REAL-TIME MULTIPUSHDOWN AND MULTICOUNTER AUTOMATA NETWORKS AND HIERARCHIES

A THESIS

Presented to

The Faculty of the Division of Graduate Studies and Research

by

Lionel Earl Deimel, Jr.

In Partial Fulfillment

of the Requirements for the Degree

Doctor of Philosophy

in the School of Information and Computer Science

Georgia Institute of Technology

August, 1975

REAL-TIME MULTIPUSHDOWN AND MULTICOUNTER

AUTOMATA NETWORKS AND HIERARCHIES

Approved:

William I. Grosky, Chairman

John M. Gwynn, Jr.

Lucio Chiaraviglio

Date approved by Chairman: 8-6-75

For Betty, to whom I promise not to do this again

ACKNOWLEDGMENTS

I would like to express my sincerest appreciation to my thesis advisor, Professor William I Grosky, and to the other members of my guidance committee, Professors Lucio Chiaraviglio and John M. Gwynn, Jr., for their advice, assistance, encouragement, and friendship. A special note of thanks is due Dr. Gwynn, who worked so closely with me while Dr. Grosky was out of the country.

I would like to thank Professors Aubrey M. Bush and Edward B. Wagstaff of the School of Electrical Engineering and Dr. Stephen N. Cole of the Engineering Experiment Station for reading the thesis and serving as examiners.

The members of the faculty of the School of Information and Computer Science who have been so kind to me deserve recognition, as does Dr. David L. Morgan, formerly of the School of Mathematics, who helped with one of the proofs.

Finally, thanks go to my wife Betty and the Reverend Woodrow McKay, Jr. of the Presbyterian Student Center for help, encouragement, and support of a non-academic nature.

TABLE OF CONTENTS

SUMMARY

A new formulation is developed for the real-time multistore pushdown automaton. The automaton is redefined as a *network* of automata, each member of which is a machine with its own finite control and pushdown store, a connection to the input head, and possibly connections to other members of the network.

The rationale behind this formulation of multistore machines is to distinguish or classify such automata according to some measure of internal complexity. It is asserted that the manner in which stores are used relative to one another reflects some kind of complexity and that this complexity can be measured by classifying the connections in the appropriate network. An "appropriate network" is a network accepting the same language as the multistore machine. It is shown that such a network may always be found and that the network formulation and conventional formulation of such machines are, in fact, equivalent. It is further shown that the network formulation is more than adequate in the sense that connections between all pairs of machines in a network are never required. Languages accepted by deterministic machines may be accepted by deterministic networks having circular or ring connections, with as few as n connections for an n-store network. Languages accepted by nondeterministic machines may be accepted by nondeterministic networks having no unconnected machines, with as few as n-1 connections for an n-store network. Real-time multipushdown automata networks are related to infinite acceptance hierarchies shown elsewhere by Aanderaa, Burkhard and

Varaiya, and Liu and Weiner.

Automata networks are used to examine in greater detail the real-time multicounter machines, which are restricted multipushdown machines. Five infinite hierarchies are exhibited, three of them new. Within each hierarchy, the class is determined by the number of stores. The hierarchies previously known are those for deterministic real-time counter networks shown for counter machines by Fischer, Meyer, and Rosenberg; and for nondeterministic real-time counter machines shown by Kain. The remaining hierarchies are for unconnected deterministic and nondeterministic networks and for linearly-connected deterministic networks. The hierarchies are shown to relate to one another in a non-trivial way.

It is suggested that the network approach to the study of multi-store automata is capable of leading to results not otherwise apparent. Further, the technique may prove more useful to the study of parallel computation than the usual polyautomata models. Future research is suggested.

CHAPTER I

INTRODUCTION

We wish to look at an area of automata theory where, we believe, the conventional formulation of automata has failed to stimulate certain interesting questions. In particular, we will examine multistore pushdown automata operating in real time. Surveying the literature suggests these machines have received less attention than they deserve. This appears to be the case because most of the "obvious" questions concerning these devices have been answered. We will present a different, but equivalent, treatment of these devices and use this formulation to derive what we believe to be significant new results. Some of these relate to general multipushdown automata, others concern a subclass of these machines, the multicounter automata.

Historical Background

The pushdown automaton (pda) is a model of a computing device which has played a major role in automata and formal language theory. Informally, a pda is a language acceptor consisting of a one-way, read-only input tape containing symbols from some alphabet Σ; a finite control represented by a state set K and transition function δ; and a pushdown store, or last-in-first-out memory, which stores symbols from the pushdown store alphabet Γ. The device is said to accept (or recognize) a string of symbols over the input alphabet, that is, a word $w \in \Sigma^*$, if reading the entire word from the input tape can drive the machine from its initial configuration to an accepting configuration. (A^*, where A is a finite alphabet set, denotes the set of all finite length strings of symbols of A. A^* includes the empty word, the string consisting of no symbols, which is denoted by ϵ.) The initial configuration usually is defined as the status of the automaton when scanning the first symbol (if any) of the input word while in some designated initial state $q_0 \in K$ with a designated initial store symbol $Z_0 \in \Gamma$ on the store. An accepting configuration is commonly defined in one of two fundamental ways — either the device enters one of a set of states designated as "final" or empties its store upon reading the final symbol of w [30].

The pushdown store predates the pushdown automaton. It may be traced back at least as far as the 1954 paper by Burks, Warren, and Wright describing the theory and operation of the Burroughs Truth Function Evaluator, a machine designed to evaluate logical expressions in parenthesis-free notation [10]. The "Register" of this Burroughs machine is essentially a pushdown store [46]. The concept is treated more explicitly in [44], where Newell and Shaw discuss pushdown store manipulation in the context of more general list-processing techniques. (See also [45].) Samelson and Bauer [51] and Oettinger [46] examine the pushdown memory in connection with syntactic analysis and translation. In the early 1960's, what had been a useful, though somewhat ad hoc programming technique was formulated into a mathematical model of a computing device analogous to the finite automata developed by Rabin and Scott [48]. This is clearly seen in the paper by Schützenberger [54], where certain relations between pda's and unambiguous context-free languages are developed. It was Chomsky [11] and Evey [15], however, who established independently the essential connection between the abstract computing device and formal languages, namely, that the set of languages acceptable by pda's is exactly the set of context-free languages.

There exists an extensive literature concerning pda's and their variants. In the current literature, the finite control is taken, in general, to be nondeterministic. (In a deterministic machine, the transition function uniquely specifies the actions to be taken; in a nondeterministic machine, the transition function specifies only a set from which valid transitions may be selected.) The Chomsky and Evey result applies to nondeterministic pda's. Early studies of deterministic pda's are those by Fischer [16], Schützenberger [54], Haines [26], and particularly that by Ginsburg and Greibach [20]. This latter paper establishes many properties of the deterministic languages (those languages accepted by deterministic pda's) such as the fact that every deterministic language is unambiguous (has an essentially unique parsing in an appropriate context-free grammar). These and other fundamental results are collected in Ginsburg's important book on context-free languages [19]. In [37], Knuth provides a class of grammars, the LR(k) grammars, which generate exactly the deterministic languages.

Pushdown automata have been studied under a number of modifications or restrictions and under combinations of such modifications and restrictions. Besides being either deterministic or nondeterministic, we may classify the input tape as one-way or two-way. "One-way" means the input head moves from left to right on the tape and never reverses itself. "Two-way" indicates the input head may move left or right. Two-way pushdown automata were first studied by Gray, Harrison, and Ibarra [24].

Harrison and Ibarra have also studied pda's with multiple input tapes and with more than one head on each input tape [29]. Other studies involving two-way pda's and multiple input heads have been done by Ibarra [32], Gwynn and Martin [25], and Martin [42]. Additional variations include reversal-bounded pda's, whose stores can alternate between lengthening and shortening only a limited number of times [4, 8, 23], and tabulator machines, which can erase many symbols from a store at once [12]. One pda variation has obtained independent status and no longer is referred to as a pushdown automaton. This is the stack automaton, first studied by Ginsburg, Greibach, and Harrison [21]. It is a pda which can examine, but not alter, the interior of its store. The model was developed to be more representative of the process of compilation of computer languages, which are not strictly context-free and which therefore cannot be accepted by any pda. Stack automata claim an extensive literature of their own.

Many early articles about pushdown automata are concerned with the acceptance sets of various models, the relation of these sets to one another, and the relation of these sets to formal grammars. Several variations have been developed as automata-theoretic analogues of developments in the theory of formal languages. Thus, for example, Rovan relates bounded pda's to bounded languages [50], and Ibarra relates controlled pushdown automata to matrix languages [31]. Other papers deal with the so-called simple deterministic languages [36], and the strict deterministic languages [27,28].

Many recent articles are concerned largely with questions of computational complexity. Such studies seek to identify resources, amount of computing time and storage, required by particular models to accept particular sets of languages. Some pda variations have appeared here as well. Cook introduces the auxiliary pushdown automaton, which has a number of work tapes in addition to its pushdown store [13]. Kameda [35] studies the counter-pushdown acceptor, which is essentially similar but which substitutes counters for auxiliary work tapes (see below). Other complexity studies involving the pda and its variants may be found in [2], [3], [14], and [33].

An important automaton which should be mentioned is the counter automaton or counter machine. Although this device was at first developed independently of the pda, it may also be viewed as a restricted version of it. The auxiliary store is called a counter and can hold any non-negative integer. The finite control, although it can increase or decrease the integer stored, can only test whether that integer is zero or positive. This machine is equivalent to a pushdown acceptor whose store alphabet consists of a single symbol. (If we wish to retain the initial store symbol as an end-marker of the store, we allow a single additional symbol.) Minsky [43] shows that a machine with two counters can simulate a Turing machine and is therefore, in some sense, uninteresting. Each counter in his construction stores an encoding of half the Turing machine tape. Schützenberger [53] is usually credited with establishing the counter machine as an object for serious independent study. Many important characteristics of these machines and their languages are established by Fischer, Meyer, and Rosenberg [18]. These and related results are developed by Kain [34]. Very recently, stack automata with one-symbol stack alphabets (stack-counter machines) have been studied [6,22].

The primary machines we wish to study here are multipushdown acceptors (multistore pda's), pda's with finitely many pushdown stores. We will also deal with the natural restriction of such machines, the multicounter acceptors (multicounter machines). Of course, Minsky's result establishes a machine with two or more counters as a Turing machine equivalent. Since a machine with two or more pushdown stores is even more general, it too has the computational power of a Turing machine. In either case, less powerful devices deserving of study may be obtained by requiring acceptance within time bounded by some function of the input length. Our main interest will be in machines operating in so-called real time. The length of the accepting computation is bounded by the length of the input. A machine under such a restriction must read exactly one symbol of the input for each transition it executes.

In general, real-time computation represents only one of many complexity classes for a given computing device. It is an intuitively appealing concept, however. In actual practice, we generally desire to complete computer calculations as quickly as possible. To say that we can do so in real time is to say we can generate results as rapidly as we can submit our input to the machine. Yamada seems to have been first to examine the concept of real-time computability [56]. He shows that certain recursive functions can always be found which cannot be computed in real time by an automaton, no matter how general its computing capabilities. (The operating rules of the machine are assumed to be recursively defined. His motion of computing a function, it should be noted, may be transformed easily into the notion of acceptance of words of a language.) Thus, not all computable functions are real-time computable. Rosenberg [49] shows the position in the classic hierarchy of the languages which can be accepted by deterministic on-line multitape Turing machines. (Such machines have a read-only input tape and a number of Turing work tapes.) The set of such real-time definable languages is, of course, a superset of the regular languages. It is also a proper subset of the context-sensitive languages (of the set of languages accepted by deterministic linear-bounded automata, in fact) and is incomparable to both the context-free languages and the deterministic context-free languages. Rabin's 1963 paper [47] raises the question of the role of auxiliary storage in real-time computation. He shows that a deterministic two-tape Turing machine operating in real time is strictly more powerful than a deterministic one-tape Turing machine operating in real time. The question of whether for $k > 1$ a $k+1$-tape machine is more powerful than a k-tape machine is known as Rabin's problem.

In [34], Kain makes four conjectures concerning languages accepted by real-time multipushdown machines. These conjectures address what seem to be the major questions concerning real-time multipushdown acceptors. They are:

(1) $NRTPD_n \subset DLBA$

(2) $DRTPD_n \subset DRTPD_{n+1}$

(3) $NRTPD_n \subset NRTPD_{n+1}$

(4) $DRTPD_n \subset NRTPD_n$

where $n \geq 1$ and DLBA, $NRTPD_k$, and $DRTPD_k$ are the sets of languages which are accepted by deterministic linear-bounded automata (dlba), nondeterministic real-time pushdown acceptors with k stores ($nrtpd_k$), and deterministic real-time pushdown acceptors with k stores ($drtpd_k$), respectively. (We will adhere to the convention of abbreviating a type of automaton in lower-case letters and representing the set of languages recognizable by the class of such machines by the corresponding upper-case letters.) Rabin's paper is ultimately the inspiration for all these conjectures. We are asking if any computing power is gained by adding pushdown stores to an automaton operating in real time. Conjecture (1) attempts to establish the position of the real-time multipushdown languages in the usual linguistic hierarchy. Notice that Rosenberg's result does not apply, as the automata involved are nondeterministic. Conjectures (2) and (3), the existence of infinite acceptance hierarchies based upon the number of pushdown stores, follow from the disposition of Rabin's problem. Conjecture (4) asserts that for a given number of pushdown stores, nondeterminism is strictly more powerful than determinism.

Book and Greibach [7] essentially settle conjecture (3) and in so doing partially resolve Rabin's problem. Their concern is with the "quasi-realtime languages," those languages accepted by nondeterministic on-line multitape Turing machines operating with finite delay. (An on-line machine operates with finite delay if it never makes more than t consecutive transitions without moving its input head, where t is some integer.) They show that every quasi-realtime language is accepted by a nondeterministic real-time (that is, $t = 0$) on-line multitape Turing acceptor, and thus, by replacing each Turing tape by two pushdown stores, is accepted by a nondeterministic real-time multipushdown acceptor.

([15] and [17] show that this replacement can be made without loss of time.) Book and Greibach also show that any quasi-realtime language may be accepted by a nondeterministic machine using one stack and one pushdown store or three pushdown stores. Thus, for $n \geq 3$, $NRTPD_n = NRTPD_{n+1}$. Conjecture (3) is shown to be incorrect, and Rabin's problem is settled for the nondeterministic case — no infinite hierarchy based upon the number of tapes available exists.

Conjecture (2) has recently been settled by Aanderaa [1]. He shows there is an infinite hierarchy in the deterministic case based upon the number of pushdown stores. This settles Rabin's problem for deterministic machines (the pushdown stores may be paired and used as Turing tapes) and distinguishes deterministic from nondeterministic real-time Turing or multipushdown machines.

The resolutions of conjectures (2) and (3) establish conjecture (4) as true, at least for $n > 2$. Conjecture (1) is open and appears to be a difficult question to resolve [5].

Goals and Results

Thus, we see that most of the obviously interesting questions relating to real-time multipushdown automata have been answered. Moreover, meaningful variants of the basic model are few. Reversal-bounded machines have received attention recently [8], but other standard pda variations cannot be applied to the conventional real-time multipushdown machine. For example, combining a two-way input head with real-time computation seems inappropriate, as utilization of the head-reversing capability means that words may be accepted without being read completely. We might suspect that some measure of internal complexity could differentiate recognition capabilities of various machines, but such measures have received little attention.

This last observation is one of wide applicability to automata theory. Machines other than finite automata have been distinguished primarily on the basis of their time and storage requirements rather than on the basis of any notion of their internal complexity. Internal complexity measures have been discussed or proposed from time to time, but such proposals have had little effect on the mainstream of research. We mention two rather similar suggestions. Shannon [55] discusses the tradeoff between the number of input symbols recognized by a universal Turing machine and the number of states of such a machine. In effect, he proposes the product of these two numbers as a measure of the complexity or efficiency of the machine. Schmitt [52] has offered a similar suggestion. He proposes a "state complexity" measure for Turing machines, the state complexity being the minimum number of states needed by any Turing machine to compute a given partial recursive function using a given input alphabet.

The multistore automaton, unlike a single-store device, exposes to view one aspect of its internal operation — namely, its utilization of its multiple stores. We may look at the use of each store in relation to the others. It seems reasonable to suggest that the cooperative use of two or more stores in such a way that they perform operations of which they are incapable alone reflects a greater machine complexity than the use of the same number of stores in isolation (in some appropriate sense). This idea will be pursued by formulating *automata networks* in which each store is operated by its own finite control. These controls all have access to the input head and possibly communication with one another. It is this communication, provided by "connections" within the network, that we wish to study. We will ask how the recognition power of a network is affected by the presence or absence of these connections.

