[illegible]

7/24/68

ABSTRACT DIGITAL COMPUTERS AND AUTOMATA

A THESIS

Presented to

The Faculty of the Division of Graduate

Studies and Research

by

Robert Charles Roehrkasse

In Partial Fulfillment

of the Requirements for the Degree

Doctor of Philosophy

in the School of Information and Computer Science

Georgia Institute of Technology

September, 1971

ABSTRACT DIGITAL COMPUTERS AND AUTOMATA

Approved:

[illegible]

Chairman: Lucio Chiaraviglio

[illegible]

Visiting Member: Trevor Evans
Chairman, Department of Mathematics
Emory University

[illegible]

Member: John M. Gwynn, Jr.

[illegible]

Member: Joseph Talavage

Date approved by Chairman: Sept 20, 1971

ACKNOWLEDGMENTS

I would like to thank Professor V. Slamecka for his continued assistance throughout my association with the School of Information and Computer Science. For their suggestive criticism and encouragement, I wish to thank the members of my doctoral guidance committee: Professors Gough, Gwynn, and Talavage.

I would like to thank my fellow doctoral student Joseph R. Horgan for his interest and assistance throughout the research. Both proved most beneficial and are greatly appreciated.

I would like to thank Professor Trevor Evans of Emory University for his willingness to review a draft of the dissertation and to serve as visiting member of the Doctoral Examining Committee.

Above all, I am especially indebted to Professor Lucio Chiaraviglio for his continual encouragement and guidance. His professional dedication and competence have given me the intellectual motivation and confidence to continue my academic endeavors.

I am very grateful for the educational opportunity provided by the United States Air Force. Under Air Force sponsorship, I have been able to earn a Master of Science and the Doctor of Philosophy from the Georgia Institute of Technology.

I furthermore believe it is most appropriate that I acknowledge the individual whose sincere interest and guidance led me to enter the military profession, Colonel James F. Van Ausdal, my former Professor of Aerospace Studies at Southern Illinois University.

TABLE OF CONTENTS

CHAPTER I

INTRODUCTION

Toward a Unifying Approach for Automata Theory

The theory of linguistic automata is a branch of automata theory that is of particular interest to Information and Computer Science. The theory of linguistic automata is concerned with the relationship between formal languages and their recognizers or acceptors (essentially abstract machines without output). This branch of automata theory received its initial impetus from Chomsky [6]. A first application to computer science was gained when the syntax of ALGOL was defined by a context-free grammar [9]. And from this beginning, the ideas of syntax-directed compilation and the concept of a compiler compiler were developed. At present formal languages and automata are so intertwined that to speak of one is to speak of the other. The results that followed emphasized the properties of language families as defined by generative schemes (grammars) or as the acceptance sets of devices (recognizers). The aim has been to characterize each class of languages by a kind of automaton through theorems of the form: If L is generated by a grammar of type X then there is an automaton of kind Y which recognizes precisely L. And, also to characterize kinds of automata by theorems of the form: If L is recognized by an automaton of kind Y then there is a grammar of type X which generates precisely L [15]. A proliferation of results has stimulated efforts directed at finding a unifying approach to linguistic automata. The principal unifying results are those obtained by the teams of Ginsburg and Greibach [11] and Hopcroft and Ullman [18].

The work of Ginsburg and Greibach is oriented towards formal languages. Rather than obtaining a language by means of a grammar or recognizer, they defined a formal language as a family of sets of words called an abstract family of languages that is closed under the language-theoretic operations of union, product, Kleene closure, ϵ-free homomorphism, inverse homomorphism, and intersection of the language with a regular set. Using this approach, they were able to specify an abstract family of languages that corresponds to each of the classical families of formal languages (i.e., regular sets, context-free, context-sensitive, and recursively enumerable sets). As the automaton counterpart of an abstract family of languages, they devised a recognizer which they call an abstract acceptor. An abstract acceptor can be viewed as a non-deterministic finite automaton with auxiliary storage. The Ginsburg and Greibach version of the language-automaton characterization is that corresponding to every abstract family of acceptors there is an abstract family of languages and conversely, corresponding to every abstract family of languages there is an abstract family of acceptors [29].

In contrast, the work of Hopcroft and Ullman has a machine orientation. They introduced a recognizer which they call a balloon automaton. A balloon automaton is basically a linear bounded automaton with auxiliary storage. Their objective was to specify subsets of balloon automata, called closed classes, that would correspond to particular classes of recognizers. They proposed that whenever some new recognizer is devised one would show that it is equivalent to a closed class of balloon automata and thus have the language-theoretic properties of balloon automata automatically proven for it. However, they were unable to determine closed sets of balloon automata whose acceptance sets coincided with each of the classical families of languages.

This type of program has some inherent problems. For suppose that the intuitive motivation used in formulating a new recognizer can be captured by some prescription for translating the recognition procedure of this new recognizer into some standard abstract machine, as for example the balloon automaton. This involves some construction which shows how one machine "simulates" another. Fisher has perceived the difficulty of this procedure quite clearly:

> However the concept of simulation of one machine by another is extremely difficult to define precisely. For too stringent a definition excludes cases in which one intuitively feels a bona fide simulation is being performed. Too liberal a definition allows the use of encodings of input and output in which the real computational work is done by the encoding and decoding algorithms and not by the machine which is supposedly performing the simulation [10].

To obviate this difficulty, we need a precise mathematical methodology for ascertaining the relationship between one machine and another. An algebraic setting seems appropriate. We need to develop a standard abstract machine which is itself an algebra and a uniform prescription or method for reformulating other machines as algebras of the same type. The reformulation of a machine should capture its computational procedure so that it is reflected in the computational procedure of this standard machine. Once machines have been so captured, we can proceed to relate machines via the concepts of universal algebra.

We note that the significant accomplishments resulting from imposing an algebraic structure on machines have only dealt with abstract machines which are in essence variants of finite automata. The lattice-theoretic approach has been one such successful endeavor [16]. The point of departure for this approach has been the set of partitions on the state set of a sequential machine that satisfy the substitution property. A sequential machine (finite automaton with output) is a quintuple $M=<S,I,O,\delta,\lambda>$ where S is the state set, I is the input alphabet, O is the output alphabet, δ is the next state function from SxI to S, and λ is the output function. The function λ is either from S to O if M is taken as the Moore model or from SxI to O if M is taken as the Mealy model of a sequential machine. A partition π on the state set S has the substitution property if and only if $x\equiv y(\pi)$ implies that $\delta(x,a)\equiv\delta(y,a)(\pi)$ for all $a\epsilon I$. Partitions satisfying the substitution property are also called admissible partitions. The set of all such admissible partitions on the state set of a sequential machine then forms a finite lattice when two binary operations are defined on this set. The lattice-theoretic approach has led to the development of a theory of machine decomposition [24]. Machine decomposition considers whether and, if so, how a sequential machine can be constructed from other component machines (again the Mealy-Moore types) interconnected in series and parallel.

The semigroup approach is the other extensively developed means for imposing an algebraic structure on machines [22]. This theory deals not only with the question of machine decomposition but also with decomposition [24]. Machine composition concerns what behavior may be realized by interconnecting an arbitrary number of component machines of specific kinds (minimal building blocks). Machine composition for the semigroup approach involves the characterization of minimum essential kinds of component machines needed to realize the behavior of an arbitrary sequential machine.

The semigroup associated with a machine is the set of all transformations of its state set induced by its input strings. Let $M = \langle S,I,O,\delta,\lambda\rangle$ be a sequential machine. If $a \epsilon I$ and $x \epsilon S$ then $(x)a = \delta(x,a)$. The semigroup operation is functional composition. Hence, if $a, b \epsilon I$ and $x \epsilon S$ then $(x)a \circ b = [(x)a]b = \delta((x)a,b) = \delta(\delta(x,a),b)$. The carrier of such a semigroup will be finite if and only if the state set S of M is finite [21]. A machine M is called a reset machine if and only if every input regarded as a transformation on the state set is an identity or a constant mapping [16]. A semigroup S divides a semigroup R if there exists a subsemigroup $R' \subset R$ and there exists an epimorphism from R' to S [21]. The principal decomposition result is that any machine (Mealy-Moore type) can always be decomposed into simpler machines unless the original machine is a two-state reset machine or its associated semigroup is a simple group, i.e., the group has no proper normal subgroups. Conversely, the principal composition result is that any decomposable machine can be realized by loop-free synthesis from two-state reset machines and a subset of the machines derivable from the simple groups that divide the associated semigroup of the original machine [21].

The concept of heterogeneous algebra has been shown to provide a formalism which is adaptable for the treatment of sequential machines [4]. An algebra is a pair $<A,\mathcal{F}>$ where A is any set known as the carrier and $\mathcal{F}$ is a set of operations defined on the carrier A. In contrast, a heterogeneous algebra is a pair $<B,\mathcal{L}>$ where B is a non-empty family of sets, each of which is called a phylum, and $\mathcal{L}$ is a set of mappings from the Cartesian product of a subset of all phyla (B) to some other phylum of B. For example, the Mealy machine $M=<S,I,O,\delta,\lambda>$ formulated as a heterogeneous algebra becomes $<\{S,I,O\},\{\delta,\lambda\}>$ where $B=\{S,I,O\}$ contains three phyla and $\mathcal{L}=\{\delta,\lambda\}$ contains two mappings: δ from SxI to S and λ from SxI to O. Such algebraic concepts as subalgebra (submachine) and morphisms can be extended to machines via the heterogeneous algebras that characterized them. Similar algebraic concepts and methods are available in the lattice-theoretic and semigroup approaches mentioned. Various results can be obtained in all three cases using these algebraic concepts. Nevertheless, with the methodology developed from these approaches, only variants of finite automata can be dealt with. We will consider a wider set of machines and show how they can be meaningfully related.

The central contribution of this research to Information and Computer Science is the explication of the theory of automata by means of the algebraic concepts of abstract digital computer and computer morphism. These concepts are shown to be sufficient to yield a unified algebraic treatment of a large portion of automata theory. We do so by showing how the classical theory of recognizers can be captured within the theory of abstract computers.

Theory of Abstract Computers

Since the theory of abstract computers has been elaborated elsewhere [20,26,27], we will merely sketch the theory here in order to provide the basic understanding needed for this application.

Definition 1. The entity $<S,A,C>$ is an abstract computer if and only if

(1) $S \neq \emptyset$, the null set

(2) $\emptyset \neq A \subseteq S^S$

(3) $C \epsilon A^S$

(4) $S \cap A \neq \emptyset$.

An abstract computer consists of a set of states S, a set of actions A, and a control unit C. The microbehavior of this device is embodied in a reading-selecting-applying routine. If we have a state $x \epsilon S$, this routine is executed by C "reading" x to obtain C(x). But as $C(x) \epsilon A$, we are "selecting" an action, and subsequently "applying" this action to x to obtain the next state $(C(x))(x) \epsilon S$. Macrobehaviors are obtained by repetition of the reading-selecting-applying routine.

An abstract computer is a programmable, centrally-controlled, iterative, synchronous, non-interactive machine. Programming is the choice of an initial state and represents the extent of external control imposed on the computer. Once an initial state has been selected, the process that is generated is completely determined by the control unit, the central control of the computer. The state-transition function T for an abstract computer <S,A,C> is given by $T(x) = (C(x))(x)$ for every $x \epsilon S$. There are many distinct computers with the same state-transition function. We say that a computer is iterative if T^{τ} is defined for every finite ordinal τ. The powers of T are defined recursively by T^{o} being the identity on S and $T^{\tau+1} = TT^{\tau}$. The computer is said to be synchronous if time (here discrete) coincides with the powers of the state-transition function T. The descriptor "synchronous" incorporates the idea that if at time t_o the computer is in state x, then at time t_{τ} the computer will be in the state $T^{\tau}(x)$. Finally, "non-interactive" means that no member of S can act on another member of S to produce some state, the sets S and A being disjoint.

For each $x \epsilon S$, the state-transition function T generates a unique infinite sequence $T^{o}(x) = x, T(x), \ldots T^{\tau}(x), \ldots$ which we call a computational process. The set CP of all computational processes is a subset of S^{ω} where ω is the first limit ordinal omega. The process ρ is in CP if for every finite ordinal $\tau \epsilon \omega$, $\rho(\tau) = T^{\tau}(\rho(o))$. Such a process is terminal if there exists a $\tau \epsilon \omega$ such that $\rho(\tau) = \rho(\tau+1)$. The computer is said to stop in the state $\rho(\tau)$ when started in state $\rho(o)$ if τ is the smallest finite ordinal for which $\rho(\tau) = \rho(\tau+1)$. Because T is a function, processes are either non-repeating or periodic after a possible initial finite delay. Terminal processes are those that eventually have period one.

Definition 2. An abstract computer <S,A,C> is a finitary action computer if and only if

(1) $S \subseteq Y^X$

(2) For every $x \varepsilon S$, $[(C(x))(x)]$ and x are elements of S that differ only on a finite subset of X.

Definition 3. A finitary action computer $<S,A,C>$ is an abstract digital computer if and only if S is a Boolean algebra.

Since an abstract digital computer is a special case of the finitary action computer, the Boolean algebra must be functional. But by a corollary to Stone's representation theory for Boolean algebras, every Boolean algebra is isomorphic to a subalgebra of an θ-value Boolean algebra where $\theta = \{0,1\}$, i.e., the simple Boolean algebra [1]. In view of the isomorphism between a Boolean algebra B and the set of all continuous functions from the dual space of B into the simple Boolean space [14], an alternative definition of abstract digital computers is as follows:

Definition 4. $<B,A,C>$ is an abstract digital computer if and only if

(1) B is a Boolean algebra

(2) $A \subseteq B^B$

(3) $C \varepsilon A^B$

(4) For any $a \varepsilon A$ and $f \varepsilon B$, there exists a finite subset K of the dual space X of B such that $(af-f)(i) = 0$ for all $i \varepsilon (X \sim K)$.

Clause (4) states that the symmetric difference of (af) and f is a function in B whose value is zero for the set-theoretic difference $(X \sim K)$ of X and K, and whose value is one only for a finite subset K of the dual space of B.

A computer may be characterized as a triplet composed of a set of states, a set of actions, and a control function or as a pair composed of a set of states and a transition function or as a set of processes. In all of its characterizations a computer is an algebra and the concepts of universal algebra are applicable.

We may consider two senses of morphism for computers. First, two computers $\langle S,A,C\rangle$ and $\langle S^*,A^*,C^*\rangle$ are said to be strongly homomorphic if there is a mapping Φ from S to S^* such that it preserves actions and control units, i.e., for every $x,y \varepsilon S$, $\Phi(C(x)(y)) = C^*(\Phi(x))(\Phi(y))$. If S and S^* have a structure, say they are Boolean algebras, then we may also require that the mapping Φ preserve this structure. A weaker sense of morphism is obtained by requiring that the mapping Φ only preserve the state-transition function T, i.e., for any $x \varepsilon S$, $\Phi(C(x)(x)) = C^*(\Phi(x))(\Phi(x))$ or equivalently $\Phi(T_C(x)) = T_{C^*}(\Phi(x))$. This weaker sense of morphism is the one used throughout this presentation. We will refer to such mappings as computer morphisms. As before, we may also require that a computer morphism preserve state-set structure. A formal definition of computer morphism is as follows:

<u>Definition 5</u>. If $\langle S,A,C\rangle$ and $\langle S^*,A^*,C^*\rangle$ are abstract computers whose state-transition functions are T_C and T_{C^*}, then Φ is a computer morphism if and only if $\Phi \varepsilon (S^*)^S$ such that $\Phi(T_C(x)) = T_{C^*}(\Phi(x))$ for every $x \varepsilon S$.

Given the concept of computer morphism, we can ascertain that every abstract computer is embeddable in an abstract digital computer. If $\langle S,T\rangle$ is an arbitrary abstract computer, let Φ be an injection from

S into θ^S such that $(\Phi(x))(y) = 1$ iff $x=y$ for any $x,y \epsilon S$. Then $\phi(x)$ is an atom of the Boolean algebra θ^S. Define another state-transition function T* so that $T^*(\Phi(a)) = \Phi(T(a))$ if $\Phi(a)$ is an atom of θ^S, and $T^*(b) = b$ if b is not an atom of θ^S. Then $<\theta^S, T^*>$ is an abstract digital computer and Φ is a computer monomorphism. Note that the actions of $<\theta^S, T^*>$ need not be more complex than one-place set and reset functions. Finite resets and sets are defined as follows:

Definition 6. For any $f \epsilon \theta^X$ and finite subsets J of X, $(R(J))(f)$ and $(S(J))(f)$ are functions in θ^X such that for any $a \epsilon X$:

$$(1) \quad ((R(J))(f))\,(a) = \begin{cases} 0, & \text{if } a \epsilon J \\ f(x), & \text{if } a \not\epsilon J \end{cases}$$

$$(2) \quad ((S(J))(f))\,(a) = \begin{cases} 1, & \text{if } a \epsilon J \\ f(x), & \text{if } a \not\epsilon J. \end{cases}$$

If we let RS be the closure under functional composition of $R \cup S$, relative to the set X, then $<\theta^X, RS, C>$ is an abstract digital computer for some mapping C from θ^X to RS. Abstract digital computers of this type will be called finite set-reset computers. With these perceptions, we may introduce the following representation theorem for abstract digital computers.

Theorem 1. Every abstract digital computer is computer isomorphic to a finite set-reset computer.

Proof: Let <B,A,C> be an arbitrary abstract digital computer. We now wish to obtain a set-reset computer for B that has the same state-transition function. Let $f \epsilon B$ and $C(f) = a \epsilon A$. Then the control unit C^* for our set-reset computer is given as:

$$C^*(f) = S(((af) \wedge f')^{-1}(1))R(((af)' \wedge f)^{-1}(1)).$$

The set $((af)\ f')^{-1}(1)$ tells what values are changed from 0 to 1, and the set $((af)' \wedge f)^{-1}(1)$ tells what values are changed from 1 to 0. Both sets will be finite because actions ($a \epsilon A$) of an abstract digital computer are finitary. Thus, $C^*(f) \epsilon RS$ and $(C^*(f))(f) = (C(f))(f)$ for every $f \epsilon B$. Taking the actions of this set-reset computer as $A^* = \{S(K)R(J) | K,J \subseteq X$, the dual space of B.K and J are finite$\}$, the control unit C^* that calls these sets and resets for elements of B is isomorphic to $<B,A,C>$ because $(C^*(f))(f) = (C(f))(f)$ for every $f \epsilon B$.

