(v)}^v)$;

(ii) if ${}^{iv}\Gamma$ is $\Delta : W \to W^*$ and $\Delta(w) = ({}^1w, \cdots, {}^nw)$, then ${}^v\Gamma(w_j^v) = ({}^1w_j^v, \cdots, {}^nw_j^v)$.

(1.11) If the label of a node $u$ of ${}^v\Gamma$ is of the form $w_\beta^\alpha$ for some $w$ in $V$ we call it a *w-node* of ${}^v\Gamma$. If $u$ is a $w$-node of ${}^{iv}\Gamma$, then $u_j^v$ is a $w$-node of ${}^v\Gamma$.

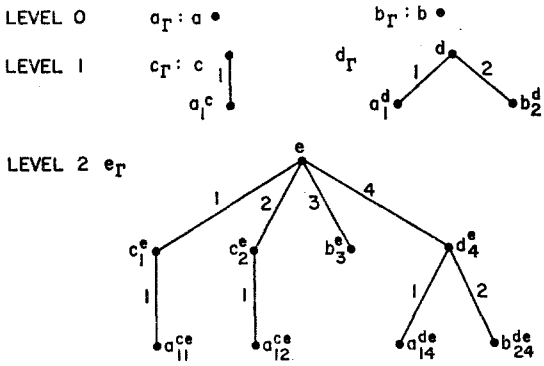
(1.12) EXAMPLE.




$V = \{a, b, c, d, e\}$

| $v$ | $a$ | $b$ | $c$ | $d$ | $e$ |
|---|---|---|---|---|---|
| $\Gamma(v)$ | $\Lambda$ | $\Lambda$ | $a$ | $ab$ | $ccbd$ |

Thus $a$ and $b$ are initial, only $e$ is terminal, and $d_\Gamma = 2$. We obtain ${}^e\Gamma$ as follows:




(1.13) DEFINITION. A *Σ-tree* is a Σ-DOAG $(\Gamma, h)$ for which $\Gamma$ is a tree.

Given a Σ-DOAG $(\Gamma, h)$ and a vertex $v$, then $({}^v\Gamma, {}^vh)$ is the Σ-tree with underlying tree ${}^v\Gamma$ for which ${}^vh(u) = h(w)$ if $u$ is a $w$-node of ${}^v\Gamma$. ${}^vh$ is thus well defined.

We may now describe the computations associated with a Σ-DOAG:

(1.14) DEFINITION. Given a Σ-DOAG $(\Gamma, h)$ and a Σ-algebra $\mathfrak{A} =$

$(Q, \alpha)$, the *evaluation* of $(\Gamma, h)$ by the algebra $\mathfrak{A}$ is the $Q$-DOAG $\Lambda\,(\Gamma, h_{\mathfrak{A}})$ where $h_{\mathfrak{A}}$ is defined as follows:

$v \in \Gamma$ of level 0; $h_{\mathfrak{A}}(v) = h(v)$ $(\Lambda)$ which is defined only if $h(v)$ has in its domain.[3]

$v \in \Gamma$ of level $k + 1$: $h_{\mathfrak{A}}(v) = h(v)[h_{\mathfrak{A}}({}^1\mathbf{v}), \cdots, h_{\mathfrak{A}}({}^n\mathbf{v})]$ which is defined only if each $h_{\mathfrak{A}}({}^i\mathbf{v})$ is defined, and $[h_{\mathfrak{A}}({}^1\mathbf{v}), \cdots, h_{\mathfrak{A}}({}^n\mathbf{v})]$ is in the range of $h(v)$.[3]

(1.15) DEFINITION. If $M = (Q, \alpha, Y, \lambda)$ is a $\Sigma$-automaton with underlying algebra $\mathfrak{A} = (Q, \alpha)$, then the evaluation of a $\Sigma$-DOAG $(\Gamma, h)$ by $M$ is the $Y$-DOAG $(\Gamma, h_M)$ where $h_M(v) = \lambda(h_{\mathfrak{A}}(v))$.

Note that the 2 definitions agree if we identify a $\Sigma$-algebra $\mathfrak{A} = (Q, \alpha)$ with the $\Sigma$-automaton $(Q, \alpha, Q, 1_Q)$ where $1_Q$ is the identity map on $Q$. Interpretation: A $\Sigma$-DOAG represents a loop-free computation structure. The labels on the initial nodes are instructions for setting up initial data. The labels on the other nodes are instructions for processing data. A $\Sigma$-algebra $\mathfrak{A}$ is simply a set of rules for interpreting the instruction set $\Sigma$. The instructions are executed level by level, to yield at each node the result of the subcomputation leading to that node. Finally, $\lambda$ may be applied to obtain the output from a given register at a given stage of the computation.