In Chapter II, we define what we mean by multistore pushdown automata. Formal definitions are given for language acceptance, configurations of such machines, and so forth. Analogous definitions are supplied for multistore pushdown automata networks. We then show by means of constructions in theorems 1 and 2 that the network formulation is equivalent to the conventional formulation. Theorems 3 and 4 establish that no pushdown automata networks need have every machine in the network connected to every other machine. Circular or ring connections are adequate for deterministic machines; nondeterministic networks need only have no isolated machines in order to accept the same languages as the corresponding conventional machines. The remainder of the chapter deals specifically with real-time automata and automata networks. Several known hierarchies are related to the new network formualtion in theorems 5, 6, and 7.

Chapter III is devoted to real-time counter networks. Theorems 8 and 9 show counter networks to be equivalent to counter machines. Theorems 10 and 11 show that the connection structures adequate for multipushdown networks are likewise adequate for multicounter networks. Five infinite acceptance hierarchies are exhibited for real-time counter networks. Two of these have been shown previously by other authors (theorems 12 and 13); three are new (theorems 15, 16, and 18). The network connections involved in these theorems are either unrestricted, linear, or nonexistent. The remainder of the chapter is devoted to showing the relation of these acceptance hierarchies to one another (theorems 19-31) and to an analysis of the significance of connections within a network.

Chapter IV mentions some additional results and suggest areas for future research.

CHAPTER II

MULTIPUSHDOWN AUTOMATA AND AUTOMATA NETWORKS

Basic Definitions

We must begin with some formal definitions.

DEFINITION 1: An *n-store pushdown automaton* (we will use *acceptor* and *machine* as interchangeable with *automaton*), pd_n, is an n+6-tuple $(K,\Sigma,\Gamma_1,\Gamma_2,\ldots,\Gamma_n,\delta,q_0,Z_0,F)$, where

(1) K is a finite set of *states*,

(2) Σ is a finite set, the *input alphabet*,

(3) Γ_i, $1 \leq i \leq n$ are finite sets, the *pushdown store alphabets*,

(4) $q_o \in K$ is the *initial state*,

(5) $Z_0 \in \Gamma_i$, $1 \leq i \leq n$ is the *initial store symbol* which appears initially on each pushdown store,

(6) $F \subseteq K$ is a set of *final states*, and

(7) $\delta: K\text{x}(\Sigma \cup \{\varepsilon\})\text{x}\Gamma_1\text{x}\Gamma_2\text{x}\ldots\text{x}\Gamma_n \to 2^{K\text{x}\Gamma_1{}^*\text{x}\Gamma_2{}^*\text{x}\ldots\text{x}\Gamma_n{}^*}$ is the transition function.

DEFINITION 2: A pd_n is *deterministic* if both of the following are true:

(1) For any $q \in K$, $s \in \Sigma \cup \{\varepsilon\}$, and $Z_i \in \Gamma_i$, $1 \leq i \leq n$, $\delta(q,s,Z_1,Z_2,\ldots,Z_n)$ contains at most one element.

(2) For any $q \in K$ and $Z_i \in \Gamma_i$, $1 \leq i \leq n$, whenever $\delta(q,\varepsilon,Z_1,Z_2,\ldots,Z_n)$ is nonempty, $\delta(q,s,Z_1,Z_2,\ldots,Z_n)$ is empty for all $s \in \Sigma$.

A pd_n which is not deterministic is *nondeterministic*. Deterministic

pd_n's will be denoted by dpd_n. Where it is necessary to make explicit that a pd_n is nondeterministic, we will denote it by npd_n.

DEFINITION 3: A pd_n (npd_n, dpd_n) is a (*nondeterministic, deterministic*) *real-time n-store pushdown automaton*, $rtpd_n$, ($nrtpd_n$, $drtpd_n$), if for all $q \in K$ and $Z_i \in \Gamma_i$, $1 \leq i \leq n$, $\delta(q,\varepsilon,Z_1,Z_2,\ldots,Z_n)$ is empty.

DEFINITION 4: A *configuration* of a pd_n M is an n+1-tuple $(q,\gamma_1,\gamma_2,\ldots,\gamma_n)$, where $q \in K$ and $\gamma_i \in \Gamma_i^*$, $1 \leq i \leq n$.

DEFINITION 5: For some pd_n M, let $q,q' \in K$ and, for $1 \leq i \leq n$, let $Z_i \in \Gamma_i$ and α_i, $\gamma_i \in \Gamma_i^*$. For $s \in \Sigma \cup \{\varepsilon\}$, we write

$$s:(q,Z_1\gamma_1,Z_2\gamma_2,\ldots,Z_n\gamma_n) \vdash_M (q',\alpha_1\gamma_1,\alpha_2\gamma_2,\ldots,\alpha_n\gamma_n)$$

if and only if $(q',\alpha_1,\alpha_2,\ldots,\alpha_n) \in \delta(q,s,Z_1,Z_2,\ldots,Z_n)$. For $s_i \in \Sigma \cup \{\varepsilon\}$, $1 \leq i \leq m$, and configurations $C_j \in K \times \Gamma_1^* \times \Gamma_2^* \times \ldots \times \Gamma_n^*$, $0 \leq j \leq m$, we write

$$s_1 s_2 \ldots s_m\colon\ C_0 \vdash_M^m C_m$$

whenever $s_i\colon\ C_{i-1} \vdash_M C_i$ for each $1 \leq i \leq m$. By convention, we write

$$\varepsilon\colon\ C_k \vdash_M^0 C_k$$

for any configuration C_k of M. For configurations C and C' and for $w \in \Sigma^*$, we write

$$w\colon\ C \vdash_M^* C'$$

whenever $w\colon\ C \vdash_M^m C'$ for some $0 \leq m$. M may be omitted from $\vdash_M$, $\vdash_M^m$, and $\vdash_M^*$ when the machine is understood.

The interpretation of $s\colon\ (q,Z_1\gamma_1, Z_2\gamma_2,\ldots,Z_n\gamma_n) \vdash_M (q',\alpha_1\gamma_1, \alpha_2\gamma_2,\ldots,\alpha_n\gamma_n)$ is that M reads s while in state q with $Z_1,Z_2,\ldots,Z_n$ at the top of the n pushdown stores. M goes into state q' and replaces Z_1 with α_1, Z_2 with $\alpha_2,\ldots,Z_n$ with α_n. The relations $\vdash_M^m$ and $\vdash_M^*$ extend the relation $\vdash_M$ to input strings of length greater than 1.

DEFINITION 6: The *language* of pd_n M *accepted by final state*, denoted T(M), is defined as $T(M) = \{w \in \Sigma^* | w:(q_0,Z_0,Z_0,\ldots,Z_0) \overset{*}{\vdash_M} (q, \gamma_1,\gamma_2,\ldots,\gamma_n)$ for $q \in F$ and $\gamma_i \in \Gamma_i^*$, $1 \leq i \leq n\}$.

DEFINITION 7: The *language* of pd_n M *accepted by empty store*, denoted N(M), is defined as $N(M) = \{w \in \Sigma^* | w:(q_0,Z_0,Z_0,\ldots,Z_0) \overset{*}{\vdash_M} (q,\varepsilon,\varepsilon,\ldots,\varepsilon)$ for $q \in K\}$.

Treatment of the concept of acceptance is not uniform in the literature. Definitions 6 and 7 provide alternatives, but other definitions are possible. For example, we could accept a word if and only if it is in both T(M) and N(M), or perhaps if any *one* of the n stores empties. In fact, our choice of definition at this point is not critical, as it is easily shown that these definitions are equivalent. This is not true of all the automata to be studied. We will somewhat arbitrarily restrict consideration to final state acceptance, as this is certainly the most general acceptance criterion.

We now introduce our most important definition.

DEFINITION 8: An *n-store pushdown automata network*, pdn_n, is a 4n+3-tuple

$$(K_1,K_2,\ldots,K_n,\Sigma,\Gamma_1,\Gamma_2,\ldots,\Gamma_n,\delta_1,\delta_2,\ldots,\delta_n,q_0,Z_0,F_1,F_2,\ldots,F_n),$$

where

(1) K_i, $1 \leq i \leq n$, is the finite *state set* of the ith automaton,

(2) Σ is the finite *input alphabet*,

(3) Γ_i, $1 \leq i \leq n$, is the finite *pushdown store alphabet* of the ith automaton,

(4) $q_0 \in K_i$, $1 \leq i \leq n$, is the *initial state* of the ith automaton,

(5) $Z_0 \in \Gamma_i$, $1 \leq i \leq n$, is the *initial store symbol* of the ith automaton,

(6) $F_i \subseteq K_i$, $1 \le i \le n$, is the *final state set* of the ith automaton, and

(7) $\delta_i: K_i \times K_{D_i(1)} \times K_{D_i(2)} \times \dots \times K_{D_i(p_i)} \times \Sigma \times \Gamma_i \to 2^{K_i \times \Gamma_i^*}$, where $0 \le p_i \le n-1$ and $D_i:\{1,2,\dots,p_i\} \xrightarrow{1-1} \{1,2,\dots,i-1,i+1,\dots,n\}$, $1 \le i \le n$, is the *transition function* of the ith machine.

Definition 8 says that a pdn_n consists of n pushdown machines sharing a common one-way input head. The actions of each machine are governed by the input, top store symbol of its own store, its internal state, and perhaps the internal states of other machines in the network. Ths functions D_i, $1 \le i \le n$, specify the dependency relations (structure) of the network.

DEFINITION 9: Let M be a pdn_n as defined above. We will say that machine j *is connected to* machine i or machine i *depends upon* machine j if and only if $D_i(k) = j$ for some $1 \le k \le p_i$. We denote the fact that machine j is connected to machine i by C_{ji}.

The following definitions are analogous to definitions 2-7.

DEFINITION 10: A pdn_n is *deterministic* if both of the following are true:

(1) For any $q \in K_i$, $q_k \in K_{D_i(k)}$, $s \in \Sigma \cup \{\varepsilon\}$, and $Z_i \in \Gamma_i$, $1 \le k \le p_i$, $1 \le i \le n$, $\delta_i(q,q_1,q_2,\dots,q_{p_i},s,Z_i)$ contains at most one element, and

(2) For any $q \in K_i$, $q_k \in K_{D_i(k)}$, and $Z_i \in \Gamma_i$, $1 \le k \le p_i$, $1 \le i \le n$, whenever $\delta_i(q,q_1,q_2,\dots,q_{p_i},\varepsilon,Z_i)$ is nonempty, $\delta_i(q,q_1,q_2,\dots,q_{p_i},s,Z_i)$ is empty for all $s \in \Sigma$.

A pdn_n which is not deterministic is *nondeterministic*, written $npdn_n$. A deterministic pdn_n will be written $dpdn_n$.

DEFINITION 11: A pdn_n ($npdn_n$, $dpdn_n$) is a (*nondeterministic, deterministic*) *real-time n-store pushdown automata network*, $rtpdn_n$ ($nrtpdn_n$, $drtpdn_n$), if for all $q \in K_i$, $q_k \in K_{D_i(k)}$, and $Z_i \in \Gamma_i$, $1 \le i \le p_i$ $1 \le i \le n$, $\delta_i(q, q_1, q_2, \ldots, q_{p_i}, \varepsilon, Z_i)$ is empty.

DEFINITION 12: A *configuration* of a pdn_n M is a 2n-tuple $(q_1, q_2, \ldots, q_n, \gamma_1, \gamma_2, \ldots, \gamma_n)$, where $q_i \in K_i$ and $\gamma_i \in \Gamma_i^*$, $1 \le i \le n$.

DEFINITION 13: For some pdn_n M, let q_i, $q_i' \in K_i$, $Z_i \in \Gamma_i$, and $\alpha_i, \gamma_i \in \Gamma_i^*$, $1 \le i \le n$. For $s \in \Sigma \cup \{\varepsilon\}$, we write

$$s: (q_1, q_2, \ldots, q_n, Z_1\gamma_1, Z_2\gamma_2, \ldots, Z_n\gamma_n) \vdash_M (q_1', q_2', \ldots, q_n', \alpha_1\gamma_1, \alpha_2\gamma_2, \ldots, \alpha_n\gamma_n)$$

if and only if $(q_i', \alpha_i) \in \delta_i(q_i, q_{D_i(1)}, q_{D_i(2)}, \ldots, q_{D_i(p_i)}, s, Z_i)$ $1 \le i \le n$. for $s_i \in \Sigma \cup \{\varepsilon\}$, $1 \le i \le m$, and configurations $C_j \in K_1 \times K_2 \times \ldots \times K_n \times \Gamma_1^* \times \Gamma_2^* \times \ldots \times \Gamma_n^*$, $0 \le j \le m$, we write

$$s_1 s_2 \ldots s_m : C_0 \vdash_M^m C_m$$

whenever s_i: $C_{i-1} \vdash_M C_i$ for each $1 \le i \le m$. By convention, we write

$$\varepsilon: C_k \vdash_M^0 C_k$$

for any configuration C_k of M. For configurations C and C' and for $w \in \Sigma^*$, we write

$$w: C \vdash_M^* C'$$

whenever w: $C \vdash_M^m C'$ for some $0 \le m$. M may be omitted from $\vdash_M$, $\vdash_M^m$, and $\vdash_M^*$ when no ambiguity results.

Notice that the one-way input head advances if and only if no machine executes an ε-transition. The network is blocked from having some machines execute ε-transitions and other machines execute non-ε-

transitions. This may or may not seem reasonable formulation. It is, however, largely irrelevant, as we will be concerned mostly with real-time networks, for which no ε-transitions are allowed.

DEFINITION 14: The *language* of pdn_n M *accepted by final state*, denoted T(M), is defined as $T(M) = \{w \in \Sigma^* | w: (q_0, q_0, \ldots, q_0, Z_0, Z_0, \ldots, Z_0) \vdash_M^* (q_1, q_2, \ldots, q_n, \gamma_1, \gamma_2, \ldots, \gamma_n)$ for $q_i \in F_i$ and $\gamma_i \in \Gamma_i^*$, $1 \le i \le n\}$.

DEFINITION 15: The *language* of pdn_n M *accepted by empty store*, denoted N(M), is defined as $N(M) = \{w \in \Sigma^* | w: (q_0, q_0, \ldots, q_0, Z_0, Z_0, \ldots, Z_0) \vdash_M^* (q_1, q_2, \ldots, q_n, \varepsilon, \varepsilon, \ldots, \varepsilon)$ for $q_i \in K_i$, $1 \le i \le n\}$.

Again, we may conceive of other acceptance definitions. For example, any machine's being in a final state could result in acceptance by the network. We will ignore such possibilities, however.

The Equivalence of Pushdown Automata and Pushdown Automata Networks

We now wish to show that multipushdown automata and multipushdown automata networks are, in fact, equivalent. Once we have shown this, we may study the latter in lieu of the former. The motivation for doing so is straightforward. The acceptance properties of real-time multipushdown acceptors are mostly known. Although these properties are of interest, they do not in any way reflect the internal complexity of the machines involved. In particular, the information placed on the stores may be used by the automaton in rather different ways. Intuitively, it appears that certain operations, such as shifting information from one pushdown store to another in order to simulate a Turing machine tape, are more sophisticated or complex than other operations, such as comparing the contents of a pushdown store with a subword of the input string. By reformulating the multipushdown automaton as a network of machines, we hope to isolate and systematically study information flow in the automaton, which we might expect to be a valid index of internal complexity.

We first show that any language accepted by a pdn_n is accepted by some pd_n without loss of time. This is done by constructing a pd_n from the pdn_n. The pushdown machine manipulates the stores exactly as the pushdown network. Its finite control "knows" the transition function of each machine in the network and, by "remembering" the state of each machine, can simulate the behavior of the network.

THEOREM 1: Given any pdn_n M. There is a pd_n M' such that T(M) = T(M'), N(M) = N(M'), and every word accepted by M in m transitions is accepted by M' in m transitions. If M is deterministic, so is M'.

PROOF: Let $M = (K_1,K_2,\ldots,K_n,\Sigma,\Gamma_1,\Gamma_2,\ldots,\Gamma_n,\delta_1,\delta_2,\ldots,\delta_n,q_0,Z_0,$
$F_1,F_2,\ldots,F_n)$.