Now that we have established part of the mathematical identity of abstract digital computers through a representation theorem, we may proceed with the task of capturing the classical theory of recognizers within the theory of abstract computers. As the representation theorem shows, the only concepts that we need to use are those of set-reset computers and computer morphisms.

CHAPTER II

THE CLASSICAL HIERARCHY OF RECOGNIZERS

Introduction

The formal grammars described by Chomsky provide a standard against which the recognition powers of automata are measured. For each class of languages generated by the Chomsky phrase-structure grammars (Type 0, 1, 2, and 3), there is a corresponding class of recognizers [17]. The induced hierarchy of recognizers ranges from Turing machines to finite automata [3]. For each class of recognizers there is both a deterministic and non-deterministic variety of machines. The languages generated by grammars of type 0 are called recursively enumerable sets and the Turing machines are the appropriate recognizers [6]. It has been shown that the deterministic and non-deterministic Turing machines have the same recognition power [15]. We deal with both varieties in order to provide a paradigm of the contrast between deterministic and non-deterministic machines and to facilitate a later proof of embedding via computer morphism. The languages generated by the grammars of Type 1 are context-sensitive. The appropriate recognizers are non-deterministic linear bounded automata [23]. It is still an open question whether non-deterministic linear bounded automata have greater recognition power than the deterministic ones [25]. The languages generated by the grammars of Type 2 are context-free. The appropriate recognizers are non-deterministic pushdown automata [7].

The non-deterministic variety of pushdown automata has greater recognition power than the deterministic one [5]. The languages generated by the grammars of Type 3 are the regular sets. The appropriate recognizers are the finite automata [8]. For finite automata, it has been shown that the non-deterministic and deterministic varieties have the same recognition power [28].

Throughout our formulation of automata theory, both the set of states of an automaton and its alphabet will be finite subsets of the natural numbers, $N = \{0,1,2,3,\ldots\}$. For convenience, the tapes of the automata are taken to be one-way infinite so that the natural members can be used to index the tape positions. Since all alphabets are finite subsets of the natural members, N^N includes every tape. The power set of a set A will be denoted as $P(A)$. The cardinality of a set A will be denoted as $\bar{\bar{A}}$. Primitive recursive subtraction will be denoted as $\dot{-}$. And, if x is an ordered n-tuple, the *ith* member of x will be denoted as ix, $1 \le i \le n$. If A is a set of n-tuples then $j\epsilon{:}A$ is the set of *jth* members of all n-tuples in A, i.e., $j\epsilon{:}A = \{x \mid (\exists w)(w \epsilon A . x = jw$ for w an n-tuple, $1 \le j \le n\}$. The letter T will be used where we are concerned with Turing machines, the letter L will be associated with linear bounded automata, P with pushdown automata, and F with finite automata. These letters with an overprinted shilling will be used for the non-deterministic variety of recognizers. The actions admissible for a particular class of recognizers will be selected from sets of functions given by ACT or ACTS.

<u>Definition 7</u>. ACT = N ∪ {LT,RT,ST} is a set of functions from $N^N \cup N$ into itself such that for every $f \varepsilon N^N$ and $i,n \varepsilon N$:

(1) $LT(f) = RT(f) = ST(f) = f$

(2) $LT(i) = \begin{cases} i-1, & \text{if } i>0 \\ i, & \text{if } i=0 \end{cases}$

(3) $RT(i) = i+1$

(4) $ST(i) = i$

(5) $n(i) = i$

(6) $(nf)(i) = \begin{cases} n, & \text{if } i=j \text{ for some head position } j \\ f(i), & \text{otherwise.} \end{cases}$

<u>Definition 8</u>. ACTS = N ∪ {LT,RT,ST} is a set of functions from $N^N \cup N$ into itself such that for every $f \varepsilon N^N$ and $i,n \varepsilon N$:

(1) $LT(f) = RT(f) = ST(f) = f$

(2) $LT(i) = \begin{cases} i-1, & \text{if } i>0 \\ i, & \text{if } i=0 \end{cases}$

(3) $RT(i) = i+1$

(4) $ST(i) = i$

(5) $n(i) = i$

(6) $(nf)(i) = \begin{cases} n, & \text{if } n>0 \text{ and } i=j+1 \text{ for some head position } j \\ f(i), & \text{otherwise} \end{cases}$

(7) $(0f)(i) = \begin{cases} 0, & \text{if } i=j \text{ for some head position } j \\ f(i), & \text{otherwise.} \end{cases}$

Turing Machines

Definition 9. QUAD = $N \times N \times N \times ACT$.

Turing machines are nm-tuples of elements of QUAD that have exactly n states and exactly m alphabetic characters. The understanding is that if $x \in QUAD$, then $1x$ represents the present state, $2x$ represents the input symbol scanned, $3x$ represents the next state, while $4x$ represents an appropriate action, given that the present state is $1x$ and the scanned symbol is $2x$. More precisely:

Definition 10. M is a deterministic Turing machine (TM) if and only if there exists $0<n$, $m \in N$ such that:

(1) M is an nm-tuple

(2) $1 \le i \le nm$, then $iM \in QUAD$

(3) $1 \le i \ne j \le n$ and $1 \le k \le (m-1)$, then $1iM = 1(i+kn)M \ne 1jM = 1(j+kn)M$

(4) $0 \le k \le (m-1)$ and $1+kn \le y \le n+kn$, then $2(1+kn)M = 2yM$

(5) $1 \le i \ne j \le m$, then $2(in)M \ne 2(jn)M$

(6) $\{3jM \mid 1 \le j \le nm\} \subseteq \{1iM \mid 1 \le i \le n\}$

(7) $(\{4jM \mid 1 \le j \le nm\} - \{LT, RT, ST\}) \subseteq \{2(in)M \mid 1 \le i \le n\}$.

Clause (3) specifies the distribution of states. Clauses (4) and (5) specify the distribution of the alphabet. Clause (6) specifies that those states admissible as next states must be members of the original state set. Clause (7) specifies that the possible actions of the machine are left moves (LT), right moves (RT), identity moves (ST), and prints of any symbol that occurs in its alphabet. Here, it is assumed that each machine has a special character among its alphabet that has the role of the blank. If we wish to discriminate a state of

M as initial, we may say that this state is 1iM. Similarly, a set of final states FS may be selected such that $FS \subseteq \{1iM | 1 \le i \le n\}$. Recognition for a Turing machine occurs when the machine M enters a final state after being placed initially on the left-most end of a tape with M in its initial state and thereafter no further moves occur. We have made no special provisions for halting states, actions, or quadruples. If we had made such provisions, deciding whether a machine halts for a given tape amounts to deciding whether the machine enters a halting quadruple. Since we have no such provisions, there will be certain distinguishable forms of quadruples for which the same question may be asked. For example, quadruples of the form <n,_,n,ST> or <n,x,n,x> where $m,x \in N$ and $ST \in ACT$ will be halting quadruples for tapes f and head position j if $(ST(f))(j) = f(j)$ and $(x(f))(j) = f(j)$. No further move occurs since state, head position, and tape remain unchanged.

Definition 11. $TM = \{M | M \text{ is a Turing machine}\}$.

Since the cardinality of TM is aleph-zero, we may sequence all Turing machines by some function T which is a bijection from N to TM. Given T, we may obtain auxiliary functions Q_T and A_T that give the cardinality of the state set and alphabet set, respectively, for any Turing machine.

Definition 12. Q_T is a mapping from TM into N such that $Q_T(T(i)) = \{1jT(i) | 1 \le j \le \overline{T(i)}\}$ for every Turing machine T(i).

Definition 13. A_T is a mapping from TM into N such that $A_T(T(i)) = \{2jT(i) | 1 \le j \le \overline{T(i)}\}$ for every Turing machine T(i).

Given T, Q_T, and A_T, we may construct a sequence T_Q of all the states of the members of TM using the natural numbers as an indexing set and a sequence T_A of all the alphabetic elements of the members of TM using the positive integers as an indexing set as follows: $T_Q = \{11T(0),12T(0),\ldots,1(Q_T(T(0)))T(0),\ldots,11T(j),12T(j),\ldots, 1(Q_T(T(j)))T(j),\ldots\}$; $T_A = \{2(Q_T(t(0)))T(0),2(2Q_T(T(0)))T(0),\ldots, 2(A_T(T(0))Q_T(T(0)))T(0),\ldots,2(Q_T(T(j)))T(j),2(2Q_T(T(j)))T(j),\ldots, 2(A_T(T(j))Q_T(T(j)))T(j),\ldots\}$.

We may relabel the states of the members of TM so that the states will be consecutive numbers. If $A\varepsilon TM$ and $A = T(i)$, then the machine $B\varepsilon TM$ obtained from A by substituting for each state $1jT(i)$ wherever it occurs in A, the state $T_Q^{-1}(1jT(i))$, $1\le j\le Q_T(T(i))$, is a machine that is equivalent to A. Hence, we may assume without loss of generality that all machines are so rewritten. If T(i) is the *ith*+1 machine, then the rewritten machine $T^*(i)$ has states $1jT^*(i) = T_Q^{-1}(1jW(i))$ for $1\le j\le[Q_T(T(i)) = Q_T(T^*(i))]$. No two rewritten Turing Machines will share states. The states of the *ith*+1 machine $T^*(i)$ will be the consecutive integers from $\sum_{j=0}^{i-1} Q_T(T(j))$ to $\left(\sum_{j=0}^{i} Q_T(T(j)\right) - 1$. Furthermore, we may relabel the alphabet of the members of TM so that the elements of the alphabet will consist of consecutive integers. If $A\varepsilon TM$ and $A = T^*(i)$, then the machine $B\varepsilon TM$ obtained from A by substituting for each alphabet character $2(jQ_T(T^*(i)))T^*(i)$ wherever it occurs in A, the character $A_T^{-1}(2(jQ_T(T^*(i)))T^*(i))$, $1\le j\le A_T(T^*(i))$, is equivalent to A. Hence, we may assume without loss of generality that all machines are so rewritten. If $T^*(i)$ is the *ith*+1 machine, then the rewritten machine $T^{**}(i)$

has alphabet characters $2\big(jQ_T(T^*(i))\big)T^{**}(i) = T_A^{-1}\big(2\big(jQ_T(T^*(i))\big)T^*(i)\big)$, for $1 \le j \le [A_T(T^*(i)) = A_T(T^{**}(i))]$. Now, in addition to not sharing states, no two rewritten Turing Machines will share alphabetic characters. Moreover, the alphabet of the *ith*+1 machine $T^{**}(i)$ will be the consecutive integers from $\left(\sum_{j=0}^{i-1} A_T(T(j))\right) + 1$ to $\sum_{j=0}^{i} A_T(T(j))$.

We may define a function T? which maps N onto the set of rewritten Turing Machines $\{T^{**}(0), T^{**}(1), T^{**}(2), \ldots\}$ such that for any $n \in N$, T?(n) is the machine in which the state n occurs. Such a mapping exists since we have rewritten all machines so they will not share any states and each $n \in N$ is the state of some machine.

<u>Definition 14</u>. For all n, $k \in N$, then $T?(n) = T^{**}(k)$ if and only if $n = 1jT^{**}(k)$ for some j, $1 \le j \le Q_T(T^{**}(k))$.

Since a Turing Machine is an ordered nm-tuple of quadruples, we may define a function INDEX which maps $N \times N - \{0\}$ into $N - \{0\} = \{1,2,3,\ldots\}$ by means of which we may select the applicable quadruple, given that the machine is in state n and is scanning tape f at head position i. Such a function exists since we have rewritten all machines so state and alphabet sets are unique. More precisely:

<u>Definition 15</u>. For all $n, i \in N$ and $f \in N^N$, $INDEX(n,f(i)) = [(n+1) - 11T?(n)] + [((f(i)+1) - 21T?(n)) - 1]Q_T(T?(n))$ such that $INDEX(n,f(i)) = j$, $1 \le j \le \overline{T?(n)}$, if and only if $1jT?(n) = n$ and $2jT?(n) = f(i)$.

The approach is to abstract completely from the process of recognition in order to isolate those parameters essential to any step in a computation of a recognizer. For it is the interaction of these parameters that in essence characterizes the recognition procedure for a class of automata. For Turing machines the essential parameters are state, head position, and tape. The interaction is restricted to the extent that given a state-symbol combination the machine may print a symbol or move one space left or right as well as change state and then repeat the cycle. An element of the Cartesian product of such parameters will hereinafter be called a configuration for the class of automata under consideration. Given this perception, we may obtain a function C^*_T from $NxNxN^N$ into itself that captures all steps in a recognition computation as it monitors changes of state, head position, and tape. C^*_T will depend on the way TM was ordered by T as well as how the states were ordered by Q_T and the alphabets by A_T to obtain the rewritten machines $\{T^{**}(0), T^{**}(1), \ldots\}$. None of these reshufflings of machines, states, and alphabets have altered the machines in any essential way because the resulting machines differ only in a relabeling of states and alphabets.

<u>Definition 16</u>. C^*_T is a mapping from $NxNxN^N$ into itself so that for any $\langle x,i,f\rangle \epsilon NxNxN^N$, $C^*_T(\langle x,i,f\rangle)$ will equal:

(1) $\langle n,j,g\rangle$, if $f''N \subseteq \{2jT?(x) \mid 1 \leq j \leq \overline{\overline{T?(x)}}\}$

where (i) $n = 3\big(\text{INDEX}(x,f(i))\big)T?(x)$

(ii) $j = \big(4\big(\text{INDEX}(x,f(i))\big)T?(x)\big)(i)$

(iii) $g = \big(4\big(\text{INDEX}(x,f(i))\big)T?(x)\big)(f)$

(2) $\langle x,i,f\rangle$, otherwise.

In order to capture the computations of the class of Turing machines within an appropriate abstract digital computer, we need a Boolean algebra in which to encode the elements of $NxNxN^N$. The elements of $N \times N \times N^N$ correspond in a natural way to the atoms of the Boolean algebra $\theta^{N \times N \times N^N}$. Given this encoding and algebra, we may define a total state-transition function T_T for an abstract digital computer which will capture the computations of the class of Turing machines in the same sense that C_T^* does.

Definition 17. E_T is a bijection from $N \times N \times N^N$ to $\theta^{N \times N \times N^N}$ restricted to atoms such that $(E_T(i))(j) = 1$ if and only if $i=j$ for all $i,j \varepsilon N \times N \times N^N$.

Definition 18. T_T is a mapping from $\theta^{N \times N \times N^N}$ into itself so that for every $b \varepsilon \theta^{N \times N \times N^N}$, $T_T(b)$ will equal:

(1) f where $f(j) = 1$ iff $j = C_T^*(b^{-1}(1))$ for all $j \varepsilon N \times N \times N^N$, if b is an atom

(2) b, if b is not an atom.

What then is the relationship between the notion of recognition for Turing machines and the notion of computation for the abstract digital computer previously defined? The recognition procedure is begun by placing the machine in its initial state, say n, left-justified on a tape t. This procedure is equivalent to selecting an initial configuration $x = \langle n,0,t \rangle$. Correspondingly, we may "programme" the abstract digital computer by selecting the initial state $E_T(x) \varepsilon \theta^{N \times N \times N^N}$. Halting means that the Turing machine makes no further moves. That is to say that no change of configuration occurs, meaning that there exists a finite ordinal τ such that $C_T^{\tau}(x) = C_T^{\tau+1}(x)$. The digital computer also stops, i.e., $T_T^{\tau}(E_T(x)) = T_T^{\tau+1}(E_T(x))$. In either case, halting is a necessary condition for recognition. For recognition by final state, we then ask if $1C_T^{\tau}(x)\varepsilon H_T$ is a final state as relabeled via Q_T and A_T. For the corresponding abstract digital computer, recognition by final state is given by the condition $1\big((T_T^{\tau}(E_T(x)))^{-1}(1)\big)\varepsilon H_T$. We will now consolidate these perceptions into a formal definition and then show that there exists an abstract set-reset digital computer whose state set is $\theta^{N\times N\times N^N}$ and whose state-transition function is T_T.

Definition 19. If $x = \langle n,0,t\rangle$ is an initial configuration of a Turing machine and $E_T(x)$ is an initial state of an abstract digital computer, then the input tape $3x$ is recognized if and only if there exists a finite ordinal τ such that $T_T^{\tau}(E_T(x)) = T_T^{\tau+1}(E_T(x))$, and $1\big((T_T^{\tau}(E_T(x)))^{-1}(1)\big)\varepsilon H_T$ is a final state as relabeled via Q_T and A_T.

Theorem 2. There exists a control function Cu_T such that $\langle\theta^{N\times N\times N^N},RS_T,Cu_T\rangle$ is a set-reset computer and $T_T(b) = (Cu_T(b))(b)$ for any $b\varepsilon\theta^{N\times N\times N^N}$.

Proof.

Case (1): $b\varepsilon\theta^{N\times N\times N^N}$ and b is an atom. Let $Cu_T(b) = S(J)R(K)$ where $J = \{C_T^{*}(b^{-1}(1))\}$ and $K = \{b^{-1}(1)\}$;

Case (2): $b\varepsilon\theta^{N\times N\times N^N}$ and b is not an atom. Let $Cu_T(b) = R(\emptyset)$ where $\emptyset$ is the null set. Therefore $(Cu_T(b))(b) = T_T(b)$, for all $b\varepsilon\theta^{N\times N\times N^N}$.

Theorem 2 shows there is at least one set-reset computer that captures all the computations of all Turing machines up to and including changes of states, head positions, and tapes. The construction consisted of defining a function on the Cartesian product of the essential parameters (states, head positions, and tapes) that would capture the computations pertinent to recognition for all Turing machines. Then we obtain a state-transition function T_T on the Boolean algebra $\theta^{N\times N\times N^N}$ that would capture the recognition process of all Turing machines. Finally, we showed that $(\theta^{N\times N\times N^N}, T_T)$ was an abstract set-reset digital computer by constructing a control function Cu_T. All these moves involved only a relabeling of states and alphabets. Thus at any stage of the construction, the resulting Turing machines were equivalent to the ones countenanced initially.