(1.16) LEMMA. *Given any* $\Sigma$-DOAG $(\Gamma, h)$, *any* $\Sigma$-*automaton* $M$, *and any node* $v \in V$ *with associated "unfolded tree"* $({}^v\Gamma, {}^vh)$ *we have*

$$ {}^vh_M(v) = h_M(v) $$

*Proof.* Tedious, but elementary, induction.

Thus any computation given by a DOAG is equivalent to a computation given by a set of disjoint trees. For proving mathematical theorems, it is usually easier to work with the trees; in looking for economical structurings for actual computations we will prefer DOAGs. In a later section we shall discuss minimization of automata, i.e., choosing the smallest $Q$ for a class of computations. But here we would emphasize the importance of the DOAGs as giving us the right class of structures within which to minimize a program for a class of machines.

The laws of a variety (cf. the discussion following 2.4) allow us to reduce DOAGs. The details of a specific algebra permit even further reductions. However, such minimization of a DOAG corresponds to the solution of a word problem, and so will not always be effectively possible.

---

[3] Note that this condition has already been guaranteed in case $\Sigma$ is graded.

(1.17) DEFINITION. A *finite* $\Sigma$-*automaton* is one with $\Sigma$-algebra $(Q, \alpha)$ for which $\Sigma$ and $Q$ are finite.

(1.18) Henceforth we shall consider $\Sigma$ to be finite.

(1.19) DEFINITION. Given a finite $\Sigma$-automaton $M = (Q, \alpha, Y, \lambda)$ and an element $y$ of $Y$, we define *the set of* $\Sigma$-DOAGs *accepted by* $M$ *with output* $y$ to be

$$D_y(M) = \{(\Gamma, h) \mid v \text{ a terminal node of } \Gamma \Rightarrow h_M(v) = y\}$$

By (1.16), $(\Gamma, h) \in D_y(M)$ iff $({}^v\Gamma, {}^vh) \in D_y(M)$ for all terminal nodes $v$ of $\Gamma$. To characterize $D_y(M)$ it will thus suffice to characterize $T_y(M)$, the set of $\Sigma$-trees in $D_y(M)$.

(1.20) DEFINITION. We say a set $S$ of $\Sigma$-trees is $\Sigma$-*recognizable* iff there exists a finite $\Sigma$-automaton $M$ and output $y$ such that $S = T_y(M)$.

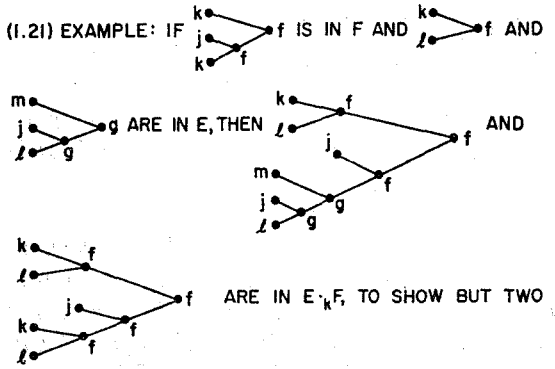
We wish to prove the analog of the Kleene theorem (1596) due to Thatcher and Wright (1966). Think of $\Sigma$-trees as branching to the left.[4] Then define:

(1.20) DEFINITION. For 2 sets $E$ and $F$ of $\Sigma$-trees and a nullary operator $k$ in $\Sigma$:

$E \cdot_k F$ = the set of $\Sigma$-trees obtained by taking trees of $F$ and replacing each initial node labelled $k$ by a tree from $E$

$E^{*k} = \bigcup_{n=0}^{\infty} E^{n;k}$ where $E^{0;k} = \{k\}$, the $\Sigma$-tree with one node, which node is labelled $k$ and $E^{n+1;k} = E \cdot_k E^{n;k}$

$E \cup F$ = the union of the sets $E$ and $F$




(1.22) DEFINITION. A set of trees is $\Sigma$-*regular* if it can be obtained

[4] Thatcher and Wright treat trees as branching to the right. However, we prefer to conform to the usage of ordinary automata theory where $x_1$ is the first symbol of the string $x_1 \cdots x_n$.

by a finite number of applications of $\cup$, $\cdot_k$ and $^{*k}$ for $k \in \Sigma_0$ from the finite sets of $\Sigma$-trees.