Let $M' = (K,\Sigma,\Gamma_1,\ldots,\Gamma_n,\delta,q_0',Z_0,F)$, where

$$K = K_1 \times K_2 \times \ldots \times K_n,$$

$$q_0' = (q_0,q_0,\ldots,q_0),$$

$$F = \{(q_1,q_2,\ldots,q_n) \in K | q_i \in F_i,\ 1 \le i \le n\},$$

and $((q_1',q_2',\ldots,q_n'),\alpha_1,\alpha_2,\ldots,\alpha_n) \in \delta((q_1,q_2,\ldots,q_n),s,Z_1,Z_2,\ldots,Z_n)$ if and only if $(q_i',\alpha_i) \in \delta_i(q_i,q_{D_i(1)},q_{D_i(2)},\ldots,q_{D_i(p_i)},s,Z_i)$

for $q_i,q_i' \in K_i$, $Z_i \in \Gamma_i$ and $\alpha_i \in \Gamma_i^*$, $1 \le i \le n$. We assert that M' is a pd_n with the properties indicated in the statement of the theorem.

Let $C = (q_1,q_2,\ldots,q_n,\gamma_1,\gamma_2,\ldots,\gamma_n)$ be a configuration of M and let $C' = ((p_1,p_2,\ldots,p_n),\beta_1,\beta_2,\ldots,\beta_n)$ be a configuration of M'. We will say that C and C' are *corresponding configurations* if and only if $q_i = p_i$ and $\gamma_i = \beta_i$ for all $1 \leq i \leq n$. Notice that if C and C' are corresponding configurations, M is in a final state if and only if M' is, by the definition of F, the final state set of M'. Also, M has all stores empty if and only if M' has all stores empty.

Suppose word w is accepted by M. Let $|w| = h$, where $|w|$ indicates the length of w, that is, the number of symbols in the word w. If C_0 is the initial configuration of M, we must have $w\colon C_0 \vdash_M^m C_a$, $h \leq m$, for some accepting configuration C_a of M. (An accepting configuration is a configuration in which the machine accepts a word.) Notice that the possibility that $h < m$ exists, as the machine may make ε-transitions, that is, may make transitions without reading input symbols. We may insert ε's (empty words) where appropriate and note that we have $s_1 s_2 \ldots s_m$: $C_0 \vdash_M^m C_a$, where $s_i \in \Sigma \cup \{\varepsilon\}$, $1 \leq i \leq m$.

We will show by induction that if w is accepted by M, w is accepted (by the same criterion) by M'. We do so by showing that M' achieves configurations corresponding to those achieved by M. Let $C_0 = (q_0, q_0, \ldots, q_0, Z_0, Z_0, \ldots, Z_0)$ be the initial configuration of M. The initial configuration of M' is $C_0' = ((q_0, q_0, \ldots, q_0), Z_0, Z_0, \ldots, Z_0)$, which is, by definition, the corresponding configuration of C_0. Now assume that for $0 \leq k < m$, $s_1 s_2 \ldots s_k$: $C_0 \vdash_M^k C_b$ and $s_1 s_2 \ldots s_k$: $C_0' \vdash_M^k C_b'$, where C_b and C_b' are corresponding configurations. In particular, let $C_b = (q_1, q_2, \ldots, q_n, Z_1\gamma_1, Z_2\gamma_2, \ldots, Z_n\gamma_n)$ and $C_b' = ((q_1, q_2, \ldots, q_n), Z_1\gamma_1, Z_2\gamma_2, \ldots, Z_n\gamma_n)$. Let s_{k+1}: $C_b \vdash_M C_c$, where $C_c = (q_1', q_2', \ldots, q_n', \alpha_1\gamma_1, \alpha_2\gamma_2, \ldots, \alpha_n\gamma_n)$. From definition 13, we have for all $1 \leq i \leq n$ $(q_i', \alpha_i) \in \delta_i(q_i, q_{D_i(1)}, q_{D_i(2)}, \ldots, q_{D_i(p_i)}, s_{k+1}, Z_i)$. From the construction of M', we must have

$((q_1', q_2', \ldots, q_n'), \alpha_1, \alpha_2, \ldots, \alpha_n) \in \delta((q_1, q_2, \ldots, q_n), s_{k+1}, Z_1, Z_2, \ldots, Z_n)$. Thus, by definition 5, s_{k+1}: $C_b' \vdash_{M'} ((q_1', q_2', \ldots, q_n'), \alpha_1\gamma_1, \alpha_2\gamma_2, \ldots, \alpha_n\gamma_n)$. But $((q_1', q_2', \ldots, q_n'), \alpha_1\gamma_1, \alpha_2\gamma_2, \ldots, \alpha_n\gamma_n) = C_c'$, the configuration corresponding to C_c. Hence, M and M' achieve corresponding configurations in the same amount of time and, since acceptance by final state or empty stores occurs in corresponding configurations, $T(M') \subseteq T(M)$ and $N(M') \subseteq N(M)$. We may use a similar argument to show that $T(M) \subseteq T(M')$ and $N(M) \subseteq N(M')$. Again, initial configurations correspond. If $s_1 s_2 \ldots s_k$: $C_0' \vdash^k_{M'} C_b'$ and $s_1 s_2 \ldots s_k$: $C_0 \vdash^k_M C_b$, where C_b' and C_b are corresponding configurations, and s_{k+1}: $C_b' \vdash_{M'} C_c'$, where the meaning of symbols is as above, then we must have $(q_i', \alpha_i) \in \delta_i(q_i, q_{D_i(1)}, q_{D_i(2)}, \ldots, q_{D_i(p_i)}, s_{k+1}, Z_i)$ for all $1 \le i \le n$, from the definition of M'. But this means s_{k+1}: $C_b \vdash_M C_c$, and, by the same reasoning as above, we must have $T(M) \subseteq T(M')$ and $N(M) \subseteq N(M')$. Taken with the previous result gives us $T(M) = T(M')$ and $N(M) = N(M')$.

Finally, assume M is deterministic. Since, for M', $((q_1', q_2', \ldots, q_n'), \alpha_1, \alpha_2, \ldots, \alpha_n) \in \delta((q_1, q_2, \ldots, q_n), s, Z_1, Z_2, \ldots, Z_n)$ if and only if, for M, $(q_i', \alpha_i) \in \delta_i(q_i, q_{D_i(1)}, q_{D_i(2)}, \ldots, q_{D_i(p_i)}, s, Z_i)$ for all $1 \le i \le n$, it is clear that $\delta((q_1, q_2, \ldots, q_n), s, Z_1, Z_2, \ldots, Z_n)$ contains at most one element if each $\delta_i(q_i, q_{D_i(1)}, q_{D_i(2)}, \ldots, q_{D_i(p_i)}, s, Z_i)$ contains at most one element. Also, if $\delta((q_1, q_2, \ldots, q_n), \varepsilon, Z_1, Z_2, \ldots, Z_n)$ is non-empty, then each $\delta_i(q_i, q_{D_i(1)}, q_{D_i(2)}, \ldots, q_{D_i(p_i)}, \varepsilon, Z_i)$ is non-empty. But since M is deterministic, for any $s \in \Sigma$, $\delta_i(q_i, q_{D_i(1)}, q_{D_i(2)}, \ldots, q_{D_i(p_i)}, s, Z_i)$ must be empty. From the definition of M', each $\delta((q_1, q_2, \ldots, q_n), s, Z_1, Z_2, \ldots, Z_n)$ is empty. By definition 2, therefore, M' is deterministic.

Q.E.D.

We complete the proof of the equivalence of the usual formulation and the network formulation by showing that a pdn_n may be found to accept the language of any pdn_n without loss of time. The necessary simulation in this case is a bit more complex than that of the previous theorem. The chief difficulty is that a given machine in the network can "know" the state of every other machine in the network but cannot "know" the top store symbol of the other pushdown stores. In general, however, the operation to be performed on any store of a pd_n depends in part on the top symbols of the other stores. This difficulty is overcome by incorporating the *logical* top store symbol from each store into the finite control to be associated with that store. By maintaining information about the top of each store in the finite controls, each machine of the network has access to enough information to simulate the finite control and one pushdown store of the pd_n.

In order to simplify certain parts of the proof, we introduce functions P (for *prefix*) and S (for *suffix*). If w is some string of symbols, P(w) is the first symbol of w, and S(w) is the string remaining when the prefix is removed. For example, if w = abc, P(w) = a and S(w) = bc. The following lemma provides formal definitions for these functions and establishes some of their properties. These properties are, in fact, intuitively obvious.

LEMMA 1: Let set A be a finite alphabet for $w \in A^*$, define P(w) and S(w) as follows:

If $w = \varepsilon$, then $P(w) = S(w) = \varepsilon$. If $w = ax$, where $a \in A$ and $x \in A^*$ then $P(w) = a$ and $S(w) = x$.

The following properties are true:

(a) $w = P(w)S(w)$ for $w \in A^*$.

(b) $P(P(w)) = P(w)$ for $w \in A^*$.

(c) $S(P(w)) = \varepsilon$ for $w \in A^*$.

(d) $P(wx) = P(w)$ for $w,x \in A^*$ and $|w| \geq 1$.

(e) $S(wx) = S(w)x$ for $w,x \in A^*$ and $|w| \geq 1$.

PROOF: (a) This follows directly from the definition. If $w = \varepsilon$, $P(w)S(w) = \varepsilon\varepsilon = \varepsilon$. If $w = ax$, for some $a \in A$ and $x \in A^*$, $P(w)S(w) = ax = w$.

(b) Either $w = \varepsilon$ or $w = ax$ for some $a \in A$ and $x \in A^*$. Suppose $w = \varepsilon$. We have

$P(P(w))$	$= P(P(\varepsilon))$	Substitution
	$= P(\varepsilon)$	Definition
	$= P(w)$	Substitution

If $w = ax$, we have

$P(P(w))$	$= P(P(ax))$	Substitution
	$= P(a)$	Definition
	$= P(ax)$	Definition
	$= P(w)$	Substitution

(c) If $w = \varepsilon$, we see that $S(P(w)) = \varepsilon$ from the definition. If $w = ax$ for some $a \in A$ and $x \in A^*$, we have

$S(P(w))$	$= S(P(ax))$	Substitution
	$= S(a)$	Definition
	$= S(a\varepsilon)$	Definition of ε
	$= \varepsilon$	Definition

(d) Let $w = ay$, where $a \in A$ and $y \in A^*$.

$P(wx)$	$= P(ayx)$	Substitution
	$= P(az)$	Let $z = yx$
	$= a$	Definition
	$= P(ay)$	Definition
	$= P(w)$	Substitution

(e) Defining w as in (d), we have

$S(wx)$	$= S(ayx)$	Substitution
	$= S(az)$	Let $z = yx$
	$= z$	Definition
	$= yx$	Substitution
	$= S(ay)x$	Definition
	$= S(w)x$	Substitution

Q.E.D.

THEOREM 2: Given any pd_n M. There is a pdn_n M' such that $T(M) = T(M')$, $N(M) = N(M')$, and every word accepted by M in m transitions is accepted by M' in m transitions. If M is deterministic, so is M'.

PROOF: Let $M = (K,\Sigma,\Gamma_1,\Gamma_2,\ldots,\Gamma_n,\delta,q_0,Z_0,F)$.

Let $M' = (K \times \Gamma_1', K \times \Gamma_2',\ldots,K \times \Gamma_n',\Sigma,\Gamma_1',\Gamma_2',\ldots,\Gamma_n',\delta_1,\delta_2,\ldots,\delta_n,$
$(q_0,Z_0),W,F_1,F_2,\ldots,F_n)$, where

$\Gamma_i' = \Gamma_i \cup \{W\}$, $W \notin \Gamma_i$, $1 \leq i \leq n$,

$F_i = \{(q,Z) \mid q \in F \text{ and } Z \in \Gamma_i'\}$, $1 \leq i \leq n$,

and $(q',\alpha_1,\alpha_2,\ldots,\alpha_n) \in \delta(q,s,Z_1,Z_2,\ldots,Z_n)$ if and only if $((q',P(\alpha_i X)),$ $S(\alpha_i X)) \in \delta_i((q,Z_i),(q,Z_1),(q,Z_2),\ldots,(q,Z_{i-1}),(q,Z_{i+1}),\ldots,(q,Z_n),s,X)$ for all $1 \leq i \leq n$ and $X \in \Gamma_i'$. We assert that M' is a pdn_n with the properties desired.

As in the previous theorem, we wish to define corresponding configurations of M and M'. Let $C = (q,\gamma_1,\gamma_2,\ldots,\gamma_n)$ be a configuration of M and let $C' = ((p_1,Y_1),(p_2,Y_2),\ldots,(p_n,Y_n),\alpha_1,\alpha_2,\ldots,\alpha_n)$ be a configuration of M'. We will say that C and C' are *corresponding configurations* if and only if $p_i = q$, $Y_i = P(\gamma_i W)$ and $\alpha_i = S(\gamma_i W)$, $1 \leq i \leq n$. From the definition of M', it is clear that in corresponding configurations, M is in a final state if and only if M' is. From the definition of corresponding configurations, it is clear also that in corresponding configurations, M has all empty stores if and only if M' does. (Note that $S(\varepsilon W) = S(W) = \varepsilon$.)

To show that M and M' accept the same languages, we again must do an induction on the number of transitions. This time we will do a single induction noting that we go from step to step using biconditionals, so that the proof could proceed forward or backward. The initial configurations of M and M' are $C_0 = (q_0,Z_0,Z_0,\ldots,Z_0)$ and $C_0' = ((q_0,Z_0),(q_0,Z_0),\ldots,(q_0,Z_0),W,W,\ldots,W)$, respectively. By inspection, we see that these are corresponding configurations. Now assume that for $0 \leq k < m$, $s_1s_2\ldots s_k\colon C_0 \overset{k}{\underset{M}{\vdash}} C_b$ and $s_1s_2\ldots s_k\colon C_0' \overset{k}{\underset{M'}{\vdash}} C_b'$, where C_b and C_b' are corresponding configurations. In particular, let $C_b = (q,\gamma_1,\gamma_2,\ldots,\gamma_n)$ and $C_b' = ((q,P(\gamma_1 W)), (q,P(\gamma_2 W)),\ldots,(q,P(\gamma_n W)),S(\gamma_1 W),S(\gamma_2 W),\ldots,S(\gamma_n W))$. Suppose $s_{k+1}\colon C_b \underset{M}{\vdash} (q',\alpha_1 S(\gamma_1),\alpha_2 S(\gamma_2),\ldots,\alpha_n S(\gamma_n)) = C_c$, where $\alpha_i \in \Gamma_i^*$. By definition 5, it is clear that this is true if and only if $(q',\alpha_1,\alpha_2,\ldots,\alpha_n) \in \delta(q,s_{k+1},P(\gamma_1),P(\gamma_2),\ldots,P(\gamma_n))$. From the definition of M', we see this is the case if and only if, for all $1 \leq i \leq n$ and $X \in \Gamma_i'$, $((q',P(\alpha_i X)),S(\alpha_i X)) \in \delta_i((q,P(\gamma_i)),(q,P(\gamma_1)),(q,P(\gamma_2)),\ldots,(q,P(\gamma_{i-1})),(q,P(\gamma_{i+1})),\ldots,(q,P(\gamma_n)),s_{k+1},X)$. Now, for the transition represented by s_{k+1}: $C_b \vdash_M C_c$ to take place, each γ_i must be other than ε, that is $|\gamma_i| \geq 1$. By lemma 1d, then, we may write C_b' as $((q,P(\gamma_1)),(q,P(\gamma_2)),\ldots,(q,P(\gamma_n)),S(\gamma_1 W),S(\gamma_2 W),\ldots,S(\gamma_n W))$. Thus, in this configuration, the ith store has $P(S(\gamma_i W))$ as its top symbol. Reading s_{k+1} therefore causes the ith machine to execute the transition $((q',P(\alpha_i P(S(\gamma_i W)))),S(\alpha_i P(S(\gamma_i W)))) \in \delta_i((q,P(\gamma_i)),(q,P(\gamma_1)),(q,P(\gamma_2)),\ldots,$ $(q,P(\gamma_{i-1})),(q,P(\gamma_{i+1})),\ldots,(q,P(\gamma_n)),s_{k+1},P(S(\gamma_i W)))$. Thus, we have s_{k+1}: $C_b' \vdash_{M'} ((q',P(\alpha_1 P(S(\gamma_1 W)))),(q',P(\alpha_2 P(S(\gamma_2 W)))),\ldots,(q',P(\alpha_n P(S(\gamma_n W)))),$ $S(\alpha_1 P(S(\gamma_1 W)))S(S(\gamma_1 W)),S(\alpha_2 P(S(\gamma_2 W)))S(S(\gamma_2 W)),\ldots,S(\alpha_n P(S(\gamma_n W)))S(S(\gamma_n W)))=$ C_c'. We will show that C_c and C_c' are corresponding configurations. To do so, we must show that $P(\alpha_i P(S(\gamma_i W))) = P(\alpha_i S(\gamma_i)W)$ and that $S(\alpha_i P(S(\gamma_i W)))S(S(\gamma_i W)) = S(\alpha_i S(\gamma_i)W)$ for all $1 \leq i \leq n$. Either $\alpha_i = \varepsilon$ or $|\alpha_i| \geq 1$. In the first case, we have

$P(\alpha_i P(S(\gamma_i W)))$	$= P(P(S(\gamma_i W)))$	Substitution
	$= P(S(\gamma_i W))$	Lemma 1b
	$= P(S(\gamma_i)W)$	Lemma 1e
	$= P(\varepsilon S(\gamma_i)W)$	Definition of ε
	$= P(\alpha_i S(\gamma_i)W)$	Substitution

and $S(\alpha_i P(S(\gamma_i W)))S(S(\gamma_i W))$	$= S(P(S(\gamma_i W)))S(S(\gamma_i W))$	Substitution
	$= S(P(S(\gamma_i)W))S(S(\gamma_i)W)$	Lemma 1e
	$= S(S(\gamma_i)W)$	Lemma 1c
	$= S(\alpha_i S(\gamma_i)W)$	Definition of ε

In the second case, we may write

$P(\alpha_i P(S(\gamma_i W)))$	$= P(\alpha_i)$	Lemma 1d
	$= P(\alpha_i S(\gamma_i)W)$	Lemma 1d

and $S(\alpha_i P(S(\gamma_i W)) S(S(\gamma_i W)) = S(\alpha_i) P(S(\gamma_i W)) S(S(\gamma_i W))$ Lemma 1e

$= S(\alpha_i) S(\gamma_i W)$ Lemma 1a

$= S(\alpha_i S(\gamma_i W))$ Lemma 1e

$= S(\alpha_i S(\gamma_i) W)$ Lemma 1e

Thus C_c and C_c' are corresponding configurations, and M accepts in m transitions if and only if M' accepts in m transitions. An argument similar to that in Theorem 1 shows that M' is deterministic if M is deterministic.