Non-Deterministic Turing Machines

Definition 20. $TRIP = N\times N\times P(N\times ACT)$.

Non-deterministic Turing machines are nm-tuples of elements of TRIP that have exactly n states and exactly m alphabetic characters. The understanding again is that if $x\in TRIP$, then 1x represents the present state, 2x represents the input symbol scanned, and 3x represents the finite number of combinations of next state and action possible, given that the present state and symbol scanned are 1x and 2x, respectively. Non-determinism here means that the machine may select any of the possible combinations of next state and action from among those in 3x on a particular occurrence of present state (1x) and symbol scanned (2x). The deterministic variety of machines is the special case of the non-deterministic variety of machines where the cardinality of 3x is one.

Definition 21. D is a non-deterministic Turing machine (NTM) if and only if there exists $0<n,m\in N$ such that:

(1) D is an nm-tuple

(2) $1\leq i\leq nm$, then $iD\in TRIP$

(3) $1\le i\ne j\le n$ and $1\le k\le(m-1)$, then $1iD=1(i+kn)D\ne 1jD=1(j+kn)D$

(4) $0\le k\le(m-1)$ and $1+kn\le y\le n+kn$, then $2(1+kn)D=2yD$

(5) $1\le i\ne j\le m$, then $2(jn)D\ne 2(in)D$

(6) $1\le j\le nm$, then $1\le(\overline{\overline{3jD}})<$aleph-zero

(7) $1\varepsilon:\{3jD|1\le j\le nm\}\subseteq\{1iD|1\le i\le n\}$

(8) $\big((2\varepsilon:\{3jD|1\le j\le nm\})-\{LT,RT,ST\}\big)\subseteq\{2(in)D|1\le i\le n\}$.

Clauses (1) and (2) are obvious. Clause (3) again specifies the distribution of states. Clauses (4) and (5) specify the distribution of the alphabet. Clause (6) specifies that the number of possible choices for next state and action pairs is finite. Note that for the deterministic case there is only one possible next state and action. Clause (7) states that the possible next states are members of the original state set. Clause (8) states that the possible actions are left moves, right moves, identity moves, and prints of any symbol that occurs in its alphabet. Again, it is assumed there is a character in the alphabet that plays the role of the blank. Comments concerning halting are the same as for the deterministic variety of machines. Recognition is by final state with the same conditions as for the deterministic case, but with the added provision that recognition occurs if there is at least one sequence of moves that satisfies the recognition criteria. We shall introduce the following definitions without explanation since they follow *mutatis mutandis* the deterministic case.

Definition 22. NTM = {D|D is a non-deterministic Turing machine}.

Definition 23. $\mathcal{T}$ is a bijection from N to NTM.

Definition 24. $Q_{\mathcal{T}}$ is a mapping from NTM into N such that $Q_{\mathcal{T}}(\mathcal{T}(i) = \{1j\mathcal{T}(i) | 1 \leq j \leq \overline{\overline{\mathcal{T}(i)}}\}$ for every non-deterministic Turing machine $\mathcal{T}(i)$.

Definition 25. $A_{\mathcal{T}}$ is a mapping from NTM into N such that $A_{\mathcal{T}}(\mathcal{T}(i)) = \{2j\mathcal{T}(i) | 1 \leq j \leq \overline{\overline{T(i)}}\}$ for every non-deterministic Turing machine $\mathcal{T}(i)$.

Definition 26. The set of rewritten machines is $\{\mathcal{T}^{**}(0), \mathcal{T}^{**}(1), \ldots\}$, i.e., $\mathcal{T}^{**}(k)$ is the machine obtained from $\mathcal{T}(k)$ by relabeling its states and alphabetic characters.

Definition 27. For all $n,k \epsilon N$, then $\mathcal{T}?(n) = \mathcal{T}^{**}(k)$ if and only if $n = 1j\mathcal{T}^{**}(k)$ for some j, $1 \leq j \leq Q_{\mathcal{T}}(\mathcal{T}^{**}(k))$.

Definition 28. For $n,i \epsilon N$ and $f \epsilon N^N$, $INDEX(n,f(i)) = [(n+1)-11\mathcal{T}?(n)] + [((f(i)+1)-21\mathcal{T}?(m))-1]Q_{\mathcal{T}}(\mathcal{T}?(n))$ such that $INDEX(n,f(i)) = j$, $1 \leq j \leq \overline{\overline{\mathcal{T}?(n)}}$, if and only if $1j\mathcal{T}?(n) = n$ and $2j\mathcal{T}?(n) = f(i)$.

The essential parameters for the non-deterministic variety of machines are the same as for the deterministic ones. The interaction between these parameters is also the same. On the other hand, non-determinism means that a configuration need not give rise merely to another single configuration, but rather it may give rise to any finite number of them. In order to accommodate this case we monitor all configurations that arise at any stage in the recognition procedure. Perhaps an analogy with the structure of trees may clarify the approach. The recognition procedure is begun by selecting an initial configuration. This initial configuration is then the root of the tree. This configuration can lead to a finite number of other configurations (nodes) that arise from the possible choices for next state and action. The resulting configurations are then seen as nodes of the next level of the tree. A configuration will give rise to as many configurations as there are possible next state-action combinations for a given present state-symbol scanned combination. Each cycle of the recognition procedure corresponds to a level of the tree. Recognition occurs if at some level we find a configuration that remains unchanged at the next cycle and the state therein is a member of the final state set. Note that halting again appears as a necessary condition for recognition. Also recall that recognition occurs if at least one path leads to acceptance. Thus, the recognition procedure terminates once a successful succession of configurations has been obtained. As we did for the deterministic variety, we will proceed to obtain an appropriate Boolean algebra and total state-transition function for the non-deterministic case.

<u>Definition 29</u>. $C^*_{\mathcal{T}}$ is a mapping from $N \times N \times N^N$ to $P(N \times N \times N^N)$ so that for any $\langle m,j,h \rangle \varepsilon N \times N \times N^N$, $C^*_{\mathcal{T}}(\langle m,j,h \rangle)$ will equal:

(1) A^*, if $h''N \leq \{2j\mathcal{T}?(m) | 1 \leq j \leq \overline{\overline{\mathcal{T}?(m)}}\}$ where

$A^* = \{x^* | (\exists z)(z \varepsilon 3(\mathrm{INDEX}(m,h(j)))\mathcal{T}?(m)$ and

$x^* = \langle 1z, 2z(j), 2z(h) \rangle)\}$

(2) $\langle m,j,h \rangle$, otherwise.

<u>Definition 30</u>. $E_{\mathcal{T}}$ is a bijection from $N \times N \times N^N$ to $\theta^{N \times N \times N^N}$ restricted to atoms such that $(E_{\mathcal{T}}(i))(j)=1$ if and only if $i=j$ for all $i,j \varepsilon N \times N \times N^N$.

<u>Definition 31</u>. $T^*_{\mathcal{T}}$ is a mapping from the atoms of $\theta^{N \times N \times N^N}$ into $\theta^{N \times N \times N^N}$ such that $T^*_{\mathcal{T}}(a) = \vee\, E_{\mathcal{T}}(x)$ for every atom a in $\theta^{N \times N \times N^N}$ and $x \varepsilon C^*_T(E^{-1}_T(a))$.

Definition 32. $T_{\mathcal{T}}$ is a mapping from $\theta^{N\times N\times N^N}$ into itself so that for every $b\varepsilon\theta^{N\times N\times N^N}$:

(1) $T_{\mathcal{T}}(b) = T^*_{\mathcal{T}}(b)$, if b is an atom

(2) $T_{\mathcal{T}}(b) = \bigvee_{a\leq b} T^*_{\mathcal{T}}(a)$ for atoms $a\leq b$, if:

(i) $(\exists c)(\exists\tau)$(c is an atom.$\tau\geq 0.T^{\tau}_{\mathcal{T}}(c) = b.c^{-1}(1)$ is an initial configuration)

(ii) $\sim (\exists a)$(a is an atom.$a\leq b.a\varepsilon T^*_{\mathcal{T}}(a).1(a^{-1}(1))$ is a final state)

(3) $T_{\mathcal{T}}(b) = b$, otherwise.

The strategy here was to define a function $C^*_{\mathcal{T}}$ from the set of configurations into the power set of the set of configurations. The configurations were encoded via $E_{\mathcal{T}}$ into the atoms of the Boolean algebra $\theta^{N\times N\times N^N}$. The function $T^*_{\mathcal{T}}$ was defined for the Boolean algebra as the *alter ego* of $C^*_{\mathcal{T}}$. A total state-transition function $T_{\mathcal{T}}$ on $\theta^{N\times N\times N^N}$ was obtained by an appropriate atomwise extension of $T^*_{\mathcal{T}}$. Note that the atoms of the Boolean algebra are the algebraic counterpart of the configurations. Thus, to monitor configurations is to monitor atoms of the appropriate Boolean algebra. The state-transition function $T_{\mathcal{T}}$ is defined so that it becomes the identity, once recognition has been attained. We may now summarize these facts of recognition in a formal definition and prove a theorem that asserts the existence of an abstract digital computer for all non-deterministic Turing machines.

Definition 33. If $x = \langle n,0,t\rangle$ is an initial configuration of a non-deterministic Turing machine and $E_{\mathcal{T}}(x)$ is an initial state of an abstract digital computer then the input tape 3x is recognized if and only if there exists a finite ordinal τ such that $T_{\mathcal{T}}^{\tau}(E_{\mathcal{T}}(x)) = T_{\mathcal{T}}^{\tau+1}(E_{\mathcal{T}}(x))$ and there exists an atom a such that $a \leq T_{\mathcal{T}}^{\tau}(E_{\mathcal{T}}(x))$ and $a \varepsilon T_{\mathcal{T}}^{*}(a)$, and $1(a^{-1}(1)) \varepsilon H_{\mathcal{T}}$ where $H_{\mathcal{T}}$ is the final state set for $\mathcal{T}?(1(a^{-1}(1)))$.

<u>Theorem 3</u>. There exists a control function $Cu_{\mathcal{T}}$ such that $<\theta^{NxNxN^{N}}, Rs_{\mathcal{T}}, Cu_{\mathcal{T}}>$ is a set-reset computer and $T_{\mathcal{T}}(b) = (Cu_{\mathcal{T}}(b))(b)$ for any $b \varepsilon \theta^{NxNxN^{N}}$.

<u>Proof</u>. Let $Cu_{\mathcal{T}}(b) = S(J)R(K)$ where $J = \{(T_{\mathcal{T}}(b))^{-1}(1)\}$ and $K = \{b^{-1}(1)\}$. Therefore $(Cu_{\mathcal{T}}(b))(b) = T_{\mathcal{T}}(b)$ for all $b \varepsilon \theta^{NxNxN^{N}}$.

We are now able to provide a characterization of the deterministic and non-deterministic varieties of a class of recognizers with the aid of an appropriate Boolean algebra and total state-transition function. The Boolean algebra for both varieties will be θ^{X} where X is the Cartesian product of the parameters essential (same for both) for the recognition procedure of that class. However, the difference between the two varieties is reflected by the total state-transition function. For the deterministic case, this function will be the identity on all elements of θ^{X} other than atoms, since for the deterministic case there is only one choice of next state and action for any given present state-symbol scanned combination. If the current configuration can be changed, it can be changed in only one way. Consequently, at every cycle of the recognition procedure, we are concerned with only one configuration. These configurations are encoded into the atoms of θ^{X} so on any iteration of the total state-transition function the computer merely passes from atom to atom.

For the non-deterministic variety, the total state-transition function will be other than the identity for, at most, elements of θ^X such that the set $b^{-1}(1)$ is finite. This is so because for any particular present state-symbol scanned combination, the number of possible next state-action combinations is finite. At any cycle of the recognition procedure only a finite number of new configurations will be introduced, meaning that on any iteration of the total state-transition function only a finite number of atoms need be monitored. The deterministic and non-deterministic varieties of a class of recognizers will be captured by abstract computers that have the same Boolean algebra for their state set, but for the deterministic case the transitions will occur only between atoms while for the non-deterministic case the transitions will occur only between those elements of the algebra which are finite.

Non-Deterministic Linear Bounded Automata

A non-deterministic linear bounded automaton can be viewed as a non-deterministic Turing machine that uses only that portion of the tape on which the initial input appears. Or more generally, a linear bounded automaton is a recognizer that uses only an amount of tape that is a linear function of the length of its input string [13]. Moreover, there is a theorem to the effect that boundary markers usually inherent in the specification of these recognizers are inessential [12]. Thus, they are genuine special cases of non-deterministic Turing machines. However, they do differ from Turing machines and this difference will be made explicit when we treat the recognition procedure of this class.

We will introduce the following definitions without comment as they follow *mutatis mutandis* the pattern of the corresponding definitions for the case of Turing machines.

<u>Definition 34.</u> TRIP = $N \times N \times P(N \times ACT)$.

<u>Definition 35.</u> Q is a non-deterministic linear bounded automaton (NLBA) if and only if there exist $0<n,m \in N$ such that:

(1) Q is an nm-tuple

(2) $1 \le i \le nm$, then $iQ \in TRIP$

(3) $1 \le i \ne j \le n$, and $1 \le k \le (m-1)$, then $1iQ=1(i+kn)Q \ne 1jQ=1(j+kn)Q$

(4) $0 \le k \le (m-1)$ and $1+kn \le y \le n+kn$, then $2(1+kn)Q=2yQ$

(5) $1 \le i \ne j \le m$, then $2(in)Q \ne 2(jn)Q$

(6) $1 \le j \le nm$, then $1 \le (\overline{\overline{3jQ}}) <$aleph-zero

(7) $1\varepsilon:\{3jQ \mid 1 \le j \le nm\} \subseteq \{1iQ \mid 1 \le i \le n\}$

(8) $\big((2\varepsilon:\{3jQ \mid 1 \le j \le nm\})-\{LT,RT,ST\}\big) \subseteq \{2(in)Q \mid 1 \le i \le n\}$.

<u>Definition 36.</u> NLBA = $\{Q \mid Q$ is a non-deterministic linear bounded automaton$\}$.

<u>Definition 37.</u> Ł is a bijection from N to NLBA.

<u>Definition 38.</u> $Q_{Ł}$ is a mapping from NLBA into N such that $Q_{Ł}(Ł(i)) = \{1jŁ(i) \mid 1 \le j \le \overline{\overline{Ł(i)}}\}$ for every non-deterministic linear bounded automaton Ł(i).

<u>Definition 39.</u> $A_{Ł}$ is a mapping from NLBA into N such that $A_{Ł}(Ł(i)) = \{2jŁ(i) \mid 1 \le j \le \overline{\overline{Ł(i)}}\}$ for every non-deterministic linear bounded automaton Ł(i).

<u>Definition 40.</u> The set of rewritten machines is $\{L^{**}(0), Ł^{**}(1),\ldots\}$, i.e., $Ł^{**}(k)$ is the machine obtained from Ł(k) by relabeling its states and alphabetic characters.

<u>Definition 41</u>. For all n,kεN, then Ł?(n) = $Ł^{**}$(k) if and only if n = 1j$Ł^{**}$(k) for some j, 1≤j≤$Q_{Ł}$($Ł^{**}$(k)).

<u>Definition 42</u>. For all n,iεN and fεN^N, INDEX(n,f(i)) = [(n+1)-1lŁ?(n)] + [((f(i)+1)-2lŁ?(m))-1]$Q_{Ł}$(Ł?(n)) such that INDEX(n,f(i)) = j, 1≤j≤$\overline{Ł?(n)}$, if and only if 1jŁ?(n) = n and 2jŁ?(n) = f(i).

We will use recognition by final state for the linear bounded automata as we did for Turing machines, but with the added stipulation that the read head for a terminal situation must be located one position to the right of the last character of the input string. In order to formalize the notion of a bound on the amount of tape available during a particular execution of the recognition procedure, we use a special representation of the input string so that the boundary condition is made exact. We have relabeled the alphabets of the machines using the positive integers rather than the natural numbers. Consequently, the natural number 0 does not appear as an alphabetic character of any machine. Thus, a tape is represented as having a finite initial segment containing only positive integers (left-justified) followed by zeros. The index of the tape position containing the last positive integer that precedes a zero will be the bound of the recognizer relative to this tape. Recognition then occurs when a recognizer halts in a final state with the value of the head position one greater than the bound of the tape being processed.

The resemblance between non-deterministic linear bounded automata and Turing machines suggests that the essential parameters of linear bounded automata should include state, head position, and tape.

The further stipulation of a bound requires that there be an additional parameter. Hence, the essential parameters are state, head position, tape, and bound. These parameters form configurations that are elements of $NxNxN^NxN$. We may now enter formal definitions.

<u>Definition 43</u>. $SEQ = \{f|f\varepsilon N^N.(\exists a)(b)(a,b\varepsilon N.(b\leq a\supset f(b)>0).$ $(b>a\supset f(b)=0))\}$.

<u>Definition 44</u>. $C^*_{\mathcal{L}}$ is a mapping from $NxNxN^NxN$ to $\mathcal{P}(NxNxN^NxN)$ so that for any $<n,i,f,z>\varepsilon NxNxN^NxN$, $C^*_{\mathcal{L}}(<n,i,f,z>)$ will equal:

(1) A, if (i) $f\varepsilon SEQ$

(ii) $f(i)\varepsilon f''N-\{0\}\subseteq\{2j\mathcal{L}?(n)|1\leq j\leq\overline{\overline{\mathcal{L}?(n)}}\}$

(iii) $\mu(f)=((MIN\{x|x\geq 0.f(x)=0\})\dot{-}1)=z$

(iv) $0\leq i\leq z$

where $A=\{x^*|(\exists z)(z\varepsilon 3(INDEX)n,f(i)))\mathcal{L}?(n).x^*=$ $<1z,2z(i),2z(f),z>)\}$

(2) $<n,i,f,z>$, otherwise.

<u>Definition 45</u>. $E_{\mathcal{L}}$ is a bijection from $NxNxN^NxN$ to θ^{NxNxN^NxN} restricted to atoms such that $(E_{\mathcal{L}}(i))(j)=1$ if and only if $i=j$ for all $i,j\varepsilon NxNxN^NxN$.