Recalling the definition (1.1) of $\Sigma_Q$, we may state:

(1.23) THEOREM. *If $\Sigma$ is a graded set, and a set $E$ of $\Sigma$-trees is recognizable by a finite $\Sigma$-automaton $M$ with state set $Q$, then $E$ is $\Sigma_Q$-regular.*

*Proof.* Let $j \in Q$, $t, s \subset Q$. Consider $\Sigma_Q$-trees, not $\Sigma$-trees. Let $M$ have underlying algebra $\mathfrak{A} = (Q, \alpha)$. Let $R^t_{s,j}$ = the set of $\Sigma_Q$-trees $(\Gamma, h)$ with $h_A(v) \in s$ for $v$ initial, $h_A(v) = j$ for the terminal $v$, and $h_A(v) \in t$ for all other $v$. We shall show, by induction, that every $R^t_{s,j}$ is $\Sigma_Q$-regular.

Basis: $R^\phi_{s,j}$ consists of trees of depth $\leqq 1$ and so is finite and thus regular for all $s \subseteq Q, j \in Q$.

Induction Step: Then for any $j$ and $s$ and any $j \notin t$

$$R^{t \cup \{k\}}_{s,j} = R^t_{s,k} \cdot_k (R^t_{s \cup \{k\},k})^{*_k} \cdot_k R^t_{s \cup \{k\},j}$$

and is regular, by induction. Note that we had to use $\Sigma_Q$-trees to make this step go through whether or not the intermediate state is in $\alpha(\Sigma_0)$. Finally,

$$T_y(M) = \bigcup_{\lambda(j)=y} R_{\alpha(\Sigma_0),j} \text{ is } \Sigma_Q\text{-regular.} \qquad \text{Q.E.D.}$$

This result suggests the following:

(1.24) DEFINITION. A set $E$ of $\Sigma$-trees is *regular* iff it is $\Sigma'$-regular for some $\Sigma'$ such that $\Sigma' - \Sigma \subseteq \Sigma_0'$.

This is, of course, the definition used by Thatcher and Wright (1966), who gave an example showing the *necessity* of extra nullary operators in the definition of regularity. However, our proof, above, of the result that every recognizable set is regular is satisfyingly shorter than theirs. Our proof of the converse is similar to theirs, but uses *startable* automata rather than their nondeterministic automata.

For our proof of the converse of (1.23) we need an auxiliary construct, corresponding to the automaton with starter input (though here we allow a starter for each element of $\Sigma_0$) used by Brzozowski (1962) in the classic case.

(1.25) Suppose we are given a $\Sigma$-automaton $M = (Q, \alpha, Y, \lambda)$. We define a new graded $\tilde{\Sigma} = (\Sigma \times 2^{\Sigma_0}, \tilde{\sigma})$ where $\tilde{\sigma}(f, s) = \sigma(f)$. Given a $\tilde{\Sigma}$-tree, a $\Sigma$-subtree is obtained by replacing the label $(f, s)$ of a node by either $f$ or some $q \in s$, and then deleting all nodes $<$ nodes labelled with some $q$. Given the set $E = T_y(M)$ we shall construct a $\Sigma_Q$-automaton $\tilde{M} = (2^Q, \tilde{\alpha}, Y, \tilde{\lambda})$ for which $T_y(\tilde{M})$ = the set of $\tilde{\Sigma}$-trees $t$ such that at least one $\Sigma$-subtree of $t$ is in $T_y(M)$. Any $\tilde{M}$ with this property is called a

*startable acceptor* for $T_y(M)$. Explicitly:

If $\sigma(f) = 0$: $\tilde{\alpha}_{(f,s)} = \{\alpha_f\} \cup s$,

If $\sigma(f) = n$: $\tilde{\alpha}_{(f,s)}(s_1, \cdots, s_n) = s \cup \bigcup_{q_j \in s_j} \alpha_f(q_1, \cdots, q_n)$,

$\tilde{M}$ has state-set $2^Q$, and for $s \subseteq Q$ we set $\tilde{\lambda}(s) = y$ iff $y \in \lambda(s)$.

(1.26) Note that if we restrict a startable acceptor of $E$ to trees over $\Sigma \times \{\phi\}$ we may consider it to be a $\Sigma$-automaton which recognizes $E$.

(1.27) THEOREM. *If a set $E$ of $\Sigma$-trees is $\Sigma'$-regular for some graded $\Sigma$, $\Sigma'$ with $\Sigma' - \Sigma \subset \Sigma_0'$ then $E$ is $\Sigma$-recognizable.*

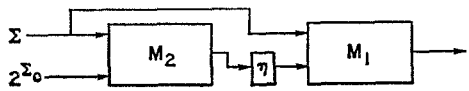
*Proof*: The proof is by induction, showing that each such $E$ has a startable acceptor. The result then follows from (1.26).

Basis: If $E$ contains only trees of depth $\leqq 1$, it clearly has a startable acceptor.