Q.E.D.

Adequate Structures for Pushdown Automata Networks

Having established the equivalence of the network formulation of multipushdown automata and the usual single-control formulation, we will restrict our attention to the latter. We now wish to consider specific types of connections within a network. In theorem 1, we assumed each machine in the network depended upon every other machine in the network. In theorem 2, the network constructed to simulate the actions of the multipushdown acceptor also has every machine connected to every other machine. Can simpler networks be equally powerful? The next theorem establishes the answer to this question to be "yes" for $dpdn_n$'s.

First, we introduce a definition.

DEFINITION 16: Let M be a pdn_n, $n \geq 2$. We will say M has a *ring structure* provided there exists a function $f: \{1,2,\ldots,n\} \xrightarrow[\text{onto}]{1\text{-}1} \{1,2,\ldots,n\}$ such that:

(1) $p_i = 1$ for $1 \leq i \leq n$,

(2) $D_{f(j)}(1) = f(j+1)$ for $1 \leq j \leq n-1$, and

(3) $D_{f(n)}(1) = f(1)$.

We will say for completeness sake that any $dpdn_1$ also has a ring structure.

THEOREM 3: Let M be a $dpdn_n$. There exists a $dpdn_n$ M' with a ring of structure such that T(M) = T(M'), N(M) = N(M'), and every word accepted by M in m transitions is accepted by M' in m transitions.

PROOF: Our proof will be somewhat less formal than previous proofs. We will rely upon the basic techniques used in the proofs of theorems 1 and 2.

Before showing how to construct M', we should examine what this theorem says. It asserts that any language accepted by a $dpdn_n$ can be accepted by a deterministic network whose machines are connected in a ring or circle — each machine is connected only to one other machine. In effect, information may flow around the network in either a clockwise or counterclockwise direction, but not both.

We will construct a network M' for which $C_{12}, C_{23}, \ldots, C_{(n-1)(n)}, C_{n1}$. (Machine 1 is connected to machine 2 and so forth.) We assume $n \geq 2$, as the theorem is trivially true for $n = 1$. The ith machine of M' will simulate the ith machine of M. In the finite control of the ith machine is coded the following information:

(1) The state of each machine of M and

(2) The top $1 + ((n+j) \bmod n)$ symbols of the pushdown store of machine $1 + ((i+j) \bmod n)$ of network M, $-1 \leq j \leq n-2$.

For example, if $n = 4$, machine 3 of M' has the following information represented in its state:

(1) The states of machines 1,2,3, and 4 of M,

(2) The top 2 symbols of store 1 of M,

(3) The top 3 symbols of store 2 of M,

(4) The top 4 symbols of store 3 of M, and

(5) The top symbol of store 4 of M.

The initial store symbol of each machine of M' is W. Whenever a control is "remembering" the top k symbols of a store which contains fewer than k symbols, the remaining otherwise unspecified symbols are represented by W's. (This technique is merely an extension of that used in theorem 2.)

In the first transition,from the configuration of M', it is clear that the control of each machine of the network can properly adjust the coding of the state and pushdown store of the corresponding machine of M. (The transition function of machine i of M can be incorporated into its transition function, the states of all machines of M are "known," the pushdown store of machine i is simulated in the control and the pushdown store, and the input symbol is known to all machines of the network.) By a similar argument, it is clear that the encoding of the states of machines of M can also be updated, as can the pushdown store segments. (Note that if the bottom symbol of a store of M is removed, no more transitions are possible). Assume that after k transitions, each machine of M' still properly encodes the desired information about M in its control. We assert this condition can be maintained for the k+1st transition,and therefore M' will accept the same language as M. It should be clear that there is no difficulty so long as no store of M grows shorter, that is, no symbol is removed from a store without being replaced by one or more symbols. Suppose this is not the case, however. Say the ith machine is "remembering" the top h symbols of the jth store of M and the transition function of the jth machine of M requires that a symbol be removed from that store. The hth symbol now becomes the h-1st, the h-1st symbol becomes the h-2nd,..., the 2nd symbol becomes the 1st. The hth symbol, however, should be replaced by the h+1st, knowledge of which is not encoded in the control of machine i. It will be noticed, however, that the machine which is connected to the ith machine of M', the i-1st machine (nth machine if i=1), incorporates knowledge of the top h+1 symbols. Since the ith machine's transition function depends upon the state of this machine connected to the ith machine, the update can indeed take place! This is true for all stores, of course, which completes the induction. The final state set of each machine of M' consists of those states encoding final states for each machine of M. Clearly all machines of M' empty their stores if and only if all machines of M do so. Thus, we must have T(M) = T(M') and N(M) = T(M').

Q.E.D.

It may seem surprising that the construction of theorem 3 may be done without loss of time. When a machine of M' moves a symbol from its physical pushdown store into the logical extension of the store in the finite control, this fact is not immediately communicated to all other machines in the network. (The information propagates no faster than one machine per transition.) The technique works, however, because the information is "sent" around the ring in advance of when it will be needed and "arrives" before it is actually required.

For nondeterministic networks, we have the following theorem which places an even weaker restriction on the type of network connections needed to accept a language of a pdn_n. Again, we introduce a definition.

DEFINITION 17: Let C* be the reflexive, symmetric, transitive closure of relation C. (Recall that C_{ij} means that machine i is connected to machine j.) If M is a pdn_n, we say that M has a *connected structure* provided that C^*_{ij} for all $1 \leq i,j \leq n$.

THEOREM 4: Let M be a pdn_n. There exists a $npdn_n$ M' with a connected structure such that T(M) = T(M'), N(M) = N(M'), and every word accepted by M in m transitions is accepted by M' in m transitions.

PROOF: Notice that the restriction on the interconnections of M' is quite minimal — no machine or group of machines may be unconnected from the others of the network. The directions of the connections, however, are irrelevant. Thus, for example, in a network of n machines, n-1 machines may depend only on the nth, which in turn depends upon none of the others.

As before, each machine of M' will simulate one of the machines of M. Except in the initial state of each machine of M' and possibly in final states or states which cannot lead to acceptance, the finite control of each machine of M' encodes the following information:

(1) The state of the simulated machine

(2) The states which the other machines of M are assumed to be in,

(3) The logical top store symbol of the simulated machine,

(4) The element from $\Sigma \cup \{\varepsilon\}$ assumed to be an argument of the transition functions for the next transition,

(5) The state the simulated machine will be in after the next transition, and

(6) The states which the other machines of M are assumed to take on after the next transition.

The final states of machines of M' are those which encode only final states of machines of M. Each machine of M', on every transition except the first, operates as follows: If the machine depends upon any other machines, it checks to see if its allocation of current and future states of the machines of M and the element from $\Sigma \quad \{\varepsilon\}$ which causes the next transition agree (items (1), (2), (4), (5), and (6) above). If they do not, it does not empty its store and enters a non-final "dead" state from which no other transitions are possible. M' continues the simulation of M only so long as all machines of M' make the same assumptions about states and input symbols. (For i and j, whenever either C_{ij} or C_{ji}, one machine is able to check this for consistency.) If these assumptions agree, they must reflect possible configurations of M, as each machine of M' totally simulates a machine of M based upon the correctness of the other states. If the predicted element of $\Sigma \cup \{\varepsilon\}$ is from Σ, the dead state is entered if the symbol read is not as predicted. Otherwise, the next states (items (5) and (6)) become the current states (items (1) and (2)) and the pushdown store will be adjusted according to the transition function of the machine simulated. (This may affect (3).) The machine "guesses" a new item (4) and "guesses" next states of other machines (item (6)). Based on this information (it may need to know the physical top store symbol here), item (5), the next state of the machine simulated, can be established. As M may be nondeterministic, this involves another "guess." If no transition is possible, this is indicated in the state if the new current states are final, otherwise a dead state is entered. This process, which has been described sequentially, can clearly be carried out in one step, albeit with a complex, nondeterministic transition function. It should be clear that M' actually does simulate the actions of M. Note that the "look-ahead" feature is necessary to assure that incorrect "guesses" may be eliminated before they are executed. It is this feature which requires the first transition to be handled differently, as we have but a single initial configuration for M'. If M accepts no words in one transition, each machine of M' on its first transition encodes all the information listed above into its state. Items (1) and (2) may be inconsistent between the machines at this point. This can be corrected on the next transition, however, after which the machine operates as described. If M does accept some words in one transition, each machine of M' chooses on the first transition (if such a word occurs) to accept the word or to treat it as the prefix of a longer word. In the former case, it enters a final dead-end state and empties its store. Otherwise, it proceeds as if no words are accepted in one transition.

Q.E.D.

As has been mentioned, multipushdown acceptors with two or more stores are equivalent to Turing machines. Thus, we now restrict our attention to real-time multipushdown machines and networks. We will also confine our attention to acceptance by final state.

Book and Greibach [7] have shown that any language accepted by $rtpd_n$'s may be accepted by some $nrtpd_3$. Theorem 4 shows that it is possible to accept such a language with a $nrtpdn_3$ such that the three machines are related by C*.

Although it will not be proved as a theorem, it should be noted that a nondeterministic network need have only one nondeterministic element — only one transition function not composed of singleton sets. One machine can "tell" the other machines which transitions to execute. Those machines can block acceptance if the transitions cannot, in fact, be carried out.

Some Related Language Hierarchies

In the next chapter, we will use the network formulation to study a restricted class of real-time multipushdown machines, namely the real-time multicounter machines. Before moving on to multicounter acceptors, however, we conclude this chapter by relating three previously identified language hierarchies to automata network theory.

THEOREM 5: There is an infinite hierarchy of languages accepted by deterministic real-time pushdown automata networks. That is, for $n \geq 1$, $DRTPDN_n \subset DRTPDN_{n+1}$.

PROOF: The existence of this hierarchy has been shown by Aanderaa [1] independently of any network formulation.

Q.E.D.

COROLLARY 1: There is an infinite hierarchy of languages accepted by $drtpdn_n$'s with ring structure.

PROOF: This follows immediately from theorems 3 and 5.

Q.E.D.

DEFINITION 18: A network of n machines in which $p_i = 0$ for all $1 \leq i \leq n$ is called an *atomized network*. (An atomized network is simply a network with no connections.)

THEOREM 6: There is an infinite hierarchy of languages accepted by nondeterministic atomized real-time pushdown networks ($nartpdn_n$'s). That is, for $n \geq 1$, $NARTPDN_n \subset NARTPDN_{n+1}$.

PROOF: Let M be a $nartpdn_n$. Let M_i be the ith machine of the network. (M_i is just an ordinary pushdown automaton.) Clearly, $T(M) = \bigcap_{i=1}^{n} T(M_i)$, since a word is accepted by M if and only if it is accepted by each M_i. But it is well-known that a $nrtpd_1$ can accept any context-free language [30]. The result then follows from the fact established by Liu and Weiner [41] that there is an infinite hierarchy of intersections of context-free languages.

Q.E.D.

THEOREM 7: There is an infinite hierarchy of languaged accepted by deterministic atomized real-time pushdown networks. That is, for $n \geq 1$, $DARTPDN_n \subset DARTPDN_{n+1}$.

PROOF: Burkhard and Varaiya establish this hierarchy [9], relate it to the Liu and Weiner hierarchy, and describe their result as dealing "with *n* deterministic real time pushdown automata operating independently in parallel."

Q.E.D.

CHAPTER III

MULTICOUNTER AUTOMATA NETWORKS

A counter is a store which can contain a single integer of arbitrary magnitude, which can be incremented or decremented in one step by an integer of limited magnitude, and which can be tested only for zero contents. Without loss of generality, we may restrict the numbers stored to non-negative integers and the increment or decrement to 1 [18]. In this chapter, we will be concerned with real-time multicounter networks accepting by final state. As we will see, such networks are an equivalent formulation of real-time multicounter automata. We will incorporate results by Fischer, Meyer, and Rosenberg [18] and Kain [34] into a unified framework by means of the network formulation. In addition, we present new results suggested by this framework.

Preliminaries

Counters and counter machines, like other constructs in automata theory, have been defined in various ways. All too often, these definitions have been informal and imprecise. We will adopt a variation of Kain's conventions here in order to show clearly the counter machine as a special case of the usual pushdown acceptor [34].

DEFINITION 19: Let M be a pd_n, $M = (K,\Sigma,\Gamma_1,\Gamma_2,\ldots,\Gamma_n,\delta,q_0,Z_0,F)$. If, for $1 \le i \le n$, $\#(\Gamma_i) \le 2$ and γ_i on the ith store implies $\gamma_i \in (\Gamma_i - \{Z_0\})^* Z_0$, then M is called an *n-store counter machine* (cm_n). By analogy to pushdown machines, we may speak of *n-store counter automata networks* (cn_n), *real-time n-store counter automata networks* ($rtcn_n$), and so forth.

In counter machines under this definition, the pushdown automaton "counts" by tallying, using a single symbol on the store. The symbol Z_0 merely acts as end-marker and is always at the bottom of the store. Alternative definitions either treat the counter as a special memory storing a count which can be tested only for zero, or describe a counter as a pushdown store limited to a single-symbol alphabet. This last description is intuitively appealing because of its simplicity, but it requires a slight redefinition of the pd_n. The reason for this is easily seen. If a single symbol is on the store but is removed without replacement in a transition, the store will be empty and the machine must halt. Thus, the machine cannot test for a zero count. We might circumvent this problem in one of the following ways:

(1) Define transition functions on Cartesian products of

$$\ldots x(\Gamma_1 \cup \{\varepsilon\})x(\Gamma_2 \cup \{\varepsilon\})x\ldots x(\Gamma_n \cup \{\varepsilon\})x\ldots$$

or

(2) Base the transition function on the top two store symbols.

Neither of these alternatives seems to preserve the pushdown machine in its usual form, so we reject them and adopt definition 17.

We first establish that real-time machines are equivalent.

THEOREM 8: Given any $rtcn_n$ M. There is a $rtcm_n$ M' such that T(M) = T(M'). If M is deterministic, so is M'.

PROOF: Let M be a $rtcn_n$. By definition 19, M is just a special type of pdn_n. Construct M' as in theorem 1. Since the construction of M' does not alter the pushdown store alphabets, M' is also a counter machine. Since the simulation of M by M' is done without loss of time, M' must be a $rtcm_n$. It follows from theorem 1 that M' is deterministic if M is.

Q.E.D.

THEOREM 9: Given any $rtcm_n$ M. There is a $rtcn_n$ M' such that T(M) = T(M'). If M is deterministic, so is M'.

PROOF: Let M be a $rtcm_n$. By definition 19, M is a special type of pd_n. The construction of M' in theorem 2 is nearly adequate to prove this theorem but is not quite acceptable, as it requires augmenting the push-down store alphabets with the symbol W. From M, we may construct a machine for which each symbol on a store, except for the end-marker Z_0, represents two symbols on a store of M [18]. It follows that no two symbols are ever removed from the store in succession. Thus, if the logical top store symbol is kept in the finite control of each machine, the fact that the end-marker is on top of the physical store can be detected and encoded into the state without ever having to remove Z_0 to represent its coming to the top of the logical store. (In the construction in theorem 2, Z_0 would be removed from the store and encoded into the state, but W would remain on the store.) Clearly, we may keep the top store symbol and state of M in the control of each machine this way and can thus construct M' in a manner otherwise similar to that of theorem 2.