<u>Definition 46</u>. $T^*_{\mathcal{L}}$ is a mapping from the atoms of θ^{NxNxN^NxN} into θ^{NxNxN^NxN} such that $T^*_{\mathcal{L}}(a)=V\,E_{\mathcal{L}}(x)$ for every atom a in θ^{NxNxN^NxN} and $x\varepsilon C^*_{\mathcal{L}}(E^{-1}_{\mathcal{L}}(a))$.

<u>Definition 47</u>. $T_{\mathcal{L}}$ is a mapping from θ^{NxNxN^NxN} into itself so that for every $b\varepsilon\theta^{NxNxN^NxN}$:

(1) $T_{\mathcal{L}}(b)=T^*_{\mathcal{L}}(b)$, if b is an atom

(2) $T_{\mathcal{L}}(b)=\underset{a\leq b}{V}\,T^*_{\mathcal{L}}(a)$ for atoms $a\leq b$, if:

(i) $(\exists c)(\exists \tau)$(c is an atom.$\tau \geq 0$.$T_{\not{V}}(c)$=b.$c^{-1}(1)$ is an initial configuration)

(ii) $\sim (\exists a)$(a is an atom.a$\leq$b.$a^{-1}(1) = \langle m,i,f,z\rangle$ where m is a final state, f(i)=0, μ(f)=z, i=z+1)

(3) $T_{\not{V}}(b)$ = b, otherwise.

Definition 48. If x=$\langle n,0,t,\mu(t)\rangle$ is an initial configuration of a non-deterministic linear bounded automaton and $E_{\not{V}}(x)$ is an initial state of an abstract digital computer then the input tape 3x is recognized if and only if there exists a finite ordinal τ such that $T_{\not{V}}^{\tau}(E_{\not{V}}(x)) = T_{\not{V}}^{\tau+1}(E_{\not{V}}(x))$ and there exists an atom a such that $a \leq T_{\not{V}}^{\tau}(E_{\not{V}}(x))$ and $a^{-1}(1) = \langle m,i,f,z\rangle$ where $m \in H_{\not{V}}$ with $H_{\not{V}}$ as the final state set for $\not{V}$?$((1(a^{-1}(1)))$, f(i)=0, μ(f)=z and i=z+1.

Theorem 4. There exists a control function $Cu_{\not{V}}$ such that $\langle \theta^{N \times N \times N^{N} \times N}, RS_{\not{V}}, Cu_{\not{V}}\rangle$ is a set-reset computer and $T_{\not{V}}(b) = (Cu_{\not{V}}(b))(b)$ any $b \in \theta^{N \times N \times N^{N} \times N}$.

Proof. The proof of this theorem parallels exactly the proof of the corresponding theorem for non-deterministic Turing machines.

Non-Deterministic Pushdown Automata

Definition 49. COMP = NxNxNx$\mathcal{P}$(NxACTxACTS).

Non-deterministic pushdown automata are nmp-tuples of elements of COMP that have exactly n states, input alphabets of exactly m characters, and pushdown alphabets of exactly p characters. The understanding here is that if x$\in$COMP, then 1x represents the present state, 2x represents the input symbol scanned, 3x represents the pushdown symbol scanned, and 4x is a set of triplets composed of the next state, the appropriate action relative to the input tape, and the appropriate action relative to the pushdown tape.

Definition 50. G is a non-deterministic pushdown automaton (NPDA) if and only if there exists $0<n,m,p\epsilon N$ such that:

(1) G is an nmp-tuple

(2) $1\leq i\leq nmp$, then $iG\epsilon COMP$

(3) $1\leq i\neq j\leq n$ and $1\leq k\leq ((mp)-1)$, then $1iG=1(i+kn)G\neq 1jG=1(j+kn)G$

(4) $0\leq k\leq (m-1)$ and $0\leq q\leq (p-1)$ and $(1+kn+qnm)\leq y\leq (n+kn+qnm)$, then $2(1+kn)G=2(1+kn+qnm)G$ and $2(1+kn)G=2yG$

(5) $0\leq i\neq j\leq (m-1)$ and $0\leq q\leq (p-1)$, then $2(1+in)G=2(1+in+qnm)G\neq 2(1+jn)G=2(1+jn+qnm)G$

(6) $0\leq k\leq (p-1)$ and $(1+knm)\leq y\leq (nm+knm)$, then $3(1+knm)G=3yG$

(7) $1\leq i\neq j\epsilon p$, then $3(inm)G\neq 3(jnm)G$

(8) $1\leq i\leq nmp$, then $1\leq(\overrightarrow{\overline{4iG}})<$aleph-zero

(9) $1\epsilon:\{4jG|1\leq j\leq nmp\}\subseteq\{1iG|1\leq i\leq n\}$

(10) $2\epsilon:\{4jG|1\leq j\leq nmp\}\subseteq\{RT,ST\}$

(11) $\big((3\epsilon:\{4jG|1\leq j\leq nmp\})-\{0,ST\}\big)\subseteq\{3iB|1\leq i\leq nmp\}$.

Clauses (1) and (2) are obvious. Clause (3) specifies the distribution of states. Clauses (4) and (5) specify the distribution of the input alphabet. Clauses (6) and (7) specify the distribution of the pushdown alphabet. Clause (8) specifies that the number of possible choices for next state and actions is finite. For the deterministic varieties there would be only one possibility. Clause (9) states that the possible next states are members of the original state set. Clause (10) states that the admissible actions relative to input tapes are stay (the epsilon-type move) and move right. These are the standard moves for this class of recognizer. An epsilon-type move is simply one where the resulting course of action is independent of the input symbol. The definition of NPDA prescribes a mapping from NxNxN to $\mathcal{P}$(NxACTxACTS). Consequently, to say the argument of the function so prescribed is independent of the input symbol is to say that the function has the same value for all inputs symbols and the particular state and pushdown symbol associated with an epsilon-type move. Thus, we need not alter our formalism to accommodate the epsilon-type moves. Clause (11) specifies that the possible actions on the stack are erase (print 0), stay, and print any symbol of the pushdown alphabet.

Again our *modus operandi* will be as before. We will relabel the states relative to natural numbers and the input and pushdown alphabets relative to the positive integers.

Definition 51. NPDA = $\{G \mid G$ is a non-deterministic pushdown automaton$\}$.

Definition 52. $\mathcal{F}$ is a bijection from N to NPDA.

Definition 53. $Q_{\mathcal{F}}$ is a mapping from NPDA into N such that $Q_{\mathcal{F}}(\mathcal{F}(i)) = \{1j\mathcal{F}(i) \mid 1 \le j \le \overline{\overline{\mathcal{F}(i)}}\}$ for every non-deterministic pushdown automaton $\mathcal{F}(i)$.

Definition 54. $A_{\mathcal{F}}$ is a mapping from NPDA into N such that $A_{\mathcal{F}}(\mathcal{F}(i)) = \{2j\mathcal{F}(i) \mid 1 \le j \le \overline{\overline{\mathcal{F}(i)}}\}$ for every non-deterministic pushdown automaton $\mathcal{F}(i)$.

We define an additional auxiliary $Z_{\mathcal{F}}$ that gives us the cardinality of the pushdown alphabet of any non-deterministic pushdown automaton as follows:

<u>Definition 55.</u> Z_{P} is a mapping from NPDA into N such that $Z_{P}(P(i)) = \{3jP(i) \mid 1 \le j \le \overline{\overline{P(i)}}\}$ for every non-deterministic pushdown automaton $P(i)$.

<u>Definition 56.</u> The set of rewritten machines is $\{P^{***}(0), P^{***}(1), \ldots\}$, i.e., $P^{***}(k)$ is the machine obtained from $P(k)$ by relabeling its states and the characters of its input and pushdown alphabets.

<u>Definition 57.</u> For all $n,k \epsilon N$, then $P?(n) = P^{***}(k)$ if and only if $n = 1jP^{***}(k)$ for some j, $1 \le j \le Q_{P}(P^{***}(k))$.

<u>Definition 58.</u> For all $n,i,j \epsilon N$ and $f,g \epsilon N^{N}$, $\text{INDEX}(n,f(i)g(j)) = [(n+1)-11P?(n)] + [((f(i)+1)-21P?(m))-1]Q_{P}(P?(n)) + [((g(j)+1)-31P?(n))-1](Q_{P}(P?(n)) \cdot A_{P}(P?(n)))$ such that $\text{INDEX}(n,f(i),g(j)) = q$. $1 \le q \le \overline{\overline{P?(n)}}$, if and only if $1qP?(n) = n$, $2qP?(n) = f(i)$ and $3qP?(n) = g(j)$.

Recognition for pushdown automata can be formulated as being either by final state or by empty store. We have chosen to use final state. In either formulation the entire finite input string must be read. Accordingly, the terminal head position must be that value at which the first blank appears on the input tape. At this time the scanned input symbol will be a blank, represented here by the symbol 0. Yet, when this situation occurs we may still need to erase symbols from the pushdown tape in order to achieve empty store. Thus, configurations where the scanned input symbol is the blank must be countenanced. As a result we would be required to introduce a pseudo blank for each machine as was done in the case of Turing machines. Further complications ensue, since the tape representation would require modification. The entire formulation of this class of recognizers would be more complex. All this seems to be needless since it is known that recognition by final state and by empty store are equivalent [19]. The requirement that the entire input string be read as a necessary condition for recognition is preserved by the method since zero is not in the input alphabet of any machine. Also, zero does not appear in any pushdown alphabet. Hence, the top of the pushdown is always $(\mathrm{Min}\{x|x\varepsilon N.f(x)=0\})\dot{-}1$. Note that the function 0 gives us the ability to erase characters from the pushdown tape. Likewise, the standard procedure for printing on the pushdown tape is to define print actions so that they overprint the top character of the pushdown and then print additional characters as part of the same action, thereby increasing the length of the pushdown tape. However, this method appears cumbersome. We have merely defined the set of functions in ACTS so that the printing is done one space ahead of the current stack top.

The essential parameters of pushdown automata are seen to be state, input head position, input tape, pushdown head position, and pushdown tape. The configurations are elements of $NxNxN^NxNxN^N$. Each such quintuplet is composed of a state, an input head position, an input tape, a pushdown head position, and a pushdown tape. We may now proceed to enter formal definitions of these perceptions.

Definition 59. C^*_p is a mapping from $NxNxN^NxNxN^N$ to $P(NxNxN^NxNxN^N)$ so that for any $<n,i,f,j,g>\varepsilon NxNxN^NxNxN^N$, $C^*_p(<n,i,f,j,g>)$ will equal:

(1) A^*, if (i) $f,g \varepsilon SEQ$

(ii) $f(i) \varepsilon f''N-\{0\} \subseteq \{2j\mathcal{P}?(n) \mid 1 \leq j \leq \overline{\overline{\mathcal{P}?(n)}}\}$

(iii) $g(j) \varepsilon g''N-\{0\} \subseteq \{3j\mathcal{P}?(n) \mid 1\ j\ \overline{\overline{\mathcal{P}?(n)}}\}$

(iv) $j = \mu(g)$

where $A^* = \{x^* \mid (\exists z)(z \varepsilon 4(\mathrm{INDEX}(n,f(i),g(j)))\mathcal{P}?(n)$ and $x^* = \langle 1z, 2z(i), f, 3z(j), 3z(g) \rangle)\}$

(2) $\langle n,i,f,j,g \rangle$, otherwise.

<u>Definition 60</u>. $E_{\mathcal{P}}$ is a bijection from $N \times N \times N^N \times N \times N^N$ to $\theta^{N \times N \times N^N \times N \times N^N}$ restricted to atoms such that $(E_{\mathcal{P}}(i))(j) = 1$ if and only if $i=j$ for all $i,j \varepsilon N \times N \times N^N \times N \times N^N$.

<u>Definition 61</u>. $T^*_{\mathcal{P}}$ is a mapping from the atoms of $\theta^{N \times N \times N^N \times N \times N^N}$ such that $T^*_{\mathcal{P}}(a) = \vee\ E_{\mathcal{P}}(x)$ for every atom $a \varepsilon \theta^{N \times N \times N^N \times N \times N^N}$ and $x \varepsilon C^*_{\mathcal{P}}(E^{-1}_{\mathcal{P}}(a))$.

<u>Definition 62</u>. $T_{\mathcal{P}}$ is a mapping from $\theta^{N \times N \times N^N \times N \times N^N}$ into itself so that for every $b \varepsilon \theta^{N \times N \times N^N \times N \times N^N}$:

(1) $T_{\mathcal{P}}(b) = T^*_{\mathcal{P}}(b)$, if b is an atom

(2) $T_{\mathcal{P}}(b) = \bigvee_{a \leq b} T^*_{\mathcal{P}}(a)$ for atoms $a \leq b$, if:

(i) $(\exists c)(\exists \tau)$(c is an atom.$\tau \geq 0$.$T_{\mathcal{P}}(c) = b.c^{-1}(1)$ is an initial configuration)

(ii) $\sim (\exists a)$(a is an atom.$a \leq b.a^{-1}(1) = \langle n,i,f,j,t \rangle$ where n is a final state, $f(i)=0$, $i=\mu(f)+1$)

(3) $T_{\mathcal{P}}(b) = b$, otherwise.

<u>Definition 63</u>. If $x = \langle n,0,f,0,g \rangle$ is an initial configuration of a non-deterministic pushdown automaton and $E_{\mathcal{P}}(x)$ is an initial state of an abstract digital computer then the input tape $3x$ is recognized if and only if there exists a finite ordinal τ such that $T^{\tau}_{\mathcal{P}}(E_{\mathcal{P}}(x)) = T^{\tau+1}_{\mathcal{P}}(E_{\mathcal{P}}(x))$

and there exist an atom a such that $a \leq T_{\mathcal{P}}^{\tau}(E_{\mathcal{P}}(x))$ and $a \in T_{\mathcal{P}}^{*}(a)$, and $1(a^{-1}(1)) \in H_{\mathcal{P}}$ where $H_{\mathcal{P}}$ is the final state set for $\mathcal{P}?(1(a^{-1}(1))$ and $2(a^{-1}(1)) = \mu(3(a^{-1}(1)))+1$ and $(3(a^{-1}(1)))(2(a^{-1}(1))) = 0$.

<u>Theorem 5</u>. There exists a control function $Cu_{\mathcal{P}}$ such that $<\theta^{N \times N \times N^N \times N \times N^N}, RS_{\mathcal{P}}, Cu_{\mathcal{P}}>$ is a set-reset computer and $T_{\mathcal{P}}(b) = (Cu_{\mathcal{P}}(b))(b)$ for any $b \in \theta^{N \times N \times N^N \times N \times N^N}$.

<u>Proof</u>. The proof of the theorem parallels exactly the proof of the corresponding theorem for non-deterministic machines.

<u>Finite Automata</u>

A finite automaton can be viewed as a machine which reads a finite input string over some alphabet. Again such strings may be taken as tapes from the set SEQ. Finite automata are merely a special subset of Turing machines (deterministic here) that only read and move right. These machines stop when the first zero of the tape is encountered, and they recognize upon stopping if they are in a final state. Thus, the construction of an abstract computer that captures all the computations of all finite automata does not differ essentially from the construction for the deterministic Turing machines. The following definitions and theorems accomplish the construction.

<u>Definition 64</u>. W is a deterministic finite automaton if and only if there exists $0<n,m \in N$ such that:

(1) W is an nm-tuple

(2) $1 \leq i \leq nm$, then $iW \in QUAD$

(3) $1 \leq i \neq j \leq n$ and $1 \leq k \leq (m-1)$, then $1iW = 1(i+kn)W \neq 1jW = 1(j+kn)W$

(4) $0 \leq k \leq (m-1)$ and $1+kn \leq y \leq n+kn$, then $2(1+kn)W = 2yW$

(5) $1 \le i \ne j \le m$, then $2(in)W \ne 2(jn)W$

(6) $\{2jW | 1 \le j \le nm\} \subseteq \{1iW | 1 \le i \le n\}$

(7) $\{4jW | 1 \le j \le nm\} = \{RT\}$.

<u>Definition 65.</u> FA=$\{W|W$ is a finite automaton$\}$.

<u>Definition 66.</u> F is a bijection from N to FA.

<u>Definition 67.</u> Q_F is a mapping from FA into N such that $Q_F(F(i)) = \{1jF(i) | 1 \le j \le \overline{\overline{F(i)}}\}$ for every finite automaton F(i).

<u>Definition 68.</u> A_F is a mapping from FA into N such that $A_F(F(i)) = \{2jF(i) | 1 \le j \le \overline{\overline{F(i)}}\}$ for every finite automaton F(i).

<u>Definition 69.</u> The set of rewritten machines is $\{F^{**}(1), F^{**}(2), \ldots\}$, i.e., $F^{**}(k)$ is the machine obtained from F(k) by relabeling its states and alphabetic characters.

<u>Definition 70.</u> For all $n,k \in N$, then $F?(n) = F^{**}(k)$ if and only if $n = 1jT^{**}(k)$ for somej, $1 \le j \le Q_F(F^{**}(k))$.

<u>Definition 71.</u> For all $n,i \in N$ and $f \in N^N$, INDEX(n,f(i)) = $[(n+1)-11F?(n)] + [((f(i)+1)-21F?(n))-1]Q_T(F?(n))$ such that INDEX(n,f(i)) = j, $1 \le j \le \overline{\overline{F?(n)}}$, if and only if $1jF?(n) = n$ and $2jF?(n) = f(i)$.

<u>Definition 72.</u> C_F^* is a mapping from $N \times N \times N^N$ into itself so that for every $x,n,i,j \in N$ and $f,g \in N^N$, $C_F^*(\langle x,i,f \rangle)$ will equal:

(1) $\langle n,j,g \rangle$, if (i) $f \in SEQ$

(ii) $f(i) \in f''N-\{0\} \subseteq \{2jF?(x) | 1 \le j \le \overline{\overline{F?(x)}}\}$

where (a) $n=3(INDEX(x,f(i)))T?(n)$

(b) $j=i+1$

(c) $g=f$

(2) $<x,i,f>$, otherwise.