Induction:

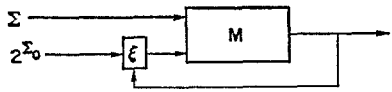
1. If $E_1 = T_{y_1}(M_1)$ and $E_2 = T_{y_2}(M_2)$ where $M_1$ and $M_2$ are startable acceptors, then $E_1 \cup E_2 = T_{\tilde{y}}(M_1 \times M_2)$ where, if $M_1 = (Q_1, \alpha_1, Y_1, \lambda_1)$ and $M_2 = (Q_2, \alpha_2, Y_2, \lambda_2)$ the product automaton $M_1 \times M_2 = (Q_1 \times Q_2, \alpha_1 \times \alpha_2, Y_1 \times Y_2, \tilde{\lambda})$, where $(\alpha_1 \times \alpha_2)_f(q_1, q_2) = (\alpha_1(q_1), \alpha_2(q_2))$ and $\tilde{\lambda}(q_1, q_2) = \tilde{y}$ iff $\lambda_1(q_1) = y_1$ or $\lambda_2(q_2) = y_2$, i.e., $M_1 \times M_2$ is just the machines $M_1$ and $M_2$ acting "in parallel".

2. Suppose $M_1$ is a startable acceptor for $E_1$ and $M_2$ is a startable acceptor for $E_2$. Then $M$, diagrammed as




is a startable acceptor for $E_2 \cdot_k E_1$ where $\eta(z) = \textit{if } z = y \textit{ then } \{k\} \textit{ else } \phi$, where $y$ is the output corresponding to acceptance of a tree in $E_2$. Formally $M = (Q_1 \times Q_2, \alpha, Y_2, \lambda_2 \circ \pi_2)$ where $\alpha_{(f,s)}(q_1, q_2) = (\alpha_{1(f,s)}(q_1), \alpha_{(f,\eta(q_1))}(q_2))$ and $\lambda_2 \circ \pi_2(q_1, q_2) = \lambda_2(q_2)$.

3. Suppose $M$ is a startable acceptor for $E$. Then $N$, diagrammed as




is a startable acceptor for $E^{*k}$ where $\xi(s, z) = s \cup \eta(z)$ where $\eta(z) = \textit{if } z = y \textit{ then } \{k\} \textit{ else } \phi$, with $y$ the output corresponding to acceptance of a tree in $E_2$. Formally $N = (Q, \alpha', Y, \lambda)$ where

$$\alpha'_{(f,s)}(q) = \alpha_{(f,s \cup \eta(\lambda(q)))}(q). \qquad \text{Q.E.D.}$$

The proof of (1.23) rests heavily on the finiteness of the state set, but only the basis step depends on our assumption that $\Sigma$ is graded (i.e., that $\mathfrak{A}$ is uniform). If we allow $\Sigma$ to be ungraded, then we must make some other assumption to ensure the truth of an analogous theorem. This corroborates Thatcher's introduction (1967) of $\Sigma$-algebras in which the condition of uniformity for a finite algebra is replaced by the demand that $\alpha_f^{-1}(q)$, for each $f \in \Sigma$ and each $q$ in $Q$, is regular in the sense of Kleene. It is then easy to see that the whole theory (i.e., Theorems 1.23 and 1.27) goes through if we use the notion:

(1.28) DEFINITION. A set $S$ of $\Sigma$-trees is *super-regular* if it can be built up by $\cup$, $\circ_k$ and $^{*k}$ for $k \in \Sigma_0$ from sets of $\Sigma$-trees of the form $(f \cup \Gamma(f), h)$ when $f \in \Sigma$ is fixed, and $h(\Gamma(f))$ runs over a subset of $\Sigma^*$ regular in the sense of Kleene.

Note that because of our branching restrictions on uniform algebras, this reduces to (1.24) if $\Sigma$ is a graded set. Now (1.23) and (1.27) combine to yield:

(1.29) THEOREM. *A set of $\Sigma$-trees is $\Sigma$-recognizable if and only if it is super-regular.*

## 2. DOAG-FUNCTIONS AND THE MINIMIZATION OF ALGEBRA AUTOMATA

We continue to regard a $\Sigma$-DOAG as a program structure, but rather than specify how each initial register is to be loaded (by labelling each initial node with an element of $\Sigma_0$) we now allow some registers to be loaded differently in different runs (by labelling an initial node with an integer $i$ if, on each run, it is to be loaded with the $i$th data item specified for each run).