Q.E.D.

Corresponding to theorems 3 and 4, we have the following two theorems.

THEOREM 10: Let M be a $drtcn_n$. There exists a $drtcn_n$ M' with a ring structure such that T(M) = T(M').

PROOF: The proof follows along the lines of that for theorem 3. In order to encode the top portion of the store into the finite control, however, we must use a technique similar to that used in the last theorem. For $n > 1$, modify the network so that each symbol on a store represents n symbols. The transition functions may then be adjusted to follow the proof of theorem 3.

Q.E.D.

THEOREM 11: Let M be a $rtcn_n$. There exists a $nrtcn_n$ M' with a connected structure such that T(M) = T(M').

PROOF: The proof follows directly from theorem 4 if the stores are handled as previously described to encode the top symbol into the finite control.

Q.E.D.

We would like to develop a general theory (an exhaustive catalog, at any rate) of the effects of interconnections within real-time counter networks. What additional computing power, if any, is provided by additional connections? The question is partially answered already. A structure more extensive than a ring structure provides no recognition power to a deterministic network beyond that provided by a ring structure. A structure having more connections than the minimum needed for a connected structure is similarly redundant for nondeterministic networks. Thus, a deterministic n-counter network need have no more than n connections (no connections, of course if n=1). A nondeterministic n-counter network need have no more than n-1 connections. From our development so far, we see that at least four cases must be examined:

(1) Deterministic real-time counter networks with unrestricted connections (the connections may be limited to ring connections, however),

(2) Nondeterministic real-time counter networks with unrestricted connections (whose connections need only link all machines in the network),

(3) Deterministic atomized real-time counter networks, and

(4) Nondeterministic atomized real-time counter networks.

Language Hierarchies Related to Multicounter Networks

Let us first look at deterministic, real-time counter networks. There exists an infinite hierarchy of acceptance classes among such machines. This was shown for counter machines independently by Fischer, Meyer, and Rosenberg [18] and Laing [39]. Here, we shall follow the development of the former.

DEFINITION 20: Let L be some language over alphabet Σ, and let $x,y,z \in \Sigma^*$. We say that x and y are *k-equivalent with respect to L*, $x\ E_k\ y \pmod L$, if, for all z such that $|z| \leq k$, $xz \in L$ if and only if $yz \in L$.

Definition 20 says that if $x\ E_k\ y \pmod L$, prefixes x and y are indistinguishable on the basis of their suffixes of length k or less.

LEMMA 2: Let M by a $drtcm_n$ with s states. Then the number of equivalence classes of E_k (mod T(M)) is less than or equal to $s(k+1)^n + 1$, which is less than ck^n for some constant c.

PROOF: First, we note that the bound in the literature is $s(k+1)^n$. The difference here, which is incidental to the lemma, is the result of our particular formulation of counter machines.

Since M can remove only one symbol at a time from a given store and since all symbols on a store are the same except for the end-marker, any store expression of length k+1 is indistinguishable in k steps from one of length greater than k+1. Note that a store may be empty (in which case no more transitions can occur), may contain only the end-marker, or may contain any number of identical symbols followed by the end-marker.

Thus, since there are n stores, at most $(k+1)^n+1$ classes of store variations are distinguishable. (All those variations having at least one empty store are indistinguishable — none allows additional transitions.) The classes may be paired with different states to give a maximum of $s(k+1)^n+1$ distinguishable equivalence classes. For some c, $s(k+1)^n+1<ck^n$.

Q.E.D.

We now establish a hierarchy arising from deterministic real-time counter networks of unrestricted structure using the languages

$$L_n^1 = \{0^{m_1}1\,0^{m_2}1\ldots 0^{m_n}B_i 0^{m_i} \mid 1\le i\le n,\ m_j\ge 1,\ 1\le j\le n\},\quad n\ge 1,$$

over alphabets $\Sigma_n = \{0,1,B_1,B_2,\ldots,B_n\}$.

THEOREM 12: $DRTCN_n \subset DRTCN_{n+1}$, $n\ge 1$.

PROOF: We prove the result for counter machines. The desired result then follows from the equivalence of counter machines and counter networks.

Clearly, $L_n^1 \in DRTCM_n$. L_n^1 is accepted by a $drtcm_n$ as follows: As each string of 0's is read, the corresponding m is placed on a counter. B_i tells the machine to compare the final string of 0's with the count of the ith counter. If the string ends with 0^{m_i}, the word of L_n^1 is accepted.

Consider two distinct words $x=0^{m_1}1\,0^{m_2}1\ldots 0^{m_{n+1}}$ and $y=0^{r_1}1\,0^{r_2}\ldots 0^{r_{n+1}}$, where all the m's and r's are not greater than some constant h. There exist s and t, $1\le s\le n$, $1\le t\le h$, such that $z=B_s0^t$, $|z|\le h+1$, $xz \in L_{n+1}^1$, and $yz \notin L_{n+1}^1$. Since there are h^{n+1} distinct words of this form, we must have $h^{n+1}\le \#(E_{h+1} \pmod{L_{n+1}^1})$. But for any $drtcm_n$ M, $\#(E_{h+1} \pmod{T(M)}) < c(h+1)^n$ by lemma 2. For large h, however, $\#(E_{h+1} \pmod{T(M)}) < c(h+1)^n < h^{n+1}$. Hence, $L_{n+1}^1 \notin DRTCM_n$. But by the previous argument, $L_{n+1}^1 \in DRTCM_{n+1}$. Since the inclusion of $DRTCM_n$ in

$DRTCM_{n+1}$ is trivial, this proves proper inclusion.

Q.E.D.

The next theorem is the analog of theorem 12 for the nondeterministic case and is a modification of a theorem in [34]. The theorem is based on the languages

$$L_n^2 = \{0^{m_1} 1\, 0^{m_2} 1 \ldots 0^{m_n} 2\, 0^{m_1} 1\, 0^{m_2} 1 \ldots 0^{m_n} 1\, 0^q \mid q \geq 0,\ m_i \geq 0, 1 \leq i \leq n\},\ n \geq 1,$$

over the alphabet $\Sigma = \{0,1,2\}$.

THEOREM 13: $NRTCN_n \subset NRTCN_{n+1}$, $n \geq 1$.

PROOF: Again, we prove the result for counter machines. It is clear that $L_n^2 \in NRTCM_n$. In fact, $L_n^2 \in DRTCM_n$. (We present both theorem 12 and theorem 13 to illustrate two techniques for proving facts about counter machines, as each of these techniques will be used later.) We will show that $L_{n+1}^2 \notin NRTCM_n$.

Let M be a $nrtcm_n$ such that $T(M) = L_n^2$. Suppose M has read some word $w \in L_n^2$ up to the 2, inclusive, and suppose k symbols of w remain to be read. If M has s states, there are at most $s(k+1)^n$ different configurations which the machine can be in which can affect the acceptance of w. This is because M can be in any state and can have a count of between 0 and k on each counter. (Since each counter can be tested only for zero, a count larger than k cannot affect the operation of M in k transitions.) This means that M can distinguish between at most $s(k+1)^n$ different w's. Consider the suffix of w which remains to be read. It consists of k symbols, n of which are 1's. These 1's may occur anywhere in the suffix to yield a valid suffix corresponding to a unique prefix. There are combination k items taken n at a time, $C(k,n)$, such suffixes or

$$C(k,n) = \frac{k!}{(k-n)!n!} = \frac{k(k-1)\ldots(k-n+1)}{n!} > \frac{(k-n+1)^n}{n!}$$

This means that we must have

$$\frac{(k-n+1)^n}{n!} < C(k,n) \leq s(k+1)^n$$

or

$$\frac{(k-n+1)^n}{(k+1)^n} < sn!$$

On the other hand, if we assume $T(M') = L^2_{n+1}$ for some n-counter machine, we have

$$\frac{(k-n+1)^{n+1}}{(k+1)^n} < s(n+1)!$$

But $s(n+1)!$ is a constant. For large values of k, the expression on the left of the above inequality increases as k, that is, it becomes arbitrarily large. Hence, M' does not exist.

Q.E.D.

We have now shown the existence of two infinite acceptance hierarchies arising from counter networks. These hierarchies, shown in theorems 12 and 13 are not identical, as we shall show in the next theorem, also a modification of a theorem in [34]. We use the languages

$$L^3_{n,p} = \{0^{m_1} 1\, 0^{m_2} 1 \ldots 0^{m_p} 1\, 0^q B_{i_1} 0^{m_{i_1}} B_{i_2} 0^{m_{i_2}} \ldots B_{i_n} 0^{m_{i_n}} \mid q \geq 0,$$
$$m_k \geq 0,\ 1 \leq k \leq p,\ 1 \leq i_j \leq p,\ 1 \leq j \leq n\},\ 1 \leq n \leq p,$$

over alphabets $\Sigma_p = \{0,1,B_1,B_2,\ldots,B_p\}$.

THEOREM 14: $DRTCN_n \subset NRTCN_n$, $n \geq 1$.

PROOF: As before we prove the result for counter machines. We assert that for $p > n$, $L^3_{n,p} \in NRTCM_n$.

A $nrtcm_n$ to accept $L^3_{n,p}$ operates as follows: The machine nondeterministically "guesses" which of the numbers $m_1, m_2, \ldots, m_p$ must be remembered. On the n stores, n of these numbers are saved. The machine can verify it has guessed correctly by checking its guess against the set of B's encountered. If it has guessed wrong, acceptance is blocked. Otherwise, the contents of the counters are compared with the input string.

Consider the possible configurations of a machine M accepting $L^3_{n,p}$ just before reading B_{i_1}. If the number of symbols read up to this time is k, we may assume without loss of generality that no store contains more than $k+1$ symbols. (If $r > 1$ symbols are added to a counter in any transition, the transition function can be coded such that each symbol except the end-marker represents r symbols on the original counter.) If the machine has s states, it may be in one of no more than $s(k+1)^n$ configurations which result in acceptance of any strings in the language. Now there are $C(k,p)$ different k-length prefixes of words, each of which requires a different set of suffixes in $L^3_{n,p}$. If M is deterministic, each prefix must lead to a different configuration. Thus, we must have

$$\frac{(k-p+1)^p}{p!} < C(k,p) \leq s(k+1)^n$$

If this is the case, we must have

$$\frac{(k-p+1)^p}{(k+1)^n} < sp!$$

But sp! is a constant. For large k, the expression on the left increases as k^{p-n}. But $n < p$ means that this gets arbitrarily large. Thus, (1) cannot hold, and M cannot be deterministic.

Q.E.D.

New Hierarchies Arising from the Network Formulation

We will now be concerned mainly with new results suggested by the network formulation of real-time multicounter machines. First, we establish the existence of infinite hierarchies for atomized machines.

THEOREM 15: $\mathrm{DARTCN}_n \subset \mathrm{DARTCN}_{n+1}$, $n \geq 1$.

PROOF: The proof is a direct result of the proof of theorem 7 found in [9]. In that proof, the set of languages

$$L_n^4 = \{1^{m_1} 2^{m_2} \ldots n^{m_n} 0 \, i^{m_i} \mid m_j \geq 0, \ 1 \leq j \leq n, \ 1 \leq i \leq n\}, n \geq 1,$$

over alphabets $\Sigma_n = \{0,1,\ldots,n\}$ are used to demonstrate a DARTPD hierarchy. That is, for $n \geq 2$, $L_n^4 \in \mathrm{DARTPD}_n$ but $L_n^4 \notin \mathrm{DARTPD}_{n-1}$. If we can show that $L_n^4 \in \mathrm{DARTCN}_n$ for $n \geq 1$, since $\mathrm{DARTCN}_n \subseteq \mathrm{DARTPD}_n$ (counter networks are restricted pushdown networks), it follows that $\mathrm{DARTCN}_n \subset \mathrm{DARTCN}_{n+1}$, $n \geq 1$. But surely this is the case. A network accepting L_n^4 operates as follows: The jth machine of the network places the number m_j on its store when j^{m_j} is read. If $i \neq j$, it enters and remains in a final state no matter what the input. If $i = j$, it compares the number of i's following 0 against m_j on its counter. It enters a final state if there are m_j 1's; otherwise it remains in a non-final state.

Q.E.D.

THEOREM 16: $NARTCN_n \subset NARTCN_{n+1}$, $n \geq 1$.

PROOF: The proof is analogous to that of theorem 15. The proof of the corresponding theorem in the pushdown case is found in [41] and uses the languages

$$L_n^5 = \{1^{m_1} 2^{m_2} \ldots n^{m_n} 1^{m_1} 2^{m_2} \ldots n^{m_n} \mid m_i \geq 1,\ 1 \leq i \leq n\}, n \geq 1,$$

over alphabets $\Sigma_n = \{1,2,\ldots,n\}$. Clearly $L_n^5 \in NARTCN_n$ for all n, but since $L_{n+1}^5 \notin NARTPDN_n$, $L_{n+1}^5 \notin NARTCN_n$.

Q.E.D.

It remains for us to illuminate the exact relationships between the hierarchies found in theorems 12, 13, 15, and 16. A more complete theory of interconnections requires us to look at one more connecting scheme, however, a linear arrangement of automata.

DEFINITION 21: A $rtcn_n$ such that C_{ij} if and only if $j = i + 1$, $1 \leq i \leq n - 1$, is called a *linear* $rtcn_n$ ($lrtcn_n$).

In definition 21, we have avoided the generality of the definition of a ring structure (definition 16). The linear nature of the linear structure does not depend upon the formal numbering given the machines of the network. Without loss of generality, therefore, we will assume $C_{12}, C_{23}, \ldots, C_{(n-1)(n)}$.

THEOREM 17: There is no distinct infinite hierarchy among $nlrtcn_n$'s.

PROOF: By theorem 11, we know that a connected structure is fully general for a $nrtcn_n$. From definition 21, it is clear that a $nlrtcn_n$ has a connected structure (C^*_{ij} for all $1 \leq i,j \leq n$). Thus, a NLRTCN hierarchy exists, but it is just the NRTCN hierarchy of theorem 13.

Q.E.D.

Theorem 18 will show that there *is* an infinite hierarchy (which will subsequently be shown to be distinct) among *deterministic* lrtcn_n's. The proof that such a hierarchy exists is involved and will require a number of technical lemmas. The proof is based on the languages

$$L_n^6 = \{0^m 1^{am} \mid m \geq 1,\ 1 \leq a \leq 2^n - 1\}, \quad n \geq 1.$$

LEMMA 3: For $n \geq 1$, there exists a dlrtcn_n M such that $T(M) = L_n^6$.

PROOF: We begin by noting that for any dlrtcn_n M', we may construct a network M which operates in a particular way and which accepts the same language. We first code the top store symbols for each machine of M' into the finite control of the corresponding machine of M, as we have done in other proofs. By so doing, the ith machine of M can always "know" whether or not the counters of machines 1,2,...,i of M' are positive or zero. (See proof of theorem 3.) Since linear connections allow information flow in one direction only, this machine "knows" nothing about machines i+1, i+2,...,n of M'. The nth machine of M can determine the states of all the machines of M' at any time. Thus, it can always determine when M should be in a final state. We may therefore make all states of machines 1,2,...,n-1 of M final. Machine n of M has both final and non-final states. It will enter a final state if and only if all the machines of M' are in final states after seeing the same input. For the remainder of this proof, we will speak of the (logical) contents of a counter, suppressing the fact that the contents of the physical counter may be different. For machines i and j, we assume machine j "knows" when machine i empties its counter if and only if $i \leq j$.

Suppose M operates as follows: While reading 0's, the ith counter increases its contents by 2^{n-i} for each 0 read. Thus, after reading 0^m, the ith counter contains $2^{n-i}m$. When the first 1 is read, the counters begin to count down. Each machine j counts either up or down by one for each 1 read until some machine $i \leq j$ empties its store, at which time it changes from incrementing to decrementing or vice versa. M accepts a word whenever *any* store empties (machine n enters a final state; other machines have only final states). Misplaced input symbols, of course, result in immediate rejection of a word by having machine n enter a "dead," non-final state. We assert that $T(M) = L_n^6$. We will show this by induction.

Let $n = 1$. After 0^m has been read, the network has m on its only counter. It then begins to decrement the counter as 1's are read, emptying its store after reading m of them. This causes the network (the single machine) to enter a final state, accepting the word 0^m1^m. Upon reading more 1's, the counter increases its count without limit, as there is no other counter to empty, and as the one counter there is will never empty so long as it is being incremented. Thus, $T(M) = L_1^6$.