Definition 73. E_F is a bijection from $N \times N \times N^N$ to $\theta^{N \times N \times N^N}$ restricted to atoms such that $(E_F(i))(j) = 1$ if and only if $i=j$ for all $i,j \varepsilon N \times N \times N^N$.

Definition 74. T_F is a mapping from $\theta^{N \times N \times N^N}$ into itself so that for every $b \varepsilon \theta^{N \times N \times N^N}$, $T_F(b)$ will equal:

(1) f where $f(j) = 1$ iff $j = C_F^*(b^{-1}(1))$ for all $j \varepsilon N \times N \times N^N$, if b is an atom

(2) b, if b is not an atom.

Definition 75. If $x = <n,0,t>$ is an initial configuration of a finite automaton and $E_F(x)$ is an initial state of an abstract digital computer, then the input tape 3x is recognized if and only if there exists a finite ordinal τ such that $T_F^{\tau}(E_F(x)) = T_F^{\tau+1}(E_F(x))$ and $T_F^{\tau}(E_F(x)) = a$, a being an atom of $\theta^{N \times N \times N^N}$, such that $1(a^{-1}(1)) \varepsilon H_F$ where H_F is the set of final states of $F?(1(a^{-1}(1)))$ and $(3(a^{-1}(1)))(2(a^{-1}(1))) = 0$.

Theorem 6. There exists a control function Cu_F such that $<\theta^{N \times N \times N^N}, RS_F, Cu_F>$ is a set-reset computer computer and $T_F(b) = (Cu_F(b))$ for any $b \varepsilon \theta^{N \times N \times N^N}$.

Proof. The proof of the theorem parallels exactly the proof of the corresponding theorem for deterministic Turing machines.

CHAPTER III

MORPHISM THEOREMS

Introduction

Earlier we discussed the induced hierarchy of recognizers that is obtained by considering the languages recognized. Turing machines are at the top of the hierarchy and finite automata are at the bottom. The hierarchical notion of recognition power ensues from the relationship between the classes of languages generated by each grammar type: the class of regular sets is a proper subclass of the context-free class of languages; the context-free class is a proper subclass of the context-sensitive class of languages; the context-sensitive class is a proper subclass of the class of recursively enumerable sets [2]. Recognition power is a reflection of the capability to recognize the classes in this chain as ordered by the inclusion relation.

In Chapter II we constructed an abstract digital computer for each class of recognizers countenanced. These computers are algebras and the natural question arises as to whether the relationship between the classes of recognizers countenanced can be captured algebraically within the theory of abstract digital computers. The principal result of this research may now be stated as follows:

The algebraic counterpart of the classical hierarchy of recognizers is obtained in the theory of abstract computers by computer morphisms which hold between the computers of finite automata, pushdown automata, linear bounded automata, and Turing machines. These morphisms relate the abstract digital computers in the expected manner.

In each of the following sections we will establish a theorem which states that the computer for a class of recognizers is appropriately morphic to the computer for the next higher class. These computer morphisms will identify for each given machine a machine of the next higher class that recognizes all and only the tapes of the given machine. Such computer morphisms will be called recognition-preserving morphisms. Their definition is as follows:

Definition 76. Let x be a recognizer, C_x its configuration function, and a and a^* initial configurations of x in which the input tapes t and t^* occur respectively. Then we say that t and t^* are x-equivalent if and only if $t(i) = t^*(i)$ for all i for which there exists a $\tau \geq 0$ such that i is any of the head positions that occur in $C_x^\tau(a)$ or $C_x^\tau(a^*)$.

We may note that for bounded recognizers, that is for recognizers that inspect only a finite segment of tape, the notion of tape equivalence just defined is effective. The notion of recognition-preserving morphisms up to tape equivalence is important to us only for bounded recognizers.

Definition 77. α is a recognition-preserving morphism from $<\theta^X,T_A>$ into $<\theta^Y,T_B>$ if and only if α is a computer morphism such that if $b\varepsilon\theta^X$ and $b^{-1}(1)$ is a configuration or set of configurations of some machine x of kind A then $(\alpha b)^{-1}(1)$ is a configuration or set of configurations of some machine y of kind B. Machine y recognizes all and only tapes recognized by x in the sense that y recognizes $t^*\varepsilon 3((\alpha d)^{-1}(1))$ and all tapes equivalent to t^* if and only if x recognizes $t\epsilon 3(d^{-1}(1))$ where $d^{-1}(1)$ is an initial configuration of x.

Recognition-Preserving Morphism for Finite Automata

Since regular sets are a proper subclass of the class of context-free languages, finite automata are less powerful than non-deterministic pushdown automata. The algebraic counterpart of this fact is given by a theorem which states that the computer of finite automata, $<\theta^{N\times N\times N^N},T_F>$, may be embedded in the computer of pushdown automata, $<\theta^{N\times N\times N^N\times N\times N^N},T_P>$, via a recognition-preserving computer morphism.

Theorem 7. There exists a recognition-preserving computer monomorphism α from $<\theta^{N\times N\times N^N},T_F>$ to $<\theta^{N\times N\times N^N\times N\times N^N},T_P>$.

Proof. Let $<n,i,f>\epsilon N\times N\times N^N$. Then n is a state of some finite automaton F?(n). Given that F?(n)ϵFA, we may construct a deterministic pushdown automaton A where jA = <1jF?(n),2jF?(n),1,{<3jF?(n),RT,ST>}> for $1\leq j\leq \overline{F?(n)}$.

We have merely reformulated the quadruples of the finite automaton F?(n) as quadruples suitable for a pushdown automaton by adding a symbol 1 to function as an initial pushdown symbol and by adding the action ST which is of no consequence. In doing so we have not changed the original behavior of the machine F?(n). The response of this constructed pushdown automaton A to a state-input symbol-pushdown symbol combination is the same as the response to this same state-input symbol combination for F?(n) because the pushdown stack is completely ignored throughout any computation. Hence, we conclude that the original behavior of F?(n) has not been altered through reformulation as a push-down automaton.

Since the set of deterministic pushdown automaton is merely a proper subset of the non-deterministic variety, we have $A\epsilon NPDA$ and $\mathcal{P}^{-1}(A)\epsilon N$, so A is the $\mathcal{P}^{-1}(A)+1$ machine whose states for $1\leq j\leq Q_{\mathcal{P}}(A)$ are the numbers given by:

(1) $[1jF?(n)-11F?(n)]$, if $\mathcal{P}^{-1}(A) = 0$

(2) $\sum_{k=0}^{(\mathcal{P}^{-1}(A))-1} Q_{\mathcal{P}}(\mathcal{P}(k)) + [1jF?(n)-11F?(n)]$, if $\mathcal{P}^{-1}(A)>0$.

The characters of the input alphabet for $1\leq j\leq Q_{\mathcal{P}}(A)$ are the numbers given by

(1) $[2(jQ_{\mathcal{P}}(A))F?(n)-21F?(n)]$, if $\mathcal{P}^{-1}(A) = 0$

(2) $\sum_{k=0}^{(\mathcal{P}^{-1}(A))-1} A_{\mathcal{P}}(\mathcal{P}(k)) + [2(jQ_{\mathcal{P}}(A))F?(n)-21F?(n)]$, if $\mathcal{P}^{-1}(A)>0$.

The single character of the pushdown alphabet is given by:

$$\sum_{k=0}^{(\mathcal{P}^{-1}(A))-1} Z_{\mathcal{P}}(\mathcal{P}(k))+1.$$

Note that this constructed pushdown automaton is one that ignores the pushdown store entirely, so a pushdown alphabet of cardinality one is adequate. We only need one initial symbol to initiate computation.

We define a mapping W_F^* from $N\times N\times N^N$ to $N\times N\times N^N\times N\times N^N$ in terms of two functions K_F and H_F. These two functions are defined relative to a particular finite automaton and pushdown automaton. If the finite automaton is F?(n) then for all $x\epsilon N\times N\times N^N$ such that the state $1x\epsilon\{1jF?(n)| 1\leq j\leq \overline{\overline{F?(n)}}\}$, we have $W_F^*(x) = K_F(x)$ if x is an appropriate configuration for F?(n) that is to be correlated with an appropriate configuration for the corresponding pushdown automaton. In contrast, $W_F^*(x) = H_F(x)$ if x is to be correlated with a configuration of the corresponding pushdown for which a subsequent configuration changed by the pushdown automaton is not desired. Consequently, W_F^* is a total function on the set $N \times N \times N^N$ when the union is taken over all such sets of configurations with respect to every finite automaton. For the finite automaton $F?(n)$ and the corresponding constructed machine $A \varepsilon NPDA$, we let $W_F^*(\langle n,i,f\rangle) = K_F(\langle n,i,f\rangle)$ where $K_F(\langle n,i,f\rangle$ is:

(1) $\langle n^*,i,f^*,o,z\rangle$, if $f''N-\{0\} \subseteq \{2jF?(n) \mid 1 \le j \le \overline{\overline{F?(n)}}\}$ and $0 \le i \le \mu(f)$ with:

$$\text{(i)}\quad n^* = \begin{cases} [n-1lF?(n)], \text{ if } \mathcal{P}^{-1}(A) = 0 \\ \displaystyle\sum_{k=0}^{(\mathcal{P}^{-1}(A))-1} Q_{\mathcal{P}}(\mathcal{P}(k)) + [n-1lF?(n)], \text{ if } \mathcal{P}^{-1}(A) > 0 \end{cases}$$

$$\text{(ii)}\quad f^*(x) = \begin{cases} [(f(x)+1)-2lF?(n)] \text{ for } 0 \le x \le \mu(f), \\ \qquad\qquad\qquad\qquad\qquad\qquad \text{if } P^{-1}(A) = 0 \\ 0 \text{ for } x > \mu(f), \end{cases}$$

$$\text{(iii)}\quad f^*(x) = \begin{cases} \displaystyle\sum_{k=0}^{(\mathcal{P}^{-1}(A))-1} A_{\mathcal{P}}(\mathcal{P}(k)) + [(f(x)+1)-2lF?(n)] \\ \qquad\qquad\qquad\qquad \text{for } 0 \le x \le \mu(f), \\ \qquad\qquad\qquad\qquad\qquad\qquad \text{if } \mathcal{P}^{-1}(A) > 0 \\ 0 \text{ for } x > \mu(f), \end{cases}$$

$$\text{(iv)}\quad z(x) = \begin{cases} 1 \text{ for } x=0, & \\ & \text{if } P^{-1}(A)=0 \\ 0 \text{ for } x>0, & \end{cases}$$

$$\text{(v)}\quad z(x) = \begin{cases} \sum_{k=0}^{(P^{-1}(A))-1} Z_P(P(k))+1 \text{ for } x=0, & \\ & \text{if } P^{-1}(A)>0 \\ 0 \text{ for } x>0, & \end{cases}$$

And $W_F^*(\langle n,i,f\rangle) = H_F(\langle n,i,f\rangle)$ where $H_F(\langle n,i,f\rangle)$ is:

(2) $\langle n^*,i,f^*,0,z\rangle$, if $f''N-\{0\} \subseteq \{2jF?(n) | 1\leq j\leq F?(n)\}$ or $i>\mu(f)$ with:

(i) n^* as given by (1) above.

$$\text{(ii)}\quad f^*(x) = \begin{cases} \sum_{k=0}^{P^{-1}(A)} A_P(P(k)) + [(f(x)+1)-2lF?(n)] & \\ \qquad\qquad \text{for } 0\leq x\leq\mu(f), & \\ & \text{if } P^{-1}(A)\varepsilon N \\ 0 \text{ for } x>\mu(f), & \end{cases}$$

$$\text{(iii)}\quad z(x) = \begin{cases} \sum_{k=0}^{P^{-1}(A)} Z_P(P(k)) + 1 \text{ for } x=0, & \\ & \text{if } P^{-1}(A)\varepsilon N. \\ 0 \text{ for } x>0, & \end{cases}$$

Lemma 1. The mapping W_F^* from $N\times N\times N^N$ to $N\times N\times N^N\times N\times N^N$ is an injection.

Proof. Let $\langle n,i,f\rangle$, $\langle m,j,h\rangle \varepsilon N\times N\times N^N$ and assume $\langle n,i,f\rangle \neq \langle m,j,h\rangle$.

Case (1.1): Assume $n\neq m$ and $F?(n) = F?(m)$.

Since $F?(n) = F?(m)$, the pushdown automata constructed using either state n or m are the same. Denote this constructed automaton as

B. From F?(n) = F?(m), we have 11F?(n) = 11F?(m). Assume $n^* = m^*$. If $\mathcal{P}^{-1}(B) = 0$ then $n^* = [n-11F?(n)]$ and $m^* = [m-11F?(m)]$. But then $[n-11F?(n)] = n^* = m^* = [m-11F?(m)]$ from which we determine that $n=m$ which is contrary to original assumption that $n \neq m$. If $\mathcal{P}^{-1}(B)>0$ then $n^* = \sum_{k=0}^{(\mathcal{P}^{-1}(B))-1} Q_{\mathcal{P}}(\mathcal{P}(k)) + [n-11F?(n)]$ and $m^* = \sum_{k=0}^{(\mathcal{P}^{-1}(B))-1} Q_{\mathcal{P}}(\mathcal{P}(k)) +$ $[m-11F?(m)]$. But by our assumption, $\sum_{k=0}^{(\mathcal{P}^{-1}(B))-1} Q_{\mathcal{P}}(\mathcal{P}(k)) + [n-11F?(n)] =$ $n^* = m^* = \sum_{k=0}^{(\mathcal{P}^{-1}(B))-1} Q_{\mathcal{P}}(\mathcal{P}(k)) + [m-11F?(m)]$ from which again follows the contradiction $n=m$. Note that $0 \leq [n-11F?(n)]$ and $0 \leq [m-11F?(m)]$;

Case (1.2): Assume $n \neq m$ and $F?(n) \neq F?(m)$.

Since $F?(n) \neq F?(m)$, we find that the machines constructed starting with states n and m are different. Denote these machines as B and D, respectively. Can these machines share any states? The most plausible possibility of intersection would be where they are consecutive machines, say $P^{-1}(B) = y \varepsilon N$ and $\mathcal{P}^{-1}(D) = y+1 \varepsilon N$. If there is any state in common, it would be bounded by the maximum value for a state of machine B and the minimum value for a state of machine D. The maximum value for a state of machine B is $\sum_{k=0}^{y-1} Q_{\mathcal{P}}(\mathcal{P}(k)) + Q_{\mathcal{P}}(\mathcal{P}(y))-1 = \sum_{k=0}^{y} Q_{\mathcal{P}}(\mathcal{P}(k))-1$ while the minimum value for machine D is $\sum_{k=0}^{y} Q_{\mathcal{P}}(\mathcal{P}(k))$. Thus, the intersection of the admissible state sets for these constructed machines is empty. Note that if $y=0$, the maximum value becomes $Q_{\mathcal{P}}(\mathcal{P}(0))-1$ and the minimum $Q_{\mathcal{P}}(\mathcal{P}(0))$. We find that no two distinct constructed pushdown automata can share states because the intersection of consecutive machines under the mapping W_F^* is empty. Therefore, we conclude that $n^* \neq m^*$ under W_F^*;

Case (2): Assume $i \neq j$.

The mapping does not change the coordinate representing head position, so $i \neq j$ remains under W_F^*;

Case (3.1): Assume $f \neq h$, $F?(n) = F?(m)$, and both tapes appropriate to $F?(n)$ and $F?(m)$.

$F?(n) = F?(m)$ means that the constructed pushdown automata are identical. So $f^*(x) = \sum_{k=0}^{(\mathcal{P}^{-1}(B))-1} A_{\mathcal{P}}(\mathcal{P}(k)) + [(f(x)+1)-21F?(n)]$ for non-zero characters while $h^*(x) = \sum_{k=0}^{(\mathcal{P}^{-1}(B))-1} A_{\mathcal{P}}(\mathcal{P}(k)) + [(h(x)+1)-21F?(m)]$ for non-zero characters. Assume $f^* = h^*$. Hence,

$$\sum_{k=0}^{(\mathcal{P}^{-1}(B))-1} A_{\mathcal{P}}(\mathcal{P}(k)) + [(f(x)+1)-21F?(n)] = f^*(x) = h^*(x) =$$

$$\sum_{k=0}^{(\mathcal{P}^{-1}(B))-1} A_{\mathcal{P}}(\mathcal{P}(k)) + [(h(x)+1)-21F?(m)]$$ for non-zero characters and $f^*(x) = 0 = h^*(x)$ otherwise. But now we see that $f(x) = h(x)$ for all $x \varepsilon N$, which contradicts the original assumption that $f \neq h$;

Case (3.2): Assume $f \neq h$, $F?(n) = F?(m)$, and one tape inappropriate to $F?(n)$ or $F?(m)$.

Assume the tape to be inappropriate to $F?(n)$. Then $f^*(x) =$

$$\sum_{k=0}^{\mathcal{P}^{-1}(B)} A_{\mathcal{P}}(\mathcal{P}(k)) + [(f(x)+1)-21F?(n)]$$ for non-zero characters while

$h^*(x) = \sum_{k=0}^{(F^{-1}(B))-1} A_F(F(k)) + [(h(x)+1)-21F?(m)]$ for non-zero characters and $f^*(x) = 0 = h^*(x)$ otherwise. Assume $f^*=h^*$. A necessary condition for $f^*=h^*$ is that the underlying alphabets have a character in common. We will show that this condition cannot occur. Again, such a common alphabetic character would be bounded by a maximum value for a character from F?(m) and a minimum from F?(n). The maximum value from F?(m) is $\sum_{k=0}^{F^{-1}(B)} A_F(F(k))$ while the minimum value from F?(n) is $\sum_{k=0}^{F^{-1}(B)} A_F(F(k))+1$. Since this necessary condition cannot be fulfilled, we cannot have $f^*(x) = h^*(x)$ for any non-zero character;

Case (3.3): Assume $f \neq h$, F?(n) = F?(m), and both tapes inappropriate to F?(n) and F?(m).

$f^* \neq h^*$ follows *mutatis mutandis* from Case (3.1);

Case (3.4): Assume $f=h$, F?(n) $\neq$ F?(m), and both tapes appropriate to F?(n) and F?(m).