For convenience, let us use $N^+$ to denote $\{1, 2, 3, \cdots\}$ and $N$ for $\{0, 1, 2, \cdots\}$ and let $[p] = \{1, 2, \cdots, p\}$ so that $[0] = \phi$. Thus, for $m < n$, every $\Sigma_{[m]}$-DOAG is a $\Sigma_{[n]}$-DOAG is a $\Sigma_{N^+}$-DOAG. Conversely, since our DOAGs are finite, each $\Sigma_{N^+}$-DOAG is a $\Sigma_{[p]}$-DOAG for some $p$. Finally $\Sigma$-DOAGs are just the same as $\Sigma_{[0]}$-DOAGs.

(2.1) Given a $\Sigma_{[p]}$-DOAG $\beta = (\Gamma, h)$ and $p$ $\Sigma_{N^+}$-trees $t_1, \cdots, t_p$ we define

$$(t_1, \cdots, t_p)\beta = ((t_1, \cdots, t_p)\Gamma, h_{(t_1,\ldots,t_p)})$$

to be the $\Sigma_{N^+}$-DOAG obtained from $(\Gamma, h)$ by replacing each initial node labelled $i \in [p]$ by a suitable copy of the tree $t_i$.

As in (1.20) we shall identify $\{k\}$ with the tree with a single node, that node being labelled $k$. Thus $(\{1\}, \cdots, \{p\})\beta = \beta$.

(2.2) Given a *computer* (a $\Sigma$-algebra $\mathfrak{A} = (Q, \alpha)$, a *program* $((\Gamma, h)$, a $\Sigma_p$-DOAG for some $p \in N)$, and an *output location specification* (an element $\xi = (w_1, \cdots, w_n)$ of $V^*$), we may define an associated function

$$\varphi^{\mathfrak{A};\xi}_{(\Gamma,h)}: Q^{[p]} \to Q^{[n]}$$

with

$$(q_1, \cdots, q_p)\varphi^{\mathfrak{A};\xi}_{(\Gamma,h)} = ([h_{(\{q_1\},\ldots,\{q_p\})}]_{\mathfrak{A}}(w_1), \cdots, [h_{(\{q_1\},\ldots,\{q_p\})}]_{\mathfrak{A}}(w_n))$$

i.e., we load each node labeled $i \in [p]$ with the appropriate initial data $q_i \in Q$, then use $\mathfrak{A}$ to evaluate the $Q$-DOAG so obtained, and then read the result of the computation off the nodes $w_1, \cdots, w_n$ in that order.

(2.3) DEFINITION. Given a $\Sigma$-algebra $\mathfrak{A} = (Q, \alpha)$ we define the *completion* of $\mathfrak{A}$, $\langle\mathfrak{A}\rangle$, to be the set of all maps $\varphi: Q^{[n]} \to Q^{[p]}$ for which there exists a $\Sigma_{N^+}$-DOAG $(\Gamma, h)$ and $\xi \in V^*$ such that

$$\varphi = \varphi^{\mathfrak{A};\xi}_{(\Gamma,h)}$$

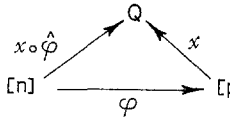
We may refer to $(\Gamma, h)$ as a *program for $\varphi$ to be run on $\mathfrak{A}$.*

We now state the theorem, whose simple verification is left to the reader, which underlies the categorical approach to algebra automata which we adopt, following Eilenberg and Wright (1967), in Part II.

(2.4) THEOREM. *Let $(\Sigma, \sigma)$ be a graded set, and $\mathfrak{A} = (Q, \alpha)$ a $\Sigma$-algebra. Then $\langle\mathfrak{A}\rangle$, the completion of $\mathfrak{A}$, is the smallest collection of maps $Q^{[n]} \to Q^{[p]}$ for $n, p \in N$ containing*

(i) *all maps $\alpha_f: Q^{\sigma(f)} \to Q$ for $f \in \Sigma$;*

(ii) *for each set-theoretic mapping $\varphi: [n] \to [p]$ the associated mapping $\hat{\varphi}: Q^{[p]} \to Q^{[n]}$ such that*



commutes,

i.e., $\hat{\varphi}: (q_1, \cdots, q_p) \mapsto (q_{\varphi(1)}, \cdots, q_{\varphi(p)})$ (*such maps $\hat{\varphi}$ are called* ***trivial*** *maps) and which is closed under the operations*:

(iii) from $\varphi_j: Q^{[p]} \to Q^{[1]}$, $j = 1, \cdots, n$ form $\varphi = (\varphi_1, \cdots, \varphi_n): Q^{[p]} \to Q^{[n]}$ by

$$(\tilde{q})\varphi = ((\tilde{q})\varphi_1, \cdots, (\tilde{q})\varphi_n)$$

(iv) from $\varphi: Q^{[p]} \to Q^{[n]}$ and $\psi: Q^{[n]} \to Q^{[r]}$ form

$$\psi \circ \varphi: Q^{[p]} \to Q^{[r]}$$

We digress briefly to recall the notion of a variety of algebras. (The reader may find a wealth of further details in Cohn (1965), Chapter IV.) In our present vocabulary, a *law* is a statement that two $\Sigma_N$+-DOAGs compute the same function. A *variety* of $\Sigma$-algebras is then the collection of all $\Sigma$-algebras which satisfy a given set of laws. As we commented after Lemma (1.16), laws allow us to reduce the size of a $\Sigma$-DOAG without changing the function computed by any algebra satisfying the laws. Thus we can reduce $\Sigma$-DOAGs using a set of laws without changing the evaluation of the DOAG by any $\Sigma$-algebra of the variety defined by those laws.

Given a variety $V$, we can then say which of the functions, obtained from the functions $\alpha_f$ of a $\Sigma$-algebra $\mathfrak{A}$ of $V$ by repeated applications of (iii) and (iv) above using the trivial functions of (ii), must be identical, simply by virtue of $\mathfrak{A}$'s membership in $V$, irrespective of the vagaries of the particular choices of $\alpha_f$. This observation allows a recasting of the notion of variety in categorical terms. This has been done by Lawvere (1963) who introduced the notion of a *theory*, and the notion of a $T$-algebra for each theory $T$. For each variety $V$ there exists a theory $T$ such that $\mathfrak{A}$ is a $T$-algebra iff it belongs to the variety $V$. The notion of a theory has been introduced into the study of algebras considered as automata by Eilenberg and Wright (1967). We shall pursue this categorical approach in Part II. Meanwhile, we shall continue our automaton-theoretic approach, and study minimization of algebra automata.

(2.5) DEFINITION. Let $\mathfrak{I}_\Sigma$ be the set of $\Sigma$-trees. A $\Sigma$-automaton $M = (Q, \alpha, Y, \delta)$ induces a function $f_M: \mathfrak{I}_\Sigma \to Y$ by the definition (cf. (1.15))

$$f_M(t) = h_M(v)$$

where $v$ is the terminal node of the $\Sigma$-tree $t = (\Gamma, h)$. We call $f_M$ the *response function of* $M$. More generally, we call any function $f: \mathfrak{I}_\Sigma \to Y$ a *response function.*

We are interested in the question: "When is a response function equal to the response function of a finite algebra automaton?" We proceed by constructing a minimal algebra automaton for each response function. The question is answered by checking to see whether or not it has a finite state set. We first note a lemma whose easy proof is left to the reader:

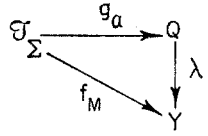
(2.6) LEMMA. *If two $\Sigma$-automata have the same response function, then they yield the same evaluation on every $\Sigma$-DOAG.*

We shall find it convenient to write $t_1 \cdots t_n\varphi$ for the $\Sigma$-tree with terminal node $v$ labelled $\varphi$, and with $t_j$ the $\Sigma$-tree depending from node $^j\mathbf{v}$.

(2.7) The *memory function* of a $\Sigma$-algebra $\mathfrak{A} = (Q, \alpha)$ is the map $g_{\mathfrak{A}}: \mathfrak{I}_\Sigma \to Q$ defined by $g_{\mathfrak{A}}((\Gamma, h)) = h_{\mathfrak{A}}(v)$ for $v$ the terminal node of $\Gamma$. We note that this entails

$$g_A(t_1 \cdots t_n\varphi) = \alpha_\varphi(g(t_1), \cdots, g(t_n)).$$

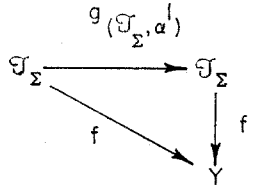
If $M = (Q, \alpha, Y, \lambda)$, the definition of $f_M$ just says that the following diagram commutes:




We may write $g_M$ for $g_{\mathfrak{A}}$ if $\mathfrak{A} = (Q, \alpha)$ for $M = (Q, \alpha, Y, \lambda)$.

(2.8) We say $M$ *realizes* $f$ if $f = f_M$.

Every response function $f: T_\Sigma \to Y$ has a trivial realization, the *free* realization $(\mathfrak{I}_\Sigma, \alpha^1, Y, f)$ where $\alpha_\varphi{}^1(t_1, \cdots, t_n) = t_1 \cdots t_n\varphi$ and since $g_{(\mathfrak{I}_\Sigma, \alpha^1)} = 1_{\mathfrak{I}_\Sigma}$

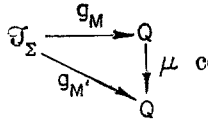



certainly commutes. $1_{\mathfrak{I}_\Sigma}$ "remembers" the whole "input history" of $f$.