Suppose the assertion is true when $n = k$. Consider the case for $n = k+1$. After 0^m is read, the store of the ith machine contains $2^{(k+1)-i}m$. The operation of machines $1,2,\ldots,k$ is independent of machine $k+1$, of course, as $k+1$ is connected to none of these machines. On the other hand, each time one of these machines empties a counter, machine $k+1$ takes the network into an accepting configuration. Notice that we can rewrite the contents of the first k counters as $2^{k-1}(2m)$, $2^{k-2}(2m),\ldots,2m$. In other words, the first k counters contain the counts a $dlrtcn_k$ of the type we are considering would have on its counters after reading 0^{2m}. In fact, so far as when the counters empty is concerned, these counters work exactly like those of such a machine which, by hypothesis, accepts L_k^6. Thus, the set of words 0^m1^{2m}, $0^m1^{2(2m)}, \ldots, 0^m1^{(2^{(k+1)}-2)m}$ must be accepted, since a machine with k counters accepts words with suffixes 1^{2m}, $1^{4m}, \ldots, 1^{(2^{(k+1)}-2)m}$ after being in the configuration in question. Since the k-counter machine accepts no other words with any of these as prefixes, the dlrtcn_{k+1} must accept no other words prefixed by $0^m1^{2m}, \ldots$ by virtue of the emptying of its first k stores. Now we consider words accepted by virtue of the emptying of the k+1st store. Initially, it acts just like the store for a machine for which $n = 1$, that is, it empties after m 1's, accepting 0^m1^m thereby. The count increases as the next m 1's are read, until 0^m1^{2m} is accepted because an earlier counter has emptied. At this point, the count is m and the counter is being decremented. Because the earlier counters empty every 2m counts, it is clear that counter k+1 empties after 0^m1^m, $0^m1^{3m}, \ldots 0^m1^{(2^{(k+1)}-1)m}$. After this, its contents increase without limit as 1's are read, since no more stores are emptied. Combining this result with the strings we know are also accepted, we see that our dlrtcn_{k+1} M accepts, for $1 \leq m$, 0^m1^m, $0^m1^{2m}, \ldots, 0^m1^{(2^{(k+1)}-1)m}$. That is, $T(M) = L_{k+1}^6$. This completes the induction.

Q.E.D.

Lemma 3 shows how the L^6 languages can be accepted by dlrtcn_n's. We shall see in theorem 18 that the essentials of the algorithm given above are *necessary* to accept L_n^6 with a dlrtcn_n.

The next lemma asserts that a dlrtcn_1 must behave periodically under certain circumstances.

LEMMA 4: Let M be a dlrtcn_1 (drtcn_1) and let $s \in \Sigma$. There exist integers p_1, p_3, $p_4 \geq 0$ and $p_2 \geq 1$ such that if M is in state q and has 0 on its counter (the end-marker only, in our formulation), M will be in some state q_1 after reading $s^{p_1+ap_2}$ for all $a \geq 0$. Further, for every a, between reading $s^{p_1+ap_2}$ and $s^{p_1+(a+1)p_2}$ inclusive, M goes through the same sequence of states $q_1, q_2, \ldots, q_{p_2}, q_1$. If the counter contains p_3 after reading s^{p_1}, it contains $p_3 + ap_4$ after reading $s^{p_1+ap_2}$.

PROOF: Let $b = \#(S_1)$, the number of states of M, let q be the state of M at time t_0, and let the input consist only of s's. There are two cases to consider.

Case 1. The counter empties a finite number of times after t_0. Thus, after some time $t_1 \geq t_0$, the counter either contains 0 and never increases its count, or contains a positive integer and never decreases its count below that integer. Let c_1 be the count stored at t_1. Within b transitions of t_1, some state q_1 of M must occur at least twice, say at times t_2 and t_3. Let $p_1 = t_2 - t_0$ and $p_2 = t_3 - t_2$. Since for all transitions after time t_1, the input and top store symbol remain the same, and since M is deterministic, the fact that M is in q_1 at t_2 and t_3 means that beginning at t_3, M will execute the same $t_3 - t_2$ transitions it executed between times t_2 and t_3. Thus between reading $s^{p_1+ap_2}$ and $s^{p_1+(a+1)p_2}$ for any $a \geq 0$, M goes through the same sequence of states $q_1, q_2, \ldots, q_{p_2}, q_1$. If the counter contains c_2 at t_2 and c_3 at t_3, we must have $p_3 = c_2$ and $p_4 = c_3 - c_2$, since executing the same sequence of transitions must always alter the count by the same amount. Notice that $p_4 \geq 0$, since $p_4 < 0$ would imply that for some a, the counter would contain $p_3 + ap_4 < c_1$, contrary to hypothesis.

Case 2. The counter empties an infinite number of times after t_0. For some b, by the time the counter has emptied b times after t_0, some state q_1 of M must have occurred at least twice at times when the counter empties. Call the first of these times t_2 and the second t_3. Let $p_1 = t_2 - t_0$ and $p_2 = t_3 - t_2$. Since at time t_3, M is in the same configuration as at t_2, since the input continues the same as between t_2 and t_3, and since M is deterministic, we again have M going through states $q_1, q_2, \ldots, q_{p_2}, q_1$ between reading $s^{p_1+ap_2}$ and $s^{p_1+(a+1)p_2}$ for any $a \geq 0$. In this case, of course, we have $p_3 = p_4 = 0$.

Q.E.D.

The next lemma establishes a periodic behavior similar to that seen in the last lemma for any deterministic real-time multicounter network. We first introduce a definition.

DEFINITION 22: Let M be a $rtcn_n$. If M is in configuration $(q_1, q_2, \ldots, q_n, \gamma_1, \gamma_2, \ldots, \gamma_n)$, we will say that M is in *state configuration* $(q_1, q_2, \ldots, q_n)$.

LEMMA 5: Let M be a $drtcn_n$ in some configuration at time t_0. If the input remains constant (say $s \in \Sigma$) and the network does not halt, there exist integers $p_1 \geq 0$ and $p_2 \geq 1$ such that M is in some state configuration C_1 after reading $s^{p_1+ap_2}$ for each $a \geq 0$ for which $t_0 + p_1 + ap_2 < t$, where t is the first time after t_0 when some counter empties (decreases from 1 to 0). Further, for all $a \geq 0$ such that $t_0 + p_1 + (a+1)p_2 < t$, between reading $s^{p_1ap_2}$ and $s^{p_1+(a+1)p_2}$ inclusive, M goes through the same sequence of state configurations $C_1, C_2, \ldots, C_{p_2}, C_1$.

PROOF: There are $p = \prod_{i=1}^{n} \#(K_i)$ possible state configurations of M. Assuming no counters empty, within $p(n+1)$ transitions after t_0, some state configuration must have occurred n+1 times. Further, on two of these occurrences, the set of counters with counts of 0 must be the same. Let these occurrences be at times t_2 and t_3, and let $p_1 = t_2 - t_0$ and $p_2 = t_3 - t_2$. Since M is deterministic, if the input remains constant and no additional stores are emptied between times t_3 and $t_3 + p_2$, M will execute the same sequence of transitions as between t_2 and t_3. This argument is valid for each successive group of p_2 transitions. The desired result follows immediately.

Q.E.D.

Now we are ready to establish the existence of the linear hierarchy. The proof of the following theorem relies upon the fact that to recognize words of L^6_{n+1} beginning with 0^m, a network accepting L^6_{n+1} must be able to "remember" the number m in order to compare multiples of it with the number of 1's following 0^m. We will show, however, that the linear structure of $dlrtcn_n$ does not allow the network to retain m long enough to recognize all such words. If these words are accepted by the network, words not in L^6_{n+1} must be accepted also.

THEOREM 18: $DLRTCN_n \subset DLRTCN_{n+1}$, $n \geq 1$.

PROOF: By lemma 3, we know that $L^6_n \in DLRTCN_n$, $n \geq 1$. For some $n \geq 1$, assume there exists some $dlrtcn_n$ M such that $T(M) = L^6_{n+1}$. We will show that M cannot exist, and hence, the inclusion, which is trivial, is also proper. Recall that

$$L^6_{n+1} = \{0^m 1^{am} \mid m \geq 1,\ 1 \leq a \leq 2^{n+1} - 1\} .$$

Claim 1: From M, we may construct a $dlrtcn_n$ M' which acts in a special way and for which T(M) = T(M'). Suppose M' has read 0^m of a word in 0^m1^*. If counters $1,2,\ldots,j$, $j \leq n$, later empty at times $t_1 \leq t_2 \leq \cdots \leq t_j$, these j counters never empty thereafter.

Begin by constructing M''' to operate in the manner of the network in the proof of lemma 3. That is, only machine n of M''' has non-final states and each machine $i \leq n$ "knows" the current state and top store symbol of machines $1,2,\ldots,i$ of M. From M''', we construct network M'''' as follows: Modify M''' so that when counter 1 empties after reading 0^m of a word in 0^m1^*, machine 1 of M'''' begins incrementing its counter on every transition, and machines $1,2,\ldots,n$ simulate the action of machine 1 of M''' in their finite controls. That this can be done follows from lemma 4. From M'''', construct M''''' in a similar way, eliminating the emptying of counter 2 after counters 1 and 2 have emptied in that order. Old machine 2 is simulated by new machines $2,3,\ldots,n$. We continue this process of constructing new networks from previous networks through n iterations. Call the resulting machine M'. Clearly, M' must behave as described in claim 1.

Claim 2: From M', we may construct a dlrtcn_n M" such that T(M) = T(M') = T(M"). In reading a word in 0^m1^*, M" empties no counter j twice, $1 \leq j \leq n$, without emptying some counter i, $i < j$, in between.

In the construction of M', we have assured that each machine j of M' "knows" the states of all machines i, $i \leq j$, of the network from which it was constructed. We modify M' so that each time machine j empties its counter after 0^m is read, it avoids emptying it again until the counter of an earlier machine in the linear chain empties its counter. (This can be done in another sequence of constructions, as above. We suppress the details.) The modification can be made, since the transition functions of earlier machines can be incorporated into the control of machine j. Looking at machine j as a dlrtcn_1, we see that its behavior must become periodic, by lemma 4. Thus, after a fixed number of transitions, counter j either never empties again or empties periodically. In the former case, machine j of M'' behaves as machine j of M'. In the latter case, since the period can be bounded, the count can be kept in the finite control as well as on the counter. These two counts will be made equal except when the counter of the old machine empties, in which case a count of 1 will be maintained on the new store. This same technique is used during the transitions before the machine becomes periodic to avoid a count of 0. When the store of some earlier machine empties, of course, machine j of M'' reverts to operating as the corresponding machine of M'.

Claim 3: There is an integer $p \geq 1$ such that for $m > p$, after reading 0^m, some counter of M'' must empty for each word in 0^m1^* accepted.

Let $p = p_1 + p_2$, where p_1 and p_2 are guaranteed by lemma 5. Suppose M'' is in state configuration C after reading 0^m1^m. By lemma 5, if no stores are emptied, the state configurations occurring after $0^m1^{p_1}$ is read recur with period p_2. Since $m > p_1 + p_2$, C must occur after reading $0^m1^{m'}$, where $m' < m$. But C must be an accepting state configuration (all components are final states of their respective machines). Thus, $0^m1^{m'} \in T(M'')$. However, $0^m1^{m'} \notin L_{n+1}^6$, so the hypothesis that no counter empties must be false. A similar argument can be made for each word in 0^m1^* that M'' must accept. This establishes claim 3.

Claim 4: For some integer p, M'' accepts no more than $2^n - 1$ words in $0^m 1^*$, where $m > p$.

Let p be as in claim 3. Let T_j be the number of times counter j empties after 0^m is read. By claim 3, the number of words in $0^m 1^*$ accepted must be no more than $T_{max} = \sum_{j=1}^{n} T_j$. By claim 2, we must have $T_j = 1 + \sum_{i=1}^{j-1} T_j$. By claim 1, we have $T_1 = 1$. It is easily seen that $T_{max} = 1 + 2 + \ldots + 2^{n-1} = 2^n - 1$.

Now L^6_{n+1} contains $2^{(n+1)} - 1 > 2^n - 1$ words in $0^m 1^*$. This contradicts the contention that $T(M'') = L^6_{n+1}$. But we have seen that $T(M) = T(M') = T(M'')$, so that $T(M) \neq L^6_{n+1}$, contrary to assumption. Hence, $DLRTCN_n \subset DLRTCN_{n+1}$.

Q.E.D.

Relations among the Hierarchies

We have now established the existence of five infinite hierarchies of acceptance classes arising from real-time counter networks. In relating these hierarchies to one another, we may ask how corresponding classes of the hierarchies relate to one another as well as how the unions of all the classes relate to one another. The two following theorems present the most obvious relationships.

THEOREM 19: For $n \geq 1$, the following are true:

(a) $DARTCN_n \subseteq DLRTCN_n$.

(b) $DLRTCN_n \subseteq DRTCN_n$.

(c) $DRTCN_n \subseteq NRTCN_n$.

(d) $DARTCN_n \subseteq NARTCN_n$.

(e) $NARTCN_n \subseteq NRTCN_n$.

PROOF: All the inclusions are obvious because in each case the acceptance class on the left arises from a network type which is a restricted form of that giving rise to the acceptance class on the right.

Q.E.D.

We may demonstrate that all the hierarchies are distinct by showing that all the set inclusions in theorem 19 are proper. This has already been done for (c) in theorem 14. Notice that showing that corresponding classes of two hierarchies are related by proper inclusion does not imply that the infinite unions of the classes of the two hierarchies are so related.

THEOREM 20: If $K_1, K_2, \ldots, K_n, \ldots$ represent acceptance classes in an infinite hierarchy, we shall represent $\bigcup_{i=1}^{\infty} K_i$ simply by K. The following are true:

(a) DARTCN $\subseteq$ DLRTCN.

(b) DLRTCN $\subseteq$ DRTCN.

(c) DRTCN $\subseteq$ NRTCN.

(d) DARTCN $\subseteq$ NARTCN.

(e) NARTCN $\subseteq$ NRTCN.

PROOF: The inclusions are obvious by the same reasoning as used in the proof of theorem 19.

Q.E.D.

We now present a number of theorems to show that the inclusions of the preceding theorems are indeed proper.

THEOREM 21: (a) $\text{DARTCN}_1 = \text{DLRTCN}_1$.

(b) DARTCN $\subset$ DLRTCN.

(c) $\text{DARTCN}_n \subset \text{DLRTCN}_n$, $n \geq 2$.

PROOF: (a) This follows immediately from the fact that a dartcn_1 and dlrtcn_1 are really the same type of device.

(b) We will show that $L_2^6 \notin \mathrm{DARTCN}_n$ for any n. But by lemma 3, $L_2^6 \in \mathrm{DLRTCN}_2$.

Assume $T(M) = L_2^6$ for some dartcn_n M.

Claim 1: There is a dartcn_n M' such that T(M) = T(M') and which never empties the same counter twice after reading 0^m for any m.

After 0^m has been read, either all 1's must be read or, without loss of generality, we may have all machines of the network halt. By lemma 4, however, machines which have emptied their stores once become periodic with constant input. Thus, we may construct the machines of M' from those of M by simulating this periodic behavior in the finite controls and maintaining a non-zero count on the counter.

Claim 2: For m sufficiently large, after 0^m is read, at least one counter of M' must empty for each word of L_2^6 accepted. This follows from lemma 5 by an argument similar to that used in theorem 18.

Claim 3: For any given m sufficiently large, suppose counters of M' empty at times $t_1, t_2, \ldots, t_r$, where the last counter to empty before time t_a, when $0^m 1^m$ is accepted, does so at t_1, and the last counter to empty before time t_b, when $0^m 1^{2m}$ is accepted, does so at t_r. (By claim 2, $r \geq 2$. We assume that between t_1 and t_r, counters empty only at the times indicated.) For increasing m, the interval between t_i and t_{i+1} for some $1 \leq i \leq r-1$ must become arbitrarily large.

From lemma 5, it is clear that $t_r - t_1$ must increase with m. Otherwise, the network state behavior must become periodic between t_r and t_b, and we may argue as in theorem 18 that words not in L_2^6 must be accepted.

But if $t_r - t_1$ becomes arbitrarily large, then the time between some t_i and t_{i+1}, must become arbitrarily large, as $r \leq n$, by claim 1.

Claim 4: There exists some m such that a word of 0^m1^* not in L_2^6 is accepted between times t_i and t_{i+1}, as defined in claim 3.