$f^*=h^*$ follows *mutatis mutandis* from Case (3.2);

Case (3.5): Assume $f=h$, F?(n)$\neq$F?(m), and one tape inappropriate to F?(n) or F?(m).

$f^*=h^*$ follows *mutatis mutandis* from Case (3.2);

Case (3.6): Assume $f=h$, F?(n)$\neq$F?(m), and both tapes inappropriate to F?(n) and F?(m).

$f^* \neq h^*$ follows *mutatis mutandis* from Case (3.2).

Therefore, if $\langle n,i,f \rangle \neq \langle m,j,h \rangle$ then $\langle n^*,i,f^*,o,z \rangle \neq \langle m^*,j,h^*,o,g \rangle$, thereby establishing the lemma.

We now define C_F as a mapping from $P(N\mathrm{x}N\mathrm{x}N^N)$ into itself such that for every $A\varepsilon P(N\mathrm{x}N\mathrm{x}N^N)$:

(1) $C_F(A) = \{C_F^*(x)\}$, if:

(i) $A = \{x\}$

(ii) x is an appropriate initial configuration

(2) $C_F(A) = \bigcup_{x\varepsilon A} \{C_F^*(x)\}$, if there exists an appropriate initial configuration y such that $\{(C_F^*)^\tau(y)\} = A$ where τ is a finite ordinal

(3) $C_F(A) = A$, otherwise.

We also defined C_P as a mapping from $P(N\mathrm{x}N\mathrm{x}N^N\mathrm{x}N\mathrm{x}N^N)$ into itself such that for every $A\varepsilon P(N\mathrm{x}N\mathrm{x}N^N\mathrm{x}N^N)$:

(1) $C_P(A) = \bigcup_{x\varepsilon A} \{C_P^*(x)\}$, if:

(i) $A = \{x\}$

(ii) x is an appropriate initial configuration

(2) $C_P(A) = \bigcup_{x\varepsilon A} \{C_P^*(x)\}$, if:

(i) there exists an appropriate initial configuration y such that $C_P^\tau(\{y\}) = A$ where τ is a finite ordinal

(ii) there does not exist $x\varepsilon A$ such that $x\varepsilon C_P^*(x)$ and $1x$ is a final state

(3) $C_P(A) = A$, otherwise.

We define W_F as a mapping from $P(N\mathrm{x}N\mathrm{x}N^N)$ into $P(N\mathrm{x}N\mathrm{x}N^N\mathrm{x}N\mathrm{x}N^N)$ such that for every $A\varepsilon P(N\mathrm{x}N\mathrm{x}N^N)$:

(1) $W_F(A) = \bigcup_{x\varepsilon A} \{K_F(x)\}$, if:

(i) $\bar{\bar{A}}=1$

(ii) $x\varepsilon A$ is an appropriate initial configuration

(2) $W_F(A) = \bigcup_{x\varepsilon A} \{K_F(x)\}$, if:

(i) $\bar{\bar{A}}=1$

(ii) there exists an appropriate initial configuration such that $C_F^\tau(\{y\}) = A$ where τ is a finite ordinal

(3) $W_F(A) = \bigcup_{x\varepsilon A} \{H(x)\}$, otherwise.

Lemma 2. If C_F is the mapping just defined from $P(NxNxN^N)$ into itself and C_P is the mapping just defined from $P(NxNxN^NxNxN^N)$ into itself, W_F is a mapping such that $W_F(C_F(A)) = C_P(W_F(A))$ for all $A\varepsilon P(NxNxN^N)$.

Proof. We note that W_F will be an injection since it is the pointwise extension of the injection W_F^*. Also, if $x\varepsilon A$ then $W_F^*(x)\varepsilon W_F(A)$ for all $A\varepsilon P(NxNxN^N)$.

Case (1): (i) $A\varepsilon P(NxNxN^N)$, (ii) $\bar{\bar{A}}=1$, and (iii) there exists an appropriate initial configuration y such that $A = C_F^\tau(\{y\})$ where τ is a finite ordinal. Then:

$$W_F(C_F(A)) = W_F(\bigcup_{x\varepsilon A} \{C_F^*(x)\}) = \bigcup_{K_F(x)\varepsilon W_F(A)} \{C_P^*(K(x))\} = C_P(W_F(A));$$

Case (2): (i) $A\varepsilon P(NxNxN^N)$, and (ii) there does not exist an appropriate initial configuration y such that $A = C_F^\tau(\{y\})$ where τ is a finite ordinal or $\bar{\bar{A}}\neq 1$. Then:

$$W_F(C_F(A)) = W_F(\bigcup_{x\varepsilon A} \{C_F^*(x)\}) = \bigcup_{H_F(x)\varepsilon W_F(A)} \{C_P^*(H_F(x))\} = C_P(W_F(A)).$$

We also conclude that the configurations of finite automata and pushdown automata are correlative, so we may effect a recognition-preserving morphism once we have devised appropriate Boolean counterparts for the mappings defined between machine configurations. We have as the *alter ego* of C_F the mapping T_F from $\theta^{N\times N\times N^N}$ into itself defined as:

$T_F(b) = f$ where $f(j) = 1$ iff $j \varepsilon C_F(b^{-1}(1))$ for all $b \varepsilon \theta^{N\times N\times N^N}$.

For C_P, we have T_P as a mapping from $\theta^{N\times N\times N^N \times N\times N^N}$ into itself defined as:

$T_P(b) = f$ where $f(j) = 1$ iff $j \varepsilon C_P(b^{-1}(1))$ for all $b \varepsilon \theta^{N\times N\times N^N \times N\times N^N}$.

The *alter ego* of W_F^* is α^* which maps the atoms of $\theta^{N\times N\times N^N}$ into the atoms of $\theta^{N\times N\times N^N \times N\times N^N}$ defined as:

$\alpha^*(b) = y$ where $y(j) = 1$ iff $j \varepsilon \{W_F^*(E_F^{-1}(b))\}$ for all atoms b of $\theta^{N\times N\times N^N}$.

An atomwise extension of α^* provides the counterpart of W_F where the prescription is:

$$\alpha(X) = \bigvee_{a \leq X} \alpha^*(a) \text{ for all } X \varepsilon \theta^{N\times N\times N^N}.$$

Since we merely used identification functions in formulating these Boolean counterparts, we conclude that α is indeed a computer monomorphism such that $\alpha(T_F(X)) = T_P(\alpha(X))$ for all $X \varepsilon \theta^{N\times N\times N^N}$. Furthermore, that α is a recognition-preserving computer morphism is evident from the construction procedure used to obtain the corresponding pushdown automaton.

Recognition-Preserving Morphism for Pushdown Automata

Since the class of context-free languages is a proper subclass of context-sensitive languages, pushdown automata are less powerful than linear bounded automata. The algebraic counterpart of this fact is a theorem which states that the computer of pushdown automata

$<\theta^{N\times N\times N^N\times N\times N^N}, T_{\mathbb{P}}>$ is homomorphic to the computer of linear bounded automata $<\theta^{N\times N\times N^N\times N}, T_{\mathbb{L}}>$. This computer homomorphism is also a recognition-preserving morphism.

Theorem 8. There exists a recognition-preserving computer homomorphism β from $<\theta^{N\times N\times N^N\times N\times N^N}, T_{\mathbb{P}}>$ to $<\theta^{N\times N\times N^N\times N}, T_{\mathbb{L}}>$.

Proof. Let $<n,i,f,j,t>\varepsilon N\times N\times N^N\times N\times N^N$ be an appropriate configuration for the pushdown automaton $\mathbb{P}?(n)$. Given that $\mathbb{P}?(n)\varepsilon$NPDA, we may construct a non-deterministic linear bounded automaton. Recall that $Q_{\mathbb{P}}(\mathbb{P}?(n))$ is the cardinality of the state set and $Z_{\mathbb{P}}(\mathbb{P}?(n))$ is the cardinality of the pushdown alphabet for machine $\mathbb{P}?(n)$. It has been shown that $\mathbb{P}?(n)$ cannot stack more than $S = [Q_{\mathbb{P}}(\mathbb{P}?(n))\cdot Z_{\mathbb{P}}(\mathbb{P}?(n))]$ pushdown alphabetic characters on the pushdown store for a given input alphabetic character and return to the same state for which this stacking operation began without the stack becoming cyclic [30]. Using this fact, we take as "states" the combination of the state of $\mathbb{P}?(n)$ and the top S symbols from the stack that results on each cycle of a computation. Note that the pushdown stack is merely a means of augmenting the "memory capability" as incorporated via states of a pushdown automaton. In order to get a natural number to correspond to these "states," we define a mapping G relative to the state, the pushdown head position, and the pushdown tape coordinates of a pushdown automaton configuration $<n,i,f,j,t>$ such that $G(n,j,t) = 2^n\cdot 3^{t(j)}\cdot 5^{t(j\dot{-}1)}\cdot\ldots\cdot P_{j+1}^{t(j\dot{-}S)}$ where P_{j+1} is the jth+1 prime number. We take the input alphabet of $\mathbb{P}?(n)$ as the input alphabet for this linear bounded automaton we are devising. The triplets for this linear bounded automaton are obtained by considering atoms $a \leq b$ where $b = T^{\tau}_{\mathcal{P}}(d)$ where $d^{-1}(1)$ is an initial configuration for $\mathcal{P}?(n)$. The resulting triplets have the form:

$$<G[1a^{-1}(1),4a^{-1}(1),5a^{-1}(1)],\ [(3a^{-1}(1))(2a^{-1}(1))],$$

$$\{<G[1c^{-1}(1),4c^{-1}(1),5c^{-1}(1)],\ M(c)>|c \in T_{\mathcal{P}}(a)\}>$$

$$\text{where } M(c) = \begin{cases} ST, \text{ if } 2c^{-1}(1) - 2a^{-1}(1) = 0 \\ RT, \text{ if } 2c^{-1}(1) - 2a^{-1}(1) = 1 \end{cases}$$

Every state-alphabetic character pair must occur in the first two positions of a triplet. We define every such pair by adding any missing pairs using as the third place member, the pair whose state is that of such a missing combination and whose action is ST.

The total number of states created by this construction procedure is at most $\left(Q_{\mathcal{P}}(\mathcal{P}?(n))\right) \cdot K$ where $K = \left(\sum_{j=1}^{S} [Z_{\mathcal{P}}(\mathcal{P}?(n))]^j\right)$, if $Z_{\mathcal{P}}(\mathcal{P}?(n))>1$; otherwise, $K=S$. The value K represents the total number of distinct strings of pushdown symbols of length L for $1 \leq L \leq S$. The summation $\left(\sum_{j=1}^{S} x^j\right) = ([(x^{S+1}-1)/(x-1)]-1)$ for $x \geq 2$.

Let B denote this machine. We now want to relabel the states of B as consecutive natural numbers so that we get a standard form of this machine. We can then locate the machine in the previously constructed sequence of linear bounded automata. The alphabet of B already consists of consecutive natural numbers because it was the input alphabet of the machine $\mathcal{P}?(n)$, which was in standard form. Let V be a mapping from the set of natural numbers $\{x|0 \leq x \leq (\bar{\bar{B}}-1)\}$ to the set of states $\{1jB|1 \leq j \leq \bar{\bar{B}}\}$ of the machine B such that $V(0) = MIN\{1jB|1 \leq j \leq \bar{\bar{B}}\}$ and $V(x+1) = \left(MIN(\{1jB|1 \leq j \leq \bar{\bar{B}}\}-\{V(i)|0 \leq i \leq x\})\right)$ for $0 \leq x \leq (\bar{\bar{B}}-1)$. We now replace each member 1iB of $\{1jB \mid 1 \le j \le \bar{\bar{B}}\}$ wherever it occurs in the machine B by the natural number $V^{-1}(1iB)$. Hence, we now have a machine whose states are the consecutive natural numbers that range from 0 to $(\bar{\bar{B}}-1)$. We now place this machine in standard form. Let A denote the machine obtained from the machine B by relabeling its states and reordering its triplets.

Since A is now in standard form, $A \in NLBA$ and $\mathcal{L}^{-1}(A) \in N$. We now define a mapping from $P(N \times N \times N^N \times N \times N^N)$ to $P(N \times N \times N^N \times N)$ in terms of two functions $K_{\mathcal{P}}$ and $H_{\mathcal{P}}$ from $N \times N \times N^N \times N \times N^N$ to $N \times N \times N^N \times N$. Consider the original appropriate configuration $\langle n,i,f,j,t\rangle$ for the pushdown automaton $\mathcal{P}?(n)$. Then $K_{\mathcal{P}}(\langle n,i,f,j,t\rangle) = \langle n^*,i,f^*,\mu(f)\rangle$ where

$$\text{(i)} \quad n^* = \begin{cases} V^{-1}(G(n,j,t)), \text{ if } L^{-1}(A)=0 \\ \sum_{k=0}^{(\mathcal{L}^{-1}(A))-1} Q_{\mathcal{L}}(\mathcal{L}(k))+[V^{-1}(G(n,j,t))], \text{ if } \mathcal{L}^{-1}(A)>0 \end{cases}$$

$$\text{(ii)} \quad f^*(x) = \begin{cases} [(f(x)+1)-21\mathcal{P}?(n)] \text{ for } 0 \le x \le \mu(f), \\ 0 \text{ for } x>\mu(f), \end{cases} \quad \text{if } \mathcal{L}^{-1}(A)=0$$

$$\text{(iii)} \quad f^*(x) = \begin{cases} \sum_{k=0}^{(\mathcal{L}^{-1}(A))-1} A_{\mathcal{L}}(\mathcal{L}(k))+[(f)x)+1)-21\mathcal{P}?(n)] \text{ for } 0 \le x \le \mu(f), \\ 0 \text{ for } x>\mu(f), \end{cases} \quad \text{if } \mathcal{L}^{-1}(A)>0.$$

The effect of $K_{\mathcal{P}}$ is to map an appropriate configuration of the pushdown automaton P?(n) to an appropriate configuration of the linear bound automaton A that corresponds to $\mathcal{P}?(n)$. However, the effect of $H_{\mathcal{P}}$ is to map configurations of $\mathcal{P}?(n)$ whether appropriate or not to inappropriate configurations for A by relabeling the alphabetic characters of the tape so that they do not occur in the alphabet of A. Machine A is precluded from making any change on such a configuration. Hence, $H_{\mathcal{P}}(\langle n,i,f,j,t\rangle) = \langle n^*,i,f^*,\mu(f)\rangle$ where

$$\text{(i)}\quad n^* = \begin{cases} V^{-1}(G(n,j,t)), \text{ if } \mathcal{L}^{-1}(A)=0 \\ \displaystyle\sum_{k=0}^{(\mathcal{L}^{-1}(A))-1} Q_{\mathcal{L}}(\mathcal{L}(k))+[V^{-1}(G(n,j,t))], \text{ if } \mathcal{L}^{-1}(A)\neq 0 \end{cases}$$

$$\text{(ii)}\quad f^*(x) = \begin{cases} \displaystyle\sum_{k=0}^{\mathcal{L}^{-1}(A)} A_{\mathcal{L}}(\mathcal{L}(k))+[(f(x)+1)-21\mathcal{P}?(n)] \text{ for } 0\le x\le\mu(f), \\ \qquad\qquad\qquad\qquad\qquad\qquad \text{if } \mathcal{L}^{-1}(A)\in N. \\ 0 \text{ for } x>\mu(f), \end{cases}$$

We again introduce the mapping $C_{\mathcal{P}}$ from $P(N\times N\times N^N\times N\times N^N)$ into itself. (See corresponding theorem on finite automata for definition.) We also introduce the mapping $C_{\mathcal{L}}$ from $P(N\times N\times N^N\times N)$ into itself. $C_{\mathcal{L}}$ is defined recursively using the same three conditions used to define $C_{\mathcal{P}}$. Using this auxiliary mapping we can define $W_{\mathcal{P}}$ as a mapping from $P(N\times N\times N^N\times N\times N^N)$ to $P(N\times N\times N^N\times N)$ such that for every $A\in P(N\times N\times N^N\times N\times N^N)$:

(1) $W_{\mathcal{P}}(A) = \bigcup_{x\in A} \{K_{\mathcal{P}}(x)\}$, if there exists an appropriate initial configuration y such that $C_{\mathcal{P}}^{\tau}(\{y\})=A$ where τ is a finite ordinal

(2) $W_{\mathcal{P}}(A) = \bigcup_{x\in A} \{H_{\mathcal{P}}(x)\}$, otherwise.

<u>Lemma 3</u>. If $C_{\mathcal{P}}$ is the mapping just defined from $P(N\times N\times N^N\times N\times N^N)$ into itself and $C_{\mathcal{L}}$ is the mapping just defined from $P(N\times N\times N^N\times N)$ into itself, $W_{\mathcal{P}}$ is a mapping such that $W_{\mathcal{P}}(C_{\mathcal{P}}(A)) = C_{\mathcal{L}}(W_{\mathcal{P}}(A))$ for every $A\in P(N\times N\times N^N\times N\times N^N)$.

Proof.

Case (1): (i) $A\varepsilon P(N\times N\times N^N\times N\times N^N)$ and (ii) there exists an appropriate initial configuration y such that $A = C^{\tau}_{\mathcal{P}}(\{y\})$ where τ is a finite ordinal. Then:

$$W_{\mathcal{P}}(C_{\mathcal{P}}(A))=W_{\mathcal{P}}(\bigcup_{x\varepsilon A}\{C^{*}_{\mathcal{P}}(x)\})=\bigcup_{K_{\mathcal{P}}(x)\varepsilon W_{\mathcal{P}}(A)}\{C^{*}_{\mathcal{L}}(K_{\mathcal{P}}(x))\}=C_{\mathcal{L}}(W_{\mathcal{P}}(A));$$

Case (2): (i) $A\varepsilon P(N\times N\times N^N\times N\times N^N)$ and (ii) there does not exist an appropriate initial configuration y such that $A=C^{\tau}_{\mathcal{P}}(\{y\})$ where τ is a finite ordinal. Then:

$$W_{\mathcal{P}}(C_{\mathcal{P}}(A)=W_{\mathcal{P}}(\bigcup_{x\varepsilon A}\{C^{*}_{\mathcal{P}}(x)\})=\bigcup_{H_{\mathcal{P}}(x)\varepsilon W_{\mathcal{P}}(a)}\{C^{*}_{\mathcal{L}}(H_{\mathcal{P}}(x))\}=C_{\mathcal{L}}(W_{\mathcal{P}}(A)).$$

We may now establish that the Boolean counterpart of $W_{\mathcal{P}}$ is a recognition-preserving computer homomorphism. Recall that the length of the stack for a pushdown automaton can only be increased by adding a symbol at the top and only be decreased by erasing a symbol at the top. In either case, only one symbol may be added or deleted from the stack on each iteration of the configuration function $C_{\mathcal{P}}$. The function $C_{\mathcal{P}}$ has been defined recursively to reflect configuration changes for computations that begin with an appropriate initial configuration; otherwise, $C_{\mathcal{P}}$ is defined as an identity function. The change of a pushdown configuration is determined by what state-input symbol-pushdown symbol combination is being scanned. This information has been preserved in the construction procedure used to obtain the linear bounded automaton A that corresponds to the pushdown automaton $\mathcal{P}?(n)$. The input alphabet of $\mathcal{P}?(n)$ has been preserved since the input alphabet of A differs from the original only in a relabeling. Hence, the information provided by the input symbol scanned is available to A. The input head position and input tape are used as parameters in the configurations of A.