More interesting is the question "what is the minimal information about 'past input history' that must be retained to give $f$?", it being understood that a mechanism (namely $\alpha$) must exist for updating this record.

(2.9) DEFINITION. We say that $M' = (Q', \alpha', Y, \lambda')$ is a *minimal realization* of $f$ if for any other realization $M = (Q, \alpha, Y, \lambda)$ for $f$ there exists a map $\mu$ of $Q$ *onto* $Q'$ such that the diagram

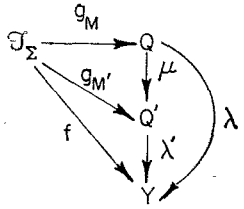
(2.10)

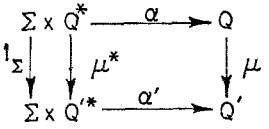



commutes.

If $Q = g_M(\mathfrak{T}_\Sigma)$, then (2.10) entais that both the following diagrams commute:

(2.11)




Let us now recall that:

(2.12) Given a $\Sigma_{[1]}$-tree $t$ and a $\Sigma$-tree $t'$, $(t')t$ denotes the tree obtained by replacing the initial vertices of $t$ labelled 1 by a copy of $t'$.

We may now generalize the Nerode equivalence (see Nerode, 1958 and Rabin and Scott, 1959). An essentially equivalent approach to minimization was obtained by Brainerd (1967), pp. 48–57, who must be given credit for first generalizing the classic treatment. However, we felt our approach to be sufficiently different in form, though not in content, to merit its inclusion here. We refer the reader to Brainerd's thesis for an explicit algorithm for minimization of a given finite algebra automaton.

(2.13) DEFINITION. Given a response function $f: \mathfrak{T}_\Sigma \to Y$, we define the *f-equivalence relation* $E_f$ on $\mathfrak{T}_\Sigma$ by setting

$$t' E_f t'' \Leftrightarrow f((t')t) = f((t'')t) \quad \text{for all} \quad t \in \mathfrak{T}_{\Sigma_{[1]}}$$

We set $Q_f = \mathfrak{T}_\Sigma / E_f$ and for $\varphi \in \Sigma_n$ we define

$$\alpha_\varphi^f([t_1], \cdots, [t_n]) = [t_1 \cdots t_n \varphi]$$

We set $\lambda^f([t]) = f(t)$.

It is clear that $\alpha_\varphi^f$ and $\lambda^f$ do not depend on the choices of $t$ or $t_j$ within an equivalence class.

(2.14) We call $\mathfrak{A}_f = (Q_f, \alpha^f)$ the *reduced* $\Sigma$*-algebra* of $f$, and $M(f) = (Q_f, \alpha^f, Y, \lambda^f)$ the *reduced automaton* of $f$.

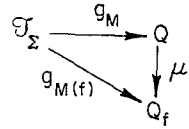
(2.15) THEOREM. *$M(f)$ is a minimal realization of $f$.*

*Proof.* Let $M = (Q, \alpha, Y, \lambda)$ be any other $\Sigma$-automaton realizing $f$. Then define $\mu: Q \to Q_f$ by the equation

$$\mu(q) = \begin{cases} g_f(t) & \text{if} \quad q = g(t) \in g(T_\Sigma) \\ \text{arbitrary in } Q_f, & \text{otherwise.} \end{cases}$$

A routine computation then verifies that $\mu$ is a well-defined map of $Q$

onto $Q_f$ which makes




commute. Q.E.D.

We may now generalize the Myhill equivalence (see Rabin and Scott, 1959 and Myhill, 1957) by operating on $\Sigma_N$+-trees rather than $\Sigma$-trees.

(2.16) DEFINITION. Let $f: \mathfrak{I}_\Sigma \to Y$ be a response function, and $Q_f$ the state-space $\mathfrak{I}_\Sigma/E_f$. For each $p \geqq 0$ and each $\Sigma_{[p]}$-tree $t$ we define a map

$$[t]_p : Q_f^{[p]} \to Q_f$$

by setting $[t]_p([t_1]_f, \cdots, [t_p]_f) = [(t_1, \cdots, t_n)t]_f$ which is well defined by the definition of $E_f$. We say $\Sigma_{[p]}$-trees $t$ and $t'$ are *f-congruent* $t \equiv_f t'$ just in case $[t]_p = [t']_p$. We denote by $T_p^f$ the set $\mathfrak{I}_{\Sigma_{[p]}}/\equiv_f$ and denote by $T^f$ the sequence $(T_0^f, T_1^f, T_2^f, T_3^f, \cdots)$.