Suppose q machines of M' empty their counters before t_{i+1} and n-q do not. We may view these machines as being a dartcn$_q$ M'' and dartcn$_{n-q}$ M'''. By lemma 5, within some interval of t_i, the state behavior of M'' becomes and remains periodic so long as 1's are read from the input tape. Since the machines of M'' must all be in final states at t_b, they must be in the same final states periodically before t_b, at least for large m. Let the period of this repetition be p_1. Likewise, for large m, the state behavior of M''' becomes periodic prior to t_a and remains so until at least t_{i+1}. Since all machines of M''' must be in final states at t_a, this set of final states recurs periodically between t_a and t_{i+1}. Let this period be p_2. Since the interval between t_i and t_{i+1} can be made arbitrarily long, the number of recurrences of final states of M'' and M''' within this interval may be made arbitrarily large. If the occurrences of all final states of both machines ever coincide, all machines of M' will be in final states, and a word not in L_2^6 will be accepted.

One may determine the possible periods of repetitive state behavior for each machine of M', as there are only a finite number of them. If m is made to be a multiple of the product of *all* the periods of *all* machines, m will be a multiple of periods p_1 and p_2.

Suppose M'' goes into all final states between t_i and t_{i+1}. (Since the interval can be made as large as we wish, we may assure that this occurrence is in an interval during which both M'' and M''' are periodic.) Let this occur jp_1 transitions from t_b. Since m is a multiple of p_1, this must be kp_1 transitions from t_a. By making m sufficiently large, we can also make j a multiple of p_2. Since m is a multiple of p_1 and p_2, it is also a multiple of p_1p_2. Hence, k is a multiple of p_2. This means M''' must be in all final states as well. Therefore, M' accepts a word not in L_2^6. But T(M) = T(M'), contradicting the assumption that $T(M) = L_2^6$.

(c) By theorem 19a, $DARTCN_n \subset DLRTCN_n$. But $L_2^6 \in DLRTCN_n$ and $L_2^6 \notin DARTCN_n, n \geq 2$. Thus, $DARTCN_n \subset DLRTCN_n$, $n \geq 2$.

Q.E.D.

THEOREM 22: (a) $DLRTCN_1 = DRTCN_1$.

(b) DLRTCN ⊂ DRTCN.

(c) $DLRTCN_n \subset DRTCN_n$, $n \geq 2$.

PROOF: (a) Any $dlrtcn_1$ is a $drtcn_1$ and vice versa.

(b) Consider the language

$L^7 = \{0^m 1^{am} | m \geq 1, \quad a \geq 1\}$.

L^7 may be accepted by a $drtcn_2$ as follows: The network places m on one counter as 0^m is read. As 1's are read, this counter is decremented and the other counter is incremented. Whenever a counter is emptied, the network accepts. Then the roles of the counters are reversed. Each time a counter empties, the other counter contains m. The network enters an accepting configuration after reading 0^m1^m, $0^m1^{2m}, \ldots, 0^m1^{am}, \ldots$. Thus $L^7 \in DRTCN_2 \subset DRTCN$.

From the proof of theorem 18, it may be seen that any $dlrtcn_n$ accepting only words of the form 0^m1^{am} for large m and certain integral values of a can do so for no more than $2^n - 1$ such values. Thus $L^7 \notin$ DLRTCN. Since $L^7 \in$ DRTCN and since by theorem 20b, DLRTCN $\subseteq$ DRTCN, we have that DLRTCN $\subset$ DRTCN.

(c) By theorem 19b, $DLRTCN_n \subseteq DRTCN_n$. But for any $n \geq 2$, $L^7 \in DRTCN_n$ but $L^7 \notin DLRTCN_n$. Hence, $DLRTCN_n \subset DRTCN_n$, $n \geq 2$.

Q.E.D.

THEOREM 23: DRTCN $\subset$ NRTCN.

PROOF: Theorem 14 has already established that $DRTCN_n \subset NRTCN_n$, $n \geq 1$. This result is inadequate to prove the present theorem, however, as all the languages used to show the proper inclusion are in both DRTCN and NRTCN.

Consider the language

$$L^8 = \{\{0,1\}^a \, 1\{0,1\}^a \mid a \geq 1\},$$

that is, the set of words consisting of a-symbol prefixes from $\Sigma^* = \{0,1\}^*$, followed by 1, followed by a-symbol suffixes from Σ. This language is accepted by a $nrtcn_1$. Such a network "guesses" when the 1 center-marker has been read. While the assumed prefix is being read, a is placed on the counter. This count is later compared to the assumed suffix. If the lengths are found to be the same, the network accepts. Hence, $L^8 \in NRTCN_1 \subseteq$ NRTCN. We assert that $L^8 \notin DRTCN_n$ for all n.

Assume there is some $drtcn_n$ such that $T(M) = L^8$. We know from lemma 2 that the number of equivalence classes of E_k (mod T(M)) is not greater than ck^n for some constant c. If for some k, the number of equivalence classes of E_k (mod L^8) is greater than this, our assumption that M accepts L^8 must be false.

Let k be odd and let $A = \{1,3,\ldots,k\}$, $\#(A) = k/2$. Let $A' \subseteq A$. There are $2^{k/2}$ such subsets of A. Suppose that for each A', we can find a $y \in \{0,1\}^*$ such that $y\{0,1\}^a \in L^8$ if and only if $a \in A'$. This would mean there are at least as many equivalence classes of $E_k \pmod{L^8}$ as there are subsets of A, namely $2^{k/2}$. We assert this is the case. Consider the following procedure to generate a y given an A': Let $A' = \{a_1, a_2, \ldots, a_p\}$, let $|y| = 2k$, and let there be exactly p 1's in y. The number of symbols of y preceding the ith 1 will be denoted m_i and computed by the formula

$$m_i = \frac{2k + a_i - 1}{2}, \quad 1 \le i \le p.$$

We assert that if $a \in A'$, then $w = y\{0,1\}^a \in L^8$. Let $a = a_i$, $1 \le i \le p$. $|y\{0,1\}^{a_i}| = 2k + a_i$. If w is to be a word of L^8, $|w|$ must be odd and the center symbol must be 1. Since a_i is odd, of course, $|w| = 2k + a_i$ is odd. The number of symbols preceding the center symbol must be $(2k + a_i - 1)/2$. But this is the number of symbols preceding a 1, since $m_i = (2k + a_i - 1)/2$. Therefore, $w \in L^8$. But it should be clear that if $w = y\{0,1\}^a \in L^8$, then $a \in A'$. This is because each word of L^8 has a 1 in the center position. The only 1's in w are the p 1's at positions $m_i + 1$ and possibly those in the suffix from $\{0,1\}^a$. The former lead to words $y\{0,1\}^{a_i} \in L^8$. The 1's in the suffix cannot act as center-markers, as they would lead to words of length at least $4k+1$. Thus, we have shown that $y\{0,1\}^a \in L^8$ if and only if $a \in A'$ and hence, there are at least $2^{k/2}$ equivalence classes of $E_k \pmod{L^8}$. For large k, $2^{k/2} > ck^n \ge \#(E_k \pmod{T(M)})$. Therefore, our assumption that some $drtcn_n$ accepts L^8 must be false, and we conclude that DRTCN $\subset$ NRTCN.

Q.E.D.

THEOREM 24: (a) DARTCN $\subset$ NARTCN.

(b) $DARTCN_n \subset NARTCN_n$, $n \geq 1$.

PROOF: (a) We know from the above theorem that $L^8 \notin$ DRTCN. Since DARTCN $\subset$ DLRTCN $\subset$ DRTCN, we must have $L^8 \notin$ DARTCN. But a $nrtcn_1$ is just a $nartcn_1$, so that $L^8 \in NRTCN_1$ (theorem 23) implies $L^8 \in NARTCN_1$. Thus, DARTCN $\subset$ NARTCN.

(b) Since $L^8 \in NARTCN_1$, $L^8 \in NARTCN_n$ for all n. But $L^8 \notin DARTCN_n$ for all n, so $DARTCN_n \subset$ NARTCN, $n \geq 1$.

Q.E.D.

THEOREM 25: (a) $NARTCN_1 = NRTCN_1$.

(b) NARTCN $\subset$ NRTCN.

(c) $NARTCN_n \subset NRTCN_n$, $n \geq 2$.

PROOF: (a) A $nartcn_1$ is a $nrtcn_1$ and vice versa.

(b) There is a $drtcn_2$ M such that $T(M) = L^9$, where

$$L^9 = \{0^{2^a} \mid a \geq 0\}.$$

Upon reading 0, M places 1 on the first counter and accepts. Upon reading additional 0's the first counter is decremented by 1 and the second counter is incremented by 2. Whenever a counter empties, M accepts and the roles of the counters are reversed. When the first word is accepted, one counter contains 1. Each successive time a counter empties thereafter, the count stored by the network has been multiplied by 2. Hence, $T(M) = L^9$. But we know that $DRTCN_2 \subset NRTCN_2 \subset$ NRTCN, so $L^9 \in DRTCN_2$ implies $L^9 \in NRTCN_2$ and $L^9 \in$ NRTCN.

Any language accepted by a $nartcn_n$ must be the intersection of n languages accepted by nartcn's, since the network accepts a word if and only if each isolated machine accepts that word. Furthermore, all the languages in $NARTCN_1$ are context-free. This is because a $nartcn_1$ is just a restricted pushdown automaton and because pushdown automata accept only context-free languages. Thus, L^9 must be the intersection of n context-free languages if L^9 is accepted by some $nartcn_n$. But Liu [40] has shown that L^9 is *not* the intersection of any finite number of context-free languages, so that $L^9 \notin$ NARTCN. Since $L^9 \in$ NRTCN, NARTCN $\subset$ NRTCN.

(c) $L^9 \in NRTCN_2$ means that $L^9 \in NARTCN_n$, $n \geq 2$. But $L^9 \notin NARTCN_n$, $n \geq 2$. Thus $NARTCN_n \subset NRTCN_n$, $n \geq 2$.

Q.E.D.

We have now shown that all the inclusions in theorems 19 and 20 are proper. We will now show some additional relations among the various acceptance classes and hierarchies.

THEOREM 26: (a) $DRTCN_1 \subset NARTCN_1$.

(b) $DRTCN_n$ and $NARTCN_n$, $n \geq 2$, are incomparable.

(c) DRTCN and NARTCN are incomparable.

PROOF: (a) Clearly, $DRTCN_1 \subseteq NARTCN_1$. In the proof of theorem 23, however, we showed that $L^8 \in NRTCN_1$ (hence, $L^8 \in NARTCN_1$), but $L^8 \notin$ DRTCN.

(b) $L^8 \notin DRTCN_n$, $n \geq 2$. We have shown that $L^9 \in DRTCN_2$, but $L^9 \notin NARTCN_n$, $n \geq 2$. Therefore it is true neither that $DRTCN_n \subset NARTCN_n$ nor that $NARTCN_n \subset DRTCN_n$, $n \geq 2$.

(c) $L^8 \in$ NARTCN, $L^8 \notin$ DRTCN, $L^9 \in$ DRTCN, $L^9 \notin$ NARTCN. Thus, neither DRTCN $\subset$ NARTCN nor NARTCN $\subset$ DRTCN.

Q.E.D.

THEOREM 27: (a) $\mathrm{DLRTCN}_1 \subset \mathrm{NARTCN}_1$.

(b) NARTCN $\not\subseteq$ DLRTCN.

(c) $\mathrm{NARTCN}_n \not\subseteq \mathrm{DLRTCN}_n$, $n \geq 2$.

PROOF: (a) A nartcn_1 is a generalization of a dlrtcn_1, so $\mathrm{DLRTCN}_1 \subseteq \mathrm{NARTCN}_1$. The inclusion is proper because $L^8 \in \mathrm{NARTCN}_1$, but $L^8 \notin \mathrm{DRTCN}_1 = \mathrm{DLRTCN}_1$.

(b) $L^8 \notin$ DRTCN. Since DLRTCN $\subset$ DRTCN, $L^8 \in$ NARTCN but $L^8 \notin$ DLRTCN.

(c) L^8 is in each NARTCN_n but in no DLRTCN_n.

Q.E.D.

We now address ourselves directly to the fundamental question: Given a rtcn_n M with machines connected in some particular way, what is the least acceptance class containing T(M)?

Suppose M is deterministic. If M has no connections, $T(M) \in \mathrm{DARTCN}_n$; if M has a linear structure, $T(M) \in \mathrm{DLRTCN}_n$; if M has a ring structure or a ring structure supplemented by additional connections, $T(M) \in \mathrm{DRTCN}_n$. As we now show, for $n \geq 2$, we may have $T(M) \notin \mathrm{DARTCN}_{n-1}$, $T(M) \notin \mathrm{DLRTCN}_{n-1}$, and $T(M) \notin \mathrm{DRTCN}_{n-1}$.

THEOREM 28: L_n^1 is accepted by some dartcn_n M'.

PROOF: Recall that $L_n^1 = \{0^{m_1} 1\, 0^{m_2} 1 \ldots 0^{m_n} B_i 0^{m_i} \mid 1 \leq i \leq n,\ m_j \geq 1,\ 1 \leq j \leq n\}$, $n \geq 1$.

M' operates as follows: As the subword preceding B_i is read, the kth machine $1 \leq k \leq n$, places m_k on its counter. When B_i is read, all machines except the ith enter and remain in a final state. The ith machine compares the number on its counter to the number of 0's following B_i. If and only if the numbers are equal, the machine enters a final state. Clearly, $T(M') = L_n^1$.

Q.E.D.

COROLLARY 2: Let M be a dartcn$_n$, $n \geq 2$. It may be the case that T(M) $\notin$ DARTCN$_{n-1}$, T(M) $\notin$ DLRTCN$_{n-1}$, and T(M) $\notin$ DRTCN$_{n-1}$.

PROOF: $L^1_n \in$ DRTCN$_n$ but $L^1_n \notin$ DRTCN$_{n-1}$.

Q.E.D.

Corollary 2 is particularly significant in light of the fact that we have found three distinct deterministic hierarchies. The L^1 languages, which have been used in the literature to establish an infinite hierarchy of drtcm$_n$'s could just as easily be used to establish the linear or atomized hierarchies. This confirms the intuition that the number of counters of such a machine is a fundamental measure of its recognition capabilities. At the same time, however, it is clear that the L^1 languages fail to distinguish between machines whose internal operations are significantly different. The L^1 languages characterize the DARTCN hierarchy better than the DLRTCN or DRTCN hierarchies. Likewise, the L^6 languages characterize the DLRTCN hierarchy better than the DRTCN hierarchy. It was in order to characterize better the real-time multipushdown and multicounter languages that the network formulation was developed. It would appear that this formulation *does* allow us to isolate meaningful acceptance classes whose existence we would not otherwise suspect. The nontriviality of the interrelations of these hierarchies may be emphasized by noting what is not clearly shown by corollary 2, namely that for $n \geq 2$, DARTCN$_n$ and DLRTCN$_{n-1}$, DARTCN$_n$ and DRTCN$_{n-1}$, and DLRTCN$_n$ and DRTCN$_{n-1}$ are incomparable.

Are there any major deterministic hierarchies we have missed? Certainly we have not yet classified all possible network structures. The following theorem suggests our three deterministic hierarchies are the most important ones.

THEOREM 29: Let C^{**}_{ij} be the transitive closure of C_{ij}, and let M be a drtcn$_n$. If C^{**}_{ii} for no $0 \leq i \leq n$, then T(M) DLRTCN$_n$. Otherwise, it may be the case that T(M) $\in$ $DRTCN_n - DLRTCN_n$.

PROOF: If C^{**}_{ii} for some i, there is a ring structure involving two or more machines embedded in the structure of M. We have shown that L^7 is in $DRTCN_n$, $n \geq 2$, but not in DLRTCN. Clearly, L^7 can be accepted by some network with the structure of M. (Machines not in the ring merely remain in final states, and those in the ring accept L^7.) Thus, it may be the case that T(M) $\in$ $DRTCN_n - DLRTCN_n$.

If C^{**}_{ii} for no $0 \leq i \leq n$, the structure of M must have no closed loops. Thus, C^{**}_{ij}, $1 \leq i,j \leq n$, implies not $-C^{**}_{ji}$. This means that C** is a transitive and asymmetric relation, sometimes called a strict partial ordering. It is known that a partial ordering may be embedded in a linear ordering, a partial ordering with the additional property that for any x and y in the field of the relation, either x is related to y or y is related to x. This embedding may be carried out algorithmically in a process called "topological sorting." (See [38] for discussion of such an embedding.) We may describe C^{**}_{ij} as meaning "machine i precedes machine j in M." In other words, there is some linear chain of machine connections from machine i to machine j. To say that partial order C** can be embedded in a linear order R is to say that (1) for all $1 \leq i,j \geq n$, either R_{ij} or R_{ji} (but not both) and (2) C^{**}_{ij} implies R_{ij}. But R is exactly the same kind of ordering we encounter in a linear network. Thus, M must be equivalent to a linear network constructed by placing its component machines in a linear order such that one machine precedes another in the ordering if its corresponding machine in M is connected to the corresponding machine of the other. Topological sorting may place machines in the linear network between corresponding connected machines of M. We have seen, however, that each machine of a linear network can "know" effectively information about all machines preceding it. Thus, we must have $T(M) \in DLRTCN_n$.