The stack of a pushdown automaton can be viewed as a means of augmenting the state set. The state set can provide a memory capability. Given this perception, we devised the states of the constructed machine A as combinations of the pushdown automaton state and the top $S = [Q_{\mathcal{P}}(\mathcal{P}?(n))\cdot Z_{\mathcal{P}}(\mathcal{P}?(n))]$ symbols of the stack. We may recall that S is the maximum number of symbols that can be placed in the stack without it becoming cyclic while we hold the input fixed letting the state vary, but requiring that the machine return to the state in which it was when the stacking operation began. Thus, any change in configuration subsequent to S stack changes must occur as a change of state or change of input tape. If such a non-cyclic change occurs, the construction procedure for obtaining the triplets for A would insure that the state set for A already contained a state which corresponds to the new state of $\mathcal{P}?(n)$ after the change. If the subsequent change was to stack only, we would have a cyclic condition. Under these circumstances, we would be repeating a string of pushdown symbols of length S that had already occurred in the computation on a previous iteration of the configuration function $C_{\mathcal{P}}$. This is so because all states of $\mathcal{P}?(n)$ would have occurred. Hence, the machine A has all the states it needs.

As we are monitoring all and only those computations that begin with an appropriate initial configuration for $\mathcal{P}?(n)$, we are assured that all and only those configuration changes specified in the original behavior of $\mathcal{P}?(n)$ are incorporated in the behavior of the corresponding constructed linear bounded automaton A. Behavior here means as usual that behavior specified for the machine $\mathcal{P}?(n)$ by a finite set of quadruples taken from NxNxNxP(NxACTxACTS) and for machine A by a finite set of triplets taken from NxNxP(NxACT).

That machine A can only perform the original computations performed by machine $\mathcal{P}?(n)$ is seen by considering the combined effects of the mappings $C_{\mathcal{P}}$, $K_{\mathcal{P}}$, $H_{\mathcal{P}}$, and $C_{\mathcal{L}}$ when they are used to define the mapping $W_{\mathcal{P}}$. $K_{\mathcal{P}}$ is used to relabel the components of appropriate pushdown configurations so they constitute appropriate configurations for the linear bounded automaton A. $H_{\mathcal{P}}$ is used to relabel the components of pushdown configurations so they constitute inappropriate configurations for the linear bounded automaton A. $W_{\mathcal{P}}$ is defined recursively in terms of $C_{\mathcal{P}}$, $K_{\mathcal{P}}$, and $H_{\mathcal{P}}$. As such, $W_{\mathcal{P}}$ maps sets of configurations that occur in a computation under $C_{\mathcal{P}}$ into sets that occur in computations under $C_{\mathcal{L}}$. Any computation that was not defined for $C_{\mathcal{P}}$ is automatically not defined for $C_{\mathcal{L}}$. The function $K_{\mathcal{P}}$ is selected when a computation is defined, and the function $H_{\mathcal{P}}$ is selected when a computation is not defined. We may conclude that configurations of the pushdown automata and linear bounded automata are appropriately mapped by $W_{\mathcal{P}}$ so that the Boolean counterpart of $W_{\mathcal{P}}$ will be a recognition-preserving computer homomorphism. We have a homomorphism rather than a monomorphism because not all the details of the stack were preserved.

Before we can define the Boolean counterpart of $W_{\mathcal{P}}$, we need the counterparts of $C_{\mathcal{P}}$ and $C_{\mathcal{L}}$. The *alter ego* of $C_{\mathcal{P}}$ is defined exactly as it was for the corresponding theorem for finite automata. For $C_{\mathcal{L}}$, we have the mapping $T_{\mathcal{L}}$ from $\theta^{N\times N\times N^N\times N}$ into itself defined as:

$T_{\mathcal{L}}(b)=f$ where $f(j)=1$ iff $j\varepsilon C_{\mathcal{L}}(b^{-1}(1))$ for all $b\varepsilon\theta^{N\times N\times N^N\times N}$.

The *alter ego* of $W_{\mathcal{P}}$ is the mapping β from $\theta^{N\times N\times N^N\times N\times N^N}$ to $\theta^{N\times N\times N^N\times N}$ defined as:

$\beta(b)=y$ where $y(j)=1$ iff $j\varepsilon W_{\mathcal{P}}(b^{-1}(1))$ for all $b\varepsilon\theta^{N\times N\times N^N\times N\times N^N}$.

Since we again used only identification functions, we conclude that β is a recognition-preserving computer homomorphism such that

$\beta(T_{\mathcal{P}}(b))=T_{\mathcal{L}}(\beta(b))$ for all $b\varepsilon\theta^{N\times N\times N^N\times N\times N^N}$.

Recognition-Preserving Morphism for Linear Bounded Automata

Since the class of context-sensitive languages is a proper subclass of the class of recursively enumerable sets, linear bounded automata are less powerful than non-deterministic Turing machines. The algebraic counterpart of this fact is a theorem which states that the computer of linear bounded automata, $<\theta^{N\times N\times N^N\times N},T_{\mathcal{L}}>$, may be embedded in the computer of non-deterministic Turing machines, $<\theta^{N\times N\times N^N},T_{\mathcal{T}}>$, via a recognition-preserving computer morphism.

Theorem 9. There exists a recognition-preserving computer monomorphism γ from $<\theta^{N\times N\times N^N\times N},T_{\mathcal{L}}>$ to $<\theta^{N\times N\times N^N},T_{\mathcal{T}}>$.

Proof. Let $<n,i,f,\mu(f)>\varepsilon N\times N\times N^N\times N$, the configuration set for NLBA, which is an appropriate configuration for $\mathcal{L}?(n)$. Given that $\mathcal{L}?(n)\varepsilon$NLBA, we may construct a non-deterministic Turing machine D whose triplets are of the form $jD = <1j\mathcal{L}?(n),2j\mathcal{L}?(n),3j\mathcal{L}?(n)>$ for $1\le j\le\overline{\overline{\mathcal{L}?(n)}}$ and $jD = <U,C,\{<U,ST>\}>$ for $\overline{\overline{\mathcal{L}?(n)}}+1\le j\le\overline{\overline{\mathcal{L}?(n)}}+Q_{\mathcal{L}}(\mathcal{L}?(n))$ where U ranges over the set of states for machine $\mathcal{L}?(n)$ and C is a constant whose value is $(MAX\{2j\mathcal{L}?(n)|1\le j\le\mathcal{L}?(n)\})+1$.

The triplets of the form <U,C{<U,ST>}> have been added so as to include a new character C which acts as the blank character of the alphabet for machine D. In order to comply with the definition of non-deterministic Turing machines, we include every combination of state for ℒ?(n) with the alphabetic character C. Note, however, that these additional triplets do not add any novelty to the behavior of machine D over the behavior of machine ℒ?(n). Recall that the tapes of the linear bounded automata were taken from the set SEQ. Such tapes had an initial finite segment of positive integers which represented the input string. This finite initial segment was followed by an infinite number of zeroes that served to mark the end of the input. The recognition criteria for linear bounded automata specified that the machine must terminate with the head position at (μ(f)+1), which is the first tape position where a zero occurred. Recognition was then determined by whether or not the machine was in a final state. As the symbol then scanned was a zero and zero does not occur in any machine's alphabet, no move was possible. The tapes appropriate for the constructed machine D have the same initial finite segment of positive integers but with all of the zeroes replaced by the character C. Interpreting these same recognition criteria for non-deterministic Turing machines, the machine D will again stop with its head position at (μ(f)+1), but the symbol scanned now will be the character C rather than the character zero. Machine D at this time will be in the same state in which ℒ?(n) would have been. The next move of machine D will be determined by one of the triplets added to the original set of ℒ?(n) because C is the symbol scanned. The specific triplet will be the one that has the state that $\mathcal{L}?(n)$ would have had under these conditions. The next state of these triplets is always the present state and the action is always ST. Hence, the state remains whatever state $\mathcal{L}?(n)$ would have had; and since the action is ST, no change occurs in the configuration. Again, recognition would be determined by whether or not machine D was in a final state. These final states are the same as those of $\mathcal{L}?(n)$. We conclude that the original behavior of $\mathcal{L}?(n)$ has not been altered through reformulation as a non-deterministic Turing machine. The reformulation was necessary to satisfy the definition of non-deterministic Turing machines. From this point on the proof parallels exactly the proof used for the recognition-preserving morphism for finite automata.

As $D \varepsilon NTM$, we have $\mathcal{T}^{-1}(D) \varepsilon N$ so that D is the $\mathcal{T}^{-1}(D)+1$ machine whose states for $1 \leq j \leq Q_{\mathcal{T}}(D)$ are the numbers given by:

(1) $[1j\mathcal{L}?(n)-11\mathcal{L}?(n)]$, if $\mathcal{T}^{-1}(D)=0$

(2) $\sum_{k=0}^{(\mathcal{T}^{-1}(D))-1} (\mathcal{T}(k))+[1j\mathcal{L}?(n)-11\mathcal{L}?(n)]$, if $\mathcal{T}^{-1}(D)>0$.

The alphabetic characters of the machine D for $1 \leq j \leq ((Q_{\mathcal{T}}(D))+1$ are the numbers given by:

(1) $[2(jQ_{\mathcal{T}}(D))D-21\mathcal{L}?(n)]$, if $\mathcal{T}^{-1}(D)=0$

(2) $\sum_{k=0}^{(\mathcal{T}^{-1}(D))-1} (\mathcal{T}(k))+[2(jQ_{\mathcal{T}}(D))D-21\mathcal{L}?(n)]$, if $\mathcal{T}^{-1}(D)>0$.

We define a mapping $W^*_{\mathcal{L}}$ from $N x N x N^N x N$ to $N x N x N^N$ in terms of the two functions $K_{\mathcal{L}}$ and $H_{\mathcal{L}}$. Again $K_{\mathcal{L}}$ and $H_{\mathcal{L}}$ are used to achieve the same effect as K_F and H_F for finite automata. For the linear bounded automaton $Ƶ?(n)$ and the corresponding constructed machine $D\epsilon NTM$, we let $W^*_{Ƶ}(<n,i,f,\mu(f)>) = K_{Ƶ}(<n,i,f,\mu(f)>)$ where $K_{Ƶ}(<n,i,f,\mu(f)>)$ is:

(1) $<n^*,i,f^*>$, if $f''N-\{0\} \subseteq \{2j\overline{\overline{Ƶ?(n)}}\}$ and $0 \le i \le (\mu(f)+1)$ with:

$$\text{(i)}\quad n^* = \begin{cases} [n-11Ƶ?(n)], \text{ if } \mathcal{T}^{-1}(D)=0, & \\ & \text{if } \mathcal{T}^{-1}(D)>0 \\ \sum\limits_{k=0}^{(\mathcal{T}^{-1}(D))-1} Q_{\mathcal{T}}(\mathcal{T}(k))+[n-11Ƶ?(n)], & \end{cases}$$

$$\text{(ii)}\quad f^*(x) = \begin{cases} [(f(x)+1)-21Ƶ?(n)] \text{ for } 0 \le x \le \mu(f), & \\ & \text{if } \mathcal{T}^{-1}(D)=0 \\ 2(\bar{\bar{D}})D \text{ for } x>\mu(f), & \end{cases}$$

$$\text{(iii)}\quad f^*(x) = \begin{cases} \sum\limits_{k=0}^{(\mathcal{T}^{-1}(D))-1} A_{\mathcal{T}}(\mathcal{T}(k))+[(f(x)+1)-21Ƶ?(n)] & \\ \qquad \text{for } 0 \le x \le \mu(f), & \\ & \text{if } \mathcal{T}^{-1}(D)>0 \\ \sum\limits_{k=0}^{(\mathcal{T}^{-1}(D))-1} A_{\mathcal{T}}(\mathcal{T}(k))+[(2(\bar{\bar{D}})D-21Ƶ?(n)] & \\ \qquad \text{for } x>\mu(f), & \end{cases}$$

and $W^*_{Ƶ}(<n,i,f,\mu(f)>) = H_{Ƶ}(<n,i,f,\mu(f)>)$ where $H_{Ƶ}(<n,i,f,\mu(f)>)$ is:

(2) $<n^*,i,f^*>$, if $f''N-\{0\} \subseteq \{2jƵ?(n) \mid 1 \le j \le Ƶ?(n)\}$ or $i>(\mu(f)+1)$ with:

(i) n^* as given by (1) above

$$\text{(ii)}\quad f^*(x) = \begin{cases} \sum\limits_{k=0}^{\mathcal{T}^{-1}(D)} A_{\mathcal{T}}(\mathcal{T}(k))+[(f(x)+1)-21Ƶ?(n)] & \\ \qquad \text{for } 0 \le x \le \mu(f), & \\ & \text{if } \mathcal{T}^{-1}(A)\epsilon N. \\ \sum\limits_{k=0}^{\mathcal{T}^{-1}(D)} A_{\mathcal{T}}(\mathcal{T}(k))+[2(\bar{\bar{D}})D-21Ƶ?(n)] \text{ for } x>\mu(f), & \end{cases}$$

<u>Lemma 4</u>. The mapping $W^*_{\mathcal{L}}$ from $N \times N \times N^N \times N$ to $N \times N \times N^N$ is an injection.

<u>Proof</u>. The proof of the lemma parallels exactly the proof of the corresponding theorem for finite automata.

We now introduce a mapping $C_{\mathcal{L}}$ from $P(N \times N \times N^N \times N)$ into itself and a mapping $C_{\mathcal{T}}$ from $P(N \times N \times N^N)$ into itself. Both the mappings $C_{\mathcal{L}}$ and $C_{\mathcal{T}}$ are defined recursively using the same three conditions that were used to define the mapping $C_{\mathcal{F}}$. (See corresponding theorem on finite automata for definition.) We now define $W_{\mathcal{L}}$ as a mapping from $P(N \times N \times N^N \times N)$ to $P(N \times N \times N^N)$ such that for every $A \epsilon P(N \times N \times N^N \times N)$:

(1) $W_{\mathcal{L}}(A) = \bigcup_{x \epsilon A} \{K_{\mathcal{L}}(x)\}$, if there exists an appropriate initial configuration y such that $C^{\tau}_{\mathcal{L}}(\{y\})=A$ where τ is a finite ordinal

(2) $W_{\mathcal{L}}(A) = \bigcup_{x \epsilon A} \{H_{\mathcal{L}}(x)\}$, otherwise.

<u>Lemma 5</u>. If $C_{\mathcal{L}}$ is the mapping just defined from $P(N \times N \times N^N \times N)$ into itself and $C_{\mathcal{T}}$ is the mapping just defined from $P(N \times N \times N^N)$ into itself, $W_{\mathcal{L}}$ is a mapping such that $W_{\mathcal{L}}(C_{\mathcal{L}}(A)) = C_{\mathcal{T}}(W_{\mathcal{L}}(A))$ for every $a \epsilon P(N \times N \times N^N \times N)$.

<u>Proof</u>.

Case (1): (i) $A \epsilon P(N \times N \times N^N \times N)$, and (ii) there exists an appropriate initial configuration y such that $A = C^{\tau}_{\mathcal{L}}(\{y\})$ where τ is a finite ordinal. Then:

$$W_{\mathcal{L}}(C_{\mathcal{L}}(A))=W_{\mathcal{L}}(\bigcup_{x \epsilon A} \{C^*_{\mathcal{L}}(x)\})= \bigcup_{K_{\mathcal{L}}(x) \epsilon W_{\mathcal{L}}(A)} \{C_{\mathcal{T}}(K_{\mathcal{L}}(x))\} = C_{\mathcal{T}}(W_{\mathcal{L}}(A));$$

Case (2): (i) $A \epsilon P(N \times N \times N^N \times N)$, and (ii) there does not exist an appropriate initial configuration such that $A = C^{\tau}_{\mathcal{L}}(\{y\})$ where τ is a finite ordinal. Then:

$$W_{\mathcal{L}}(C_{\mathcal{L}}(A))=W_{\mathcal{L}}(\bigcup_{x \epsilon A} \{C^*_{\mathcal{L}}(x)\})= \bigcup_{H_{\mathcal{L}}(x) \epsilon W_{\mathcal{L}}(A)} \{C^*_{\mathcal{T}}(H_{\mathcal{L}}(x))\}=C_{\mathcal{T}}(W_{\mathcal{L}}(A)).$$

We also conclude that configurations of linear bounded automata and non-deterministic Turing m chines are correlative, so we may effect a recognition-preserving morphism once we have devised appropriate Boolean counterparts for the mappings defined between machine configurations. We have as the *alter ego* of $C_{\mathcal{L}}$ the mapping $T_{\mathcal{L}}$ from $\theta^{N\times N\times N^N\times N}$ into itself defined as:

$$T_{\mathcal{L}}(b)=f \text{ where } f(j)=1 \text{ iff } j\epsilon C_{\mathcal{L}}(b^{-1}(1)) \text{ for all } b\epsilon\theta^{N\times N\times N^N\times N}.$$

For $C_{\mathcal{T}}$, we have $T_{\mathcal{T}}$ as a mapping from $\theta^{N\times N\times N^N}$ into itself defined as:

$$T_{\mathcal{T}}(b)=f \text{ where } f(j)=1 \text{ iff } j\epsilon C_{\mathcal{T}}(b^{-1}(1)) \text{ for all } b\epsilon\theta^{N\times N\times N^N}.$$

Finally, we obtain the computer morphism γ as the *alter ego* of $W_{\mathcal{L}}$ as a mapping from $\theta^{N\times N\times N^N\times N}$ to $\theta^{N\times N\times N^N}$ defined as:

$$\gamma(b)=y \text{ where } y(j)=1 \text{ iff } j\epsilon W_{\mathcal{L}}(b^{-1}(1)) \text{ for all } b\epsilon\theta^{N\times N\times N^N\times N}.$$

Since we merely used identification functions in formulating these Boolean counterparts, we conclude that γ is indeed a computer monomorphism such that $\gamma(T_{\mathcal{L}}(b)) = T_{\mathcal{T}}(\gamma(b))$ for all $b\epsilon\theta^{N\times N\times N^N\times N}$. That γ is also a recognition-preserving morphism is evident from the construction procedure used to obtain the corresponding non-deterministic Turing machine.