The reader may readily verify the following theorem:

(2.17) THEOREM. *Every map* $\varphi: Q_f^{[p]} \to Q_f^{[n]}$ *of the completion* (2.3) $\langle \mathfrak{A}_f \rangle$ *of the reduced* $\Sigma$*-algebra* $\mathfrak{A}_f$ *for* $f$ *can be uniquely expressed in the form* (cf. 2.4 (iii))

$$\varphi = (\varphi_1, \cdots, \varphi_n)$$

*where each* $\varphi_j: Q_f^{[p]} \to Q_f$ *is a member of* $T_p^f$.

We can best appreciate this result if we recall the situation in ordinary automata theory. There we always have a unique initial state, $q_0$ say. Thus when we specify a string $x_1 \cdots x_n$ we may really mean

$$q_0 x_1 \cdots x_n$$

the element of $\mathfrak{I}_\Sigma$ uniquely defined by $x_1 \cdots x_n$, which may be considered as representing a function $Q^{[0]} \to Q$, or we may mean

$$1\, x_1 \cdots x_n$$

the element of $\mathfrak{I}_{\Sigma_{[1]}}$ uniquely defined by $x_1 \cdots x_n$, which may be considered as representing a function $Q^{[1]} \to Q$. In the first case we are led to the Nerode relation and in the second case we are led to the Myhill relation. What the general algebraic approach shows is that, in the general case, the Nerode and Myhill equivalence relations correspond to the first two steps, $T_0^f$ and $T_1^f$, of an infinite series, rather than the two separate entities they often seem to be in the ordinary theory. We note that $\Sigma_{[p]}$-trees $t'$ and $t''$ are $f$-equivalent iff for all $\Sigma_{[1]}$-trees $t'$, and all $\Sigma$-trees $t_1, \cdots, t_p$, we have

$$f[((t_1, \cdots, t_p)t')t] = f[((t_1, \cdots, t_p)t'')t]$$

We now have all the automaton-theoretic apparatus necessary to facilitate the connection between the automaton approach and the categorical approach to the theory of algebras considered as automata. In Part II, we recall and extend the Eilenberg–Wright categorical approach, with emphasis on concepts necessary to properly treat the dynamics of our algebra automata.

RECEIVED: February 6, 1968

## REFERENCES

THATCHER, J. W. AND WRIGHT, J. B. (1966), "Generalized Automata Theory with an Application to a Decision Problem of Second-Order Logic," IBM Research RC 1713.

THATCHER, J. W. (1967), "A Further Generalization of Finite Automata," IBM Research RC 1846.

MEZEI, J. AND WRIGHT, J. B. (1965), "Generalized ALGOL-like languages," IBM Research Paper RC 1528.

THATCHER, J. W. (1967), "Characterizing Derivation Trees of Context-Free Grammars through Generalized Finite Automata Theory," IBM Research Note NC 719.

BÜCHI, J. R. (1960), "Mathematical Automata Theory," Lecture notes, University of Michigan.

EILENBERG, S. AND WRIGHT, J. B. (1967), Automata in General Algebras. *Inform. Control* **11**, 452–470.

LAWVERE, F. W. (1963), Functorial semantics of algebraic theories. *Proc. Natl. Acad. Sci. (USA)*, **50**, 869–872.

BIRKHOFF, G. (1935), On the structure of abstract algebras. *Proc. Cambridge Phil. Soc.* **31**, 433–454.

KLEENE, S. C. (1956), Representation of events in nerve nets and finite automata *in* "Automata Studies" (C. E. Shannon and J. McCarthy, Eds.), Princeton Univ. Press, pp. 3–41.

BRZOZOWSKI, J. (1962), A survey of regular expressions and their applications. *IRE Trans. Electron. Computers* **EC-11**, 324–335.

COHN, P. M. (1965), "Universal Algebra," Harper, New York.

NERODE, A. (1958), Linear automaton transformations. *Proc. Am. Math. Soc.* **9**, 541–544.

RABIN, M. O. AND SCOTT, D. (1959), Finite automata and their decision problems. *IBM J. Res. Dev.* **3**, 114–125.

BRAINERD, W. S. (1967), "Tree Generating Systems and Tree Automata," Ph.D. Thesis, Purdue University.

MYHILL, J. (1957), "Finite Automata and the Representation of Events," WADC Tech. Report 57-264.

GIVE'ON, Y. (1967), Categories of semimodules: The categorical structure properties of transition systems. *Math. Systems Theory* **1**, 67–78.