Q.E.D.

We now summarize what we know about deterministic real-time counter networks: There are three overlapping but distinct infinite hierarchies — one arising from machines having no connections whatever, one arising from machines having connnections with no closed loops, and one arising from machines whose connections include closed loops. Adding connections to an atomized structure in general adds to computing power. Doing so to a linear structure produces a more powerful structure only if closed loops are created thereby. Adding connections to a ring structure does not increase computing power. Adding additional machines to any structure produces a more powerful structure.

Suppose M is nondeterministic. If M has no connections, $T(M) \in NARTCN_n$; if M has a connected structure, $T(M) \in NRTCN_n$. The set of languages used in the literature to show the NRTCN hierarchy, the L^2 languages, could be used to demonstrate the existence of either this hierarchy of the NARTCN hierarchy. This leads to the following theorem.

THEOREM 30: Let M be a $nartcn_n$, $n \geq 2$. It may be the case that $T(M) \notin NRTCN_{n-1}$.

PROOF: Clearly L_n^2 may be accepted by a $nartcn_n$. But we have shown that $L_n^2 \in NRTCN_n$ and that $L_n^2 \notin NRTCN_{n-1}$.

Q.E.D.

From theorem 30, we see that for $n \geq 2$, $NARTCN_n$ and $NRTCN_{n-1}$ are incomparable. We now prove one last theorem about nondeterministic real-time multicounter networks.

THEOREM 31: Let M be a $nrtcn_n$. If M has at least one connection, it may be the case that $T(M) \in NRTCN_n - NARTCN_n$.

PROOF: If M has one connection, two connected machines can accept L^9 and the remaining machines can stay in final states. Since $L^9 \notin NARTCN$, we may have $T(M) \in NRTCN_n - NARTCN_n$.

Q.E.D.

Summarizing, we may say that there are two distinct infinite hierarchies in the nondeterministic case, one contained in the other — one arising from machines having no connections and one arising from machines with a connected structure. Adding connections to an atomized structure produces a more powerful structure. Doing so to a connected structure, however, is redundant. Adding additional machines to any structure produces a more powerful structure.

We conclude this chapter with some informal remarks to suggest what various counter networks intuitively can and cannot do.

Lack of connections prevent machines from sharing information. Since information stored on a counter must be removed to be used, lack of connections prevent information from being saved for future use. Thus, $L_2^6 \notin DARTCN$. For deterministic machines, linear connections *do* allow information to be used up to a fixed number of times. Any connection which adds a closed loop, allows unlimited use of stored information and, in general, takes the acceptance set out of the linear hierarchy. Hence, $L_n^6 \in DLRTCN_n$, but $L^7 \notin DLRTCN$. The number of counters limits the amount of information a network can store. If the amount of information that must be stored *at one time* is limited, however, nondeterminism may be substitutable for additional counters. We have $L_{n,p}^3 \in NRTCN_n$, $n < p$, but although $L^3_{n,p} \in DRTCN_p$, $L^3_{n,p} \notin DRTCN_n$. Nondeterminism *cannot* be *generally* substituted for additional counters, however, as we see in the proof of theorem 13. Nor can nondeterminism generally replace connections, as we see in theorem 26.

This last remark leads us to look at one unresolved matter relating to the real-time counter network hierarchies, the relation of NARTCN and DLRTCN. We have seen in theorem 27, that DARTCN $\not\subseteq$ DLRTCN and that for $n \geq 2$, $NARTCN_n \not\subseteq DLRTCN_n$. Is it the case that for $n \geq 2$, $DLRTCN_n \subset NARTCN_n$ and thus DLRTCN $\subset$ NARTCN; or is it the case that DLRTCN is incomparable with NARTCN, and $DLRTCN_n$ is incomparable with $NARTCN_n$, $n \geq 2$? If DLRTCN $\subset$ NARTCN, nondeterminism *can* serve in lieu of connections under some rather general conditions. This possibility becomes plausible when one looks for counterexamples and finds them difficult to come by. The L^6 languages will not do the job, as they may be shown to be in NARTCN. In all probability, there *is* a counterexample, however. We conjecture the following language is one, in fact:

$$L^{10} = \{w_1 2 w_2 \mid w_1 \in \{0,1\}^*,\ w_2 = 0^z,\ z = \#(Z(w_1))\},$$

where $w_1 = s_1 s_2 \ldots s_n,\ n \geq 0,\ |w_1| = n$,

$Z(w) = \{i \mid 1 \leq i \leq n,\ T_i(w) = 0\}$, and

$$T_i(w) = \begin{cases} 0 & \text{if } i = 0 \text{ or if } T_{i=1}(w) = 0,\ s_i = 1,\ 1 \leq i \leq n \\ T_{i-1}(w) - 1 & \text{if } T_{i-1}(w) > 0,\ s_i = 1,\ 1 \leq i \leq n \\ T_{i-1}(w) + 1 & \text{if } s_i = 0,\ 1 \leq i \leq n \end{cases}$$

Rather than describe L^{10}, we describe how it is accepted by a $dlrtcn_2$ M. As w_1 is read, the first counter is incremented by 1 for each

0 read and decremented by 1 for each 1 read. If the counter contains 0, it is not decremented. For each symbol read that results in a count of 0 on the first counter, the second counter is incremented by 1. After 2 is encountered in the input string, the second machine compares the number of 0's seen to the number on the second counter. If these numbers are equal, the word is accepted.

The suffix of a word of L^{10}, then, depends upon characteristics of the prefix which appear to require a counter to recognize. Since this recognition also appears to require the counter to increase and decrease in length, it does not seem that this recognition and the counting associated with it can be done using the same machine. That is, the connection in M is probably required. It is interesting to note that L^{10} is a deterministic context-free language.

CHAPTER IV

SUGGESTED RESEARCH

The use of the network formulation of real-time multicounter machines allows the identification of several hierarchies which seem very "natural." The number and nature of connections within a network does appear to be a reasonable index of internal complexity. This suggests that the network approach may lead to additional results when applied to other multistore machines. For example, some results have been shown for real-time pushdown networks. It is reasonable to conjecture that other analogs of the theorems of Chapter III may be proved for such networks. The existence of a deterministic linear real-time pushdown network hierarchy would not be at all surprising. Results such as this could be interesting in themselves, but they may, in addition, shed light on the power of connections between automata generally. We would like to answer such questions as what is the utility of additional connections within a network irrespective of the type of auxiliary stores involved.

A more complete theory of automata networks must await such further studies; the present research can be considered only preliminary. In particular, the development of the theory should include better linguistic characterization of the languages accepted by networks. We would like theorems about the closure properties of such languages under various operations. Ideally, we would also like to find formal grammars which exactly capture various acceptance classes. Automata networks will be intellectually more attractive if they are shown to reflect, in a "natural" way, significant linguistic properties. Many closure results are easily enough obtained, although their contribution to the theory is unclear at this stage of development. As an example, we may note that each of the real-time multicounter automata hierarchies we have shown, *except* the deterministic atomized one, is closed under union.

It has been emphasized that studying multistore automata as networks allows us to see properties of machines not otherwise obvious. We may go beyond this by noting that this technique allows us to restrict machines in certain ways in order to study variants not otherwise of interest. For example, multipushdown and multicounter machines always have been studied under some time restriction because the unrestricted machines are equivalent to Turing machines. For certain network structures, however, we may remove this restriction. Deterministic linear multicounter networks may be studied without any time restriction, as it may be shown that no $dlcn_n$ accepts L^7. Furthermore, upon developing appropriate definitions, it would appear to be meaningful to examine such networks having multiple input heads or two-way input heads.

The possibility of gaining insight into computer networks or parallel processing by studying automata networks should not be overlooked. This view is hinted at by Burkhard and Varaiya [9] but not developed systematically. Automata networks may provide a more realistic model of parallel computation than other forms of polyautomata such as tessellation automata.

REFERENCES

1. Aanderaa, S. On k-tape versus (k-1)-tape real-time computation. *SIAM-AMS Proc.* 7 (1974), 75-96.

2. Aho, A., J. Hopcroft, and J. Ullman. Time and tape complexity of pushdown automaton languages. *Inf. and Control* 13 (1968), 186-206.

3. Aho, A., J. Hopcroft, and J. Ullman. On the computational power of pushdown automata. *J. Computer and Syst. Sci.* 4 (1970), 129-136.

4. Baker, B. and R. Book. Reversal-bounded multipushdown machines. To appear in *J. Computer and Syst. Sci.*

5. Book, R. Personal communication.

6. Book, R. and S. Ginsburg. Multi-stack-counter languages. *Math. Syst. Theory* 6 (1972), 37-48.

7. Book, R. and S. Greibach. Quasi-realtime languages. *Math Syst. Theory* 4 (1970), 97-111.

8. Book, R., M. Nivat, and M. Paterson. Intersections of linear context-free languages and reversal-bounded multipushdown machines. *Sixth Annual ACM Symp. on Theory of Computing* (1974), 290-298.

9. Burkhard, W. and P. Varaiya. Complexity problems in real time languages. *Inf. Sciences* 3 (1971), 87-100.

10. Burks, A., D. Warren, and J. Wright. An analysis of a logical machine using parenthesis-free notation. *Math. Tables and Other Aids to Computation* 8 (1954), 53-57.

11. Chomsky, N. Context-free grammars and push-down storage, *MIT Res. Lab. Electron. Quart. Progress Rep.* 65 (1962), 187-194.

12. Cole, S. Deterministic pushdown store machines and real-time computation. *J. ACM* 18 (1971), 306-328.

13. Cook, S. Characterizations of pushdown machines in terms of time-bounded computers. *J. ACM* 18 (1971) 4-18.

14. Cook, S. Linear time simulation of two-way pushdown automata. *Proc. IFIP Congress 71* 1 (1972), 75-80.

15. Evey, J. *The theory and application of pushdown store machines.* Doctoral thesis, Harvard University, Cambridge, Mass., 1963.

16. Fischer, P. On computability by certain classes of restricted Turing machines. *Proc. 4th Annual Symp. on Switching Circuit Theory and Logical Design* (1963), 23-32.

17. Fischer, P. Turing machines with restricted memory access. *Inf. and Control* 9 (1966), 364-379.

18. Fischer, P., A. Meyer, and A. Rosenberg. Counter machines and counter languages. *Math. Syst. Theory* 2 (1968), 265-283.

19. Ginsburg, S. *The mathematical theory of context-free languages.* McGraw-Hill, New York, 1966.

20. Ginsburg, S. and S. Greibach. Deterministic context free languages. *Inf. and Control* 9 (1966), 620-648.

21. Ginsburg, S., S. Greibach, and M. Harrison. Stack automata and compiling. *J. ACM* 14 (1967), 172-201.

22. Ginsburg, S. and F. Rose. The equivalence of stack-counter acceptors and quasi-realtime stack-counter acceptors. *J. Computer and Syst. Sci.* 8 (1974), 243-269.

23. Ginsburg, S. and E. Spanier. Finite-turn pushdown automata. *SIAM J. on Control* 4 (1966), 429-453.

24. Gray, J., M. Harrison, and O. Ibarra. Two-way pushdown automata. *Inf. and Control* 11 (1967), 30-70.

25. Gwynn, J. and D. Martin. Two results concerning the power of two-way deterministic pushdown automata. *Proc. ACM National Conf.* (1973), 342-345.

26. Haines, L. *Generation and recognition of formal languages.* Doctoral thesis, MIT, Cambridge, Mass., 1965.

27. Harrison, M. and I. Havel. Real-time strict deterministic languages. *SIAM J. on Computing* (1972), 333-349.

28. Harrison, M. and I. Havel. Strict deterministic grammars. *J. Computer and Syst. Sci.* (1973), 237-277.

29. Harrison, M. and O. Ibarra. Multi-tape and multi-head pushdown automata. *Inf. and Control* 13 (1968), 433-470.

30. Hopcroft, J. and J. Ullman. *Formal languages and their relation to automata.* Addison-Wesley, Reading, Mass., 1969.

31. Ibarra, O. Controlled pushdown automata. *Inf. Sciences* 6 (1973), 327-342.

32. Ibarra, O. On two-way multihead automata. *J. Computer and Syst. Sci.* 7 (1973), 28-36.

33. Igarashi, Y. and N. Honda. Deterministic multitape automata computations. *J. Computer and Syst. Sci.* 8 (1974), 167-189.

34. Kain, R. *Automata theory: machines and languages.* McGraw-Hill, New York, 1972.

35. Kameda, T. Pushdown automata with counters. *J. Computer and Syst. Sci.* 6 (1972), 138-150.

36. Korenjak, A. and J. Hopcroft. Simple deterministic languages. *Seventh Annual Symp. on Switching and Automata Theory* (1966), 36-46.

37. Knith, D. On the translation of languages from left to right. *Inf. and Control* 8 (1965), 607-639.

38. Knith, D. *The art of computer programming*, vol. 1 Addison-Wesley, Reading, Mass., 1968.

39. Laing, R. Realization and complexity of cumulative events. *U. of Mich. Technical Rep. 03105-48-T*, 1967.

40. Liu, L. *Finite-reversal pushdown automata.* Doctoral thesis, Princeton University, Princeton, N. J., 1968.

41. Liu, L. and P. Weiner. An infinite hierarchy of intersections of context-free languages. *Math. Syst. Theory* 7 (1973), 185-192.

42. Martin, D. *Universal multihead automata.* Doctoral thesis, Ga. Institute of Technology, Atlanta, Georgia, 1974.

43. Minsky, M. Recursive unsolvability of Post's problem of "Tag" and other topics in the theory of Turing machines. *Annals of Math.* 74 (1961), 437-455.

44. Newell, A. and J. Shaw. Programming the logic theory machine. *Proc. West Joint Computer Conf.* (1957), 230-240.

45. Newell, A., J. Shaw, and H. Simon. Empirical explorations of the logic theory machine: a case study in heuristic. *Proc. West. Joint Computer Conf.* (1957), 218-230.

46. Oettinger, A. Automatic syntactic analysis and the pushdown store. *Proc. Symp. Applied Math.* 12 (1961), 104-129.

47. Rabin, M. Real time computation. *Israel J. Math.* 1 (1963), 203-211.

48. Rabin, O. and D. Scott. Finite automata and their decision problems. *IBM J. of Res. and Dev.* 3 (1959), 114-125.

49. Rosenberg, A. Real-time definable languages. *J. ACM* 14 (1967), 645-662.

50. Rovan, B. Bounded push down automata. *Kybernetika* 4 (1969), 261-265.

51. Samelson, K. and F. Bauer. Sequential formula translation. *Comm. ACM* 3 (1960), 76-83.

52. Schmitt, A. The state complexity of Turing machines. *Inf. and Control* 17 (1970), 217-225.

53. Schützenberger, M. Finite counting automata. *Inf. and Control* 5 (1962), 91-107.

54. Schützenberger, M. On context-free languages and push-down automata. *Inf. and Control* 6 (1963), 246-264.

55. Shannon, C. A universal Turing machine with two internal states, in *Automata studies*, C. Shannon and J. McCarthy, eds. Princeton University Press, Princeton, N. J., 1956.

56. Yamada, H. Real-time computation and recursive functions not real-time computable. *IRE Trans. on Electronic Computers* 11 (1962), 753-760.

VITA

Lionel Earl Deimel, Jr. was born in New Orleans, Louisiana on October 28, 1946. After graduating from Bemjamin Franklin High School in New Orleans in 1964, he attended the University of Chicago, from which he was awarded a Bachelor of Arts degree in physics in 1968. He entered Georgia Institute of Technology the same year, but left in 1969 to enter military service. He served as a bandsman in the U. S. Army at Ft. McPherson, Georgia and Ft. Shafter, Hawaii. He was separated from the Army with the rank of Specialist Fifth Class in 1971 and returned to Georgia Tech to resume his studies. He received a Master of Science degree and a Doctor of Philosophy degree in Information and Computer Science in 1972 and 1975, respectively.

Mr. Deimel has had a paper published in *Information Processing Letters* and has presented a paper at the ACM Computer Science Conference '75 in Washington, D.C. While at Georgia Tech, he has served successively as a Research Assistant, Teaching Assistant, and Instructor. He has accepted a faculty appointment in the Computer Science Department of North Carolina State University at Raleigh, beginning August, 1975.