CHAPTER IV

CONCLUDING REMARKS

Computer Morphisms and Turing Machine Varieties

On the basis of the previous theorems we have an algebraic counterpart of the classical hierarchy of recognizers in the theory of abstract computers. We have established recognition-preserving computer morphisms between the computers of finite automata and pushdown automata, between the computers of pushdown automata and linear bounded automata, and between the computers of linear bounded automata and non-deterministic Turing machines. That the latter morphism involves the non-deterministic variety of Turing machine prompts us to consider whether or not computer morphisms may be used to characterize the relation between the deterministic and non-deterministic varieties of Turing machines. As previously noted, both varieties have the same recognition power. We would like to be able to reflect this relationship with a recognition-preserving morphism from the computer for the non-deterministic variety to the computer for the deterministic variety.

Let us consider the method used to show the equivalence of the recognition capabilities of deterministic and non-deterministic Turing machines. The objective is to devise a deterministic machine that recognizes all and only those tapes recognized by the non-deterministic machine. Using a theorem that a K-tape deterministic Turing machine has no more recognition power than a one-tape deterministic Turing machine, the standard method to show that a three-tape deterministic machine x recognizes all and only those tapes recognized by a non-deterministic machine y [15]. Tape one is the original input tape of machine y. Tape two is always a finite sequence of integers modulo $(\mathrm{MAX}\{(\overline{\overline{3jy}})|1\le j\le\overline{\overline{y}}\})$ which represent the choice to be selected for a particular present state-symbol scanned combination. The choices for each such present state-symbol scanned combination are assumed to be indexed by the natural numbers between zero and $(\mathrm{MAX}\{(\overline{\overline{3jy}})|1\le j\le\overline{\overline{y}}\})-1$. Tape three is a working tape on which a particular computation is carried out. The simulation procedure has machine x copy tape one (which must be assumed to be finite otherwise the copying would never terminate) onto tape three. Then machine x generates a sequence of integers modulo $(\mathrm{MAX}\{(\overline{\overline{3jy}})|1\le j\le\overline{\overline{y}}\})$ on tape two. These sequences of integers are systematically generated by length and by natural order within sequences of the same length. Not every such sequence need correspond to a possible sequence of moves for machine y because there are not necessarily $(\mathrm{MAX}\{(\overline{\overline{3jy}})|1\le j\le\overline{\overline{y}}\})$ choices for each state-symbol combination. Once this sequence is generated, machine x attempts to recognize the input string on tape three by using, on the τ*th* cycle, the τ*th* integer on tape two to decide which choice to select from those possible for the present state-symbol scanned combination that occurs on the τ*th* cycle. If machine y had accepted the input string on tape one using this sequence of moves, so would machine x. If acceptance does not occur the entire procedure is repeated, but the sequence of moves is changed by regarding tape two as a counter which is incremented on each subsequent attempt at recognition. The number of integers on tape two corresponds to the number of moves to be made in an attempt at recognition. First, all paths of length one are tried, then all paths of length two, etc. Recognition, if it occurs, occurs after a finite number of moves. Hence, if such a finite sequence of moves leads to recognition, it will eventually occur as a sequence of moves determined by tape two. In this sense machine x recognizes all and only those tapes recognized by machine y.

It is apparent that machine x makes many intermediate moves while generating the sequence of integers on tape two and copying strings on tape three that are not directly related to the actual recognition computation of y. Consequently, it is impossible to use this approach in order to obtain a computer morphism between the computer for y and the computer of x because the two computers would not be synchronous. A computer morphism requires that the morphic computers be appropriately synchronous. If we retain synchronicity, we may have a weak recognition-preserving morphism in the sense that recognition occurs for the image machine whenever recognition occurs for the pre-image machine, but not conversely. That is, the image machine recognizes at least those tapes recognized by the pre-image machine yet it may also recognize others.

For example, let x be a non-deterministic Turing machine whose state set is S and whose alphabet is the set A. An abstract digital computer which models machine x is $<\theta^{S \times N \times A^N}, \mathcal{T}>$ whose state transition function $\mathcal{T}$ is defined as usual for a non-deterministic machine. Let

INIT = $\{b \mid b \varepsilon \theta^{S \times N \times A^N}$ and $b^{-1}(1)$ is an initial configuration$\}$ and APR = $\{c \mid c = \mathcal{T}^{\tau}(b)$ for some $b \varepsilon$INIT and $\tau \geq 0\}$. Let A^* be a new alphabet defined as $A^* = A \cup \{a^* \mid a^* = a + \bar{\bar{A}}$ for $a \varepsilon A\}$. We may now define a mapping K from the set APR to a new set of tapes $(A^*)^N$. If $b \varepsilon$INIT then:

$$(1) \quad (K(b))(0) = \begin{cases} ([3(b^{-1}(1))](0))^*, & \text{if } \mathcal{T}(b) \neq b \\ [3(b^{-1}(1))](0), & \text{otherwise.} \end{cases}$$

and

$$(2) \quad (K(b))(i) = [3(b^{-1}(1))](i), \text{ if } 0 < i \varepsilon N.$$

If $b \varepsilon$APR and $b \varepsilon \mathcal{T}^{\tau+1}(x)$ for $c \varepsilon$INIT then:

$$(1) \quad (K(b))(i) = (K(\mathcal{T}^{\tau}(c)))(i), \text{ if } 0 \leq i \leq \tau$$

$$(2) \quad (K(b))(i) = [3(c^{-1}(1))](i), \text{ if } i > \tau + 1$$

$$(3) \quad (K(b))(\bar{\bar{\tau}}+1) = \begin{cases} ([3(c^{-1}(1))](\bar{\bar{\tau}}+1))^*, & \text{if } \mathcal{T}^{\tau}(c) \neq T^{\tau+1}(c) \\ [3(c^{-1}(1))](\bar{\bar{\tau}}+1), & \text{otherwise.} \end{cases}$$

We now define a mapping Q from K''APR to $(A^*)^N$ such that $[Q(K(\mathcal{T}^{\tau}(b)))](\bar{\bar{\tau}}) = [K(\mathcal{T}^{\tau}(b))](\bar{\bar{\tau}})$ for all finite ordinals τ and $b \varepsilon$INIT. We may now construct a deterministic Turing machine y whose state set is $\{0,1\}$ and whose alphabet is A^* with initial state 0 and final state set $\{0,1\}$. The quadruples are given by the set $\{<0,a^*,1,RT>, <0,a,0,a>, <1,a^*,1,RT>, <1,a,1,a>\}$ where a, $a^* \varepsilon A^*$.

Note that if machine x stops after τ steps after having been started on the tape t, then machine y stops after τ steps after having been started on the tape $t^* = Q(K(\mathcal{T}^{\tau}(b)))$ for any finite ordinal τ where $b^{-1}(1)$ is an initial configuration and $t = 3(b^{-1}(1))$. However, the machine y also stops on tapes that are not in the set Q''(K''APR) and whose standard representation for machine y is also not in Q''(K''APR).

Suppose t is not recognized by machine x. Then the tape t^* given by $t^*(0) = (t(0))^*$ and $t^*(i) = t(i)$ for $i>0$ is recognized by y but it is not in Q''(K''APR) nor are any of its equivalent tapes in Q''(K''APR). Yet, it is possible to exhibit a computer monomorphism from the computer $<\theta^{S\times N\times A^N}, \mathcal{T}>$ to the computer $<\theta^{\theta\times N\times Q''(K''APR)}, T>$ whose state-transition function T is defined as other than the identity only on the atoms of $\theta^{\theta\times N\times Q''(K''APR)}$.

We have not succeeded in showing that there do not exist recognition-preserving morphisms that relate the computers for non-deterministic Turing machines to appropriate computers for deterministic machines. It can be easily shown that computers for non-deterministic machines are monomorphically related to computers which are reasonable candidates. That is to say, we can construct abstract digital computers whose transition functions are defined as other than the identity only on the atoms and stop or fail to stop appropriately. But we have not been able to devise ways of constructing deterministic Turing machines for these reasonable candidates. We also have not been able to show that such constructions are impossible. We have not attempted to solve the problem of how to proceed generally from appropriate abstract digital computers to the recognizers which they may be said to be modelling.

As an example, let x be a non-deterministic Turing machine whose state set is S and whose alphabet is the set A. An abstract digital computer which models machine x is $<\theta^{S\times N\times A^N}, \mathcal{T}>$ where the state-transition function $\mathcal{T}$ is defined as usual for a non-deterministic machine. Let $B = \theta^{S\times N\times A^N}$. Define a mapping H from B to θ^B such that

$(H(x))(y)=1$ iff $x=y$ for any $x,y\varepsilon B$. H is an injection as it is merely an identification function. Consider the abstract digital computer $<\theta^B,T>$ whose state transition function T is defined as $T(H(x))=H(\mathcal{T}(x))$ if $H(x)$ is an atom of θ^B and $T(b) = b$ if b is not an atom of θ^B. Note that the computer $<\theta^B,T>$ stops when and only when the computer $<B,\mathcal{T}>$ stops, and H is a computer monomorphism. The abstract digital computer $<\theta^B,T>$ is a reasonable candidate for a deterministic machine because its state-transition function T is defined as other than the identity only for atoms of θ^B.

Summary

The objective has been to recover the classical theory of recognizers within the theory of abstract digital computers. We have done so using only the algebraic concepts of abstract digital computer and computer morphism. In doing so, we have also addressed the question of how machines of different kinds can be algebraically related to reflect their capability relative to a particular process. We have considered the question relative to the process of recognition used by the machines of the hierarchy of recognizers induced by the hierarchy of languages.

Abstract digital computers provide a standard machine in which the abstract machines of the hierarchy can be modelled. The modelling mirrors the details of the computation procedure of the original machines. The essential parameters a machine uses are identified and subsequently encoded into an appropriate Boolean algebra with an additional operator. The additional operator on the algebra is defined so that it will reflect the interaction of a machine's essential parameters during the course of a computation. The Boolean algebra is the carrier of the abstract digital computer and the operator is its state-transition function.

An abstract digital computer has been constructed for each class of recognizers countenanced. The algebraic counterpart of the classical hierarchy of recognizers has been obtained in the theory of abstract computers by means of computer morphisms which hold between the computers of finite automata, pushdown automata, linear bounded automata, and Turing machines. The construction of an abstract digital computer that models a single recognizer is effective. But the construction of an abstract computer that models a whole class of recognizers is not effective since the class of recognizers must be ordered. Nevertheless, the morphisms that relate abstract digital computers in the manner expected are effective. Indeed, the relevant morphisms are obtained via constructions that depend on single machines. Effective coding techniques are used on single machines to obtain the machine in the next higher class of recognizers which recognizes all and only the tapes recognized by the original.

From this investigation we conclude that the algebraic concepts of abstract digital computer and computer morphisms are sufficient to yield a unified algebraic theory of a large portion of automata theory.

REFERENCES

1. Abbott, J. C. 1969. *Set, Lattices, and Boolean Algebras.* Boston: Allyn and Bacon, Inc. pp. 197-207.

2. Aho, A. V. and Ullman, J. D. 1968. "The Theory of Languages." *Mathematical Systems Theory* 2:97-125.

3. Arbib, M. A. 1969. *Theories of Abstract Automata.* New Jersey: Prentice-Hall, Inc. pp. 3-21.

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

5. Booth, T. L. 1967. *Sequential Machines and Automata Theory.* New York: John Wiley & Sons pp. 413-416.

6. Chomsky, N. 1959. "On Certain Formal Properties of Grammars." *Information and Control* 2:137-167.

7. Chomsky, N. 1962. "Context-Free Grammars and Pushdown Storage." *Quarterly Progress Rept. No. 65, M.I.T. Research Lab. Electronics.* pp. 187-194.

8. Chomsky, N. and Miller, G. A. 1958. "Finite State Languages." *Information and Control* 1:91-112.

9. Chomsky, N. and Schutzenberger, M. P. 1963. "The Algebraic Theory of Context-Free Languages." *Computer Programming and Formal Systems,* edited by P. Braffort and D. Hirschberg. Amsterdam: North Holland. pp. 122-214.

10. Fisher, P. C. 1965. "On Formalisms for Turing Machines," *J.ACM* 12:570.

11. Ginsburg, S. and Greibach, S. 1967. "Abstract Families of Languages." *IEEE Conf. Record of Eighth Annual Symp. on Switching and Automata Theory.* Austin, Tex., October, 1967. pp. 128-139.

12. Ginsburg, S. and Rose, G. F. 1966. "Preservation of Languages by Transducers." *Information and Control* 9:153-176.

13. Gross, M. and Lentin, A. 1970. *Introduction to Formal Grammars.* New York: Springer-Verlag. p. 153.

14. Halmos, P. R. 1966. *Algebraic Logic.* New York: Chelsea. p. 52.

15. Harrison, M. A. 1968. "Characterizations of Languages by Grammars and Automata" in *Foundations of Information Systems Engineering.* Univ. of Michigan Engineering Summer Conference June 17-28, 1968. 29 pp.

16. Hartmanis, J. and Stearns, R. E. 1966. *Algebraic Theory of Sequential Machines.* New York: Prentice-Hall, Inc.

17. Hopcroft, J. E. and Ullman, J. D. 1967. "A Survey of Formal Language Theory." *Proceedings of First Annual Princeton Conf. on Information Sciences and Systems.* Princeton, N. J. March, 1967. pp. 68-75.

18. Hopcroft, J. E. and Ullman, J. D. 1967. "An Approach to a Unified Theory of Automata." *Bell System Technical Journal* 46:1763-1829.

19. Hopcroft, J. E. and Ullman, J. D. 1969. *Formal Languages and Their Relation to Automata.* Menlo Park: Addison-Wesley. pp. 74-75.

20. Horgan, J., Roehrkasse, R. and Chiaraviglio, L. 1971. "Abstract Digital Computers and Turing Machines." (Forthcoming.)

21. Kalman, R. E., Falb, P. L., and Arbib, M. A. 1969. *Topics in Mathematical Systems Theory.* New York: McGraw-Hill. pp. 17-20; 163-233.

22. Krohn, K. B. and Rhodes, J. L. 1962. "Algebraic Theory of Machines." *Proceedings of the Symp. on Mathematical Theory of Automata.* Polytechnic Institute of Brooklyn, N.Y. April, 1962. pp. 341-384.

23. Kuroda, S. Y. 1964. "Classes of Languages and Linear Bounded Automata." *Information and Control* 7:207-223.

24. Nievergelt, J. 1965. "Partially Ordered Classes of Finite Automata." *IEEE Conf. Record of Sixth Annual Symp. on Switching and Automata Theory.* Ann Arbor, Mich., October, 1965. pp. 229-234.

25. Peters, S. 1968. "Mathematical Linguistics" in *Mathematics of the Decision Sciences,* part 2. Lectures in Applied Mathematics 12: 368.

26. Poore, J. H. Jr. 1970. "Toward an Algebra of Computation." Doctoral Thesis. School of Information and Computer Science, Georgia Institute of Technology.

27. Poore, J. H. Jr. and Chiaraviglio, L. 1971. "Abstract Digital Computers and Their Algebras." (Forthcoming.)

28. Rabin, M. O. and Scott, D. 1959. "Finite Automata and Their Decision Problems." *IBM Journal of Research and Development* 3:114-125.

29. Rose, G. F. 1970. "Abstract Families of Processors." *Journal of Computer and System Sciences* 4:193-204.

30. Stearns, R. E., Hartmanis, J. and Lewis, P. M. 1965. "Hierarchies of Memory Limited Computations." *IEEE Conf. Record of Sixth Annual Symp. on Switching and Automata Theory.* Ann Arbor, Mich., October, 1965. pp. 179-190.

VITA

Robert Charles Roehrkasse is a regular officer in the United States Air Force presently serving in the grade of captain. He was born in Waterloo, Illinois, on June 1, 1941. He was graduated from Red Bud High School, Red Bud, Illinois, in 1959. In 1965 he was graduated from Southern Illinois University with a Bachelor of Arts in Mathematics and was commissioned a second lieutenant in the Air Force. He was immediately assigned to the Georgia Institute of Technology to obtain a Master of Science in Information Science. Upon completion of the degree in 1966, he was assigned to the Armament Development and Test Center, Eglin Air Force Base, Florida. He served initially as Chief of the Computer Systems and Programs Branch and later as Chief of the Data Automation Division. He was subsequently selected under a special Air Force education program to pursue the Doctor of Philosophy in Information and Computer Science at the Georgia Institute of Technology beginning in June of 1969. Upon completion of this degree, he will report for duty as an instructor at the Air Force Academy in the Department of Astronautics and Computer Science.

Captain Roehrkasse is a Distinguished Military Graduate of the Air Force Reserve Officers Training Corps at Southern Illinois University. He has been a member of the national honorary mathematics fraternity *Pi Mu Epsilon*. During his assignment to the Armament Development and Test Center, he was awarded the Air Force Commendation Medal for his managerial effectiveness and had his program for cost reduction featured in an Air Force periodical. He currently has a publication pending in his area of specialization.